UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

UNIVERSITY OF MODENA AND REGGIO EMILIA

Department of Physics, Mathematics and Computer Science

Master's Degree in Computer Science (LM-18)

Blockchain-based Notarization for Digital Product Passports

Supervisor:
Prof. Giacomo Cabri

Candidate:
Fabio Zanichelli
Student ID 185445

ACADEMIC YEAR 2024/2025

Abstract

Digital Product Passports (DPP) act like a digital identity for products that provide a lot of information on their lifecycle and supply chain. DPP are gaining popularity because they allow consumers to access information directly at the point of sale, often scanning a QR code placed on the label of the product. This transparency may lead to a transition to a green economy because it allows consumers to make informed purchasing based on environmental metrics.

Building upon an existing DPP software tailored for the fashion industry, this thesis focuses on data trustworthiness and integrity; the main objective is to implement a notarization system that ensures non-repudiation, so the companies cannot alter the data or denying the fact that they reported them. It is important to note that this thesis does not address in any way the truthfulness of the data, but rather the immutability of the information once recorded. To achieve this, this thesis proposes a decentralized approach using blockchain technology. By eliminating the problem of a single point of failure, this research proposes a robust mechanism to ensure trust in the transparency of the DPP data.

This research focuses on the usability of the system for small and medium companies that often lack the economic, infrastructural, and technical resources to implement proprietary solutions. To mitigate these barriers, the concept of an aggregation center is introduced.

Contents

1 Introduction **1**

1.1 Context 1

1.2 Objective 2

2 State of art **3**

2.1 Digital signature 3

2.1.1 Authentication and integrity 3

2.1.2 Certification Authorities considerations 4

2.2 eIDAS 4

2.2.1 eIDAS standards 5

2.2.2 The Italian case 5

2.2.3 Article 25 and Article 26 eIDAS 6

3 Requirements and tools **8**

3.1 Requirements 8

3.2 Tools 8

3.3 Blockchain 9

3.3.1 Blockchain technology 9

3.3.2 Tendermint blockchain 10

3.4 Passkeys 10

3.4.1 Passkey technology 10

3.4.2 Passkeys for users 11

4 Architecture **13**

4.1 Aggregation centers 13

4.2 Permissioned blockchain 14

4.3 Audit log 15

4.4 Database 17

5 Implementation **19**

5.1 Notarization status 19

5.2 Signature 21

5.2.1 Passkey registration 21

5.2.2 Selection of data to notarize 22

5.2.3 Notarization of documents 24

5.2.4 Signature with passkeys 25

5.3 Notarization 26

5.3.1 ABCI server 26

5.3.2 Daily notarization 27

5.3.3 Merkle tree 28

5.3.4 Atomicity of the operation 30

5.4 Tendermint configuration 31

5.5 Addition of a new aggregation center 32

6 Containerization and hosting **36**

6.1 Containerization 36

6.2 Hosting 40

6.2.1 Cloud Provider 40

6.2.2 Domain name 41

6.2.3 Nginx configuration 42

6.3 CI/CD 44

6.3.1 Build and push 45

6.3.2 Deploy 46

7 Results **49**

7.1 Remote Procedure Call 49

7.2 Technical result 50

7.2.1 One change notarization 50

7.2.2 Two-changes notarization 51

7.3 User point of view 53

8 Conclusions **57**

Acknowledgements **59**

A Appendix **60**

References **67**

List of Figures

4.1 Architecture of the blockchain 14

4.2 Size of notarization. The red function explains the audit log scalability, while the blue function the full snapshot approach 16

4.3 ER diagram of the database 17

5.1 Notarization status 20

5.2 A visual representation of the Merkle tree 29

7.1 AuditLog instance of the notarization of ZUKK001 51

7.2 Batches ZUKK002 and ZUKK003 in SignedModification 53

7.3 List of batches in the DPP software 54

7.4 Details of ZUKK001 in the DPP software 54

7.5 Passkey screen for user 56

List of Code

5.1 Notarization status code 20
5.2 Options used for passkey signature 22
5.3 `get_data_to_sign` for batches model 22
5.4 Notarization of document in external cloud storages 24
5.5 The scheduler implementation for the blockchain 28
5.6 The implementation of Merkle tree 29
5.7 Atomicity of the notarization 30
5.8 Persistent peers in config.toml fine 32
5.9 Methods used for adding validators 33
6.1 Service blochchain_app in `docker-compose.yml` 36
6.2 Service tendermint and tendermint_init in `docker-compose.yml` 37
6.3 Service web in `docker-compose.yml` 38
6.4 Services nginx and certbot in `docker-compose.yml` 39
6.5 Nginx redirection of HTTP connection 42
6.6 Nginx management of HTTPS connections 43
6.7 Build and push job in the CI/CD pipeline 45
6.8 Deploy job in the CI/CD pipeline 47
7.1 Notarization of a single change in the blockchain 50
7.2 Notarization of two changes in the blockchain 52
A.1 Notarization of a single change in the blockchain 60
A.2 Notarization of two changes in the blockchain 63

1. Introduction

1.1 Context

In an increasingly globalized market, consumers are often required to choose among a vast array of products without having full access to the production data behind them. Providing detailed information about these processes offers a significant advantage to consumers, enabling them to make informed decisions based on the materials used and the production methods.

The carbon footprint also plays a crucial role in this context. Consumers may choose the product to buy prioritizing the product that has the minimum environmental impact; that will incentivize companies to adopt eco-friendly policies. The principle of transparency as a tool to reduce environmental degradation is further supported by the European Union. In 2024, the EU enacted the Ecodesign for Sustainable Products Regulation (ESPR), a set of measures with the aim of reducing pollution by improving production processes and increasing the durability of products placed on the market [**RegolamentoESPR2024**]

To achieve the goal of consumer transparency, the realization of a Digital Product Passport (DPP) emerges as an effective tool. This is implemented through a QR code on the product label, which users can scan with their smartphones to access product data [6]. Consequently, consumers gain access to information with freedom and autonomy in the store. This direct access mitigates the need to rely on store assistant, who might otherwise provide biased or inaccurate information.

It is evident that achieving reliable data requires monitoring the entire supply chain, from the raw material producer to the company that produces the final product. A similar mechanism is already implemented in the food industry, in which every step must be tracked to facilitate root-cause analysis in the event of safety or quality issues.

This thesis focuses on the fashion sector; therefore, the consumer has access to critical data such as the materials used, the water consumption, and all other key indicators.

This mechanism can solve the problem of transparency, but its effectiveness depends on the reliability of the data provided by companies. In fact, companies themselves have to upload their data to the information system, and authorities can verify that the data are correct. Authorities have

the power to apply sanctions to companies that report false data.

1.2 Objective

The objective of this thesis is **not** to find a methodology for data verification (there will be appropriate authorities), but to define a framework to certify that a company has reported a data at a precise date and time. Furthermore, the aim is also to find a mechanism for tracking the historical changes of product data. This procedure is formally defined as "notarization". This aspect is crucial because, if not implemented, companies can retroactively modify data just to comply with authorities control, disregarding the data transparency for the consumer.

The aim is to implement this process using blockchain technology, focusing on small and medium companies that might not have the resources to implement proprietary solutions.

This thesis is structured as follows. The second chapter shows the current situation on notarization. The third chapter outlines the technical requirements and know-how to implement the notarization. The fourth chapter discusses the architecture used. The fifth chapter details the implementation of the project. The sixth chapter explains the results obtained. Finally, the seventh chapter summarizes and concludes this thesis.

2. State of art

The current state of-the-art for digital notarization is the digital signature. It provides authentication and non-repudiation, bringing more security compared to handwritten signatures, which are subject to forgery. Instead, it is computationally infeasible to break a digital signature because of its cryptographic nature. Furthermore, every change to the document made after the signature is mathematically detectable. A blockchain-based approach must satisfy these same cryptographic criteria while introducing the advantages of a distributed architecture, which eliminates the risks associated with a centralized point of failure.

2.1 Digital signature

2.1.1 Authentication and integrity

Digital signature authentication is ensured by asymmetric cryptography. This mechanism relies on a key pair: the private key, known exclusively by the signatory, allows for the creation of the signature, while the corresponding public key enables verification. Consequently, the signatory retains sole control over the signing process. The signature function takes as input the result of the cryptographic hash of the document and the private key. This process yields the digital signature as its output. It is fundamental to emphasize that the digest of the document is strictly linked to the document itself and, due to the mathematical properties of the hashing functions, every change to the document changes the digest completely. This ensures data integrity, because if the digest changes, then the signature changes consequently, and the signature verification process will fail.

Digital signature requires a trusted thirty-part authority, known as Certification Authority (CA), that guaranties the correlation between the key used and the signatory, and must be trusted by both the signatory and the verifier. A vulnerability with this approach is represented by the choice of the authority; in fact, it is possible that a certification authority deliberately compromises the signature to advantage (or disadvantage) some companies for economical or political reasons. This represents a single point of failure in the digital signature approach.

2.1.2 Certification Authorities considerations

As introduced in the previous section, choosing which Certification Authority to trust is complex because it can have compromised behavior. There are two primary mechanisms designed to mitigate these risks and improve overall trust in a CA. The first mechanism is the Certification Revocation List (CRL), which provides a list of certificates that are revoked and, consequently, no longer valid. A significant drawback with this approach is that the access at the CRL can be denied by a DoS attack, making the CRL less reliable. Furthermore, the time gap between the insertion of a certificate in the list and the moment a client downloads the list is also a security issue, Online Certificate Status Protocol (OCSP) was developed with the aim of implement a real-time verification of the certificate but there is the same DoS vulnerability. Treating a "no response" of an OCSP responder like a non valid certificate can potentially paralyze signatures until the end of the attack.

The second mode is represented by Certification Transparency (CT). Every CA that releases a certificate must publish it in a public log to make it valid. This approach ensures that all CA operations are publicly auditable, making misconduct visible to everyone. This framework does not cover the issue of a revoked certificate.

Finally, the eiDAS regulation has a List of Trusted Services, a list of providers that are trusted by the government. Providers who have bad behavior or security issues are excluded from that list, and services will deny authentications with these providers.

2.2 eIDAS

With the aim to digitalize procedures, the European Union enacted the Electronic Identification, Authentication and Trust Services (eIDAS) regulation. This establishes a set of standards to be respected in order to ensure secure digital identity and electronic trust services. This provides a standard for citizens and companies to authenticate in the public administration system for signing and submitting documents with the same legal standing as handwritten signature [1]. Furthermore, eIDAS provides cross-border interoperability in all members of the European Union; consequently, a signing operation in one nation is considered valid in all other members of the EU.

2.2.1 eIDAS standards

The standards to use are not enacted in the eIDAS regulation itself but are written in the Commission Implementation Decision document. The European Commission released the first version of that in 2015 (Implementing Decision (EU) 2015/1506) and allows the signing of documents in PDF, XML and CSM format only. These standards refer to the European Telecommunications Standards Institute (ETSI) specifications and are the following:

- PAdES ,described in ETSI EN 319 142 for PDF documents;
- XAdES, described in ETSI EN 319 132 for XML documents;
- CAdES, described in ETSI EN 319 122 for CSM documents;

It is important to note that the use of other standards outside those specified by the European Commission is not forbidden; however, it can complicate legal proceedings. In case of a dispute, the party that used a not-approved standard will be asked to prove the correctness and reliability of the standard used to a judge.

Later, in 2024, a new regulatory framework (Regulation (EU) 2024/1183, known as eIDAS 2.0) was enacted, and consequently the Commission Implementation Decision document was updated in late 2024. The new version includes the possibility of signing JSON documents that address the massive increase in JSON usage in modern informatics, such as RESTful APIs (used in the implementation of the Digital Passport Product. The standard for signing JSON document is JAdES and is described in ETSI TS 119 182.

2.2.2 The Italian case

As a member of the European Union, Italy was among the first countries to adopt a digital identity to respect the eIDAS regulation. AgID (Agenzia per l'Italia Digitale) implemented SPID (Servizio Pubblico di Identità Digitale), a system that allows citizens and businesses to authenticate on public administration portals. This system securely links a phisical or legal person to their digital identity. Obtaining a SPID account does not require dedicated hardware, but can be accessed via a smartphone, tablet, or personal computer. The system is based on a public-private partnership;

only some providers (Identity Providers) approved by the government can issue SPID credentials and must recognize the person in the registration phase, ensuring a link between the person and his digital account.

Having a SPID account allows users to sign documents using an Advanced Electronic Signature (Firma Elettronica Avanzata or FEA), a digital signature that provides a high level of trust. According to eIDAS regulations, every EU member must recognize it. However, it is important to distinguish it from the Qualified Electronic Signature.

Later, Italy promoted the use of CIE (Carta d'Identità Elettronica), a new version of the traditional identity card, that allows a more secure signature and authentication; this is known as Qualified Electronic Signature (Firma Elettronica Qualificata, or FEQ).

FEQ is more secure due to the fact that the identity card is a physical device linked to one person and uses the embedded NFC chip to ensure hardware-based authentication, while SPID does not have this technology. Only FEQ has the same legal value as the traditional handwritten signature.

For the security considerations cited above and financial reasons, with the aim of unifying the national system, the Italian government has signaled a preference for CIE over SPID.

2.2.3 Article 25 and Article 26 eIDAS

Article 25 of the eIDAS regulations represents a fundamental point because it establishes the rules for the validity of the digital signature. Article 26 cites the requirements for a digital signature. The articles are as follows [1].

Article 25

Legal effects of electronic signatures

1. An electronic signature shall not be denied legal effect and admissibility as evidence in legal proceedings solely on the grounds that it is in an electronic form or that it does not meet the requirements for qualified electronic signatures.

2. A qualified electronic signature shall have the equivalent legal effect of a handwritten signature.

3. A qualified electronic signature based on a qualified certificate issued in one Member State shall be recognized as a qualified electronic signature in all other Member States.

Article 26

Requirements for advanced electronic signatures

An advanced electronic signature shall meet the following requirements:

(a) it is uniquely linked to the signatory;

(b) it is capable of identifying the signatory;

(c) it is created using electronic signature creation data that the signatory can, with a high level of confidence, use under his sole control; and

(d) it is linked to the data signed therewith in such a way that any subsequent change in the data is detectable.

According to article 26, letter c, it is evident that using a password is insufficient because passwords lack security because they do not ensure that the device or the credential remain under the sole control of the signatory. Unlike cryptographic keys stored on a physical device, passwords can be easily shared, intercepted, or used by unauthorized parties from any location. Furthermore, the choice of password can represent an issue; even if software has requirements for them such as length or presence of capital letters or symbols, passwords can be easy to break, and in computer security, the use of password as authentication factor is always considered weak.

This thesis does not specifically address letter d of article 26. However, the software and architecture proposed in the following chapter can be integrated with a human verification to ensure full compliance with the eIDAS regulation.

3. Requirements and tools

3.1 Requirements

The project is specifically designed for small and medium companies, so it has to prioritize user-friendliness and minimize the computational resources required by the company.

The objective is to guaranty the authenticity and integrity of the data uploaded before notarizing them to the blockchain network. Users can input data through an interface similar to standard resource management software, ensuring a familiar user experience. Users can also upload PDF documents for some certification obtained by the company. The system has to take into account the possibility of a human error such as an employee of a company reporting incorrect data in the software.

For the purpose of this research, it is considered that companies do not have the resources to manage a dedicated block node; Consequently, the only technical requirements that companies have to support are a stable internet connection and the ownership of devices that support passkeys (explained in section 3.4).

3.2 Tools

This work is based on an existing software designed for DPP in the fashion sector. The core application is written using the Django framework, using Python as a programming language. Python was also chosen due to its extensive ecosystem, which includes mature libraries for implementing the Tendermint ABCI protocol. Additionally, JavaScript is also used on the client side to manage the use of passkeys to make the browser communicate with the windows for the user.

The containerization and hosting workflow rely on the following tools:

- Docker for containerization;
- GitHub for versioning and GitHub Action for deployment;
- Amazon Web Service (AWS) for hosting;

For the blockchain, Tendermint[1] engine is used. At the time of writing, Tendermint is a component of the Cosmos SDK ecosystem. The choice of tendermint is driven by the need to integrate the notarization system into an existing software architecture; this is possible thanks to ABCI (Application BlockChain Interface), a protocol that functions as an API that allows the integration of blockchains in other applications.

3.3 Blockchain

3.3.1 Blockchain technology

Blockchain is a technology that has gained popularity in recent years. Originally popularized by Bitcoin for financial transactions because of the absence of a central host to trust, its distributed nature makes it an ideal solution for notarization. A blockchain network is made up of multiple nodes and every node stores a copy of the entire chain. The chain is composed of many interconnected blocks where each block contains the result of the cryptographic hash of the previous one and his data[2]. The result of the hash function of a node is then used to the next block.

$$hash_n = hash(hash_{n-1} + data_n)$$

Due to the mathematical hash property, any modification to data causes the change of the result of the hash function, and, consequently, this makes the ledger tamper-evident and virtually impossible to alter retroactively. The interconnection of the blocks by the hash function makes the blockchain very difficult to break.

To append a new block to the blockchain, a node must propose the block, and then other nodes vote to accept the block or not. A block must have been accepted by a certain threshold (consensus) of the node to be included in the chain. Typically, consensus is reached when at least two-thirds of the nodes on the blockchain accept the new block. This guaranties that even if a node is down, the blockchain is not paralyzed.

[1] https://tendermint.com/

[2] In reality, there are other data in a block but for the comprehension of the mechanism, they are omitted.

3.3.2 Tendermint blockchain

The Tendermint engine is used to implement a private and permissioned blockchain. The consensus threshold is set at two thirds of total nodes in the network; this makes the blockchain Byzantine Fault Tolerant (BFT) ensuring his operativity even if part of the nodes are down offline or act maliciously. Tendermint blockchain is based on a Proof of Authority (PoA) consensus model. Nodes prove their identity using digital signature; other nodes know the public key to verify signatures, ensuring that only blockchain that are part of the blockchain can propose blocks. The accepted nodes are defined in the configuration file along with several key operational parameters. Other important parameters to note are the timeout that define the duration of consensus step and the listening ports for the Tendermint server.

3.4 Passkeys

Passkeys are gaining prominence in computer security as a modern alternative for the purpose of authentication. They offer a higher level of security than passwords.

3.4.1 Passkey technology

Passkeys are based on asymmetric cryptography. Any devices that support this technology, in the registration phase, generate a pair of keys: a private key and a public key. Online servers only store the public key and use it to verify the authentication process. The two keys are mathematically linked together, but it is computationally impractical to derive the private key starting from the public key.

A passkey is stored in a specialized hardware module of a device and cannot be used outside of the authentication purpose. To use a passkey, a verification is always needed; this verification is often satisfied with fingerprint, facial recognition, or device PINs. The most famous specialized hardware for storing passkeys are:

- Secure Enclave, used in Apple products;
- Trusted Platform Module (TPM), used in most PCs;

- Trusted Execution Environment (TEE) on Android devices.

The authentication process follows a challenge-response mechanism. To authenticate, the server generates a random *challenge* and only the owner of the private key can sign that correctly. It works like a black-box: the challenge enters the secure environment (the "box") where it is signed, and then the output exits the SE and goes back to the server. The server checks the signature using the public key stored. It is important to note that the private key is never shared.

Due to the fact that private keys are never shared outside the secure environment, if a user has multiple devices, he has to register as many passkeys as the devices he own (each device has a different passkey). However, modern systems allow passkeys synchronization, solving the usability issue of multiple passkeys for a single account.

The technical standard that defines the implementation of passkeys is WebAuthn, which is a core component of the FIDO2 project [5].

3.4.2 Passkeys for users

Passkeys are an optimal solution to solve the most recurrent problems that affect users using passwords.

Firstly, passkeys eliminate the risk of being forgotten. Users often use biometric factors to unlock them, so this process does not rely on human memory, so it is almost impossible that a passkey is unavailable due to lack of user memory. Using different credentials for different services is one of the fundamental rules in computer security; however, users often use the same password to avoid remembering everything, which is uncomfortable. Passkeys solve this problem because they are unique for every service, and the biometric factor cannot be forgotten.

Secondly, passkeys are cryptographically better than passwords because they have the randomness that passwords do not have. This enhances the defenses against attacks where a cracker brute-forces the password using a list of most common passwords used like "password123".

Finally, passkeys are natively resistant to phishing. During the registration phase, informations about the service are stored so the passkey is bounded to the service used. If a user is under a phishing attack and visits a fraudulent clone website (like a bank website graphically similar to the original), passkeys simply don't work and the user cannot authenticate. This is due to the fact

that passkeys check the information stored during the registration phase and, in the case of a clone website, there is no passkey stored for the phishing site. So, the user will not be able to authenticate on the site.

Most modern devices and operating systems currently support passkeys. Among the most popular, there are:

- Apple devices with iOS 16;
- Android devices with Android 9+;
- Windows 10 and Windows 11 devices through Windows Hello;
- Physical token that supports FIDO2 standards.

4. Architecture

As stated in the requirements, the system must bridge the gap between the complex blockchain infrastructure and the practical needs of small and medium companies. The following sections explain how this objective is achieved.

4.1 Aggregation centers

The architecture bases its core idea on the concept of aggregation centers, specialized entities responsible for hosting the DPP software for affiliated companies. Ideally, a company is affiliated with a single aggregation center, although the proposed architecture would also work if a company decides to use the data in two different servers (and thus to the aggregation centers). This flexibility can only be ensured if a single data is uploaded to a single server.

Furthermore, aggregation centers act as a blockchain node in the network, so there is a 1:1 relationship between the centers and the nodes. Consequently, the computational overhead, storage requirements, and network maintenance associated with the blockchain are managed entirely by centers, leaving companies without the burden of the technological requirements necessary for the blockchain implementation. Consequently, aggregation centers host the DPP software under the Software as a Service (SaaS) model.

It is important to note that every aggregation center implements the same DPP software, so, from a technical point of view, there is no difference in affiliating in different aggregation centers (the process remain consistent disregarding of the center chosen). Aggregation centers can charge companies for hosting and notarization services, which can lead to new business opportunities. In order to keep the costs for companies at a sustainable level, the presence of healthy competition between the centers is fundamental.

The price, as well as the chosen aggregation center, does not affect the notarization process in any way. In the blockchain network, all nodes have the same voting power and the same probability of becoming a proposer node, so the chosen center does not guaranty any advantage from both technical and legal point of view.

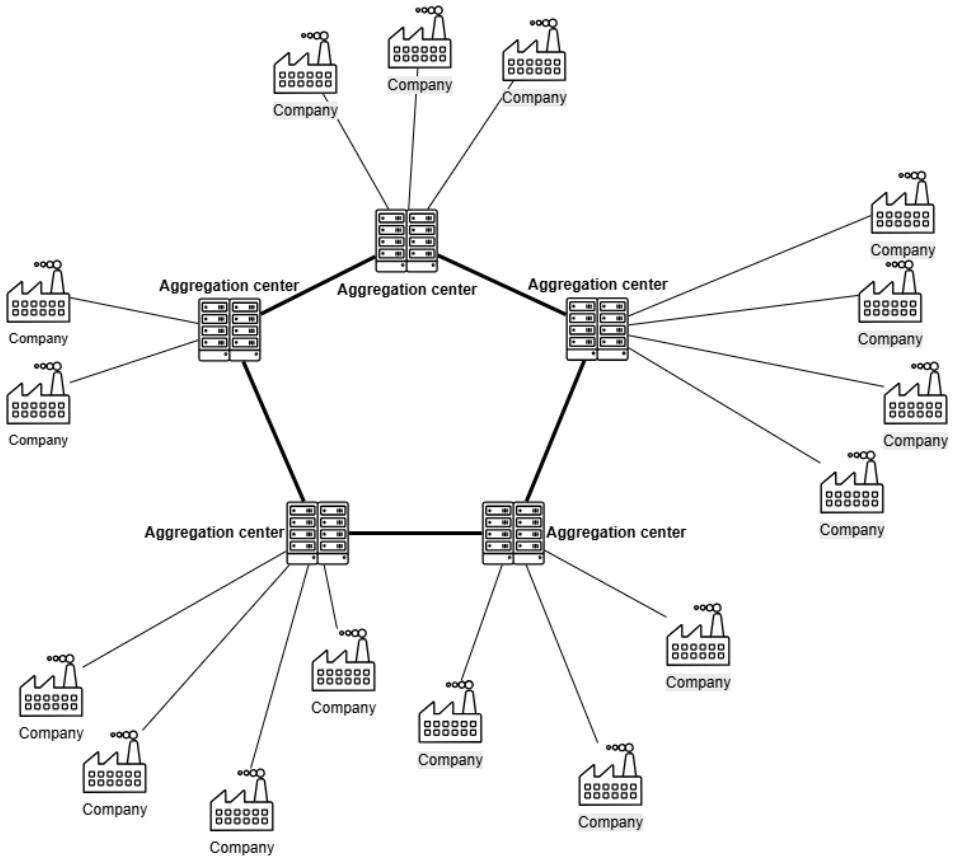

Figure 4.1: Architecture of the blockchain

The security of the system is demanded by the aggregation center, not by the affiliated companies. The responsibility for system security and his functioning is a duty of the aggregation center, rather than the affiliated companies. The picture 4.1 summarize the architecture graphically, clearing the most important aspect that are:

- Companies send messages only to their aggregation centers;
- Aggregation centers are the only node of the blockchain;
- Companies are **not** direct blockchain nodes.

4.2 Permissioned blockchain

The notarization system is based on a permissioned blockchain, where only aggregation centers can participate as nodes. The use of a permissioned blockchain allows for a controlled environment

where each node knows each other. It is possible to create a node without voting power to allow synchronization and verifications of competent authorities; this node will receive every block but has no right to participate in the voting process and cannot be a proposer node. This node will act as an observer.

The fact that every node has the same voting power, as explained in the previous section, guaranties the neutrality of the network because no node can dominate the network.

Due to the fact that the blockchain is not open-access, the admission of a aggregation center (thus, a new node) is a manual operation. Each existing aggregation center has to update their accepted nodes; this is a necessary policy to make the blockchain fair. Consequently, an aggregation center must reach an agreement with the existing nodes to be inserted in the blockchain; the agreement may include money transaction or other economical aspect but for the neutrality concept, this does not affect the notarization validity, too. The economic aspect is not the fundamental point of this research, but it is clear that introducing a new aggregation has the following consequence:

- Increasing the number of nodes enhances the resilience, credibility, and popularity of the network. A direct consequence is that a major popularity can lead to an increase in the number of affiliated companies for an aggregation center, increasing the profit;
- Every new node can be potentially disadvantageous because it can become a competitor; companies may change their center causing economic losses to the previous one.

It is important to emphasize that this architecture is not intended to function as a unique global blockchain. Consequently, the software implemented and the blockchain used are not meant to be the sole blockchain used in all fashion companies, so it is possible that there will be more than one blockchain for the same purpose.

4.3 Audit log

The blockchain does not store full snapshots of the entire database because it contains a large amount of data, and, to fully notarize the network, it would be necessary to take a snapshot of every database in the blockchain. Furthermore, the notarization system implies that the data in the

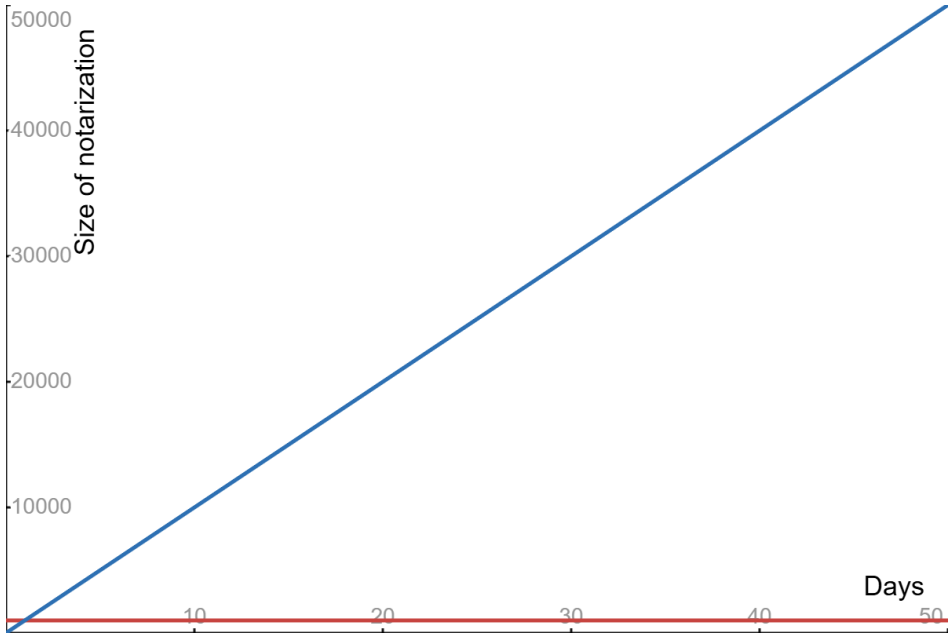

Figure 4.2: Size of notarization. The red function explains the audit log scalability, while the blue function the full snapshot approach

database increase over time due to the fact that, for traceability reasons, data are append-only and are not deleted.

To solve this problem, the proposed notarization framework takes into account only changes in the data, making the blockchain as an audit log. This approach makes things to notarize constant over time, while the database size grows linearly, ensuring that the size of data notarized is proportional to the daily activity and not to the size of the databases.

Consequently, storing full snapshots of entire databases is not scalable. For example, if one thousand additions are added to the database every day, there are one thousand data to notarize every day. After one hundred days, the database has collected one million items. As shown in Figure 4.2, the two functions diverge.

Furthermore, this approach is in line with the core principle of notarization. Traditionally, a document is signed when something changes (i.e. a property, an ownership, or money transactions). Notarizing something that has not changed will cause an overhead that increases over time; if something is notarized, it is implicit that it hasn't changed.

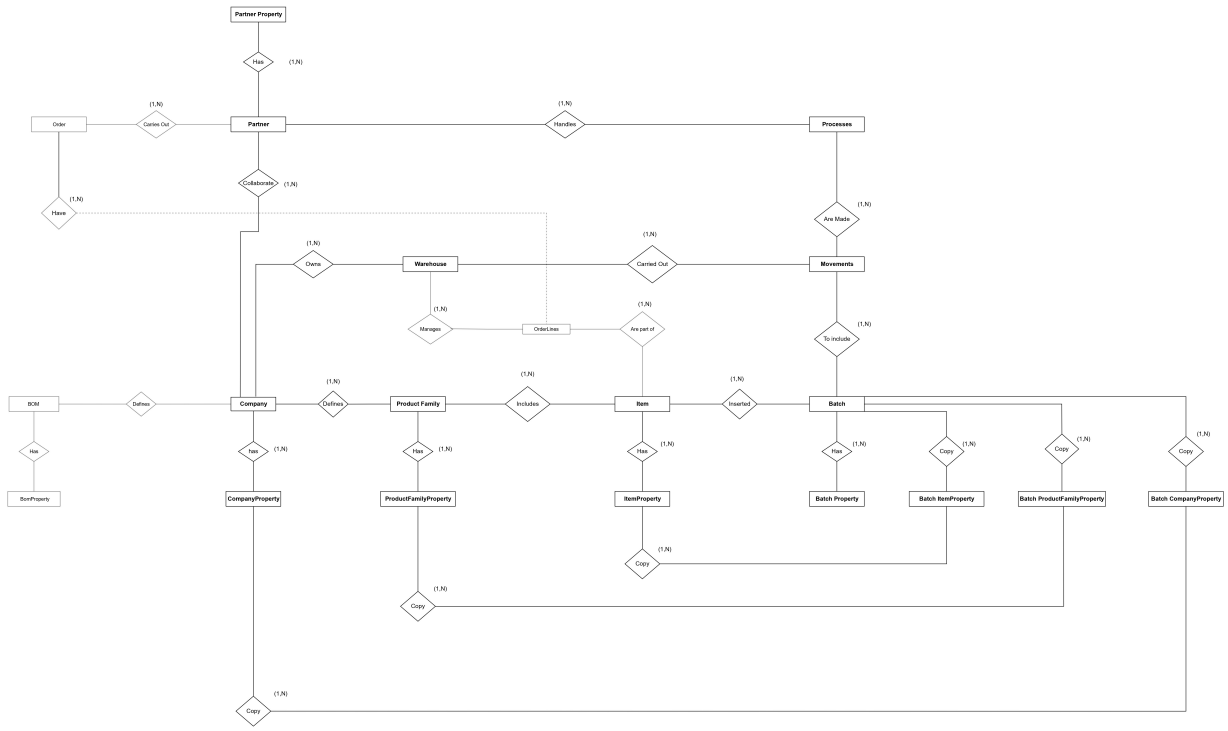

Figure 4.3: ER diagram of the database

4.4 Database

For notarization purposes, not all stored data are subject to notarization. This is because the software also has the purpose of being a management system, so it memorizes also operational and administrative data that are not relevant to the legal requirements of DPP notarization.

Figure 4.3 illustrates the Entity-Relationship (ER) diagram of the existing software that explains the structure of the database. Entities highlighted in bold characters are the entities designed to be notarized. The proposed notarization process is flexible: if other entities will have the legal obligation to be notarized, the software can be simply modified to include such new entities. The modification has to be made in the code (thus, by a programmer) but the software is designated in a way that allows fast integrations.

The existing relationships between the database entities create implicit dependencies that must be reflected in the notarization process. For instance, a batch is a sub-component of an item, so its notarization will be inconsistent if its parent entities are not notarized. For that reason, instances

within child entities will be ineligible for notarization until the notarization of their respective parents.

This approach is in contrast with the requirement of simplicity explained in the previous chapter, but it is necessary to keep logical consistency between all notarized data.

5. Implementation

This chapter discusses how the notarization system is implemented, following the requirements and the architectural design established in chapters 3 and 4, respectively.

5.1 Notarization status

All data that are subject to notarization (according to figure 4.3) follow a lifecycle, implemented by creating four possible stages in which the data can be; these stages are called *notarization status*.

The four statuses are the following:

- Pending;
- Ready;
- Blockchain;
- Deleted.

The pending status is the starting status for all the data and it has to be considered as a draft phase. Because of that, the data in pending status can be modified or deleted without any consequences. The creation of pending status matches the requirement that the system has to mitigate human error.

The Ready status represents data which are ready to be notarized by the blockchain. To move to this stage, the data have to be signed using the passkey system. Like in pending status, data modification and deleting is allowed since the effective notarization in the blockchain; making these actions results in a signature invalidation and a roll back to the pending status. It is possible to roll back to the pending status, also pressing a dedicated button if there is a necessity to do such operations, without deleting or modifying that.

The blockchain status represents data notarized on the blockchain network that have not been modified or canceled. At this point, the record in the local Django database is perfectly synchronized with the immutable evidence on the network. Data in this status can still be modified or canceled,

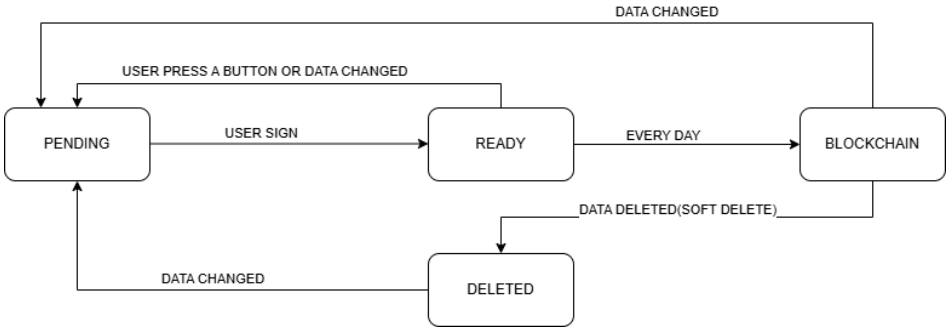

Figure 5.1: Notarization status

but this does not affect the notarization process already completed; instead, a new draft will be created (pending status) to begin a new notarization process.

The deleted status explains that the notarized data is deleted. In reality, this works like a flag system, highlighting the data for which the company has the will to delete. This works like a soft deletion mechanism, in which data are not physically deleted from the database because it will be inconsistent since notarized data have to remain accessible for transparency and traceability reasons. Since data in deleted status are not physically deleted, there is the possibility of modifying those data; such action causes the creation of a new draft of that (pending status). It is important to note that only notarized data will reach deleted data, since canceling data from other status physically delete the data.

Since the existing software is built on the Django framework, the statuses are implemented using the Django models in the `models.py` file. The code is in the snippet 5.1

The figure 5.1 graphically explains the four statuses and how the data change them.

Code 5.1: Notarization status code

```
class NotarizationStatus(models.TextChoices):
    PENDING = 'PND', 'Pending'
    READY = 'RDY', 'Ready'
    BLOCKCHAIN = 'BLK', 'Blockchain'
    DELETED = 'DEL', 'Deleted'
```

5.2 Signature

To transit data from a *pending* status to *ready* status, a digital signature is required to prove its authenticity.

As stated in section 2.2, making this transition only by pressing a button after the initial password authentication does not meet the eIDAS regulation. Consequently, a signature mechanism based on Passkey technology has been implemented. The choice of using passkey technology is due to its better user experience, improved security, and the fact that this technology is widely supported by many devices. This mechanism certifies authenticity, making the transaction in the database immutable before the effective notarization process.

5.2.1 Passkey registration

In order to enable passkeys signature, companies have to register their passkeys using the interface in the company detail screen.

It is possible to register multiple passkeys for every company, allowing them to choose whether to use a cloud-based credential synchronization feature or not.

Passkeys are implemented using the webauthn[1] library for python. The passkey is strictly linked to the domain name of the website and other several metadata fields required by the WebAuthn standard. The fields registered are:

- **rp_id**, a unique identifier of the service, that typically matches the domain name as in this case,
- **rp_name**, a name used to recognize the service in a human-readable format to allow passkey identification for the final user;
- **user_id**, an identifier of the authenticated user that, in this service, is the active company that is authenticated,
- **user_name** and **user_display_name**, a name used to recognize the authenticated user in a human-readable format.

[1] https://pypi.org/project/webauthn/

- **authenticator_selection**, a parameter that defines the authentication requirements for the user to use the signature functionality. To fully meet the eIDAS regulation, it is fundamental to set this value to REQUIRED; this ensures that the user must authenticate himself (using biometrics or PINs) and the device is at the sole control of the user.

Code 5.2: Options used for passkey signature

```
options = generate_registration_options(
    rp_id='tesidpp.it',
    rp_name="Sistema Notarizzazione",
    user_id=str(company.company_code).encode('utf-8'),
    user_name=company.company_legalname,
    user_display_name=company.company_legalname,
    authenticator_selection=AuthenticatorSelectionCriteria(
        user_verification=UserVerificationRequirement.REQUIRED
    )
)
```

The code snippet 5.2 shows the implementation of the options for passkeys.

In order to complete the registration phase, the server generates a cryptographic challenge that the client must sign using his private key. The server verifies the signature using the public key. The security of this operation is ensured by the constraint explained in section 3.4. Using Windows 11, the authentication is required twice; the first to unlock the Trusted Platform Module and the second to register the passkey.

5.2.2 Selection of data to notarize

Every entity in the database subject to notarization has a function called `get_data_to_sign` that returns a JSON with all the attributes of the entities that must be notarized. JSON was selected due to its ubiquitous role in modern web services and its native compatibility with RESTful APIs.

Code 5.3: `get_data_to_sign` for batches model

```
def get_data_to_sign(self):
```

```
properties = self.properties.all().order_by('id')
data: dict[str, Any] = {
    'id': self.id,
    'status': self.status,
    'company': self.company.company_legalname,
    'batch_code': self.batch_code,
    'item': self.item.item_name,
}

properties_list = []
for prop in properties:
    if prop.property_value_type == "U":
        doc_hash = notarize_single_pdf(prop)
        properties_to_add = model_to_dict(prop)
        properties_to_add['document_hash'] = doc_hash.hex()
    else:
        properties_to_add = model_to_dict(prop)
        properties_list.append(properties_to_add)
data['properties'] = properties_list
return json.dumps(data, sort_keys=True,
    cls=DjangoJSONEncoder)
```

The lines in the code snippet 5.3 (an instance of the `get_data_to_sign` function) take into account the attributes of the entity and his nested relationship, such as entities and their specific properties. In the example shown, following the figure 4.3, Batch and BatchProperties represent this specific nested relationship. To ensure consistency between data, the signature of an entity includes all its specific properties.

A critical aspect of this process is the canonicalization of JSON. The returned JSON contains a dictionary that includes the attributes to notarize ordered alphabetically by the keys; this is ensured by the `sort_keys=True` parameter in the return statement of the function. The order of the attribute represents a fundamental point for the objective of notarization, since different orders will generate

different output of the hashing functions, due to the mathematical properties discussed in section 3.3. Consequently, a strict order allows the signature and his verification to work correctly.

5.2.3 Notarization of documents

As shown in the code snippet 5.3, some properties can be an url that points to a document in a cloud storage, such as Google Drive. Since the document can be modified without altering the url, it is clear that notarizing only the url cannot ensure the immutability of the document. So, the signature includes the cryptographic hash of the document itself.

It is important to note that to correctly calculate the hash of the document, it is necessary to download it instead of using the preview of the browser. In fact, the preview contains metadata related to the browser that inevitably changes within multiple visits, making the use of the preview inadequate.

To download a file stored in Google Drive it is necessary to slightly change the url to point to a direct download, so a function returning the download link is implemented. The document is hashed using the sha256 function; the use of this hash function allows splitting the document into little segments (chunks) to handle large files in a more efficient way. This is possible for the following mathematical property of the sha256 function that allows an iterative calculation of the hash [8].

$$H_i = sha256(H_{i-1}||chunk_i)$$

The code snippet shows 5.4 the implementation of this functionality.

Code 5.4: Notarization of document in external cloud storages

```
def create_link_download_for_drive(link):
    id_item = re.search(r'/d/([^/]+)', link)

    if id_item:
        file_id = id_item.group(1)
        return f"https://drive.google.com/uc?export=
            download&id={file_id}"
```

```
    return link


def notarize_single_pdf(link_pdf):
    link_download = create_link_download_for_drive(link_pdf)
    digest = hashlib.sha256()
    with requests.get(link_download,
        stream=True, timeout=60) as r:
            r.raise_for_status()
            for piece in r.iter_content(chunk_size=65536):
                if piece:
                    digest.update(piece)

    digest_bytes = bytes.fromhex(digest.hexdigest())
    return digest_bytes
```

5.2.4 Signature with passkeys

Every change in entities subject to notarization is transformed into a JSON using the function explained in the previous section.

When a user decides to sign a document, the device will use its secure specialized hardware to sign the document, requiring the authentication of the user. Authentication can be ensured with the same requirements as needed for the registration phase (in Windows 11, only one authentication is required instead of the double authentication explained in section 5.2.1).

To maintain a register of signatures and their cryptographic proofs, an entity called SignedModification (not shown in figure 4.3 because it was not in the starting DPP software) is implemented. This entity is crucial for auditing and synchronization. Its attributes are:

- **id**: the primary key;

- **content_type_id**: a Django identifier used to reference the entity in which the transaction is made;
- **object_id**: the identifier of the instance subject to the modifications;
- **data_snapshot**: a JSON string that contains the details of the modification (the output of `get_data_to_sign`);
- **hash_snapshot**: the SHA-256 hash of the data_snapshot field, serving as the unique digital fingerprint of the transaction;
- **signature_id**, the identifier of the passkey used to sign the document;
- **timestamp**: the exact date and time of the signature;
- **audit_log**, a field used to store the exact transaction in which the signature is notarized;
- **is_notarized**, a boolean flag that memorizes if a signed modification has been notarized successfully on the blockchain;
- **leaf_index**, a number that explains the index of the modification in the Merkle tree once notarized;
- **proof**, the root of the Merkle tree in which the modification is notarized.

Considering the fundamental role of this entity in ensuring integrity and non-repudiation, it is fundamental that this entity has a robust backup as a safety net, alongside a disaster recovery strategy.

5.3 Notarization

5.3.1 ABCI server

Every node in the blockchain network has an ABCI Server, user for the integration between Tendermint and the Django application. For the implementation, the abci package from the Tendermint library is used.

The server listens on port 26658 and is implemented on the same machine as the Django application. This co-location allows the DPP service and the blockchain node to share the same IP address, simplifying the deployment within the Aggregation Center's environment.

A Tendermint application requires the implementation of three methods to manage the transaction lifecycle and, consequently, the functionality of the blockchain. There are the following:

- **check_tx**;
- **deliver_tx**;
- **commit_tx**.

The first provides controls on the transaction, filtering transactions that do not respect the right format or are not originated by an authorized node. Given the permissioned nature of the blockchain and the fact that the transactions are all generated by the DPP software, the implemented function always returns 0, a value interpreted as a success code. The second contains the business logic of the transactions used to update the global state of the blockchain, while the third finalizes the changes and increments the block height. It is important to note that this particular implementation handles application state since every aggregation center has its database with its data, with the blockchain acting as an audit log. The only thing that each node must save is the history of changes for every aggregation center. So, for the purpose of notarizing the DPP data, these functions do not implement a logic with the objective of maintaining a common state (like in the Bitcoin context) and only return a value that shows that everything is correct. This is due to the fact that the creation the transaction of the blockchain is created in the same machine and the blockchain is permissioned, so only few nodes can create blocks.

5.3.2 Daily notarization

For implementation purposes, the fact that each company can change a large amount of data in a single day is considered. Consequently, the blockchain has to be scalable. Processing a blockchain transaction for every individual database can lead to issues for aggregation center and the need to bigger hardware requirements.

To mitigate this, the core idea is to generate transactions only once a day, at a predetermined time, instead of broadcasting immediately after the signature. For this implementation, transactions are made every day at 6 P.M. because it represents the typical end of a work day. The latency for a change to the database to be notarized is, for a legal and practical aspect, negligible, as the time elapsed between the creation of a production batch and its sale is significantly longer than the notarization delay. Furthermore, this framework has the advantage of making the possibility of reporting incorrect data even smaller. In fact, there is a two-layer protection against human error:

- Data are notarized only after a signature, meaning that an human has to confirm the data reported;
- An erroneous signature can be deleted until 6 P.M. of the day, providing a further safety net.

As shown in the code snippet 5.5, this implementation is achieved using the background scheduler, part of the AP scheduler library[2].

Code 5.5: The scheduler implementation for the blockchain

```
scheduler = BackgroundScheduler()
scheduler.add_job(notarization, 'cron', hour=18, minute=0)
scheduler.start()
```

5.3.3 Merkle tree

In order to improve the process for verifying a change in data, every transaction is created using the Merkle tree.

A Merkle tree is a structure that allows one to prove that a change is made in that transaction without fully knowing every case. This is due to the use of a binary tree. Every change signed by companies is a leaf of the tree and his hashed together with her neighbored leaf. This process is repeated recursively until the root of the tree is reached. This allows proving with a logarithmic cost (it is necessary to know only log n nodes, if n is the number of changes), while without the Merkle tree the cost is linear because it is necessary to know all the changes. The picture 5.2 explains the structure of a Merkle tree [4].

[2]https://pypi.org/project/APScheduler/

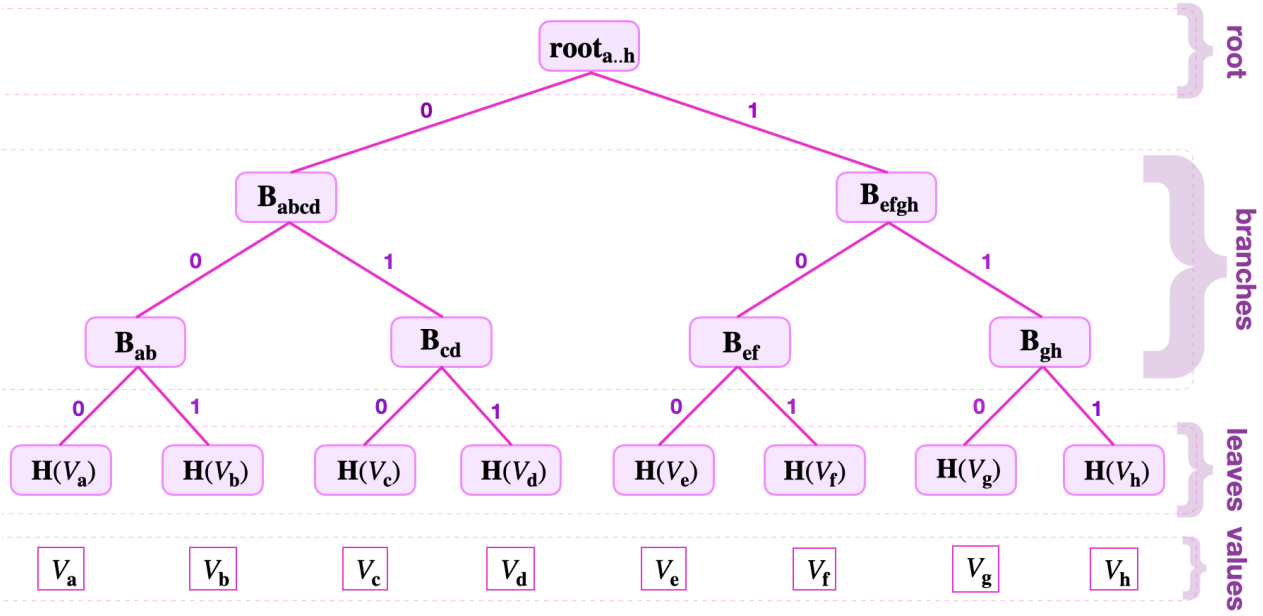

Figure 5.2: A visual representation of the Merkle tree

In DPP software, the Merkle tree is implemented using the pymerkle library[3]. Every signed modification, after conversion to JSON format, is hashed (using the sha256 function) and stored in the Merkle tree. Then, only the value of the root is memorized and then sent to the blockchain.

The code snippet 5.6 shows the implementation of the Merkle tree. This code is part of the core notarization function.

Code 5.6: The implementation of Merkle tree

```
ready = SignedModification.objects.filter(is_notarized=False)
.order_by('timestamp')
tree = MerkleTree(algorithm='sha256')
if not ready:
    return
for item in ready:
    item_json = json.dumps(item.get_data_to_sign(),
        sort_keys=True)
    tree.append(item_json.encode())
```

[3]https://pymerkle.readthedocs.io

```
merkle_root = tree.root.value.hex()
```

5.3.4 Atomicity of the operation

All notarized data must have a status consistent with the notarization process. Consequently, it is necessary to update the SignedModification entity by setting the flag `is_notarized` to true, and change the notarization status for every single instance to the BLOCKCHAIN status.

It is important to consider the possibility of some issues during the notarization process. Since signed modifications are grouped together, a failure at any point can block the process to all the others, leading to an inconsistent state where an instance has the flag `is_notarized` set to true while it has not effectively sent to the Tendermint server. Consequently, it is necessary to wait to update the state of the instances until all have notarized. In a scenario where notarization issues can occur, changes to status and flag must be rolled back to the previous value, making the entire notarization an atomic operation. As a consequence, if an instance that is notarized correctly at 6 P.M., a new attempt to notarize will be done the following days until they are notarized correctly on the blockchain network.

This is achieved using the Django `transaction.atomic` method, as shown in the code snippet 5.7. This snippet shows the sending of the transaction to the blockchain and follows the operations in the code snippet 5.6.

Code 5.7: Atomicity of the notarization

```
with transaction.atomic():
    audit_log = AuditLog.objects.create(
        transaction=json.dumps({
            "merkle_root": merkle_root,
            "leaves": [
                json.dumps(item.get_data_to_sign(),
                    sort_keys=True)
                for item in ready
            ],
        }),
```

```
        hash_transaction=merkle_root_b64,
    )

    for i, item in enumerate(ready):
        proof = tree.prove_inclusion(i + 1)
        item.audit_log = audit_log
        item.proof = proof.serialize()
        item.is_notarized = True
        item.notarized_at = datetime.datetime.now()
        item.save()

        instance = item.content_object
        instance.status = NotarizationStatus.BLOCKCHAIN
        instance.save()
    requests.get(url, params={'tx': merkle_root_hex}, timeout=5)
```

5.4 Tendermint configuration

The Tendermint engine is highly customizable by changing values in the `config.toml` file.

By default, Tendermint periodically creates new blocks with a default interval of one second to verify the activeness of the network for the consensus mechanism. This leads to a large amount of blocks that cause an unnecessary overhead of storage used, potentially making the verification process by the competent authorities more difficult. For that reason, the blockchain network is set to not create empty blocks setting the value `create_empty_blocks = false`.

Furthermore, by default, Tendermint waits some time to execute a block with the objective of grouping more transactions. For the implementation of the DPP notarization, this is redundant, since the Merkle tree is handled at the time of sending transaction, as explained in section 5.3.3. This approach slightly reduces the latency of the notarization.

Unlike public networks that rely on dynamic peer discovery, this implementation implements controlled ingress controlled ingress. The field `persistent_peers` contains a list of the other

nodes in the network, following the structure in snippet 5.8.

Code 5.8: Persistent peers in config.toml fine

```
persistent_peers="pubkey1@IPnode1:port1, pubkey2@IPnode2:port2"
```

5.5 Addition of a new aggregation center

To add a new aggregation center (and its corresponding node) into the blockchain network, it is necessary to satisfy all the conditions explained in section 4.1. Consequently, the new node must have the same operational rights as the existing participants. Specifically:

- The right to propose a new block at the same interval as other nodes;
- The right to validate a new block;
- The same voting power as the other node.

Adding the new node in the `persistent_peers` file in the `config.toml` file (as in the snippet 5.8 is not enough, because it provides network connectivity but does not ensure participation in the voting process and also does not have the ability to propose a new block. The research tries to implement a solution for this issue.

A "special" transaction is implemented, and the node receiving this transaction must accept the new validator node.

To differentiate between this special transaction and the common transaction about changes in the respective database, this transaction is signed using an asymmetric cryptographic signature, implemented via the `criptography` library for python[4]. The management of the private key used for this operation presents a significant governance challenge, due to the fact that the private key has to be known by only one node, while the others know only the corresponding public key. This could lead to a scenario where an aggregation center can potentially monopolize the network. To mitigate this, it could be possible to leave this possibility to a node dedicated to the competent authorities or, following agreements between the aggregation centers, trust a single node that can guaranty the security of the operation.

[4]https://pypi.org/project/cryptography/

The code snippet 5.9 shows the functions used to add a new validator node. The function `add_validator` is called by a command implemented in the manage.py file.

Code 5.9: Methods used for adding validators

```
def deliver_tx(self, tx):
    try:
        data = json.loads(tx.decode())
        if data.get("action") == "add_validator":
            message = json.dumps({
                "action": "add_validator",
                "pub_key": data["pub_key"],
            }, sort_keys=True).encode()

            public_key = Ed25519PublicKey.from_public_bytes(
                bytes.fromhex(ADMIN_PUBLIC_KEY)
            )
            public_key.verify(
                bytes.fromhex(data["signature"]),
                message
            )
            self.pending_validators.append(data["pub_key"])
    except json.JSONDecodeError:
        self.last_root = tx

    return ResponseDeliverTx(code=0)

def end_block(self, req):
    updates = []
    for pub_key_b64 in self.pending_validators:
        pub_key_bytes = base64.b64decode(pub_key_b64)
```

```
            pub_key = PublicKey()
            pub_key.ed25519 = pub_key_bytes

            update = ValidatorUpdate()
            update.pub_key.CopyFrom(pub_key)
            update.power = 10

            updates.append(update)

        self.pending_validators = []
        return ResponseEndBlock(validator_updates=updates)

def add_validator(pub_key_b64):
    message = json.dumps({
        "action": "add_validator",
        "pub_key": pub_key_b64,
    }, sort_keys=True).encode()

    private_key = Ed25519PrivateKey.from_private_bytes(
        bytes.fromhex(settings.ADMIN_PRIVATE_KEY)
    )
    signature = private_key.sign(message).hex()

    data = json.dumps({
        "action": "add_validator",
        "pub_key": pub_key_b64,
        "signature": signature
    })
    data_hex = "0x" + data.encode().hex()
    base_url = settings.TENDERMINT_RPC_URL.rstrip('/')
```

```
requests.get(
    f"{base_url}/broadcast_tx_commit",
    params={'tx': data_hex},
    timeout=5
)
```

6. Containerization and hosting

This chapter discusses the containerization and the deployment strategy for DPP software with its notarization system.

To verify the functionality, a small blockchain composed of two nodes (representing two distinct aggregation centers) is implemented to simulate a real scenario.

6.1 Containerization

The first step for the deployment is to containerize the software with the objective of providing environment consistency across all the nodes where the software will be hosted. To achieve this, Docker[1] is used. The reason beyond this choice is its wide popularity that makes it an industrial standard for containerization.

The containerization of this work requires the creation of more containers, each for a specific service. To provide orchestration between all services, the configuration file `docker-compose.yml` is used. This file defines a multi-container architecture where the Django application, the Tendermint core, and the ABCI server coexist and are dependent on each other.

This section explains the implementation of each service, starting from the one without a condition to start.

The snippet 6.1 shows the starting of the blockchain server. This is implemented using a pre-built image and executing the dedicated file in the application; that starts the abci server, bridging the gap between the DPP software and the Tendermint engine.

Code 6.1: Service blochchain_app in `docker-compose.yml`

```
blockchain_app:
    image: ghcr.io/fabiozanic2000/dppsoftware:latest
    command: python core/server_blockchain.py
    env_file:
      - .env
```

[1] https://www.docker.com/

```
    volumes:
      - .:/app
```

The snippet 6.2 discusses the implementation of Tendermint. To correctly setup a blockchain node, the tendermint service is split into two different parts: one for the initialization and one for the execution of the node. The service `tendermint_init` provides the initialization of a node by checking the presence of the `genesis.json` file and, eventually, create the file. This file is fundamental because it contains parameters that are necessary to uniquely identify the network and his characteristics . The tendermint service starts once the initialization service has terminated correctly. This contains is necessary to launch the node and expose the port to allow the connectivity between the nodes; the ports exposed are the 26657 and 26658, which are the default port for the Tendermint engine. The service also depends on the blockchain_app (in the snippet 6.1) because the node can start only if the blockchain server has started correctly.

Code 6.2: Service tendermint and tendermint_init in `docker-compose.yml`

```
tendermint_init:
    image: tendermint/tendermint:v0.34.0
    volumes:
      - ./tendermint_config:/tendermint_container
    entrypoint: /bin/sh
    command: >
        -c "if [! -f /tendermint_container/config/genesis.json];
        then
            tendermint init --home /tendermint_container;
            fi"

tendermint:
    image: tendermint/tendermint:v0.34.0
    volumes:
      - ./tendermint_config:/tendermint_container
    command: node --home /tendermint_container
```

```
      --proxy_app=tcp://blockchain_app:26658
      --rpc.laddr=tcp://0.0.0.0:26657
    ports:
      - "26657:26657"
      - "26656:26656"
    depends_on:
      tendermint_init:
        condition: service_completed_successfully
      blockchain_app:
        condition: service_started
```

The snippet 6.3 illustrates the implementation of the web service, which hosts the DPP software. This service represents the primary point of interaction for both companies and consumers.

It is important to emphasize that the SQLite database and the static files are mapped to the host machine; this ensures the presence of data, avoiding data loss in the case of a container being restarted or updated. This is achieved using Docker volumes.

The environmental variable `TENDERMINT_URL` is crucial to bridge the gap between the blockchain part of the system.

The web service starts only when the Tendermint container used for notarization (explained in snippet 6.2) has started correctly and, consequently, is ready to notarize data. This dependency prevents potential error in the boot phase.

Code 6.3: Service web in `docker-compose.yml`

```
web:
    image: ghcr.io/fabiozanic2000/dppsoftware:latest
    volumes:
      - ./db.sqlite3:/app/db.sqlite3
      - staticfiles:/app/staticfiles
    environment:
      - TENDERMINT_URL=http://tendermint:26657
    depends_on:
```

```
      tendermint:
        condition: service_started
```

The snippet 6.4 shows how the nginx server is implemented and the TLS certificate is obtained. The nginx service acts as the gateway for the connection in HTTP (port 80) and HTTPS (port 443). Consequently, it manages the TLS connection, separating the encryption process from the Django application. The files necessary for the HTTPS connection are shared volumes.

The use of HTTPS is mandatory for the use of passkeys. For that reason, connections to port 80 are redirected to the secure protocol. To obtain TLS certificate, the system integrates Certbot, a client for the Let's Encrypt CA.

The use of HTTPS provides secure communications between the client and the server (aggregation center), providing the authenticity of the latter and the confidentiality of the messages. The `certbot` service is implemented to automatically check the validity of the certificate and renew it if necessary.

Code 6.4: Services nginx and certbot in `docker-compose.yml`

```
nginx:
    image: nginx:alpine
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf:ro
      - ./certbot/conf:/etc/letsencrypt:ro
      - ./certbot/www:/var/www/certbot:ro
      - staticfiles:/app/staticfiles:ro
    depends_on:
      - web
    restart: unless-stopped

  certbot:
```

```
image: certbot/certbot
volumes:
  - ./certbot/conf:/etc/letsencrypt
  - ./certbot/www:/var/www/certbot
entrypoint: "/bin/sh -c 'trap exit TERM; while :; do
    certbot renew; sleep 12h & wait $${!};
    done;'"
```

6.2 Hosting

This section details the cloud infrastructure used to host the DPP software and the notarization nodes.

6.2.1 Cloud Provider

The DPP software, together with the notarization service, is deployed using Amazon Web Service (AWS)[2].Specifically, The service used is AWS Lightsail[3], a solution that offers predictable monthly costs and a simplified interface.

With the objective of minimize the cost, it was attempted to use a lower-tier instance. In particular, the instance with the minimum cost that provides only 512 MB of RAM was considered as clearly insufficient for running smoothly the DPP software and the Tendermint engine. Consequently, a more powerful instance was chosen. Each instance has the following specification:

- Ubuntu 24.04 LTS (Long Term Support) as an operating system;
- 60 GB of storage;
- 2 GB of RAM;
- 2 vCPU;
- 3 TB of transfer.

[2]https://aws.amazon.com/

[3]https://aws.amazon.com/lightsail/

The choice of those requirements is considered a good tradeoff for the two-node blockchain simulation. However, it is important to note that if this software is used by a large aggregation center, the requirements needed may increase significantly making the specification previously no longer sufficient.

An AWS Lightsail instance provides a static public IP address with IPv4 and IPv6 versions. This ensures consistency of the connection between the node because the IP address will change at every restart if it is not static.

Each instance provides a firewall. Following the basic standard security principle, the firewall has a deny all approach, with connection to the system whitelisted only if they represent a typical use case. Consequently, only the following ports are open to connection:

- Port 22, to allow the SSH connection to configure the remote machine. For testing convenience, this port is accessible from every IP address, but in a production environment this has to be avoided;
- Port 80, for HTTP connections. Every connection to HTTP will be redirected to its secure counterpart;
- Port 443, for HTTPS connections;
- Port 8000, for Django;
- Port 26656, 26657, and 26658 for the tendermint engine.

6.2.2 Domain name

To facilitate user access, the infrastructure is assigned to a registered domain name. This provides a human-readable name to digit in the browser and is essential for the issuance of valid TLS certificates by Let's Encrypt, as discussed in the previous sections.

Domain registration and management are performed using the Aruba service[4]. The DNS configuration utilizes A Records to map each hostname to its respective static IPv4 address provided by AWS.

In the simulated scenario, the two nodes are accessible through the following domain name:

[4]https://www.aruba.it/

- *www.tesidpp.it*;
- *secondonodo.tesidpp.it*.

Although the second node uses a subdomain of the first, it simulates a real scenario with two distinct aggregation centers.

6.2.3 Nginx configuration

Nginx serves as a web server and reverse proxy for the DPP system. It was selected because of its performance, its wide popularity, and his status as an industrial standard. The server's behavior is governed by the `nginx.conf` file. This section details the implementation of that file, referring to the primary node. With the exception of the domain name, the configuration is identical for each node in the blockchain network.

As illustrated in the code snippet 6.5 and as explained in the previous sections, the server handles HTTP connections upgrading each one to the HTTPS protocol. This is achieved in the last line of the snippet where the redirection code `301 Moved Permanently` is used. This code instructs the browser to submit a new request using the secure protocol [3].

Code 6.5: Nginx redirection of HTTP connection

```
http {
    include /etc/nginx/mime.types;
    default_type application/octet-stream;
    server {
        listen 80;
        server_name tesidpp.it www.tesidpp.it;

        location /.well-known/acme-challenge/ {
            root /var/www/certbot;
        }

        location / {
```

```
        return 301 https://$host$request_uri;
    }
}
```

The snippet 6.6 shows the nginx management of HTTPS connections. To provide the best security level, only modern versions of the TLS protocol are supported due to the vulnerability of older versions [7]. This prevents a downgrade attack[5]. Specifically, only versions 1.2 and 1.3 are supported.

In the location section, requests are forwarded to the Django application using its default port 8000. This is due to minimize modification to the existing DPP software. To maintain compatibility with existing DPP software, Nginx acts as a transparent proxy that injects critical headers. This is fundamental for Django to identify that the original request was secure.

Code 6.6: Nginx management of HTTPS connections

```
server {
    listen 443 ssl http2;
    server_name tesidpp.it www.tesidpp.it;

    ssl_certificate
        /etc/letsencrypt/live/tesidpp.it/fullchain.pem;
    ssl_certificate_key
        /etc/letsencrypt/live/tesidpp.it/privkey.pem;

    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers HIGH:!aNULL:!MD5;
    ssl_prefer_server_ciphers on;

    location /static/ {
        alias /app/staticfiles/;
        expires 30d;
```

[5]Downgrade attacks try to force the use of an old version of TLS to exploit its vulnerabilities.

```
        add_header Cache-Control "public, immutable";
    }

    location / {
        proxy_pass http://web:8000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For
            $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto https;
    }
}
```

6.3 CI/CD

To automatize the deployment of the work and ensure a consistent deployment across all nodes, a Continuous Integration and Continuous Deployment (CI/CD) pipeline was implemented. Since GitHub is already used for the versioning of the software, GitHub Actions was selected for the implementation of the CI/CD framework.

Using such a mechanism has many advantages, from standardizing the build process to minimizing the manual effort to update the software in the remote instances.

To use GitHub Actions it is necessary to use the `deploy.yml` file in which the pipeline is defined. This file is automatically triggered by GitHub at the time of every push.

This section illustrates how such a pipeline is implemented. The workflow is divided into two parts:

- Build and push, which automates the containerization of the software;
- Deploy, that automates the deployment to the remote AWS instance.

6.3.1 Build and push

The part `build-and-push` is illustrated in the snippet 6.7.

In this part of the pipeline the software is containerized, transforming the source code to a deployable artifact. and then the result is stored in the appropriate registry in GitHub.

The process follows a standardized containerization workflow:

1. Source Checkout, to fetch the latest version of the repository;

2. Registry Authentication, to securely upload the resulting image, the runner authenticates to GHCR using a token. In order to follow the basic principle of security, the credentials are not hard-coded; instead, the job uses a GitHub Secret (`TOKEN_REGISTRY`). This avoids a scenario in which someone can read credentials and use them maliciously.

3. Image Construction and Publication, using the `docker/build-push-action` step. The system builds the Docker image based on the specifications explained in section 6.1. Once the build is completed successfully, the image is pushed to the registry. The image just built is then tagged as the latest.

Code 6.7: Build and push job in the CI/CD pipeline

```
build-and-push:
    runs-on: ubuntu-latest
    permissions:
      contents: read
      packages: write
    steps:
      - name: Checkout del codice
        uses: actions/checkout@v4

      - name: Login al Registry di GitHub
        uses: docker/login-action@v3
        with:
```

```
      registry: ghcr.io
      username: ${{ github.actor }}
      password: ${{ secrets.TOKEN_REGISTRY }}

  - name: Build e Push dell'immagine
    uses: docker/build-push-action@v5
    with:
      context: ./DPPSoftware
      push: true
      tags: ghcr.io/fabiozanic2000/dppsoftware:latest
```

6.3.2 Deploy

The second stage of the CI/CD pipeline serves to correctly deploy the image obtained in the previous section to the AWS Lightsail instance, consequently handling the distribution of the updated software.

The listing 6.8 illustrates the code of the job `deploy`, executed with a secure SSH tunnel. It is important to emphasize that this stage is executed only after the completion of the `build-and-push` stage, as indicated by the instruction `needs: build-and-push`. The deployment workflow is structured as follows:

1. Authentication to the remote machine, using the SSH protocol. For the same reason explained in section 6.3.1, the critical variables that must be used for SSH authentication are stored in the GitHub variables and not hard-coded;

2. Injection of the environmental variable into the remote machine in the `.env` file;

3. Authentication to the GitHub Container Registry;

4. Pull of the image from GHCR;

5. Cleaning of legacy containers that can generate conflict with the new version of the software;

6. Starting the containers, following the order specified in section 6.1;

7. Execution of Django management commands to synchronize the relational schema of the database and web assets without manual effort.

Code 6.8: Deploy job in the CI/CD pipeline

```
deploy:
    needs: build-and-push
    runs-on: ubuntu-latest
    steps:
      - name: Deploy via SSH
        uses: appleboy/ssh-action@master
        with:
          host: ${{ secrets.LIGHTSAIL_HOST }}
          username: ${{ secrets.LIGHTSAIL_USER }}
          key: ${{ secrets.LIGHTSAIL_SSH_KEY }}
          script: |
            echo "ADMIN_SECRET=${{ secrets.ADMIN_SECRET }}"
                >> /home/ubuntu/DPPSoftware/DPPSoftware/.env
            echo "ADMIN_PRV_KEY=${{secrets.ADMIN_PRV_KEY}}"
                >> /home/ubuntu/DPPSoftware/DPPSoftware/.env
            echo "ADMIN_PUBLIC_KEY=${{secrets.ADMIN_PUBLIC_KEY}}"
                >> /home/ubuntu/DPPSoftware/DPPSoftware/.env
            echo "${{secrets.TOKEN_REGISTRY}}" | docker login
                ghcr.io -u ${{github.actor}} --password-stdin
            cd /home/ubuntu/DPPSoftware
            git pull origin ${{github.ref_name}}
            cd DPPSoftware
            docker compose pull
            docker compose up -d --remove-orphans
            docker compose exec -T web python manage.py
                collectstatic --noinput
```

```
docker compose exec -T web python manage.py migrate
docker image prune -f
```

7. Results

This chapter details the experimental results obtained after the deployment of the notarization system, from both a technical and a user point of view. To demonstrate the correct functionality, some test batches are used to simulate the framework in a potential real case. The scenario considered is the one detailed in chapter 6, where two nodes each represent a distinct aggregation center.

The illustrated test is based on the notarization of three batches called ZUKK001, ZUKK002 and ZUKK003.

7.1 Remote Procedure Call

Tendermint provides a built-in RPC (Remote Procedure Call) interface that allows users to query the node's status and retrieve cryptographic evidence through standard HTTP GET requests. The requests are to be made to port 26657, which is the default port for the Tendermint RPC. The following list contains the crucial endpoint:

- `/block`, it serves the last block added;
- `/blockchain?minHeight=X\&maxHeight=Y`, to see a range of blocks;
- `/status`, to see the status of the blockchain;
- `/block?height=X`, to see the content of a specific block;
- `/tx?hash=0x<tx>`, to find the notarization of a specific transaction.

The last point is fundamental because it provides querying on a specific transaction; since the blockchain data are public, this allows the competent authorities to certificate the notarization of a specific transaction (thus, a change in the database).

7.2 Technical result

7.2.1 One change notarization

This section illustrates the notarization process for a single batch.

In this case, this is the only batch notarized and consequently it is the only leaf of the Merkle tree. So, the Merkle tree root should be identical to the resulting hash of the batch. Consequently, in the transaction field, the same value is stored in the AuditLog model in the Django SQLite integration database (figure 7.1).

To verify the result, the RPC protocol is used. The listing 7.1 shows the response, so the content of the node that has notarized the change. It is important to note that the listing does not contain the full content of the node, highlighting only the important part to show the notarization. The complete node content is illustrated in the appendix A.1.

Code 7.1: Notarization of a single change in the blockchain


```
{
  "jsonrpc": "2.0",
  "id": -1,
  "result": {
    "block_id": {
      "hash": "CD0B150F02325A69A1DC1564BFFA5065C0456F5E1C70DCFBE6
  F9B6C840547C00",
      "parts": {
        "total": 1,
        "hash": "0EF2C23AE5E19490FB814C0DD9A752AD8F737BEAD8B57EE6
  73ED736A8496D4EB"
      }
    },

    "data": {
        "txs": [
```

```
            "McLdxa6bdSR1iKBwt/l1Hwh8PH54J1Q7qLuRlFvAgh0="
          ]
        },
        "evidence": {
          "evidence": []
        }
    }
}
```


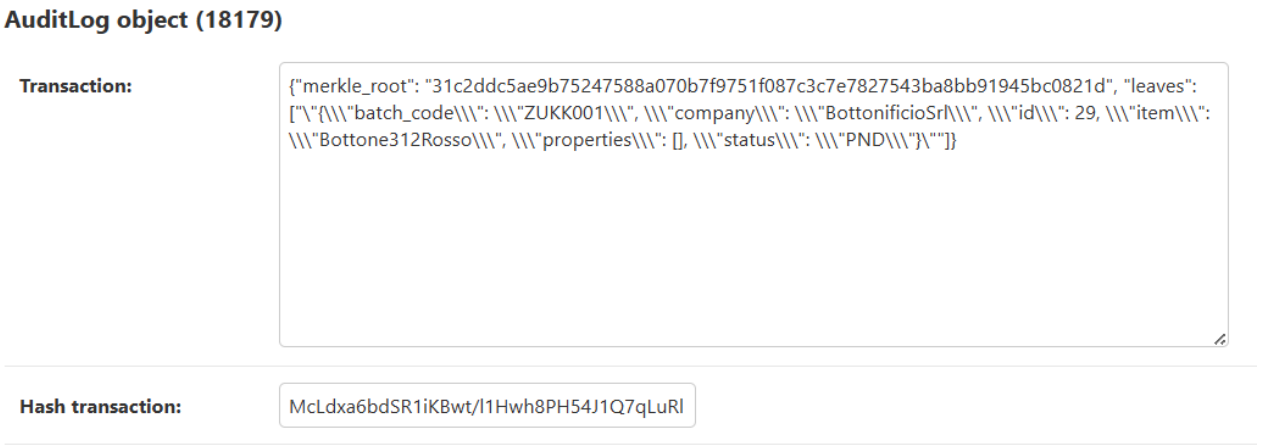

Figure 7.1: AuditLog instance of the notarization of ZUKK001

7.2.2 Two-changes notarization

This section details the notarization of two distinct modifications in the database (creation of batches ZUKK002 and ZUKK003) in the same block.

In this scenario, the Merkle root is mathematically different from the individual hash of the single modification because the Merkle root combines the two individual hashes together, as explained in section 5.3.3. Consequently, the Merkle root represents the integrity of both modifications simultaneously. The listing 7.2 contains the core data of the node that notarize the two blocks; as in the previous scenario, the full content of the node is available in the appendix A.2. The figure 7.2 illustrates the two batches notarized in the SignedModification model.

Code 7.2: Notarization of two changes in the blockchain


```
{
  "jsonrpc": "2.0",
  "id": -1,
  "result": {
    "block_id": {
      "hash": "C9B7F0A212BE928CD8DC8267CECCC6B8605B6D1343A8B80A08
   DB6CD6E7562F1F",
      "parts": {
        "total": 1,
        "hash": "0CC01863F859C68D03C8A214C01AEDC2D87958C264770E6D
   9AC0011146FF8B9B"
      }
    },

    "data": {
      "txs": [
        "2+ahyTatmlVCExw0Sv1CdF2+ZVCcGo0t1hT6g1gKwQw="
      ]
    },
    "evidence": {
      "evidence": []
    }
  }
}
```

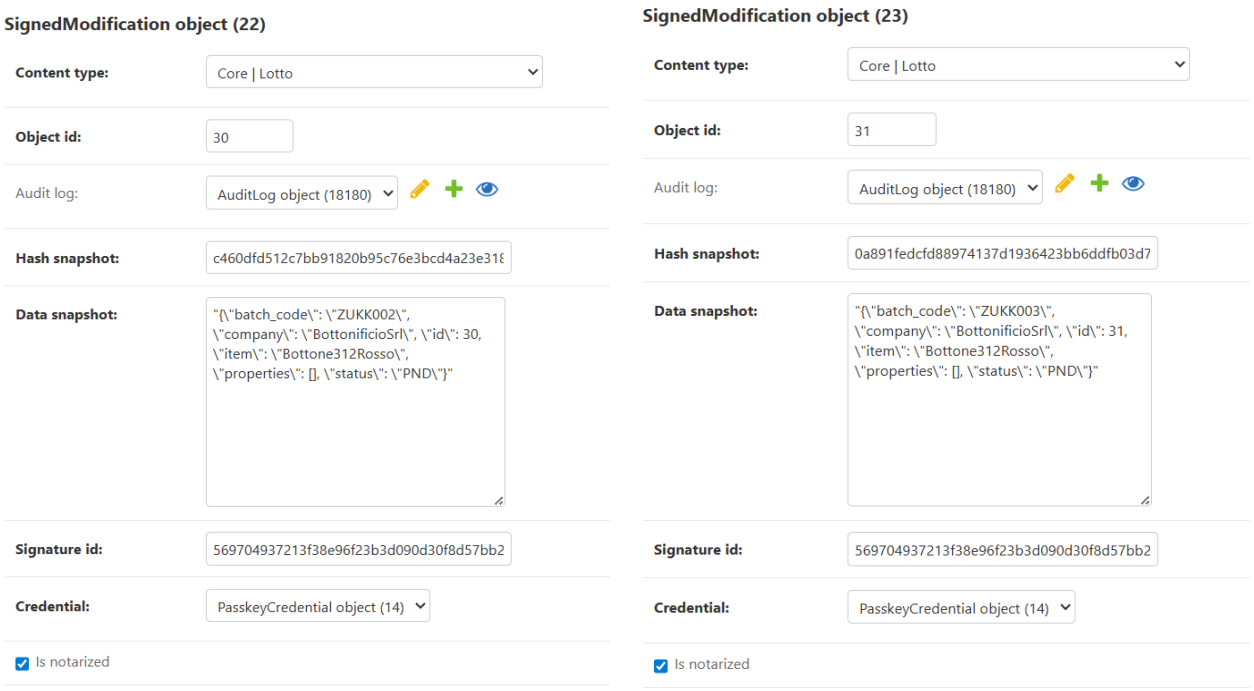

Figure 7.2: Batches ZUKK002 and ZUKK003 in SignedModification

7.3 User point of view

Figure 7.3 shows the user interface of the batch list page. It is important to emphasize that this interface is very similar to the previous one; the only difference is the addition of the last two columns of the table that serves the notarization process. This is due to respecting the requirements of the simplicity to use, because a user does not have to learn a new interface.

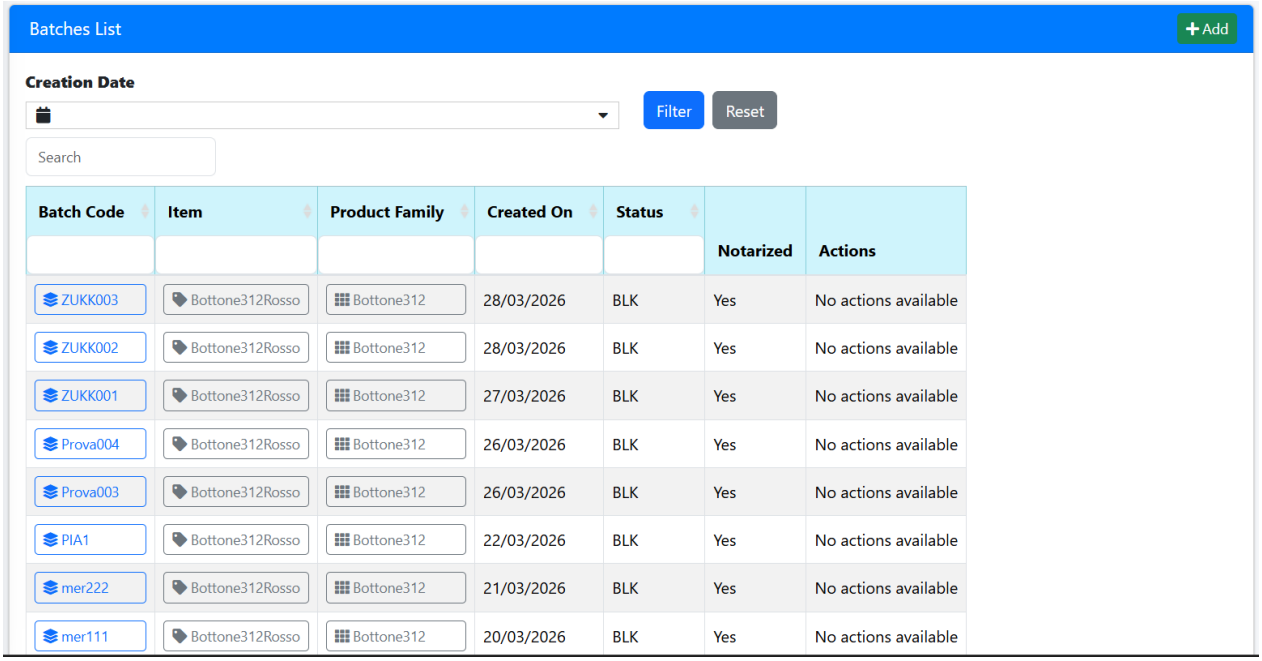

Figure 7.3: List of batches in the DPP software

When the user presses the *sign* button, in the case of Windows as operating system, a Windows Hello prompt is triggered. The user has to authenticate using biometric data or a PIN. Then, the signature is automatically made without requiring any technical knowledge from the user. Figure 7.4 shows the Windows Hello prompt screen.

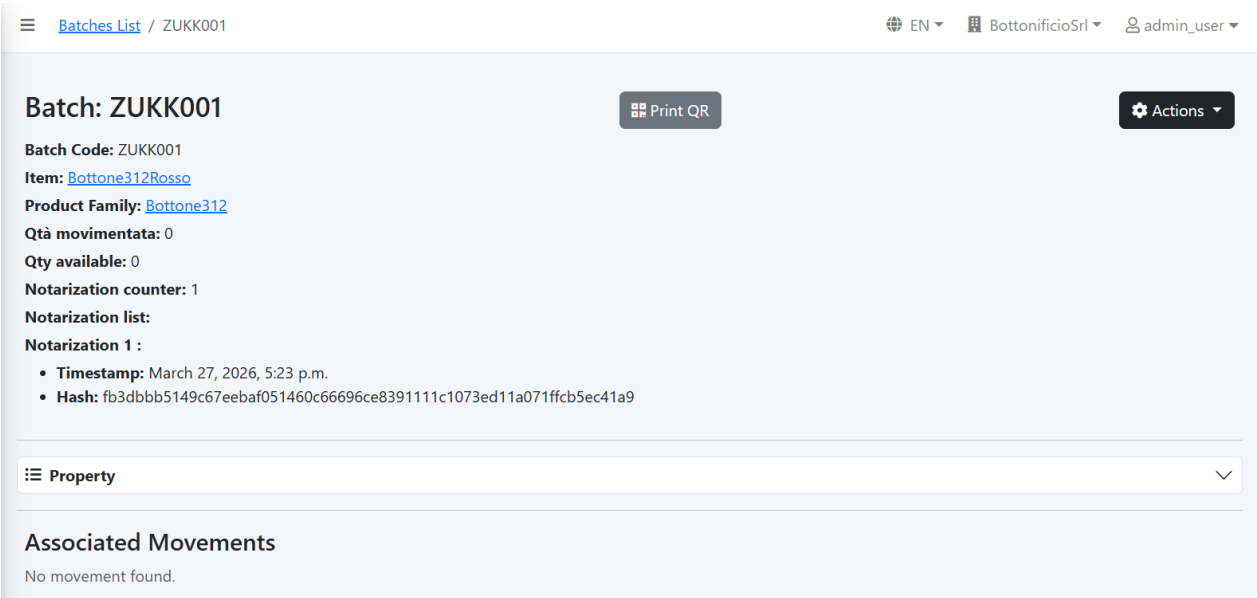

Figure 7.4: Details of ZUKK001 in the DPP software

Finally, in the detail page of an instance, such as batch ZUKK001 used as an example, the history of changes in the database is detailed along with the date and time of this change. Figure 7.5 illustrates the user interface in the batch detail page.

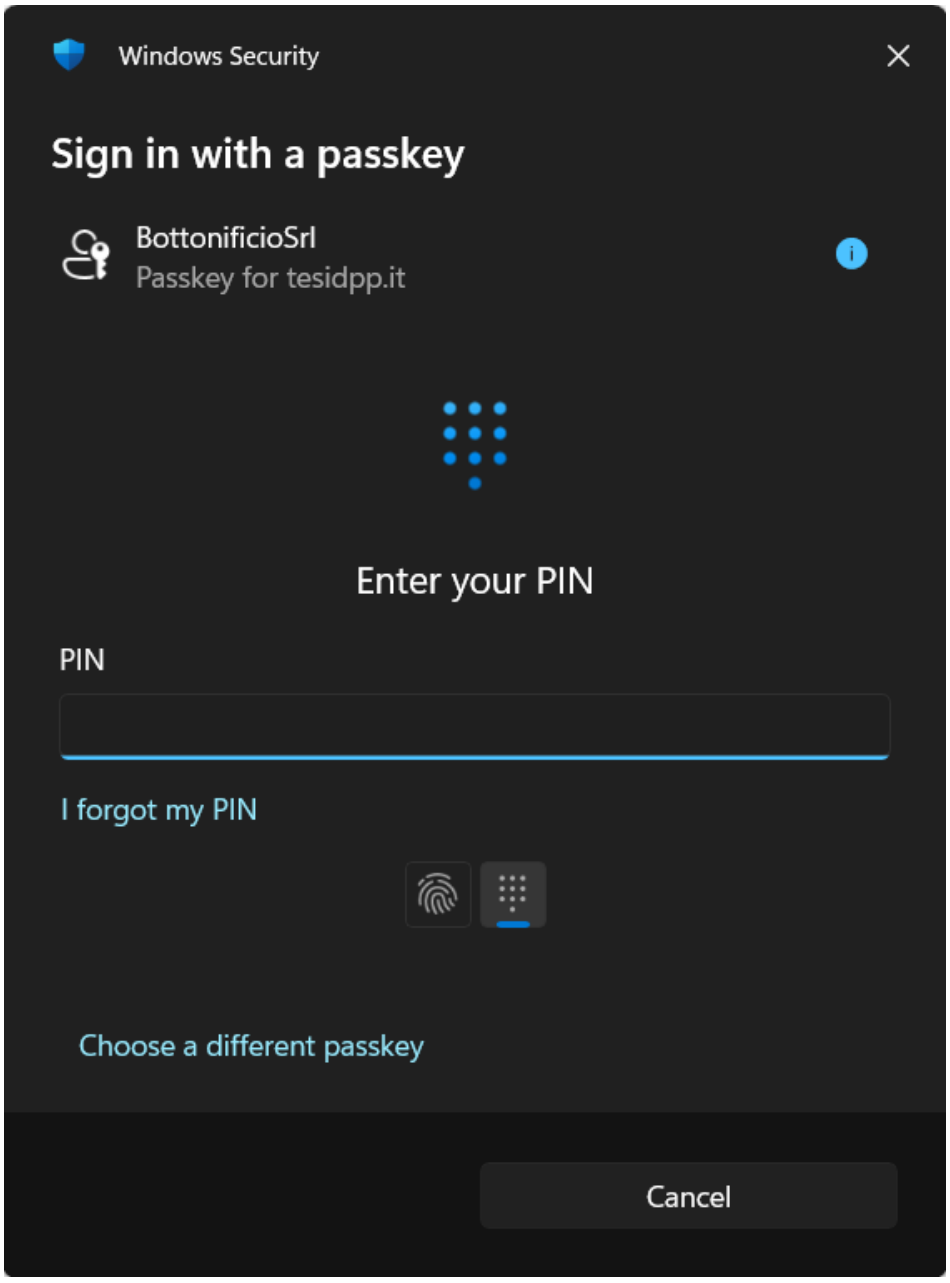

Figure 7.5: Passkey screen for user

8. Conclusions

As illustrated in chapter 1, Digital Product Passport sees his popularity increase rapidly, giving consumers the possibility to make more conscious choices, incentivizing, as a consequence, companies to use sustainable materials and to use practices that reduce the environmental impact of the production.

The notarization process can lead to increased trust for consumers, making the use of Digital Product Passport even more popular.

The framework explained in previous chapters demonstrates how it can be possible to use a decentralized notarization that replaces the traditional digital signature that can generate some skepticism for consumers based on its native single point of failure.

It is important to emphasize that a crucial aspect in enhancing the trust of consumers is the decision not to redesign the user experience for consumers, focusing only on change in the backend in the management system, together with little modifications in the interface for company employees. So, for the final consumer in the store, all aspects of usability are untouched: users only have to scan the QR code on the product label and see the corresponding data on their devices, typically smartphones.

The proposed framework respects the current laws, in particular the electronic IDentification, Authentication and trust Services regulation (eIDAS) explained in Section 2.2. In fact, the integration of passkeys is implemented to match the requirements for a digital signature, proving integrity, authenticity, and non-repudiation. The use of passkey is fundamental not only for consumers, but also for companies to prevent potential malicious behavior by their aggregation center.

Based on this work, a scientific article was written under the title *Exploiting Blockchains to Enable Notarization in the Digital Product Passport*. This article has been submitted to the 34^{th} IEEE International Conference on Enabling Technologies Infrastructure for Collaborative Enterprises[1] which is, as written in the official website, an annual international conference on state-of-the-art research in enabling technologies for collaboration. The article summarizes the core idea detailed in this thesis.

[1] https://www.olab-dynamics.net/wetice2026/

The article was written in collaboration with:

- Giacomo Cabri, professor of Università degli Studi di Modena e Reggio Emilia and supervisor of this thesis;
- Angelo Ferrando, professor of Università degli studi di Modena e Reggio Emilia;
- Massimo Garuti, referent of Democenter[2] for the Digital Transformation area.

[2]https://www.democentersipe.it/

Acknowledgements

Dopo aver concluso la tesi mi sono fermato un momento.

E, in quel momento, mi sono reso conto di aver concluso un percorso; un percorso che, senza certe persone, probabilmente non sarebbe mai stato possibile.

Mi sembra quindi doveroso prendersi un momento per dire grazie a tutte queste persone.

Ringrazio in primis il prof. Giacomo Cabri per i suoi preziosi consigli durante la progettazione, l'implementazione e la stesura della tesi.

Ringrazio Massimo Garuti, il prof. Angelo Ferrando e Luca Azzani per avermi affiancato durante i lavori sul software.

Un ringraziamento speciale va a mia mamma, a mio papà, ai miei nonni e a tutta la mia famiglia per avermi sostenuto in questo percorso sotto ogni punto di vista e per sostenermi, sempre, ogni giorno.

Inoltre, voglio ringraziare Annapia per essermi sempre stata vicina ed aver ascoltato, con cadenza quasi quotidiana, tutti i miei risultati raggiunti e le mie lamentele. Senza di te questo percorso sarebbe stato molto diverso, e averti al mio fianco lo ha sicuramente reso migliore.

Un ringraziamento va a Sara, Domenico, Nicolò e Matteo per essere una spalla su cui contare ogni giorno e per tutti i momenti divertenti passati insieme. Momenti che hanno permesso di distrarmi dalla classica giornata *standard* di lavoro e studio.

Un grazie va anche a Giovanni, Andrea, Davide, Elena, Marika, Luca, Chiara, Alessia, Alice, Sara, Lucia, Riccardo, Letizia, Filippo, Matteo, Rossana, Alessandro, Laura, Elisa, Mattia, Karen e Xe per le risate regalate durante questi anni e per aver parlato di tantissime cose ma quasi mai di tesi, rendendo questo percorso molto meno faticoso.

Ringrazio Francesco, Alex ed Elena per aver condiviso con me questo percorso di studi e per l'aiuto reciproco nei momenti più difficili dell'università. Col tempo siete diventati amici oltre che colleghi.

Infine, desidero ringraziare tutte quelle persone che mi hanno supportato in un modo o nell'altro che, purtroppo, non sono riuscito a menzionare. Se questo percorso è giunto al termine, è anche merito vostro.

A. Appendix

Code A.1: Notarization of a single change in the blockchain


```
{
  "jsonrpc": "2.0",
  "id": -1,
  "result": {
    "block_id": {
      "hash": "CD0B150F02325A69A1DC1564BFFA506
        5C0456F5E1C70DCFBE6F9B6C840547C00",
      "parts": {
        "total": 1,
        "hash": "0EF2C23AE5E19490FB814C0DD9A752AD8F737BEAD8B57EE6
   73ED736A8496D4EB"
      }
    },
    "block": {
      "header": {
        "version": {
          "block": "11"
        },
        "chain_id": "test-chain-hXybdB",
        "height": "64350",
        "time": "2026-03-26T21:02:45.224902269Z",
        "last_block_id": {
          "hash": "22FF08527BFAEC1BC26AF8894190FF8F812BA43E84CCB8
   5C91DDDDF18E23591E",
          "parts": {
```

```
        "total": 1,
        "hash": "37AB834F8E5DE31D8E8377083D9F462239E2AB5139D5
7D0BBE2493D1D42DBE6D"
      }
    },
    "last_commit_hash": "8399CF22FF871CE4E0B9C457FE1DE37F5425
70D743101256459311D718BE297F",
    "data_hash": "6600A1922CACF3C988D884A6D5FE2A8A845091E2A5F
666E0AF0C143B49B6C912",
    "validators_hash": "3FB6F30FB7E00A5379613D5B06A9E36AAD315
EC615B400F427920C516CA77A1",
    "next_validators_hash": "3FB6F30FB7E00A5379613D5B06A9E36
AAD4315EC615B400F427920C516CA77A1",
    "consensus_hash": "048091BC7DDC283F77BFBF91D73C44DA58C3DF
8A9CBC867405D8B7F3DAADA22F",
    "app_hash": "9D5D20481AC8D64829BA560B3AC4093F18B97E3ED2AC
6F0B2D357E1AEBA8674B",
    "last_results_hash": "E3B0C44298FC1C149AFBF4C8996FB92427
AE41E4649B934CA495991B7852B855",
    "evidence_hash": "E3B0C44298FC1C149AFBF4C8996FB92427AE41E
4649B934CA495991B7852B855",
    "proposer_address": "6816FBD562C8C4CEB15D1EE148E2C3AE154D
49B8"
  },
  "data": {
    "txs": [
      "McLdxa6bdSR1iKBwt/11Hwh8PH54J1Q7qLuRlFvAgh0="
    ]
  },
  "evidence": {
```

```
      "evidence": []
    },
    "last_commit": {
      "height": "64349",
      "round": 0,
      "block_id": {
        "hash": "22FF08527BFAEC1BC26AF8894190FF8F812BA43E84CCB8
5C91DDDDF18E23591E",
        "parts": {
          "total": 1,
          "hash": "37AB834F8E5DE31D8E8377083D9F462239E2AB5139D5
7D0BBE2493D1D42DBE6D"
        }
      },
      "signatures": [
        {
          "block_id_flag": 2,
          "validator_address": "592DD0667A0CB614EB74AEEAEAA6E73
D7A73B727",
          "timestamp": "2026-03-26T21:02:45.325234633Z",
          "signature": "eMDmvxDATxBMRbUpQFCFXxNQXEN2o+guDZj8cg/
36Grh63ug9/hKECsBLgzEv9MdUYg9br4VuHvjLE580bhCBw=="
        },
        {
          "block_id_flag": 2,
          "validator_address": "6816FBD562C8C4CEB15D1EE148E2C3
AE154D49B8",
          "timestamp": "2026-03-26T21:02:45.224902269Z",
          "signature": "VYmv7+n0YfwWO8xQxgD4mW7hHvba+PlB1vnVXEq
8+uIGftyiW1CNBfy1BloIT9ErgiM5r2qZhthNoQks0fd7Dg=="
```

```
            }
          ]
        }
      }
    }
}
```


Code A.2: Notarization of two changes in the blockchain


```
{
  "jsonrpc": "2.0",
  "id": -1,
  "result": {
    "block_id": {
      "hash": "C9B7F0A212BE928CD8DC8267CECCC6B8605B6D1343A8B80A08
   DB6CD6E7562F1F",
      "parts": {
        "total": 1,
        "hash": "0CC01863F859C68D03C8A214C01AEDC2D87958C264770E6D
   9AC0011146FF8B9B"
      }
    },
    "block": {
      "header": {
        "version": {
          "block": "11"
        },
        "chain_id": "test-chain-hXybdB",
        "height": "64352",
        "time": "2026-03-27T16:24:45.214384739Z",
        "last_block_id": {
```

```
        "hash": "E791BEF2A63CD6F2B2ECDD03FC293625F365977E18CFF8
7422025A79F946A57B",
        "parts": {
          "total": 1,
          "hash": "0D1F7374439376357D27C6EDDB942EB37148473274B2
E1F492E6BEBE3332912B"
        }
      },
      "last_commit_hash": "18A4772DA034B391C0E0E9EF6C4216DAF3B2
2C04DDE731E362F051914722D734",
      "data_hash": "EB3D6FE000855830E83D7D69B4D5A6A5F4D7A85806
AA86574B2D8A48A13AB53E",
      "validators_hash": "3FB6F30FB7E00A5379613D5B06A9E36AAD431
5EC615B400F427920C516CA77A1",
      "next_validators_hash": "3FB6F30FB7E00A5379613D5B06A9E36
AAD4315EC615B400F427920C516CA77A1",
      "consensus_hash": "048091BC7DDC283F77BFBF91D73C44DA58C3DF
8A9CBC867405D8B7F3DAADA22F",
      "app_hash": "31C2DDC5AE9B75247588A070B7F9751F087C3C7E7827
543BA8BB91945BC0821D",
      "last_results_hash": "E3B0C44298FC1C149AFBF4C8996FB92427
AE41E4649B934CA495991B7852B855",
      "evidence_hash": "E3B0C44298FC1C149AFBF4C8996FB92427AE41E
4649B934CA495991B7852B855",
      "proposer_address": "6816FBD562C8C4CEB15D1EE148E2C3AE154D
49B8"
    },
    "data": {
      "txs": [
        "2+ahyTatmlVCExw0Sv1CdF2+ZVCcGo0t1hT6g1gKwQw="
```

```
      ]
    },
    "evidence": {
      "evidence": []
    },
    "last_commit": {
      "height": "64351",
      "round": 0,
      "block_id": {
        "hash": "E791BEF2A63CD6F2B2ECDD03FC293625F365977E18CFF8
7422025A79F946A57B",
        "parts": {
          "total": 1,
          "hash": "0D1F7374439376357D27C6EDDB942EB37148473274B2
E1F492E6BEBE3332912B"
        }
      },
      "signatures": [
        {
          "block_id_flag": 2,
          "validator_address": "592DD0667A0CB614EB74AEEAEAA6E73
D7A73B727",
          "timestamp": "2026-03-27T16:24:45.307754081Z",
          "signature": "8TwhMzybR+RBXEWz/LJfKjZWXrUULtGyKycDe
uVjZyI3qnqaWPvj7664Rs7D1NFnUr8fAFG5IgWTMWT0pjNkBw=="
        },
        {
          "block_id_flag": 2,
          "validator_address": "6816FBD562C8C4CEB15D1EE148E2C3
AE154D49B8",
```

```
                "timestamp": "2026-03-27T16:24:45.214384739Z",
                "signature": "nXJtLPG/XJkQ83CMY4R5SuICyuBQnvDNd2ajMpn
    cOoHSZ5m7+RFVAtkUr/6XsC98ep59n23Wb8v6/ZGWS2ATAw=="
              }
            ]
          }
        }
      }
}
```

Bibliography

[1] European Parliament and Council of the European Union. *Regulation (EU) No 910/2014 of the European Parliament and of the Council of 23 July 2014 on electronic identification and trust services for electronic transactions in the internal market and repealing Directive 1999/93/EC*. Official Journal of the European Union, L 257. Article 25, comma 2". 2014. URL: https://eur-lex.europa.eu/legal-content/IT/TXT/?uri=CELEX:32014R0910.

[2] European Union. *Regulation (EU) 2024/1781 of the European Parliament and of the Council of 13 June 2024 establishing a framework for the setting of ecodesign requirements for sustainable products*. Official Journal of the European Union, L 2024/1781. 2024. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32024R1781.

[3] R. Fielding, M. Nottingham, and J. Reschke. *HTTP Semantics*. RFC 9110. RFC Editor, 2022. URL: https://www.rfc-editor.org/info/rfc9110.

[4] S. Geerthana and S. Ananthi. *MTTBA- A Key Contributor for Sustainable Energy Consumption Time and Space Utility for Highly Secured Crypto Transactions in Blockchain Technology*. 2022. URL: https://www.researchgate.net/publication/363888842 (visited on 03/22/2026).

[5] Jeff Hodges et al. *Web Authentication: An API for accessing Public Key Credentials - Level 2*. W3C Recommendation. W3C, Apr. 2021. URL: https://www.w3.org/TR/webauthn-2/.

[6] Jérémy Legardeur and Pantxika Ospital. *Digital product passport for the textile sector*. Study PE 757.808. Panel for the Future of Science and Technology (STOA). European Parliamentary Research Service (EPRS), June 2024. DOI: 10.2861/947638. URL: https://www.europarl.europa.eu/thinktank/it/document/EPRS_STU(2024)757808.

[7] M. Moriarty and S. Farrell. *Deprecating Transport Layer Security (TLS) 1.0 and 1.1*. RFC 8996. RFC Editor, 2021. DOI: 10.17487/RFC8996. URL: https://www.rfc-editor.org/info/rfc8996.

[8] National Institute of Standards and Technology. *Secure Hash Standard (SHS)*. Tech. rep. FIPS PUB 180-4. Standard ufficiale per l'algoritmo SHA-256. U.S. Department of

Commerce, 2015. DOI: 10.6028/NIST.FIPS.180-4. URL:
https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf.