Sapienza University of Rome

Information engineering, computer science and statistics department
Computer Science

A DECENTRALIZED PRIVACY-PRESERVING AGE VERIFICATION SYSTEM

ZKBlockMature

Advisor
Prof. Di Ciccio

Candidate
Moriconi Nicolò, Paolacci Mattia
1615286, 1664449

Academic Year 2023-2024

Contents

1 Preface

1.1 Overview

ZKBlockMature is a decentralized application (DApp) designed as a privacy-preserving solution for age verification, leveraging advanced cryptographic methods such as Zero-Knowledge Proofs (ZKP) and Pedersen Commitments. This project serves as a simplified example within a broader vision to establish a secure, decentralized digital identity on the blockchain.

The overarching concept is to create a digital identity framework that enables individuals to commit their personal data securely on a blockchain. In this larger framework, ZKBlockMature represents a specific implementation case focusing on age verification, where users can demonstrate their age eligibility without revealing sensitive details, such as their exact birthdate. This example highlights the potential of digital identities to validate attributes transparently while upholding user privacy. Nowadays, regulations, such as European ones, are increasingly strict regarding personal data protection and access to certain platforms, which may be restricted below a specific age threshold. ZKBlockMature is a solution intended to address this type of need.

By employing blockchain technology, ZKBlockMature not only ensures the transparency, security, and immutability of stored commitments but also preserves the confidentiality of user data. This approach exemplifies how future digital identity systems could enable individuals to authenticate various attributes in a privacy-centric manner.

Outline of the Report

The structure of this report is as follows:

1. **Preface:** Introduces the context and importance of the ZKBlockMature project.

2. **Background:** Provides a detailed explanation of foundational cryptographic methods, including a history of relevant technologies, an overview of Ethereum and smart contracts, and discussions on Zero-Knowledge Proofs (ZKP) and Pedersen Commitments within blockchain technology.

3. **Presentation of the Context:** Describes the overall aim of the DApp, its key features, and the high-level architecture, along with the various types of blockchains and the application domain.

4. **System Architecture:** Outlines the architectural components and interactions within ZKBlockMature, detailing the ambitious concept, high-level design, content authentication, integration aspects, and specific use cases.

5. **Implementation:** Describes the technical implementation of the ZKBlockMature application, including phases such as content authentication, integration, frontend logic, smart contract logic, error handling, and results.

6. **Known Issues and Limitations:** Examines the constraints of the current system and discusses the limitations faced during implementation.

7. **Future Improvements and Considerations:** Proposes potential future enhancements to expand functionality and address existing limitations.

1.2 Team members and main responsibilities

Due to the complexity of this project, particularly concerning the implementation of the commitment and zero-knowledge proof mechanisms, continuous collaboration was essential to ensure a coherent approach. After multiple brainstorming sessions, in which the overall solution was refined, we established the application's workflow. Nicolò focused on developing the zero-knowledge proof and commitment components, while Mattia was responsible for the frontend design using Angular, as well as the integration of the zero-knowledge proof and commitment algorithms. The development of the smart contracts was undertaken through pair programming sessions to maintain alignment and rigor in the implementation process.

2 Background

2.1 A Bit of History

Blockchain technology has evolved over several decades, with its roots traceable back to the early 1990s. In 1991, researchers Stuart Haber and W. Scott Stornetta proposed a cryptographically secure method to timestamp digital documents, preventing backdating or tampering. This early work laid the foundation for the development of blockchain's core concept — an immutable, cryptographically secured chain of data blocks.

However, it wasn't until 2008 that blockchain technology entered mainstream awareness with the release of Bitcoin by an anonymous entity known as Satoshi Nakamoto. Bitcoin's design introduced the world to a decentralized, peer-to-peer digital currency system, relying on blockchain as a public ledger to ensure transparency and prevent double-spending. Bitcoin's blockchain achieved consensus through Proof-of-Work (PoW), where participants (miners) compete to solve cryptographic puzzles to validate transactions.

Following the success of Bitcoin, numerous alternative blockchain platforms emerged. The most significant of these was Ethereum, launched in 2015 by Vitalik Buterin. Ethereum introduced a revolutionary concept — the ability to execute code (smart contracts) on the blockchain, thereby extending the technology's utility far beyond digital currency. This programmable blockchain laid the groundwork for decentralized applications (DApps), making it possible to automate complex tasks without the need for intermediaries.

In recent years, advancements like Proof-of-Stake (PoS), sidechains, and Layer 2 solutions have improved blockchain's scalability, security, and efficiency. The adoption of blockchain has expanded to a range of industries, including supply chain management, healthcare, finance, and government, making it a cornerstone technology for future decentralized systems.

2.2 Ethereum and Smart Contracts

Ethereum's introduction in 2015 marked a pivotal moment for blockchain technology. Unlike Bitcoin, which was designed specifically as a decentralized digital currency, Ethereum allows developers to build decentralized applications (DApps) that operate through the execution of smart contracts.

A smart contract is a self-executing code with predefined conditions that automatically enforces the terms of an agreement between parties. These contracts are stored and replicated on the blockchain, ensuring transparency, security, and immutability. Smart contracts on Ethereum are written in a programming language called Solidity, which enables developers to create complex and versatile decentralized applications that can interact with digital assets, identities, and data.

One of Ethereum's key innovations is its ability to handle decentralized finance (DeFi) applications, non-fungible tokens (NFTs), and other blockchain-based assets. Ethereum enables programmable money and trustless agreements between parties, eliminating the need for intermediaries like banks or notaries. As a result, Ethereum has become the dominant platform for deploying decentralized applications, allowing for the automation of various industries.

In the context of ZKBlockMature, Ethereum's smart contracts play a crucial role in managing user commitments, verifying cryptographic proofs, and ensuring that interactions are immutable and secure.

For example, when a user commits their birthdate using a Pedersen Commitment, Ethereum's blockchain records this information in a tamper-proof way. Smart contracts also handle the Zero-Knowledge Proofs (ZKP), ensuring the system verifies a user's age without revealing their personal data.

3 Presentation of the context

3.1 General Description

ZKBlockMature is a decentralized solution focused on enhancing privacy in age verification by applying cryptographic techniques. By integrating Zero-Knowledge Proofs (ZKP) and Pedersen Commitments, this project enables users to prove their age without compromising sensitive personal data, such as their birthdate. This approach aligns with the broader vision of establishing a digital identity system on the blockchain, where individuals can verify the properties of their identity in a secure, transparent, and decentralized manner.

3.2 Aim of the DApp and Key Features

The main objectives of ZKBlockMature center around enhancing privacy, security, and accessibility in age verification. The architecture of ZKBlockMature is designed to function entirely on a decentralized blockchain network, leveraging smart contracts to manage the validation of commitments and verification of zero-knowledge proofs. This structure ensures that age verification remains public and accessible, yet private, as it protects user data from unauthorized disclosure. The modular architecture allows integration with a broader identity management system, laying the groundwork for future expansion into a comprehensive digital identity solution.

The primary features are as follows:

- **Privacy-Preserving Verification**: Enables age verification using Zero-Knowledge Proofs, eliminating the need for personal data disclosure.
- **Enhanced Security and Source of Truth with Pedersen Commitments**: In addition to preserving data privacy, Pedersen Commitments serve as a verifiable source of truth. By embedding user information (such as age) in a commitment, the system provides a secure, unalterable record. Official verification of identity documents allows physical validation of the commitment's contents, ensuring the committed data reflects accurate identity information.
- **Decentralization and Trustlessness**: Leverages blockchain technology to establish a transparent, trustless environment, where after an initial verification by a certified officer, further interactions remain fully decentralized and free from central authority intervention.
- **User Accessibility**: Provides an interface that balances security with user-friendliness, facilitating easy interaction with the DApp for non-expert users.
- **Immutability and Tamper-Resistance**: Employs cryptographic techniques to ensure that age verification is tamper-proof, making the data immutable once recorded.

3.3 Types of Blockchain

Blockchain networks can be classified into two categories: permissionless and permissioned. Permissionless blockchains (like Bitcoin and Ethereum) allow anyone to participate as a node and engage in the consensus process. These networks operate in a decentralized and trustless manner, but they may face scalability issues and high transaction fees due to the public nature of the system.

In contrast, permissioned blockchains restrict access to only authorized participants. These private or consortium blockchains are typically used by enterprises, where certain entities are trusted to validate transactions. Permissioned blockchains provide enhanced privacy, higher transaction throughput, and improved efficiency compared to public blockchains, making them ideal for industries like banking, supply chain management, and healthcare, where data confidentiality is critical.

In our concept, the need for a permissioned blockchain arises when verifying commitments by a trusted authority. This approach ensures that only authorized entities can certify the data, providing a secure and reliable foundation for validation within the ZKBlockMature framework.

3.4 Application Domain

In recent years, the need for secure, privacy-preserving digital identity verification has become increasingly important across various industries. Traditional identity verification methods rely heavily on centralized authorities, which raises significant concerns regarding data privacy, user control, and security. The digital identity domain encompasses a range of applications, including age verification, access to restricted content, compliance in regulated industries, and user authentication in online platforms. These applications demand solutions that ensure the authenticity of user-provided information while preserving individual privacy.

Blockchain technology, with its decentralized and immutable properties, has emerged as a promising solution to address these challenges. Specifically, the application of blockchain in digital identity management offers a secure, transparent, and user-controlled approach, reducing reliance on central authorities. Zero-knowledge proofs (ZKPs) and cryptographic commitments, such as the Pedersen Commitment protocol, enable users to prove certain information (e.g., age, identity attributes) without revealing sensitive data in plaintext. This approach not only enhances privacy but also aligns with the growing regulatory focus on data protection, such as the European Union's General Data Protection Regulation (GDPR).

The application of ZKPs and blockchain technology in identity verification allows users to retain control over their data, presenting only the necessary proofs to fulfill access requirements while safeguarding personal information. The integration of blockchain-based identity solutions is particularly relevant in sectors that handle sensitive data, such as finance, healthcare, and e-commerce. These solutions are instrumental in preventing identity fraud, streamlining compliance, and ensuring user trust.

This report explores the implementation of a blockchain-based age verification system that leverages zero-knowledge proofs and cryptographic commitments to verify user age without compromising privacy. By using these advanced cryptographic techniques, the system provides a secure, decentralized, and user-centric approach to identity verification, highlighting the broader applicability of blockchain in the digital identity domain.

3.5 Pedersen Commitment

Pedersen Commitment is a cryptographic protocol designed to enable a party to commit to a chosen value while keeping that value hidden. This property is essential for ensuring both data privacy and integrity in secure digital systems. Introduced by Torben Pedersen [1], this commitment scheme is particularly useful in blockchain systems and cryptographic protocols requiring verifiable, non-revealing commitments.

The strength of the Pedersen Commitment lies in its ability to balance confidentiality and verifiability. By creating a commitment, a user can confirm that they possess specific information without disclosing it. Later, they may choose to reveal both the value and the parameters used to generate the commitment, allowing others to verify its accuracy without the possibility of tampering.

A commitment scheme like this involves two fundamental properties:

- **Hiding**: It ensures that the value remains concealed, so others cannot deduce the committed information.
- **Binding**: It guarantees that the committed value cannot be changed once the commitment is made.

The Pedersen Commitment is calculated as follows:

$$C = g^v h^r \mod p$$

where:

- C represents the commitment,
- g and h are generators in a specified elliptic curve group, chosen such that their discrete logarithmic relationship is unknown,
- v is the value being committed, such as a number or piece of private information,
- r is a randomly chosen blinding factor that masks the committed value,
- p is a large prime number defining the finite field.

The security of Pedersen Commitment relies on the difficulty of the discrete logarithm problem, which ensures that no information about v can be derived from C without knowledge of r.

3.5.1 Applications and Use of Jacobian ECC

Pedersen Commitments are widely employed in privacy-oriented blockchain technologies, such as Monero, due to their non-revealing yet verifiable nature. Additionally, in many applications, Jacobian elliptic curve cryptography is used for commitment calculations. Jacobian curves provide computational advantages and increased security, following principles from the study "Comparative Study of ECC and Jacobian Elliptic Curve Cryptography." [2] The use of these curves enhances efficiency and security in commitment generation due to the unique properties of elliptic curve groups.

3.6 Zero-Knowledge Proofs

Zero-Knowledge Proofs (ZKP) are cryptographic protocols that allow one party, known as the *prover*, to demonstrate to another party, the *verifier*, that they possess certain knowledge without revealing any details about that knowledge. This unique capability makes ZKPs especially valuable in scenarios where privacy and data confidentiality are critical.

The fundamental concept behind ZKP is to provide a convincing assurance to the verifier about a particular statement's truth without disclosing any additional information. For example, a prover could confirm they know the solution to a problem without revealing what the solution actually is.

A well-designed Zero-Knowledge Proof protocol satisfies the following key properties:

- **Completeness**: If the statement being proven is true, an honest verifier will be convinced by an honest prover.
- **Soundness**: If the statement is false, a dishonest prover cannot convince the verifier of the truth of the statement, except with a negligible probability.
- **Zero-Knowledge**: No information about the prover's knowledge is leaked to the verifier beyond the validity of the statement itself.

One widely used Zero-Knowledge Proof system is *Groth16*, an efficient ZKP protocol that enables secure and succinct proofs. By leveraging such protocols, complex statements can be validated without any exposure of sensitive data. This makes Zero-Knowledge Proofs a cornerstone for secure and privacy-preserving applications, particularly in cryptography and blockchain technology.

4 System Architecture

4.1 The Ambitious Concept

The field of Zero-Knowledge Proofs (ZKPs) presents a captivating aspect of blockchain technology. Our exploration began with the examination of projects leveraging this cryptographic method. The core principle of ZKPs lies in the ability to validate a statement without disclosing sensitive information.

The concept for our project emerged when a member of our team was prompted by ChatGPT to verify their age, specifically to prove that they were at least 18 years old. A disclaimer assured the user that the data would be temporarily stored in the browser and deleted after authentication. This sparked our initiative to transfer the process onto the blockchain, ensuring that no central authority would manage sensitive data, while simultaneously making the entire protocol public for interaction and information retrieval without compromising privacy.

While we developed a method to demonstrate age using ZKPs without revealing it, a crucial question remained: how could we guarantee that the age declared by the user is accurate? To address this, we explored the concept of commitments and discovered that Monero, a well-known blockchain, employs commitments to obscure transaction amounts.

4.2 High-Level Design

With all essential components identified, we are now prepared to present a high-level overview of our protocol implementation. In the following sections, we will detail the main steps of the project, focusing first on the foundational components: content authentication and integration. These sections will introduce the design and workflow in general terms, explaining each part's role and functionality. Finally, a dedicated section will illustrate a practical use case, showcasing our current project as a specific implementation of this broader concept. While this example demonstrates the protocol's core functionalities, it also acknowledges certain limitations in this early version.

4.2.1 Content Authentication

The primary objective of the Content Authentication phase is to establish a secure, verifiable link between sensitive personal information and a blockchain-based commitment. This process not only ensures that sensitive data remains hidden but also provides a means for authorized entities to validate this information when necessary. By committing critical data to the blockchain, individuals can retain control over their personal information, while third parties can verify its authenticity through a secure, structured protocol. In this section, we outline each step of the authentication workflow, from committing data to validation by authorized officers, thereby enabling a reliable and privacy-preserving verification mechanism.

1. Commit sensitive data onto a blockchain.
2. Visit an authorized office to validate the contents of the commitment. In our application, we commit only the birth date; however, we envision a broader scope encompassing all critical information on an identification card.

3. To facilitate validation, present the officer with the identification card and the random number used to generate the commitment.

4. The officer performs the commitment and verifies whether the value matches the one recorded on the blockchain.

5. If the commitment is valid, the officer records the transaction ID of our commitment onto a private-access blockchain (write access only), which remains publicly accessible for reading. This confirms that the commitment's content has been verified and the information is legitimate.

4.2.2 Integration

The Integration phase focuses on utilizing the authenticated commitment and Zero-Knowledge Proofs (ZKP) to verify specific properties of the committed data without revealing the data itself. This phase outlines how the commitment and ZKP can be effectively incorporated within an application to enable seamless and secure validation of sensitive information. By leveraging blockchain-based commitments and ZKP techniques, developers can build applications that grant conditional access based on authenticated properties of user data. The following steps demonstrate how an application can integrate our protocol, enabling data verification on-chain and offering privacy-preserving validation mechanisms within the app.

1. Integrate our validation tool into the website.

2. A validation window will prompt the user to enter the random number associated with the commitment, the transaction ID of the corresponding commitment, and additional values stored in the commitment for validation.

3. The initial step is to ensure that the value declared by the user corresponds to the committed values. This is accomplished by invoking the commit validator smart contract, allowing progression only if the result is positive.

4. After confirming the accuracy of the declared value, it is necessary to verify certain properties of this data. In this case, ZKPs can be employed.

5. The next step involves generating ZKP signals based on the submitted values, utilizing the Snarkjs library within our web application.

6. Finally, we invoke the smart contract checker, and if the proof is valid with a positive outcome, the entire process is deemed successful. Otherwise, access to the website's content is denied.

4.2.3 Use Case

To clarify the overall objective of our application, we present a real use case that we have implemented as a demonstration. Consider the scenario where we need to perform an age check similar to that of ChatGPT and many other applications.

1. In the validation window that appears, users are prompted to enter their birth date, the random number used in the commitment, and the transaction ID corresponding to their commitment.

2. The first step is to verify that the entered birth date corresponds to the committed date. This is done by invoking the commit validator smart contract, which allows progression only if the result is positive.

3. Once we confirm that the declared birth date is accurate, we still need to verify if the user is at least 18 years old.

4. The subsequent step is to generate ZKP signals, using the current date and the birth date. The Snarkjs library in our web application will facilitate this process.

5. Finally, we call the age checker smart contract, and if the user is determined to be at least 18 years old, the entire process is successful; otherwise, access to the website's content is denied.

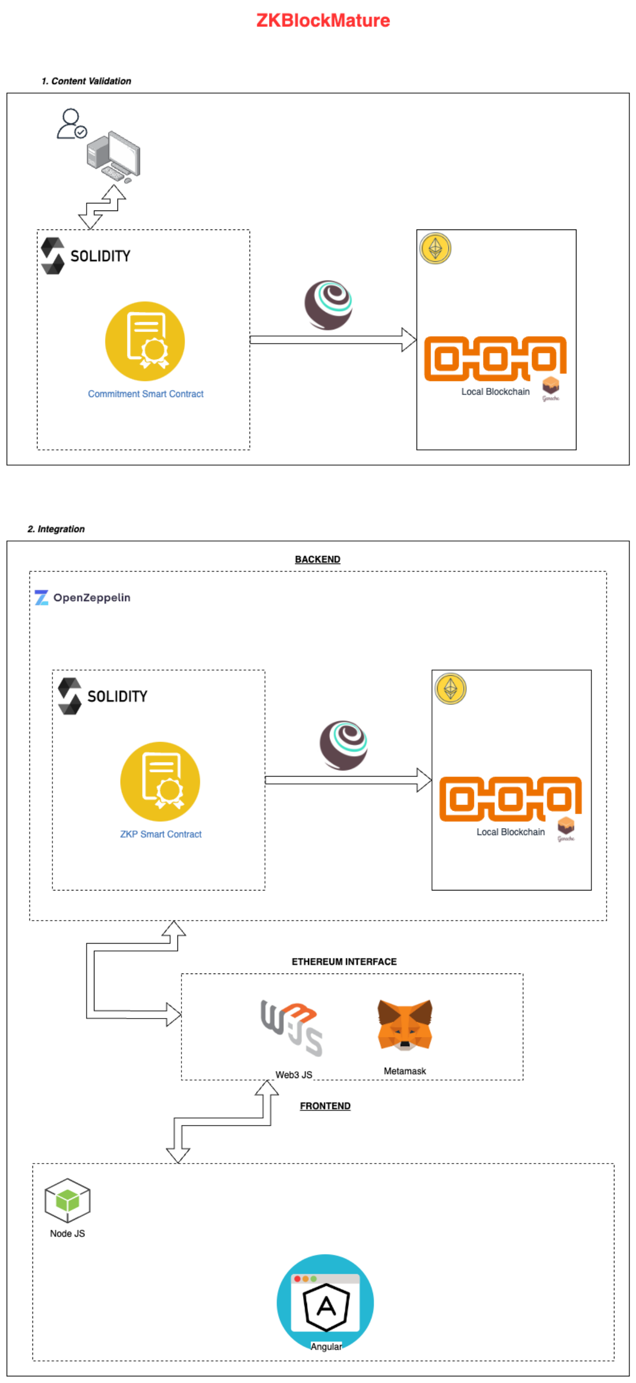

Figure 4.1: HLD

5 Implementation

5.1 Content Authentication Phase

The content authentication phase begins with writing the commitment to the blockchain using a dedicated script. In this phase, the user's date of birth is recorded. Truffle and Ganache have been used for compiling, deploying smart contracts, and managing a local blockchain environment. At this point, we can consider the setup of the commitment to be complete.

5.2 Integration Phase

Following the content authentication phase, the Integration Phase involves interaction with the application's frontend. Users must provide the address on the blockchain where the commitment is stored, the random number used during the commitment process, and select a date of birth.

The frontend then invokes the smart contract responsible for validating the commitment. If the validation is successful, the frontend code proceeds to the Zero-Knowledge Proof (ZKP) phase, generating the necessary signals to compare the provided date of birth with the current date. This process allows for the verification of whether the user is of legal age or not.

It is essential that the entered date of birth matches the one recorded in the commitment to ensure the successful outcome of the validation test.

5.3 Frontend Logic

The frontend logic is handled within an Angular component that manages user interactions and connects to the blockchain using Web3 and MetaMask. Users are required to enter specific information, including their date of birth, a random number generated during the commitment phase, and the blockchain address where the commitment is stored.

When a user selects a date, the `onDateChange()` function captures this input, while the `onSubmit()` function initiates the commitment validation process and interacts with the Zero-Knowledge Proof (ZKP) generation and verification components.

5.3.1 Commitment Validation

The commitment validation is performed via the `checkCommitment()` method, which constructs a date string in the format YYYYMMDD and converts it to a BigInt to match the requirements of the smart contract. This function then communicates with the deployed `PedersenCommitment` contract by invoking its `verify()` method.

The `verify()` method accepts three parameters: the random number, the generated BigInt from the date, and the transaction ID. If the commitment is validated successfully, the frontend proceeds with ZKP generation; otherwise, it terminates the process and displays an error.

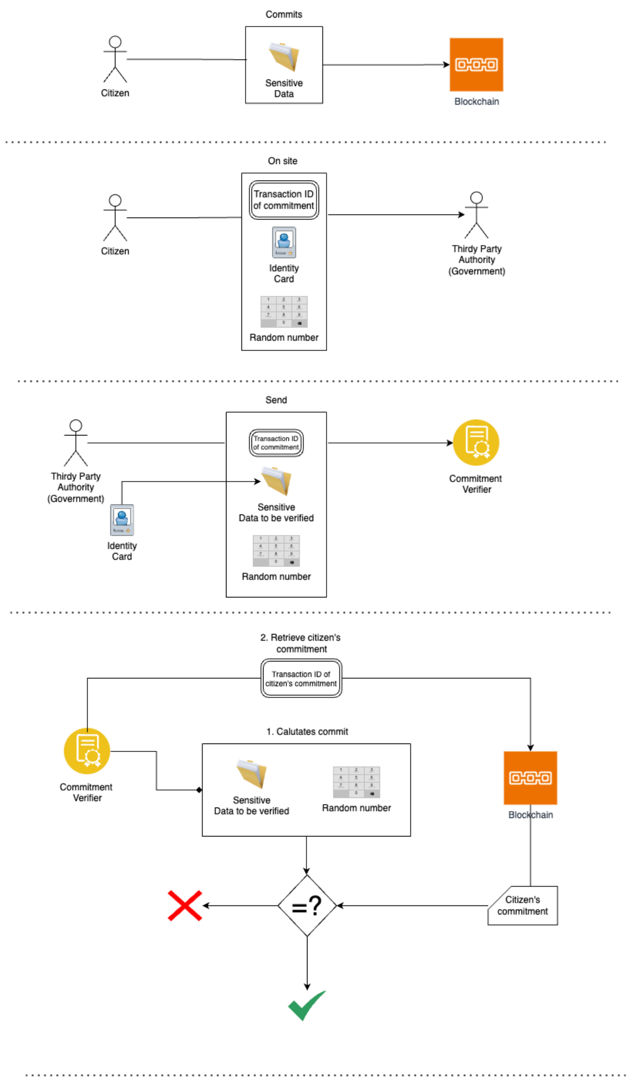
Commits
Citizen
Sensitive Data
Blockchain
On site
Transaction ID of commitment
Identity Card
Random number
Citizen
Thirdy Party Authority (Government)
Send
Thirdy Party Authority (Government)
Identity Card
Transaction ID of commitment
Sensitive Data to be verified
Random number
Commitment Verifier
2. Retrieve citizen's commitment
Transaction ID of citizen's commitment
1. Calutates commit
Commitment Verifier
Sensitive Data to be verified
Random number
Blockchain
=?
Citizen's commitment

Figure 5.1: UML

5.3.2 Proof Generation and Verification

Upon successful commitment validation, the application proceeds to generate a zero-knowledge proof by invoking `generateProof()`. This function utilizes SnarkJS to create the proof, passing in the input signals derived from the user's date of birth and the current date.

- **WASM and ZKey Files**: The WASM and ZKey files are required for proof generation. In this implementation, these are specified as `checkAge.wasm` and `checkAge_001.zkey`, which are initialized as properties in the component.
- **Proof Verification**: Once the proof is generated, the frontend verifies it using the `verify()` method from SnarkJS. This method takes the proof, public signals, and a verification key (`vKey`), ensuring that the generated proof is valid before sending it to the smart contract.

If the proof is valid, the function returns both the proof and the public signals for further processing.

5.3.3 Preparing and Sending Proof Data

Before sending the proof and public signals to the smart contract, the `parsingCallData()` function is called to convert the proof and signals into a format compatible with the Ethereum blockchain. This function uses

SnarkJS's `exportSolidityCallData()` function to parse and extract the arguments `a`, `b`, and `c`, along with the public input signals. These parsed values are then sent to the smart contract for validation.

The `sendData()` function is responsible for interfacing with the `ZKBlockMature` contract, deploying the proof data to the contract's `submitProof()` function. This function checks the validity of the proof, emitting a `ProofVerification` event with a result.

5.4 Smart Contract Logic

The backend logic is supported by two main smart contracts: `PedersenCommitment` and `ZKBlockMature`.

5.4.1 PedersenCommitment Contract

The `PedersenCommitment` contract is responsible for the initial commitment verification. This contract stores the elliptic curve point used for generating Pedersen commitments and includes functions such as:

- `getPoint()`: Retrieves the generator point.
- `setPoint()`: Sets the generator point if not previously set.
- `verify()`: Verifies that the commitment aligns with the birth date and random value.

5.4.2 ZKBlockMature Contract

The `ZKBlockMature` contract handles the submission of ZKP proof data and verifies the proof to determine whether the user meets the age requirement. Upon receiving the proof and public signals from the frontend, the contract's `submitProof()` function evaluates the proof's validity and checks that the constraint condition (i.e., age) is satisfied.

If the proof is valid, an event `ProofVerification` is emitted, returning the proof result, which indicates whether the user is over the legal age threshold.

5.5 Error Handling and Results

Throughout the process, errors are captured and logged in both the frontend and smart contract functions. For instance, any failure in the proof generation or validation triggers an error message, and unsuccessful proof validation prevents further progression.

In the event of a successful proof verification, the frontend will display a message based on the public signal returned from the smart contract, indicating whether the user is of legal age.

This modular structure ensures that the application is able to validate user information securely and efficiently using zero-knowledge proofs, while relying on the Pedersen commitment scheme for initial commitment validation.

5.6 Commitment

5.6.1 Elliptic Curve Cryptography (ECC)

Elliptic Curve Cryptography (ECC) underpins the security and efficiency of the cryptographic operations used in this project. ECC is based on the mathematical structure of elliptic curves, which are curves defined over a finite field, following an equation of the form:

$$y^2 = x^3 + ax + b \tag{5.1}$$

where a and b are constants that satisfy the curve's discriminant conditions to ensure no singular points. ECC is chosen for its high level of security per bit of key length, making it suitable for resource-constrained environments where performance and efficiency are critical.

5.6.2 Elliptic Curve Operations

The core operations on an elliptic curve include point addition, scalar multiplication, and inversion, which are essential for creating and verifying commitments. Below are key operations utilized in the implementation:

- **Point Addition**: Given two points P and Q on the curve, point addition computes $P + Q$. If P and Q are the same point, the operation becomes point doubling.
- **Scalar Multiplication**: This operation multiplies a point P by a scalar k, which is crucial for generating commitments and verifying them.

These operations are implemented following optimized algorithms, such as Jacobian coordinates, to enhance performance by avoiding costly inversion operations.

5.6.3 Pedersen Commitment Smart Contract

The Pedersen Commitment scheme leverages ECC to create secure, non-interactive commitments. The following pseudocode provides an overview of the Pedersen Commitment smart contract implemented:

```
contract PedersenCommitment {
    // Elliptic Curve parameters
    Point G; // Generator point
    Point H; // Secondary generator

    // Commitment Function
    function commit(uint x, uint r) public view returns (Point) {
        // Commitment formula: C = x * G + r * H
        return add(mul(G, x), mul(H, r));
    }

    // Verification Function
    function verify(Point C, uint x, uint r) public view returns (bool) {
        // Check if C == x * G + r * H
        return C == add(mul(G, x), mul(H, r));
    }
}
```

This smart contract implements the Pedersen Commitment protocol, where:

- `commit`: Generates a commitment C from a value x and a random number r by performing elliptic curve operations on points G and H.
- `verify`: Confirms the integrity of the commitment by recomputing the commitment with the provided x and r.

By leveraging these ECC-based commitments, the contract provides privacy-preserving guarantees while allowing an authority to verify the committed values as needed.

6 Known Issues and Limitations

In this section, we discuss the main challenges and limitations encountered in the implementation of the commitment protocol on the blockchain. Despite our efforts, several key issues remain that impact the effectiveness of the solution. These include:

- **Reliance on Third-Party Verification:** We were not fully able to eliminate the need for third-party verification. An additional step is still required to officialize the commitment, necessitating intervention by a trusted authority and detracting from the intended autonomy of the protocol.
- **Transparency and Data Exposure on Blockchain:** The calculated data for the commitment is visible on the blockchain, exposing it to all network participants. Due to blockchain's inherent transparency, this limits the privacy of the information encoded in the commitment.
- **Lack of Commitment Aggregation:** The current implementation does not take advantage of commitment aggregation properties, which could enhance efficiency and scalability by optimizing storage and verification processes.
- **Exposure of Sensitive Data During Validation:** During the frontend validation process, it is necessary to "open" the commitment by revealing sensitive data, as this information is not encrypted prior to interaction with the smart contract. This compromises privacy and highlights a significant area for improvement in future development.

7 Future Improvements and Considerations

While the current implementation of the commitment protocol demonstrates its potential, there are several areas for future improvements and considerations that could enhance its efficiency and usability:

- **Lower-Level Mathematical Operations:** One of the primary considerations for future work is to implement the mathematical operations associated with the commitment protocol at a lower level using assembly instructions. This change could significantly improve efficiency, as gas costs in the current implementation are relatively high. By optimizing the mathematical computations directly in the smart contract, it may be possible to reduce overall transaction costs and improve performance.
- **Delegation of Commitment to an Oracle:** To address the issue of exposing sensitive information, it may be beneficial to delegate the commitment process to an oracle. This approach could prevent sensitive data from being stored on-chain, thereby enhancing privacy and making the application more viable for real-world usage. The oracle could handle the commitment and verification processes securely, reducing the risk of data exposure.
- **Exploring Additive Properties of Commitments:** Future work should investigate the possibility of utilizing the additive properties of commitments more effectively. This includes the potential to incorporate non-numeric values into commitments, expanding the scope of what can be committed. Understanding how to leverage these properties can lead to more versatile and powerful implementations of the protocol.
- **Integration of Permissioned Blockchains:** Integrating the commitment protocol with permissioned blockchains could provide a solution for managing data privacy. Permissioned blockchains allow for private writing and public reading, which can ensure that sensitive data is kept secure while still enabling transparency. Examples of permissioned blockchains that could be considered for this purpose include Hyperledger Fabric, Corda, and Quorum. These platforms can facilitate a controlled environment where only authorized parties have access to sensitive information, enhancing the overall security and functionality of the system.

These considerations not only aim to address the limitations identified in the current implementation but also to position the project for future growth and broader adoption in the rapidly evolving landscape of digital identities and blockchain technology.

Bibliography

[1] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. 1991.

[2] A. P. Zele and A. P. Wadhe. Comparatively study of ecc and jacobian elliptic curve cryptography. 2015.