Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем факультету інформатики

“Реалізація програмного застосунку digital wallet для зберігання та перевірки електронних записів (Verifiable Credentials), випущених на платформі European Blockchain Services Infrastructure”

Текстова частина до курсової роботи
за спеціальністю «Комп’ютерні науки» - 122

Керівник курсової роботи
старший викладач
Гороховський К.С.
_____ (Підпис)
“___” _____ 2024 року

Виконав студент
КН-3 Микитишин А. П.
“___” _____ 2024 року

Київ 2024

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем факультету інформатики

ЗАТВЕРДЖУЮ
старший викладач
_____ Гороховський К.С.
„_____” _____ 2024 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на курсову роботу

студенту Микитишину Артему Павловичу
факультету інформатики 3 курсу бакалаврської програми

ТЕМА: **Реалізація програмного застосунку digital wallet для зберігання та перевірки електронних записів (Verifiable Credentials), випущених на платформі European Blockchain Services Infrastructure**

Зміст ТЧ до курсової роботи:

Abstract
Introduction
Chapter 1. Theoretical concepts
Chapter 2. European approach: a case study
Chapter 3. Sample issuer wallet implementation
Conclusion
Bibliography
Appendices

Дата видачі „___” _____ 2024 р.
Керівник_____
(підпис)
Завдання отримав_____
(підпис)

Календарний план виконання роботи

№	Назва етапу курсової роботи	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу	Листопад 2023	
2.	Огляд літератури на теоретичні теми	Січень 2024	
3.	Написання першого розділу	Лютий 2024	
4.	Огляд джерел на практичні теми	Березень 2024	
5.	Написання другого розділу	Квітень 2024	
6.	Планування практичної частини і вибір засобів розробки	Квітень 2024	
7.	Реалізація практичної частини	Травень 2024	
8.	Написання третього розділу	Травень 2024	
9.	Надання роботи науковому керівнику для перевірки	Травень 2024	
10.	Корегування роботи відповідно до зауважень	14.05.2024	
11.	Захист курсової роботи	21.05.2024	

Студент ____Микитишин А. П._____

Керівник ___Гороховський К. С._____

“_____”_____ 2024

Київ 2024

Table of contents

ABSTRACT **6**

INTRODUCTION **7**

CHAPTER 1. THEORETICAL CONCEPTS **9**

1.1. Digital Identity **9**

1.2. Approaches to digital identity **9**

1.2.1. Centralized 9

1.2.2. Federated 10

1.2.3. Decentralized 10

1.2. Self-Sovereign Identity **11**

1.3. Verifiable credentials **11**

1.4. Verifiable presentations **13**

1.5. Decentralized identifiers (DIDs) **15**

1.6. Trust model **17**

1.7. Digital wallets **19**

CHAPTER 2. EUROPEAN APPROACH: A CASE STUDY **21**

2.1. EUROPEAN BLOCKCHAIN SERVICES INFRASTRUCTURE **21**

2.2.1. EBSI’s core technical services 23

2.2.2. EBSI Trust Model 23

2.2.3. VC issuance process 26

2.2. EU Digital Identity Wallet **29**

CHAPTER 3. SAMPLE ISSUER WALLET IMPLEMENTATION **31**

3.1. Specification **31**

3.1.1. API endpoints 32

3.1.2. Front-end specification 32

3.2. Issuer API implementation **33**

3.3. Front-end implementation **33**

CONCLUSION **35**

BIBLIOGRAPHY **36**

APPENDIX A **42**

APPENDIX B **47**

Abstract

This thesis examines the implementation of a digital issuer wallet that is compliant with the European Blockchain Services Infrastructure (EBSI). The study begins with an exploration of concepts related to digital identities, verifiable credentials, verifiable presentations, decentralized identifiers (DIDs), decentralized trust models, and digital wallets. We present a detailed case study on the European approach regarding decentralized digital identities, particularly EBSI and the European Digital Identity (EUDI) wallet. We analyze the standards and trust model EBSI relies upon and explore the specifications of its issuance flow. The practical component of this work includes the development of a simplified issuer implementation, consisting of a minimal issuer API and a website enabling the issuance of verifiable credentials that follow the EBSI standards, utilizing Node.js and VC issuance libraries for the backend and Flutter for frontend development.

Introduction

In today's world, with the rapid development of digital technologies, reliable identity verification is becoming increasingly important. As nearly all areas of our lives—from banking to healthcare—become digitized, a rethinking of traditional approaches to identification is required. Centralized identification methods controlled by large corporations and state governments often face various problems of confidentiality, data security, and interoperability.

In dealing with these challenges, decentralized identity systems come into play. They are based on a distributed ledger technology that gives every user personal data ownership rights: an innovative model in this area. This method can solve issues typical for centralized data systems and helps people regain control over their digital identity while also decreasing the operating costs of many identification processes.

Under this pretext, the European Commission launched the European Blockchain Services Infrastructure (EBSI), a project aimed at unifying Europe's approach to digital identification. An important arm of this initiative is the development of digital wallets that adhere to these standards and make practical use of decentralized digital identities. Yet, creating such systems faces significant technical and regulatory hurdles that cannot be overcome without detailed study or innovative methods.

This research aims to explore the topic of decentralized digital identities and EBSI's approach to them in detail. Moreover, it sets out to develop a functioning prototype of an issuer wallet that meets EBSI standards and solves the problems of existing centralized systems by providing a reliable and efficient way to create, sign, and share digitally verifiable credentials. This work is of great importance as it

affects the broader use of verifiable digital IDs in people's everyday lives, helping to improve digital privacy and security in Europe.

The novelty of this research lies in integrating theoretical knowledge with practical application by creating a working prototype that demonstrates the potential and benefits of decentralized digital identification in the real world. The results of this research are helpful for the future development of digital identification technologies and can serve as a basis for further innovations in this area.

Chapter 1. Theoretical Concepts

1.1. Digital Identity

A digital identity is often defined as a digital reference to a person. [1] While an identity can be viewed as a description of a person or object, a digital identity is the collection of data about someone or something available in a digital format. For example, a credit card number, a university diploma, and an email address can all be attributes that together form a person's digital identity.

The notion of digital identity is frequently used in the context of security as a means of identification or authentication. For example, a person might need to prove ownership of their email address to access a website, or an institution might request a digital diploma to verify the person's qualifications.

It is important to note that a subject of a digital identity need not be a human. It can be virtually anything: an organization, a pet, a building, or a product. For example, a product sold at a store may have a digital identity that customers can use to verify its origin and manufacture date.

1.2. Approaches to digital identity

Currently, there are 3 main approaches to digital identity, according to the report at the 2018 World Economic Forum: centralized, federated, and decentralized. [2]

1.2.1. Centralized

The one in which all the identity data is stored and managed by one organization. Such organizations can be governmental or private (often banks and social media). Examples include Ukrainian Diia, Estonian eID, and Facebook log-

in. This approach is also referred to as the “siloed identity”. [1] The main strength of this model is its widespread adoption. Its main weaknesses are low user control, different standards across systems, and a high risk of data leakage.

1.2.2. Federated

The one in which multiple centralized systems are linked together, establishing mutual trust and becoming trust anchors. For example, the International Civil Aviation Organization provides standards for international cross-border travel, that, when implemented by different parties, enable mutual acceptance of digital identity systems. [3] This approach also allows for partial repurposing of credentials. For example, a driver’s license whose original purpose was demonstrating the ability to drive a vehicle may be used as proof of age. The most important advantage of this approach is wider access to services for users compared to the centralized model. All problems of the centralized approach are still relevant to the federated approach. On top of that, a new issue is the difficulty of establishing legal agreements between organizations that slows down the evolution of such systems.

1.2.3. Decentralized

The one in which the users control their identity by storing all their attestations (or credentials) in a data store (a digital wallet) on their device. Traditional trust anchors discussed above simply provide these credentials but have no control over them. The individual then chooses which attestations to disclose and to whom. [4] The main benefits of this model are increased user control over their data, easier identity management, and a decreased amount of information shared by users. The downsides of this approach include low adoption (mostly pilots and proof-of-concepts at the moment), challenges of offline access, underlying system complexity, and legal challenges of adoption.

We will focus mostly on the decentralized approach as it is the one EBSI is part of.

1.2. Self-Sovereign Identity

"Self-Sovereign Identity" (SSI) is a vague term, often used interchangeably with "decentralized identity". A. Preukschat and D. Reed provide the following "literal translation" of the term "self-sovereign identity" in their cognominal book: *A person's identity that is neither dependent on nor subjected to any other power or state.* [5] Even though efforts have been made towards strictly defining SSI (most notably, [6] and [7]), there is still a lot of controversy surrounding the status quo of this name [1]. Therefore, this work instead uses the term "decentralized digital identity" whenever applicable.

1.3. Verifiable credentials

In everyday life, people use credentials to prove some claims about themselves. A passport issued by a state's government proves the person's citizenship, a diploma proves an educational degree, and a driver's license proves the ability to drive motor vehicles. The authors of the "Self-Sovereign Identity" book [5] state that "the term *credentials* extends to any (tamper-resistant) set of information that some authority claims to be true about the subject of the credential—and which in turn enables the subject to convince others (who trust that authority) of these truths."

It is worth noting that, as with digital identity, the subject of a credential need not be a human. For example, a manufacturer may issue credentials about the contents of their product, or a veterinarian may issue credentials about a pet's vaccinations.

The common terms used in the context of credentials and verifiable credentials are “issuer”, “subject”, “holder”, and “verifier”. An issuer is an authority that produces the credentials. A subject is an entity (someone or something) a credential is about. A holder is a person or an organization to which the credential is issued. Most often the subject and the holder are the same entity, but this may not be the case, as in the example of pet vaccinations. A verifier is an organization that wants to verify the validity of a credential.

Credentials contain claims that state some assertions about the subject of the credential. Typically these assertions are about entitlements (education degree, medical benefits), relationships (citizenship, marriage), or attributes (name, age).

For verifiers to trust the credentials, they must be verifiable. The verifiers need to determine the issuer of the credential, its authenticity (i.e. that it hasn’t been tampered with after being issued), and that it hasn’t expired or been revoked. Although physical credentials provide some proof of legitimacy in the form of watermarks and stamps, they can be forged and are thus not tamper-proof. Another way to verify physical credentials is to contact the issuer directly, but this approach is usually costly in terms of time and money, and sometimes even unfeasible.

To address the issues with physical credentials and digitize them, the concept of verifiable credentials (VCs) has emerged. The World Wide Web Consortium (W3C), an organization defining standards for technologies related to the web, has issued a formal Recommendation regarding Verifiable Credentials. In its most recent version [8], a verifiable credential is defined as “[…] a tamper-evident credential that has authorship that can be cryptographically verified.” Thus, a VC is a much more reliable and practical form of a credential.

The same W3C Recommendation governs the structure of a VC. [8] It includes credential metadata, claims, and proofs. The metadata describes the VC itself - primarily its unique identifier, but also its issuer, expiry date, revocation method, etc. Claims are the most important part of the VC and are essentially the same as claims in other types of credentials. Proofs are mechanisms for cryptographically verifying the credential. Usually, there is only one proof - a digital signature of the issuer. Below is a figure depicting how these elements of a VC correspond to elements on a physical credential - a US driver's license. [5] Note that this particular physical credential does not have an equivalent of an issuer signature or proof, which can be found in the form of a watermark or a seal on other physical credentials.

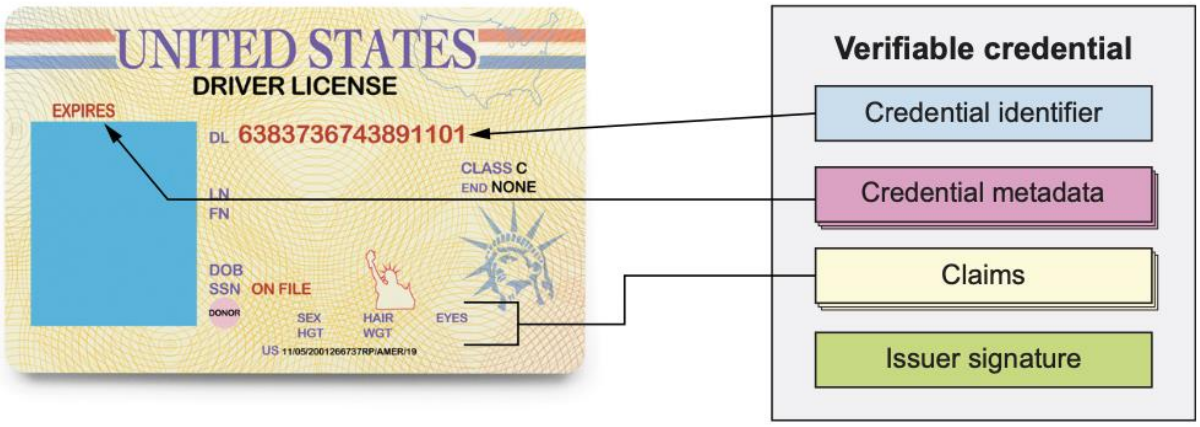

Figure 1-1. A mapping between the elements of a verifiable credential and a physical credential.

1.4. Verifiable presentations

One of the biggest advantages of the decentralized approach to digital identity is selective disclosure - the ability for the user to disclose only the relevant parts of personal information, thus increasing privacy. Verifiable credentials have a special mechanism for this - verifiable presentations (VPs). As described in W3C's draft

recommendation [8]: “The expression of a subset of one's persona is called a verifiable presentation. Examples of different personas include a person's professional persona, their online gaming persona, their family persona, or an incognito persona.”

Essentially, they are just wrappers for VCs created by the holder with the sole purpose of being sent to the verifier. A verifiable presentation contains its metadata (context, type, ID), a set of VCs, a set of proofs, and potentially arbitrary additional data. The general structure can seen in the picture from [5]:

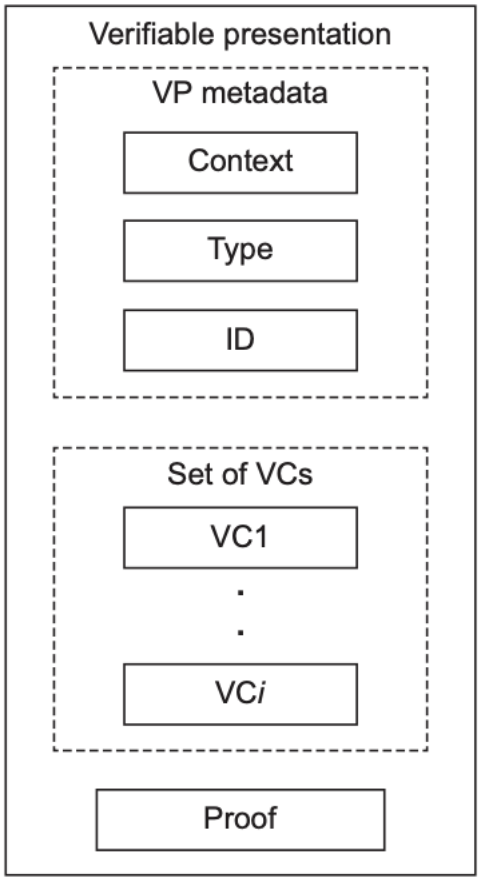

Figure 1-2. Basic VP structure

1.5. Decentralized identifiers (DIDs)

Just like subjects within centralized systems can be identified using a system-level unique identifier (like a Social Security Number, a phone number, or an email address), some unique identifier must also exist for use within decentralized systems. Such an identifier is called a Decentralized Identifier (DID) and is again formally described in a W3C Recommendation [9]. The document defines DID as "a globally unique persistent identifier that does not require a centralized registration authority and is often generated and/or registered cryptographically". Simply put, a DID is just a string that uniquely identifies one subject (i.e. a person, organization, or a thing) and links to more information about the subject, like a key in a hash table. DIDs are generated by their owners using any compliant system of the owner's choice.

According to the W3C Recommendation [9], the DIDs are part of a larger VC ecosystem and were designed with the following goals in mind: decentralization, control, privacy, security, proof-based, discoverability, interoperability, portability, simplicity, and extensibility. Another advantage of DIDs is their permanent nature - an identifier must never change regardless of any changes regarding its subject. [5]

Because DIDs are a part of the decentralized identity framework, they offer most of the same benefits that decentralized systems have over federated and centralized ones. Decentralized identifiers are issued and managed by their owners, not by third-party organizations. They are much simpler to use because a user of a decentralized system may have as little as one DID to use across all services, instead of a separate identifier for each centralized or federated system. Moreover, a user may have as many DIDs as they want to completely separate their personas, or identities, from each other, which is particularly difficult, if at all possible, in

centralized and federated systems. Lastly, because DIDs are controlled by their owners, they are much less prone to identity theft and are thus much more secure.

Interestingly, the W3C specification [9] does not restrict the implementation to any specific technologies, thus making it possible to create DIDs based on some preexisting identifiers in centralized or federated systems, like passport numbers, creating a pathway for bridging the different approaches.

"A DID is a simple text string consisting of three parts: 1) the `did` URI scheme identifier, 2) the identifier for the DID method, and 3) the DID method-specific identifier." [9] The DID method is the specific implementation of the DID. It is used in the DID string to define its syntax. The DID is linked to a DID document, a set of data describing the subject of the DID, most notably, the ways to authenticate the subject cryptographically. Below is a simple example of a DID and its corresponding DID document (written in JSON-LD [10]), again taken from [9]:

Figure 1-3. DID example

```
{
  "@context": [
    "https://www.w3.org/ns/did/v1",
    "https://w3id.org/security/suites/ed25519-2020/v1"
  ]
  "id": "did:e
  "authentication
    // used to authenticate as did:...fghi
    "id": "did:example:123456789abcdefghi#keys-1",
    "type": "Ed25519VerificationKey2020",
    "controller": "did:example:123456789abcdefghi",
    "publicKeyMultibase": "zH3C2AVvLMv6gmMNam3uVAjZpfkcJCwDwnZn6z3wXmqPV"
  }]
}
```

Figure 1-4. Corresponding DID document example

1.6. Trust model

For issuers, holders, and verifiers to effectively communicate in a digital system, some trust between these parties must exist. A model that describes such trust is often visualized as the "trust triangle" with nodes representing issuers, identity holders, and verifiers. For example, see this picture below from "A Tutorial on the Interoperability of Self-sovereign Identities" [11]. It depicts the 3 main actors, their interactions, and the examples of actors.

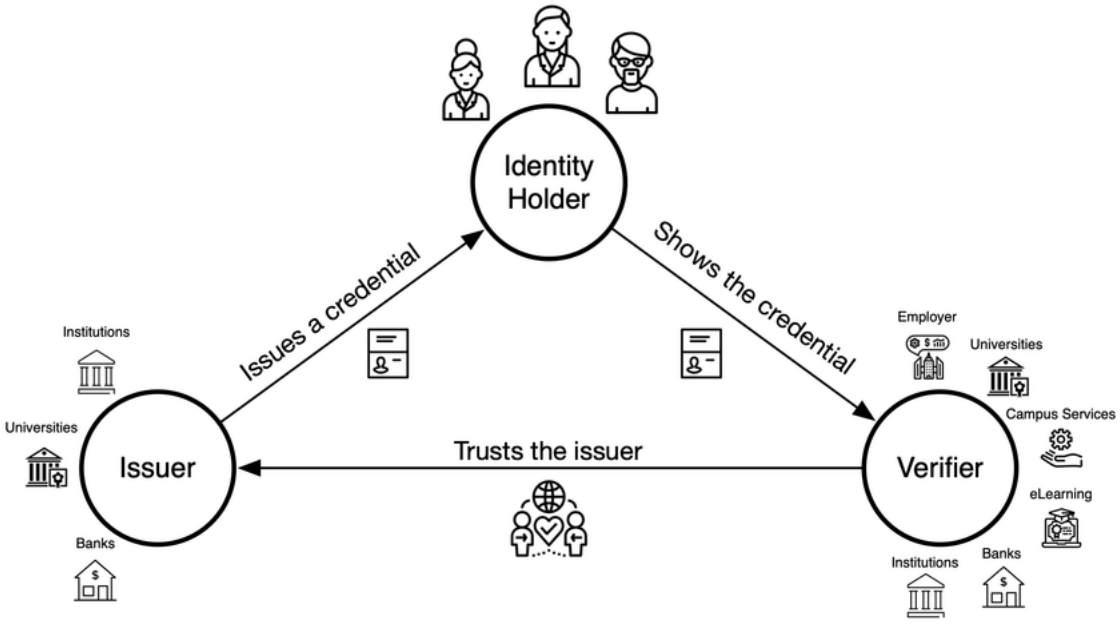

Figure 1-5. Identity trust triangle

It is important to note that the above schema applies to all identity verification processes, not limited to the digital decentralized identity model. However, in a decentralized digital system, it is virtually impossible for verifiers to directly know and trust every single issuer. The challenge of guaranteeing trust between verifiers and issuers without their direct contact is at the core of the decentralized identity

model. The most common solution to it is to have issuers and verifiers communicate indirectly via a blockchain. One simple definition of blockchain is a "highly tamper-resistant transactional distributed database that no single party controls" [5]. Its distributive and secure nature makes it perfect for a decentralized identity system. W3C Recommendation on VCs [8] generalizes blockchain to a verifiable data registry. Below is the picture from the "Self-Sovereign Identity" book [5], depicting the communication model between issuers, holders, and verifiers, utilizing a verifiable data registry for mediation and DIDs for identification.

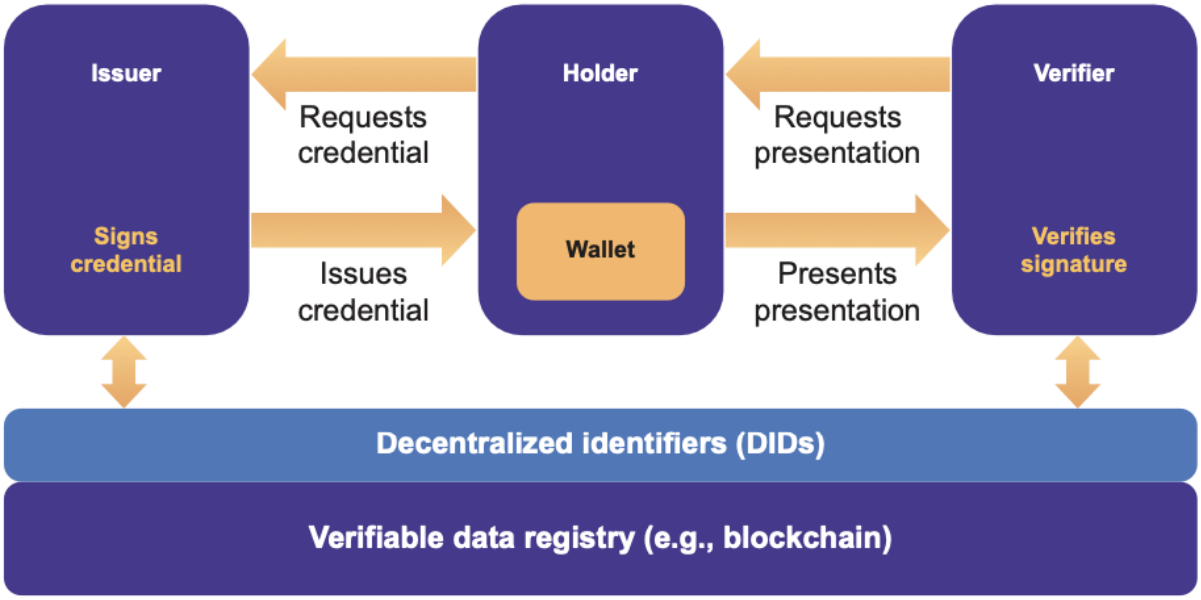

Figure 1-6. A decentralized identity trust model that utilizes a verifiable data registry

However, a verifiable data registry is not enough for a completely functional decentralized system. Each organization may still have its own standards and data formats, making it hard to cooperate with others without some explicit agreement between the organizations. Here is where governance frameworks (or trust frameworks) come into play. The Trust Over IP Foundation defines a governance

framework as "a set of business, legal, and technical [definitions], [policies], [specifications], and contracts by which the members of a trust community agree to be governed in order to achieve their desired objectives" [12]. In short, a governance framework describes standards for communication between different parties of an identity system. These frameworks are the reason why VCs can be truly scalable.

1.7. Digital wallets

The core technology that lets users take control of their digital identity is a digital wallet. Just as one stores their physical credentials, money, and banking cards in a physical wallet, digital wallets are designed for storing digital credentials, money, and banking cards. There have been many different implementations of digital wallets over the past decade, each offering different features - from holding plane tickets to buying cryptocurrencies. The type of digital wallet relevant to our discussion of decentralized digital identities and verifiable credentials is often referred to as a “decentralized identity wallet” [13] or an “SSI wallet” [5, 14]. Its core features are secure storage and presentation of VCs. This may not sound like much but as the authors of the “Self-Sovereign Identity” book [5] put it, “Digital wallets and agents are to SSI what browsers and servers are to the web: the basic tools we need to make the whole infrastructure work.”

It is worth noting that the authors of the aforementioned book also define “digital agents” that work with digital wallets. In their terms, a wallet is just a secure storage a user doesn’t interact with directly. Instead, users interact with digital agents that perform actions with their wallets (like creating and presenting a verifiable presentation) on their behalf. However, in practice, the functionalities of both are combined into one application which is called a digital wallet. That is why we do not differentiate between digital wallets and agents in this work.

Decentralized identity wallets are not limited to the storage of VCs. Some other things they may store include cryptographic keys, DIDs, digital copies of physical credentials, personal contact information, and data usually stored in password managers (like user's credentials on various websites).

Digital wallets are a part of a larger ecosystem of decentralized identity and verifiable credentials, therefore, they share the main overarching design principles:

Open and portable - they must support open standards for VCs, DIDs, and other related things. This makes them interoperable regardless of the specific organization developing the product. For users, this results in data portability - they may change the specific wallet they use at any time.

One wallet, one experience [15] - the standard experience should be the same across all wallets from different vendors.

Security and privacy by design - as digital wallets contain highly sensitive private data, their security and privacy are the main design goals of any implementation.

Consent-driven - every single action done by the wallet must have the explicit approval of its owner. This is in contrast to how most online services operate now concerning user data.

Chapter 2. European Approach: A Case Study

Many European countries have developed their own digital service platforms (Estonian eID, Spanish Cl@ve, Ukrainian Diia). The problem with all of them, however, is that they are essentially following the centralized approach to digital identity – all the data is stored in centralized governmental siloes and are subject to all the same problems as any other centralized identity solution. The same applies not only to governmental services, but also to universities, hospitals, and employers. Many of them have built their separate digital services that are, unfortunately, unable to interoperate. The issue of interoperability is even more pressing in the context of globalization and digitalization. With the EU's open borders principle, there must be a secure and reliable way to use digital credentials outside the country where they were issued.

To address these issues, the European Commission has announced multiple projects and regulations that are currently under development. The main ones are the European Blockchain Services Infrastructure and an EU Digital Identity Wallet.

2.1. European Blockchain Services Infrastructure

European Blockchain Services Infrastructure (EBSI) is a governance framework aimed at unifying approaches to digital identity and verifiable credentials across member states. "EBSI is an initiative of the European Commission and European Blockchain Partnership to create a peer-to-peer network of distributed nodes across Europe, supporting applications focused on selected use-cases." [16] It provides a set of open standards, regulations, and recommendations that together form a foundation of a huge digital system. Initiated by the European Commission and the European Blockchain Partnership, EBSI is still in its development stage but already has many tools to use.

The dominant idea behind EBSI is to facilitate the development of Web 3.0 and a decentralized approach to digital identities. It is built with the same overarching main design goals in mind: control of data, reusability, extensibility, and easy identity management.

To facilitate this goal, EBSI heavily utilizes many open standards regarding digital identities, verifiable credentials, privacy, and security. The main standards used include: [17]

- W3C standards: Decentralized Identifiers, Verifiable Credentials Data Model, Presentation Exchange
- OpenID Connect standards: OpenID Connect SIOP, OpenID Connect for Verifiable Presentations, OpenID Connect for Verifiable Credentials, Issuance
- eIDAS standards: JAdES, eID authentication and identification
- JWT RFC family standards: IETF RFC 7515-7520

The network's EBSI nodes, which support various pluggable protocols and a set of APIs, currently primarily use Hyperledger Besu (with IBFT 2.0 consensus) and Fabric. [20]

The main pilot use cases for EBSI are identity (government IDs), education (digital diplomas, certificates, student ID cards, transcripts), and employment (digital employment history).

Because EBSI so heavily relies on open standards, everything discussed in the previous chapter still applies to it. What’s specific to EBSI is its trust model and what are the responsibilities of the EBSI itself.

2.2.1. EBSI's core technical services

Regardless of the specific use case, the general flow of operation is the same across all EBSI services. First, the actor uses the end application. Second, the application utilizes EBSI's API. Third, an API calls a smart contract. Fourth, a smart contract performs the requested operation and stores the transaction on the EBSI ledger. To achieve maximum data safety, one cannot skip any of the steps of this process. It is impossible to modify the ledger or deploy smart contracts directly without calling the API first. To access the API, an application needs to be conformant with all EBSI standards. This approach ensures a wide acceptance of standards and a high level of security.

2.2.2. EBSI Trust Model

In general, EBSI trust model looks as follows: [18]

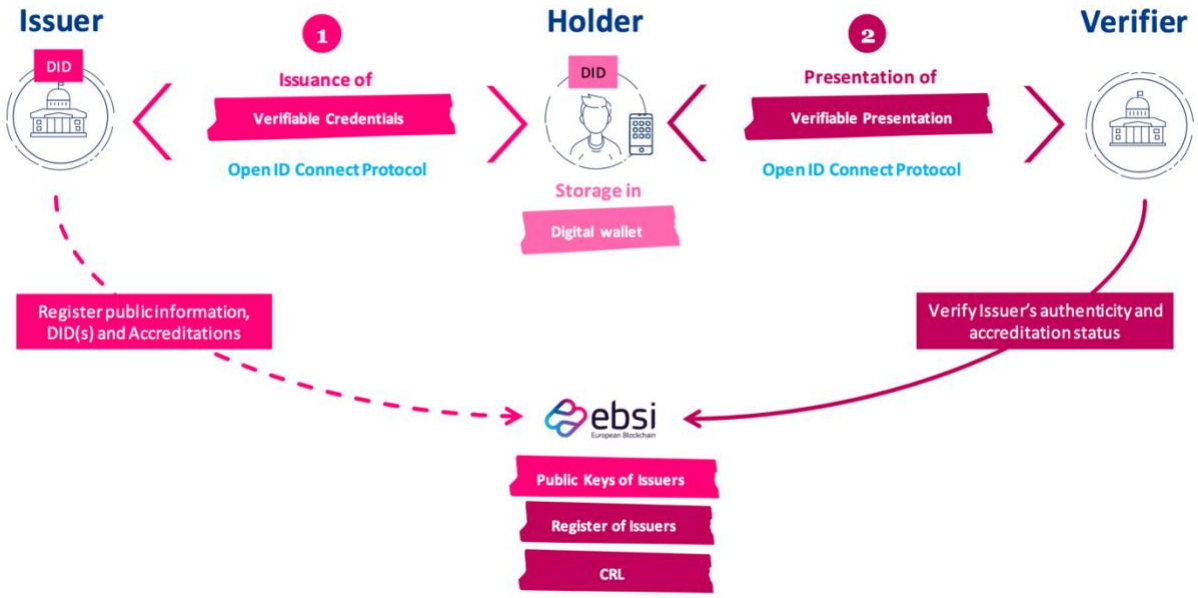

Figure 2-1. EBSI trust model

This is the same trust model as in Figure 1-6. A decentralized identity trust model that utilizes a verifiable data registry, with EBSI instead of a generic verifiable data registry. The picture clearly displays how different parties communicate and what is stored on the EBSI blockchain.

Issuers and verifiers communicate with the holder wallet via the OpenID Connect Protocol (one of the main protocols upon which EBSI is built). EBSI itself is responsible for storing the public keys of trusted issuers, verifiable accreditations of issuers, trusted schemas, and revoked credentials. It is worth noting that the EBSI blockchain does not store the VCs themselves, as this is not needed with the current trust model. Instead, it is enough for verifiers to only check the issuer's authenticity and whether the credential has been revoked or not. Such design improves simplicity and decreases the operating costs of the infrastructure.

For this model to work, however, there must exist a way to designate issuers as trusted. For this, a special role exists - Trusted Accreditation Organization (TAO) [19]. In short, TAO is responsible for accrediting the trusted issuers (TIs) and registering Trusted Schemas. TAOs are usually high-level governmental authorities that each manage their specific domain. For example, the Ministry of Education of Poland would accredit Polish educational institutions and possibly issue new credential schemas related to education. A simple illustration of the relationship between TAOs and TIs is given below:

Figure 2-2. Relationship between TAOs and TIs

A more complete EBSI trust model that includes TAOs and separate registries stored on the EBSI blockchain is given below:

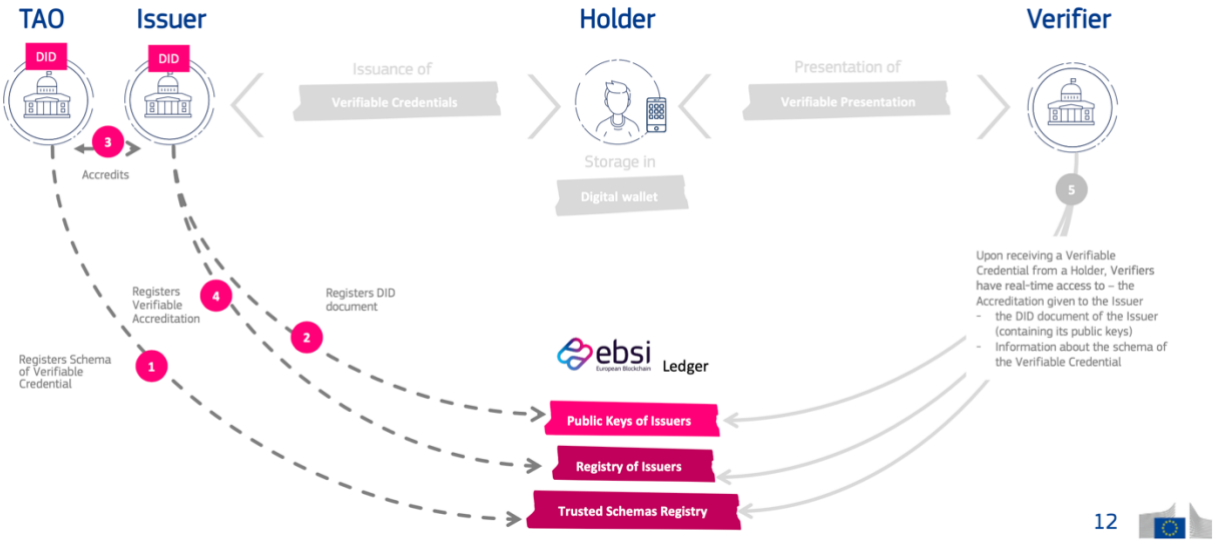

Figure 2-3. Complete EBSI trust model

The main benefits of the discussed trust model design are:

- Ease of management – once the trust chain is set up, subsequent operations become easy, requiring only one or two API calls.
- Ease of verification – EBSI guarantees the authenticity of verified issuers, so verifiers can check the validity of credentials independently of the issuers with only a few API calls to EBSI.
- Difficulty of forgery – this is based on the core design principles of VCs themselves.

2.2.3. VC issuance process

The EBSI's Verifiable Credential Issuance specification [21] is designed to be compliant with OpenID for Verifiable Credential Issuance specification [22]. Both specifications describe an API issuers need to implement. They also split the issuing side into two entities: Authorization Server and Credential Issuer, which may or may not be combined into one service. To ensure communication between parties without them knowing about each other in advance, the specifications rely heavily on redirects and QR codes.

The specifications define the following main endpoints:

- Credential Offer Endpoint – used by the issuer to initiate the issuance;
- Authorization Endpoint – used to authorize user within the issuer's system;
- Token Endpoint – used by the wallet to get an access token;
- Credential Endpoint – used by the wallet to receive a VC;
- Batch Credential Endpoint – same, but for multiple VCs;
- Deferred Credential Endpoint – used to issue a VC after a delay;
- Metadata Endpoints – used to retrieve metadata about the issuer and the authorization service.

There are two different code flows of the process: authorized and pre-authorized. Authorized code flow is initiated by a wallet. Thus, a wallet must know about the issuer beforehand and suggest the user get a credential from that issuer. In this flow, the wallet first sends an authorization request to the authorization server. The authorization typically includes multiple steps, one of which is for the user to log in to the centralized issuer service. Below is an example of the authorization flow from [21]:

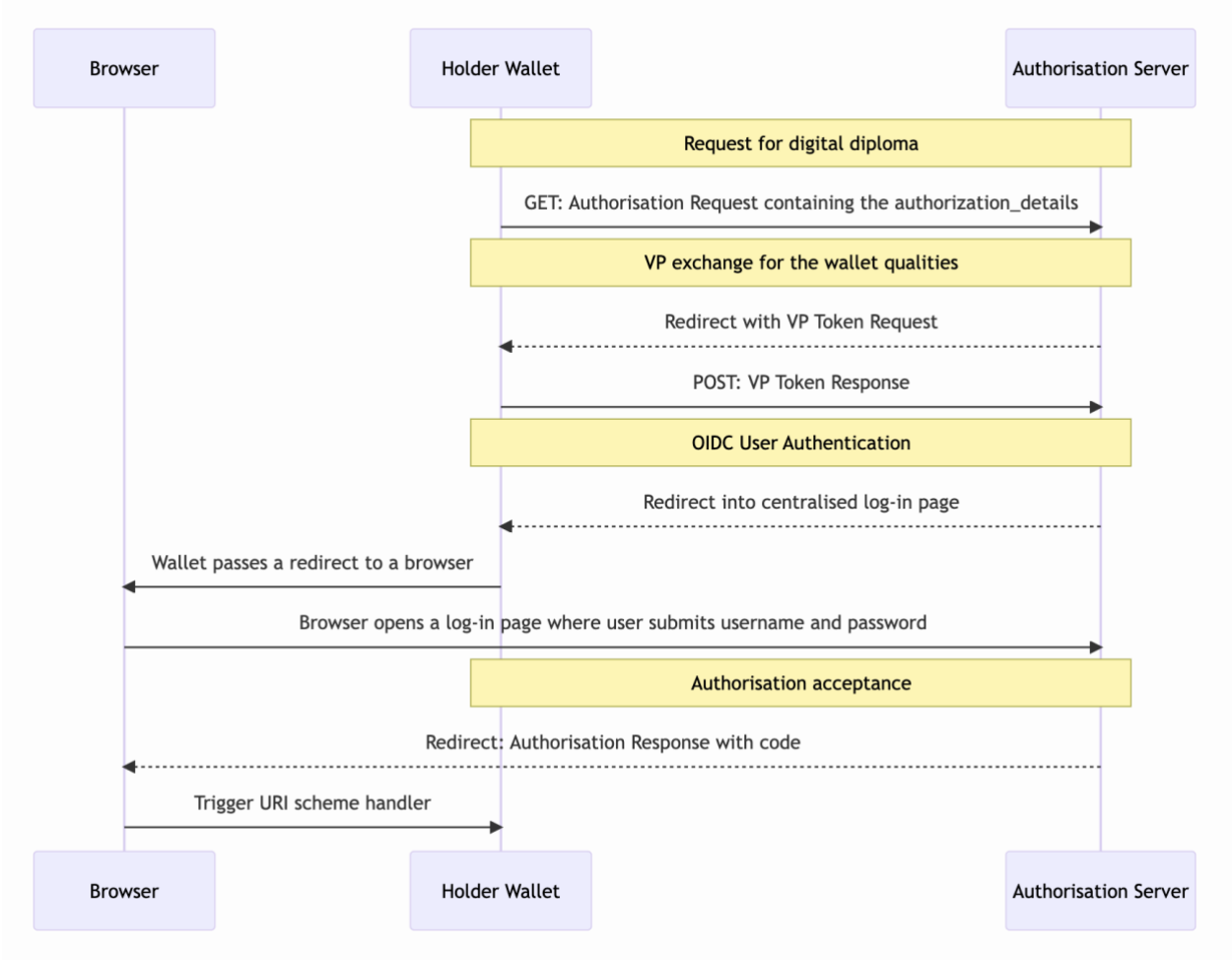

Figure 2-4. Authorization flow example

After the successful authorization response, the wallet sends the token request to the issuer server together with the authorization code acquired in the previous step. If successful, the issuer returns the access token. Finally, the wallet sends a credential request with the access token from the step before. If everything is properly validated, the issuer returns a VC. Combining all these steps, the complete authorized flow looks as follows [22]:

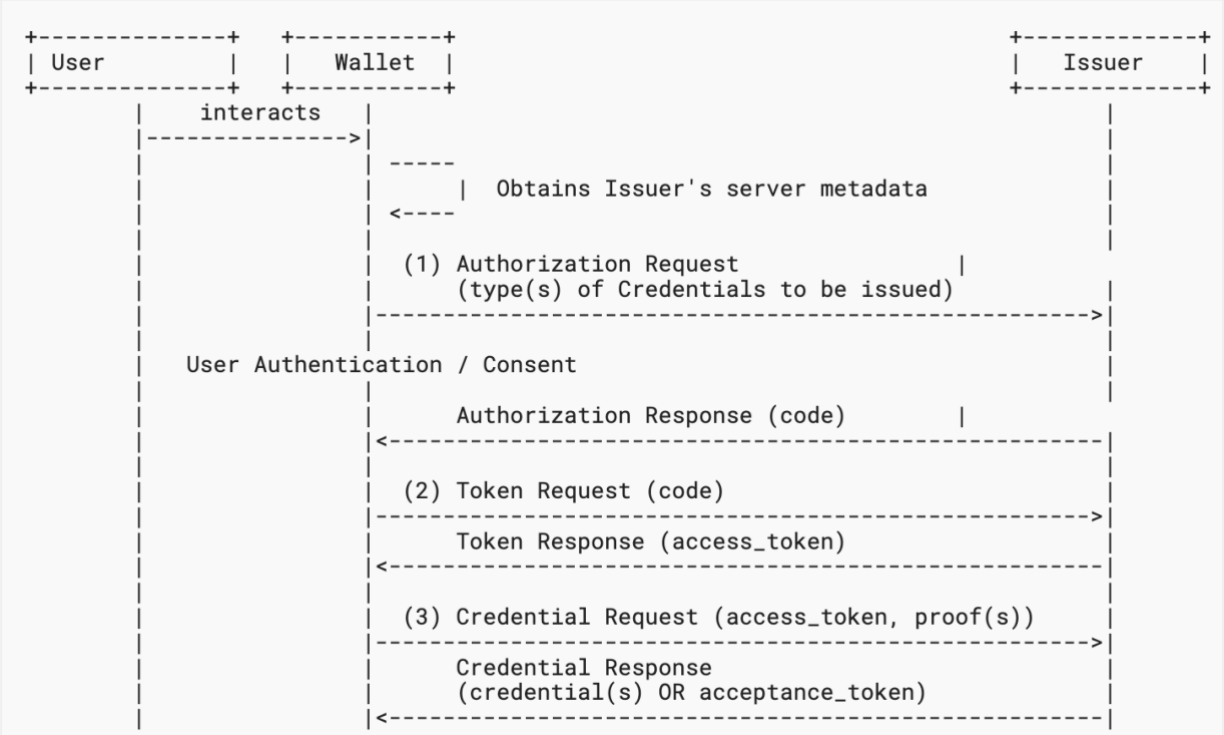

Figure 2-5. Complete authorized issuance flow

The pre-authorized flow is initiated by the issuer. In this flow, the user is authenticated before the flow begins, so the authorization endpoint is not needed. The flow starts with the issuer service making a credential offer to the wallet. This is achieved with either a redirect if the issuer and wallet are on the same device, or a QR code if different devices are used. The credential offer contains information for the wallet on how to get the metadata about the issuer and credential. Next, the wallet sends a token request to the issuer. The difference from the token request in authorized flow is that instead of an authorization code, a predetermined code is sent to the issuer. Finally, the credential request step is the same as in the authorized flow. The figure below shows the pre-authorized flow [22]:

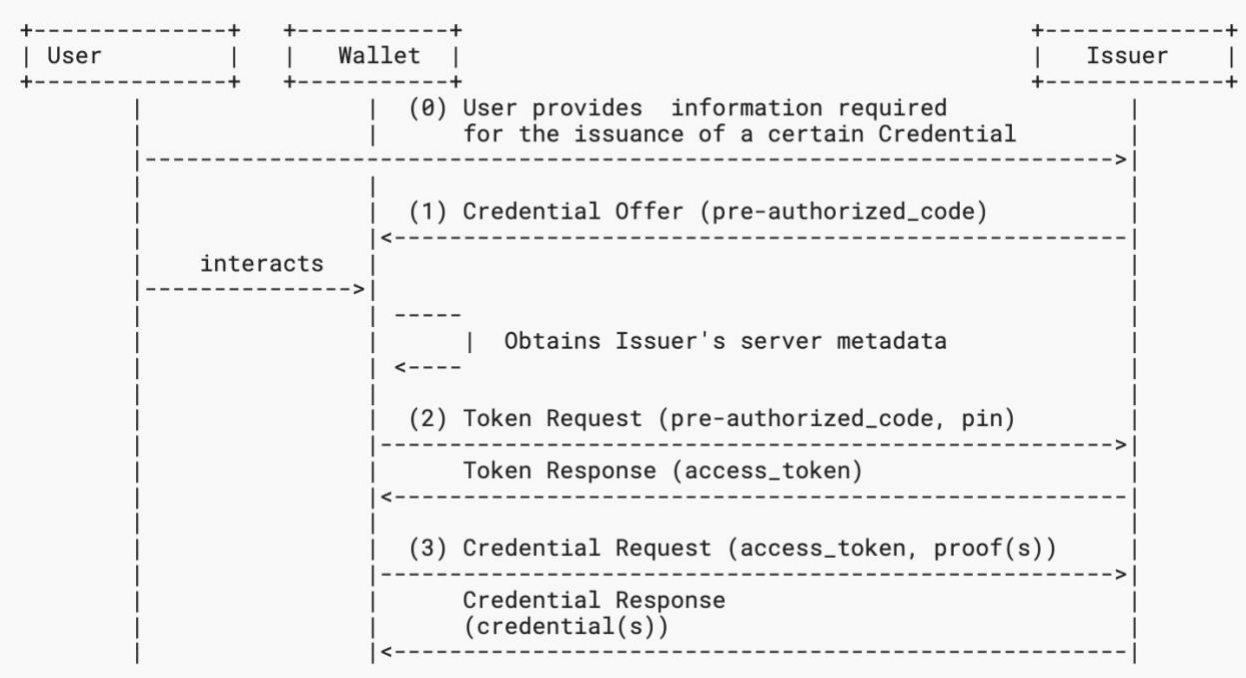

Figure 2-6. Complete pre-authorized issuance flow

Aside from these endpoints, the issuer service must also implement the `/.well-known/openid-configuration` and `/.well-known/openid-credential-issuer` metadata endpoints. They are used by wallets for retrieving the metadata about the issuer. The former provides metadata about the issuer, and the latter – about the authorization server.

2.2. EU Digital Identity Wallet

EU Digital Identity Wallet (EUDI Wallet or EUDIW) is another product proposed as part of the European Commission's focus on digital transformation. Simply put, the EUDI Wallet is an EU-wide framework of a digital wallet for individuals capable of storing verifiable credentials about their owners. The EUDI

Wallet aligns closely with EBSI, following the same open standards and specifications. The wallet's specification is a major part of the eIDAS regulation [23], focusing on innovation, interoperability, and transparency. The wallet framework focuses on the same ideas discussed throughout this thesis: interoperability, users' control of data, privacy, scalability, and security. Currently, members from all 27 EU member states, Norway, Iceland, the UK, and Ukraine work on this initiative.

The specification of the EUDIW is governed by the Architecture and Reference Framework [25], a technical specification of the proposed wallet implementation. As part of the framework, a reference implementation is made publicly available [26]. An important point worth making is that this specification is only a recommendation, and it is not required to precisely follow the proposed architecture as long as the implementation follows the necessary standards.

Currently, the concept is being tested on 4 large-scale pilot projects: The EU Digital Identity Wallet Consortium (EWC), POTENTIAL**,** NOBID**,** and DC4EU**.** It is set out to make the implementations widely available by the year 2026.

The project has some technical challenges, mostly regarding the choice of technologies and standards to use. For example, aside from the W3C Verifiable Credentials Data Model (discussed previously), there is also an ISO/IEC 18013-5 (mDL) standard [27] for digital credentials. Another problem is that the standards used are very complicated and thus require highly skilled engineers to develop them. Yet another issue, related to the previous ones, is that the development process is slow and doesn't always fit the predefined timeline.

Chapter 3. Sample Issuer Wallet Implementation

3.1. Specification

Issuers typically employ organizational wallets for issuing and managing issued credentials. In the context of VCs, the following system components are involved: a wallet's issuer API, an organizational backend system, wallet's and organization's front-end systems for the organization's managers, a front-end system for the organization's users, and the end users' wallets.

Consider the following example: Alice has just graduated from a university that supports digital diplomas. A person working at the university's administration uses their account to record that Alice has graduated via the university's front-end service. The update is then saved to the university's backend system. Depending on whether the organizational wallet has direct access to the university's databases, the manager at the university may need to also record the graduation manually via the wallet's front-end service that makes the necessary requests to the wallet's backend. After the graduation has been digitally recorded, Alice can log in to the university website (i.e. a front-end service) and request her diploma VC. The university website makes the required calls to the wallet's API, issuing the credential to Alice, which she then stores on her individual wallet.

For simplicity, our sample implementation only consists of the wallet's issuer API and a front-end website that allows anyone to issue any credentials they would like, thus noticeably simplifying the issuance process. This approach ignores the organizational own systems completely, focusing specifically on the verifiable credentials issuance process. Its main downside is that it is not representable of the whole process. However, this limitation can be neglected because of our focus on the issuance process.

An important clarification worth making is that our sample issuer is not a "verified issuer". The program is a minimalistic example of issuing fake credentials, not the real credentials about real subjects. Moreover, it does not implement all required endpoints to be EBSI compliant. To be precise, it does not implement the deferred and batch issuance requests, only supports the JWT format of VCs, and does not support the same-device issuance. However, the endpoints that are implemented do follow the standards.

3.1.1. API endpoints

The API design mimics that of walt.id Issuer API [28]. Here is the list of implemented endpoints:

- `GET /.well-known/openid-configuration`
- `GET /.well-known/openid-credential-issuer`
- `GET /authorize`
- `POST /token`
- `POST /credential`
- `POST /openid4vc/jwt/issue`
- `GET /openid4vc/credentialOffer`

They follow the endpoints discussed in 2.2.3. VC issuance process, so there is no need to repeat that information.

3.1.2. Front-end specification

The front-end service is a simple website that allows users to either choose from a preselected standard credential schema or create a credential with arbitrary claims represented as key-value pairs. In case of a nonstandard credential, the users may also select the name of the credential. After the users fill in and submit the

required data, the website makes a call to the API's `POST /openid4vc/jwt/issue` endpoint. The API returns the credential offer URI, which the website makes into a QR code and displays to the user. The user can then scan the QR code with their phone and add the credential to their wallet.

3.2. Issuer API implementation

The main consideration when choosing the development tools for the issuer API is the availability of properly functioning libraries with good documentation. Implementing the API as per specification from scratch would be infeasible in the scope of this thesis, so finding a good SDK was crucial. We rejected multiple libraries before we finally found the one we decided to use.

For an issuer API implementation, we selected Node.js [29] as the main framework and multiple Digital Bazaar libraries [30] for issuing and sharing verifiable credentials. Node.js, specifically together with the Express framework [31] that we also used, offers an easy and straightforward way to quickly develop a small development server. Another big advantage of using JS is the ease of working with JSON, which is very important in the context of VCs, which are usually represented in JSON-LD format. We also used the `vc-js`, `body-parser`, and `jsonwebtoken` libraries [32-34] for more specialized functions.

3.3. Front-end implementation

We selected Flutter [35] as the implementation framework for our front-end website. Its main advantage is the cross-platform support – the same codebase works on all end platforms, like Web, Android, iOS, Windows, Linux, and MacOS. This is important for an issuer wallet because we may want to issue credentials from any of

those platforms. Another big selling point in the context of a small prototype project is how easy and fast it is to set up a fully functioning and visually appealing product. Flutter's out-of-the-box widget library provides a big set of customizable components that can be used without installing any additional tools or libraries. Flutter uses Dart as its development language. It is appealing because of its static typing and similarity with other popular high-level languages, like Java and C#.

The implementation uses Flutter's `http` library [36] for making API requests, and a `qr_flutter` library [37] for displaying a QR code with the credential offer endpoints.

Conclusion

In conclusion, this paper has gone in-depth into the topic of digital identity, which is still young but has the potential to become omnipresent soon. We have pointed out the significant advantages of decentralized systems over traditional centralized approaches. We thoroughly discussed verifiable credentials, verifiable presentations, decentralized identifiers, trust models, and digital wallets. Such details lay an essential foundation for understanding why these elements are critical components in digital identity verification.

Through the case study on European Blockchain Services Infrastructure (EBSI) and European Digital Identity (EUDI) wallet, we have seen how these theoretical concepts find practical application within a regulatory and operational setup. In analyzing EBSI standards (including the trust model upon which they are built) and recommended issuance flow details by EBSI, we shed light on the implementation intricacies and adoption problems that this new approach still faces.

A simplified issuer implementation was used to demonstrate the applicability of the discussed theoretical concepts. Development of an issuer API and a frontend website through Node.js and Flutter have illustrated the practical ways these technologies can be used in verifiable credentials issuance, demonstrating how the complex technical specifications result in easy-to-use applications.

Bibliography

[1] J. Sedlmeir, R. Smethurst, A. Rieger, and G. Fridgen, “Digital Identities and Verifiable Credentials,” Business & Information Systems Engineering, vol. 63, Oct. 2021, doi: https://doi.org/10.1007/s12599-021-00722-y.

[2] World Economic Forum, “Identity in a Digital World. A new chapter in the social contract,” Sep. 2018. Accessed: May 13, 2024. [Online]. Available: https://www3.weforum.org/docs/WEF_INSIGHT_REPORT_Digital%20Identity.pdf

[3] International Civil Aviation Organization, “Machine Readable Travel Documents. Part 7: Machine Readable Visas ,” 2021. Accessed: May 13, 2024. [Online]. Available: https://www.icao.int/publications/Documents/9303_p7_cons_en.pdf

[4] A. J. Zwitter, O. J. Gstrein, and E. Yap, “Digital Identity and the Blockchain: Universal Identity Management and the Concept of the ‘Self-Sovereign’ Individual,” *Frontiers in Blockchain*, vol. 3, May 2020, doi: https://doi.org/10.3389/fbloc.2020.00026.

[5] A. Preukschat and D. Reed, *Self-Sovereign Identity*. New York: Manning Publications Co. LLC, 2021.

[6] C. Allen, “The Path to Self-Sovereign Identity,” *Life With Alacrity*, Apr. 26, 2016. https://www.lifewithalacrity.com/article/the-path-to-self-soverereign-identity/ (accessed May 13, 2024).

[7] A. Tobin and D. Reed, “The Inevitable Rise of Self-Sovereign Identity,” Sep. 2016. Accessed: May 13, 2024. [Online]. Available: https://sovrin.org/wp-content/uploads/2018/03/The-Inevitable-Rise-of-Self-Sovereign-Identity.pdf

[8] M. Sporny, D. Longley, D. Chadwick, and O. Steele, *Verifiable Credentials Data Model v2.0*. World Wide Web Consortium, 2024. Accessed: May 13, 2024. [Online]. Available: https://www.w3.org/TR/vc-data-model-2.0/

[9] M. Sporny, D. Longley, M. Sabadello, D. Reed, O. Steele, and C. Allen, *Decentralized Identifiers (DIDs) v1.0*. World Wide Web Consortium, 2022. Accessed: May 13, 2024. [Online]. Available: https://www.w3.org/TR/did-core/

[10] M. Sporny, D. Longley, G. Kellogg, M. Lanthaler, P.-A. Champin, and N. Lindström, *JSON-LD 1.1*. World Wide Web Consortium, 2020. Accessed: May 13, 2024. [Online]. Available: https://www.w3.org/TR/json-ld11/

[11] H. Yıldız, A. Küpper, D. Thatmann, S. Göndör, and P. Herbke, “A Tutorial on the Interoperability of Self-sovereign Identities,” *OPAL (Open@LaTrobe) (La Trobe University)*, Aug. 2022, doi: https://doi.org/10.36227/techrxiv.20430825.v1.

[12] Trust Over IP Foundation, “GSWG Glossary,” *trustoverip.github.io*. https://trustoverip.github.io/gswg/glossary#governance-framework (accessed May 14, 2024).

[13] M. Korir, S. Parkin, and P. Dunphy, “An Empirical Study of a Decentralized Identity Wallet: Usability, Security, and Perspectives on User Control,” in *Usenix*, Aug. 2022, pp. 195–211. Accessed: May 13, 2024. [Online]. Available: https://www.usenix.org/conference/soups2022/presentation/korir

[14] European Blockchain Associaton, “SSI Wallets,” *European Blockchain Association*. https://europeanblockchainassociation.org/ssi-wallets/ (accessed May 13, 2024).

[15] I. B. Pulse, “Episode 1: The future of protecting your wallet and identity,” *IBM Blog*, Apr. 24, 2019. https://www.ibm.com/blog/episode-1-the-future-of-protecting-your-wallet-and-identity/ (accessed May 14, 2024).

[16] European Blockchain Services Infrastructure, “QAs - EBSI -,” *ec.europa.eu*. https://ec.europa.eu/digital-building-blocks/sites/display/EBSI/QAs?option=659622446 (accessed May 14, 2024).

[17] European Blockchain Services Infrastructure, “EBSI Explained: Verifiable Credentials. An introduction,” Jan. 2023. Accessed: May 13, 2024. [Online]. Available: https://ec.europa.eu/digital-building-blocks/sites/download/attachments/600343491/Chapter%200%20-%20Verifiable%20Credentials%20An%20introduction.pdf

[18] European Blockchain Services Infrastructure, “EBSI Explained. Chapter 1: EBSI Verifiable Credentials,” Jun. 2022. Accessed: May 13, 2024. [Online]. Available: https://ec.europa.eu/digital-building-blocks/sites/download/attachments/600343491/Chapter%201%20-%20EBSI%20VC%20.pdf?api=v2

[19] European Blockchain Services Infrastructure, “EBSI Explained. Chapter 5: Issuers trust model,” Jun. 2022. Accessed: May 13, 2024. [Online]. Available: https://ec.europa.eu/digital-building-blocks/sites/download/attachments/600343491/Chapter%205%20-%20Issuer%20Trust%20Model%20%20.pdf?api=v2

[20] European Blockchain Services Infrastructure, “Home - EBSI -,” *ec.europa.eu*. https://ec.europa.eu/digital-building-blocks/sites/display/EBSI/Home (accessed May 13, 2024).

[21] European Blockchain Services Infrastructure, “How to issue Verifiable Credentials,” *hub-test.ebsi.eu*, Apr. 05, 2024. https://hub-test.ebsi.eu/conformance/learn/verifiable-credential-issuance (accessed May 13, 2024).

[22] T. Lodderstedt, K. Yasuda, and T. Looker, *OpenID for Verifiable Credential Issuance*. OpenID Connect, 2022. Accessed: May 13, 2024. [Online]. Available: https://openid.net/specs/openid-4-verifiable-credential-issuance-1_0-10.html

[23] European Commission, “eIDAS Regulation | Shaping Europe’s digital future,” *digital-strategy.ec.europa.eu*, Apr. 04, 2024. https://digital-strategy.ec.europa.eu/en/policies/eidas-regulation (accessed May 13, 2024).

[24] “EU Digital Identity Wallet Consortium,” *EUDI Wallet Consortium*, Mar. 29, 2022. https://eudiwalletconsortium.org/ (accessed May 13, 2024).

[25] *Architecture and Reference Framework*. European Commission, 2024. Accessed: May 13, 2024. [Online]. Available: https://eu-digital-identity-wallet.github.io/eudi-doc-architecture-and-reference-framework/1.3.0/arf/

[26] V. Kanellopoulos, “EUDI Wallet Reference Implementation,” *GitHub*, Jun. 2023. https://github.com/eu-digital-identity-wallet/.github/blob/main/profile/reference-implementation.md (accessed May 14, 2024).

[27] ISO and IEC, *ISO/IEC 18013-5:2021. Personal identification — ISO-compliant driving licence. Part 5: Mobile driving licence (mDL) application*. International Organization for Standardization, 2021. Accessed: May 13, 2024. [Online]. Available: https://www.iso.org/standard/69084.html

[28] walt.id, "walt.id Issuer API," *issuer.portal.walt.id*. https://issuer.portal.walt.id/swagger/index.html#/ (accessed May 14, 2024).

[29] Node.js, "Node.js," *Node.js*, 2023. https://nodejs.org/en

[30] Digital Bazaar, "Digital Bazaar, Inc.," *GitHub*. https://github.com/digitalbazaar (accessed May 14, 2024).

[31] OpenJS Foundation, "Express - Node.js web application framework," *Expressjs.com*, 2017. https://expressjs.com/

[32] Digital Bazaar, "vc-js," *npm*. https://www.npmjs.com/package/vc-js (accessed May 13, 2024).

[33] Express.js, "body-parser," *npm*, Apr. 26, 2019. https://www.npmjs.com/package/body-parser (accessed May 13, 2024).

[34] Auth0, "jsonwebtoken," *npm*, Sep. 2023. https://www.npmjs.com/package/jsonwebtoken

[35] Flutter, "Flutter - Beautiful native apps in record time," *Flutter.dev*, 2019. https://flutter.dev/ (accessed May 13, 2024).

[36] pub.dev, "http | Dart Package," *Dart packages*, Feb. 15, 2024. https://pub.dev/packages/http (accessed May 13, 2024).

[37] theyakka.com, “qr_flutter | Flutter Package,” *Dart packages*, May 14, 2023. https://pub.dev/packages/qr_flutter

Appendix A

Issuer API implementation

issuer-api-node/app.js

```
import vc from '@digitalbazaar/vc';
import openid4vc from '@digitalbazaar/openid4vc';
import {Ed25519VerificationKey2020} from '@digitalbazaar/ed25519-verification-key-2020';
import {Ed25519Signature2020} from '@digitalbazaar/ed25519-signature-2020';

const express = require('express');
const bodyParser = require('body-parser');
const jwt = require('jsonwebtoken');

const keyPair = await Ed25519VerificationKey2020.generate();
const suite = new Ed25519Signature2020({key: keyPair});

const hostname = "localhost:3000";

const app = express();

app.use(express.json());
app.use(express.urlencoded({ extended: true }));

app.get('/.well-known/openid-configuration', (req, res) => {
   res.sendFile(__dirname + '/openid-configuration.json');
});

app.get('/.well-known/openid-credential-issuer', (req, res) => {
   res.sendFile(__dirname + '/openid-credential-issuer.json');
});

app.get('/authorize', (req, res) => {
   const { responseType, clientId, issuerState, redirectUri } = req.query;
```

```
    try {
        if (!responseType || !clientId || !redirectUri) {
            throw new Error("Missing parameters");
        }

        if (responseType.includes('code')) {
            if (clientId.startsWith('did:key') && clientId.length === 186) {
                const idTokenRequestKid =
openid4vc.getSessionCredentialPreMapping(issuerState).issuerKey.getKeyId();
                const privateKey = openid4vc.getSessionCredentialPreMapping(issuerState).issuerKey;
                const authResp = openid4vc.processCodeFlowAuthorizationWithIdTokenRequest(req.query,
idTokenRequestKid, privateKey);

                handleRedirect(authResp, req, res);
            } else {
                const authResp = openid4vc.processCodeFlowAuthorization(req.query);

                handleRedirect(authResp, req, res);
            }
        } else if (responseType.includes('token')) {
            const authResp = openid4vc.processImplicitFlowAuthorization(req.query);

            handleRedirect(authResp, req, res);
        } else {
            throw new Error("not supported");
        }
    } catch (error) {
        console.error("Authorization error: ", error.toString());
        res.status(400).json({ error: error.message });
    }
});

function handleRedirect(authResp, req, res) {
    const redirectUri = req.query.redirectUri;
    res.redirect(`${redirectUri}?code=${authResp.code}`);
}
```

```
app.post('/token', (req, res) => {
    const params = req.query;

    try {
        const tokenResponse = openid4vc.processTokenRequest(params);

        const decoded = jwt.decode(tokenResponse.accessToken);
        if (!decoded || !decoded.sub) {
            throw new Error();
        }

        const sessionId = decoded.sub;
        const nonceToken = tokenResponse.nonce;
        if (!nonceToken) {
            throw new Error();
        }

        openid4vc.mapSessionIdToToken(sessionId, nonceToken);

        res.json(tokenResponse);
    } catch (error) {
        res.status(400).send(error.toString());
    }
});

app.post('/credential', (req, res) => {
    const authHeader = req.headers.authorization;
    const accessToken = authHeader ? authHeader.split(' ')[1] : null;

    if (!accessToken || !openid4vc.verifyTokenSignature(accessToken)) {
        return res.status(401).send('Unauthorized');
    }

    try {
        const credentialRequest = req.body;
        const credentialResponse = openid4vc.generateCredentialResponse(credentialRequest, accessToken);
```

```
        res.json(credentialResponse.toJSON());
    } catch (error) {
        res.status(400).send(error.toString());
    }
});

app.post('/openid4vc/jwt/issue', (req, res) => {
    const offer_uri = createCredentialOfferUri(req.body);
    res.status(200).send(offer_uri);
});

app.get('/credentialOffer', (req, res) => {
    const sessionId = req.query.id;
    if (!sessionId) {
        return res.status(400).send('Missing id param');
    }

    try {
        const issuanceSession = openid4vc.getSession(sessionId);
        if (!issuanceSession) {
            throw new Error('No session found');
        }

        const credentialOffer = issuanceSession.credentialOffer;
        if (!credentialOffer) {
            throw new Error('Credential offer not found');
        }

        res.json(credentialOffer);
    } catch (error) {
        res.status(400).send(error.toString());
    }
});

const createCredentialOfferUri = async (issuance_request) => {
    const session = openid4vc.initiateSession(issuance_request, 5*60*60, true);
    const sessionId = session.id;
```

```
    const signedCredential = await signCredential(issuance_request.vc);

    openid4vc.setIssuanceData(sessionId, signedCredential);

    return `openid-credential-
offer://${hostname}/?credential_offer_uri=${hostname}/openid4vc/credentialOffer?id=${sessionId}`;
}

const signCredential = async (credential) => {
    const signedVC = await vc.issue({credential, suite, documentLoader});
    return signedVC;
}

const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
    console.log(`Server running on port ${PORT}`);
});
```

Appendix B

Frontend website implementation

main.dart

```
import 'package:flutter/material.dart';
import 'my_app.dart';

void main() {
  runApp(const MyApp());
}
```

my_app.dart

```
import 'package:flutter/material.dart';
import 'key_value_form.dart';

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
        useMaterial3: true,
      ),
      home: const KeyValueForm(),
    );
  }
}
```

key_value_form.dart

```
import 'dart:convert';

import 'package:flutter/material.dart';
import 'send_vc_request.dart';

class KeyValueForm extends StatefulWidget {
  const KeyValueForm({super.key});

  @override
  _KeyValueFormState createState() => _KeyValueFormState();
}

class _KeyValueFormState extends State<KeyValueForm> {
  List<MapEntry<TextEditingController, TextEditingController>> fields = [];

  @override
  void initState() {
    super.initState();
    addField();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text('Credential Creation Form')),
      body: Padding(
        padding: const EdgeInsets.all(16.0),
        child: SingleChildScrollView(
          child: Column(
            children: [
              ListView.builder(
                physics: NeverScrollableScrollPhysics(),
                shrinkWrap: true,
                itemCount: fields.length,
                itemBuilder: (context, index) {
```

```
            return Padding(
              padding: const EdgeInsets.only(bottom: 8.0),
              child: KeyValueRow(
                keyController: fields[index].key,
                valueController: fields[index].value,
                onChanged: checkAndAddField,
              ),
            );
          },
        ),
        const SizedBox(height: 32.0),
        ElevatedButton(
          onPressed: submitData,
          child: const Text('Submit'),
        ),
      ],
    ),
  ),
 ),
);
}

void checkAndAddField() {
 if (fields.every((element) =>
    element.key.text.isNotEmpty && element.value.text.isNotEmpty)) {
  addField();
 }
}

void addField() {
 var newEntry = MapEntry(TextEditingController(), TextEditingController());
 setState(() {
  fields.add(newEntry);
 });

 newEntry.key.addListener(checkAndAddField);
 newEntry.value.addListener(checkAndAddField);
```

```
  }

  void submitData() {
    Map<String, String> data = {};
    for (var element in fields) {
      if (element.key.text.isNotEmpty && element.value.text.isNotEmpty) {
        data[element.key.text] = element.value.text;
      }
    }

    String credentialContent = jsonEncode(data);
    sendVCRequest(context, credentialContent);
  }
}

class KeyValueRow extends StatefulWidget {
  final Function onChanged;
  final TextEditingController keyController;
  final TextEditingController valueController;

  const KeyValueRow({
    Key? key,
    required this.onChanged,
    required this.keyController,
    required this.valueController,
  }) : super(key: key);

  @override
  _KeyValueRowState createState() => _KeyValueRowState();
}

class _KeyValueRowState extends State<KeyValueRow> {
  @override
  Widget build(BuildContext context) {
    return Row(
      children: [
        Expanded(
```

```
          child: TextField(
            controller: widget.keyController,
            decoration: const InputDecoration(labelText: 'Key'),
            onChanged: (text) => widget.onChanged(),
          ),
        ),
        const SizedBox(width: 16),
        Expanded(
          child: TextField(
            controller: widget.valueController,
            decoration: const InputDecoration(labelText: 'Value'),
            onChanged: (text) => widget.onChanged(),
          ),
        ),
      ],
    );
  }
}
```

send_vc_request.dart

```
import 'package:code/qr_screen.dart';
import 'package:flutter/material.dart';
import 'package:http/http.dart' as http;

void sendVCRequest(BuildContext context, String credentialContent) async {
  var url = 'localhost:3000/openid4vc/jwt/issue';
  var vc = getVC(credentialContent);
  print(vc);

  var response = await http.post(
    Uri.parse(url),
    headers: {"Content-Type": "application/json"},
    body: vc,
  );
```

```
  if (response.statusCode == 200) {
    navigateToQRScreen(context, response.body);
  } else {
    print('Failed to submit data');
  }
}

String getVC(String credentialContent) {
  return '''
{
  "issuanceKey": {
    "type": "local",
    "jwk": "{\\"kty\\":\\"OKP\\",\\"d\\":\\"mDhpwaH6JYSrD2Bq7Cs-pzmsjlLj4EOhxyI-
9DM1mFI\\",\\"crv\\":\\"Ed25519\\",\\"kid\\":\\"Vzx7l5fh56F3Pf9aR3DECU5BwfrY6ZJe05aiWYWzan8\\",\\"x\\":\\"T3T4-
u1Xz3vAV2JwPNxWfs4pik_JLiArz_WTCvrCFUM\\"}"
  },
  "issuerDid": "did:key:z6MkjoRhq1jSNJdLiruSXrFFxagqrztZaXHqHGUTKJbcNywp",
  "vc": {
    "@context": [
      "https://www.w3.org/2018/credentials/v1",
      "https://purl.imsglobal.org/spec/ob/v3p0/context.json"
    ],
    "id": "will be replaced",
    "type": [
      "VerifiableCredential",
      "OpenBadgeCredential"
    ],
    "name": "JFF x vc-edu PlugFest 3 Interoperability",
    "issuer": {
      "type": [
        "Profile"
      ],
      "id": "will be replaced",
      "name": "Credential",
    },
    "issuanceDate": "will be replaced",
    "expirationDate": "will be replaced",
```

```
    "credentialSubject": $credentialContent
  },
  "mapping": {
    "id": "<uuid>",
    "issuer": {
      "id": "<issuerDid>"
    },
    "credentialSubject": {
      "id": "<subjectDid>"
    },
    "issuanceDate": "<timestamp>",
    "expirationDate": "<timestamp-in:365d>"
  }
}
  ''';
}

void navigateToQRScreen(BuildContext context, String data) {
  Navigator.push(
    context,
    MaterialPageRoute(
      builder: (context) => QRScreen(qrData: data),
    ),
  );
}
```

qr_screen.dart

```
import 'package:flutter/material.dart';
import 'package:qr_flutter/qr_flutter.dart';

class QRScreen extends StatelessWidget {
  final String qrData;

  const QRScreen({super.key, required this.qrData});
```

```
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text('QR Code')),
      body: Center(
        child: QrImageView(
          data: qrData,
          version: QrVersions.auto,
          size: 200.0,
        ),
      ),
    );
  }
}
```