



BRNO UNIVERSITY OF TECHNOLOGY

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

**PROOF-OF-UNIQUENESS:
A DECENTRALIZED APPROACH TO
PRIVACY-PRESERVING IDENTITY VERIFICATION
BASED ON ZK-SNARKS**

DŮKAZ JEDINEČNOSTI:

DECENTRALIZOVANÝ PŘÍSTUP K OVEŘENÍ TOTOŽNOSTI

SE ZACHOVÁNÍM SOUKROMÍ ZALOŽENÝ NA ZK-SNARKS

BACHELOR'S THESIS

BAKALÁŘSKÁ PRÁCE

AUTHOR

AUTOR PRÁCE

ADAM VOŽDA

SUPERVISOR

VEDOUCÍ PRÁCE

doc. Ing. IVAN HOMOLIAK, Ph.D.

BRNO 2026

Bachelor's Thesis Assignment



170306

Institut: Department of Intelligent Systems (DITS)

Student: **Vožda Adam**

Programme: Information Technology

Title: **Proof-of-Uniqueness: A Decentralized Approach to Privacy-Preserving Identity Verification based on zk-SNARKs**

Category: Security

Academic year: 2025/26

Assignment:

1. Get familiar with principles of blockchains, their consensus protocols, and existing identity verification approaches on blockchains.
2. Study principles of zk-SNARKs and their variants.
3. Get familiar with the MSc thesis of Juraj Mariani aimed at Proof-of-Social-Capital; focus on the Sybil resistance and proof-of-uniqueness.
4. Investigate possible design options for resolving the Sybil resistance problem and discuss their pros and cons. Focus on the performance overhead and privacy.
5. Select one design option and make a proof-of-concept implementation of the proposed approach and evaluate its performance.
6. Discuss limitations of the approach and propose future extensions.

Literature:

- W3C. A Primer for Decentralized Identifiers. <https://w3c-ccg.github.io/did-primer/>
- Alizadeh, M., Andersson, K. and Schelén, O., 2022. Comparative analysis of decentralized identity approaches. *IEEE Access*, 10, pp.92273-92283.
- WorldCoin Foundation. World ID is a digital proof of human for the internet. <https://world.org/world-id>
- Luong, D.A. and Park, J.H., 2023. Privacy-preserving identity management system on blockchain using Zk-SNARK. *IEEE Access*, 11, pp.1840-1853.
- Petkus, M., 2019. Why and how zk-snark works. *arXiv preprint arXiv:1906.07221*.
- Mariani, Juraj, and Ivan Homoliak. "Proof-of-Social-Capital: Privacy-Preserving Consensus Protocol Replacing Stake for Social Capital." *arXiv preprint arXiv:2505.12144* (2025).
- dle pokynů vedoucího
- Juraj Mariani, MSc thesis, Privacy-Preserving Consensus Protocol based on Social Capital

Detailed formal requirements can be found at <https://www.fit.vut.cz/study/theses/>

Supervisor: **Homoliak Ivan, doc. Ing., Ph.D.**

Head of Department: Kočí Radek, Ing., Ph.D.

Beginning of work: 1.11.2025

Submission deadline: 13.5.2026

Approval date: 3.11.2025

Abstract

This thesis addresses Sybil resistance in Proof-of-Social-Capital and related decentralized systems that need to enforce one registration per user without revealing real-world identity data. It proposes a privacy-preserving Proof-of-Uniqueness architecture combining Verifiable Credentials, zk-SNARKs, and a threshold OPRF network.

The solution proves credential ownership, authorizes access to TACEO:OPRF nodes, and binds the resulting OPRF transcript to a hidden identity input using zk-SNARK. A Solidity smart contract verifies the enrollment proof, checks wallet authorization and issuer trust, then stores a nullifier-bound identity record.

The implemented prototype integrates a React client, Noir circuits, Ethereum smart contracts, and an extended TACEO:OPRF testnet with credential ownership authorization. Evaluation shows that the architecture is feasible with current tooling, while finding ideal credential issuer and private key management remains the main drawback of the proposed solution.

The main contribution is a reusable architecture for privacy-preserving uniqueness verification that eliminates disclosure of personal data.

Abstrakt

Tato práce se zabývá ochranou proti Sybil útokům v systému Proof-of-Social-Capital a příbuzných decentralizovaných systémech, které musí zajistit jednorázovou registraci uživatele bez odhalení jeho skutečné identity. Navrhuje architekturu pro soukromé ověření unikátnosti uživatele pomocí Verifiable Credentials, zk-SNARKs a prahové OPRF sítě.

Řešení umožňuje uživateli prokázat vlastnictví platného Verifiable Credentialu a získat autorizovaný přístup k uzlům TACEO:OPRF. Výsledek OPRF protokolu je svázán se skrytou identitou uživatele a použit k vytvoření registračního důkazu. Ten ověřuje Solidity smart contract, který zároveň kontroluje autorizaci peněženky, důvěryhodnost vydavatele a ukládá záznam identity navázaný na identifikátor.

Součástí práce je prototyp tvořený React klientem, Noir obvody, Ethereum smart contracty a rozšířenou testovací sítí TACEO:OPRF s modulem pro autorizaci na základě vlastnictví credentialu. Experimentální vyhodnocení ukazuje, že architektura je se současnými nástroji realizovatelná, přičemž hlavními nedostatky zůstávají nalezení vhodného vydavatele Verifiable Credentials a správa soukromých klíčů.

Hlavním přínosem práce je znovupoužitelná architektura pro ověřování unikátnosti uživatele s minimalizací odhalených osobních údajů.

Keywords

blockchain, identity verification, sybil resistance, verifiable credentials, zk-SNARKs, proof-of-uniqueness, oblivious pseudorandom function

Klíčová slova

blockchain, ověření identity, odolnost vůči sybil útokům, ověřitelné údaje, zk-SNARKy, důkaz jedinečnosti, zaslepená pseudonáhodná funkce

Reference

VOŽDA, Adam. *Proof-of-Uniqueness:*

A Decentralized Approach to

Privacy-Preserving Identity Verification

based on zk-SNARKs. Brno, 2026. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor doc. Ing. Ivan Homoliak, Ph.D.

Proof-of-Uniqueness: A Decentralized Approach to Privacy-Preserving Identity Verification based on zk-SNARKs

Declaration

I declare that I have prepared this bachelor's thesis independently under the supervision of Ing. Ivan Homoliak, Ph.D. Additional information was provided to me by Ing. Juraj Mariani. I have listed all literary sources, publications, and other materials from which I have drawn. For the linguistic revision of the technical report, I used the tools Gemini and Grammarly. During the development of the submitted program code, I used the tool GitHub Copilot.

.....
Adam Vožda
May 12, 2026

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Ing. Ivan Homoliak, Ph.D., for his professional guidance, valuable advice, and continuous support throughout the development of this thesis. I would also like to thank my supervisor and Ing. Juraj Mariani for their prior work on the concept of Proof-of-Social-Capital, which served as the motivation and foundation for this thesis.

Contents

1	Introduction	5
1.1	Thesis Organization	5
2	Introduction to Blockchains and Smart Contracts	7
2.1	Fundamental Concepts of Blockchain Technology	7
2.2	Bitcoin as the First Decentralized Blockchain System	7
2.3	Ethereum and Programmable Blockchain Platforms	9
2.4	What are Ethereum Smart Contracts?	10
3	Identity Management Approaches	13
3.1	What is Digital Identity and Self-Sovereign Identity?	13
3.2	Centralized, Federated, and Existing Decentralized Identity Models	14
3.3	W3C Standards for Decentralized Identity	16
4	Zero-Knowledge Cryptography and zk-SNARKs	19
4.1	Introduction to Zero-Knowledge Proofs	19
4.2	From Code to Proof: How it Works	20
4.3	Comparison of Modern Zero-Knowledge Proof Systems	21
4.4	Deep Dive: zk-SNARKs	23
5	Proof-of-Social-Capital	25
5.1	System Overview and Motivation	25
5.2	System Design and Economic Model	25
5.3	Sybil Attack Resistance Mechanism	26
5.4	Limitations and Thesis Contribution	27
6	Options for Proof-of-Uniqueness	28
6.1	Definition and System Requirements	28
6.2	Options for Identity Issuers	29
6.3	Storage Architectures for Identity Commitments	31
6.4	Oblivious Pseudorandom Function for Privacy-Preserving Deduplication	33
7	Proposed Approach	37
7.1	Design Overview	37
7.2	Credential and Holder Binding Design	37
7.3	Protocol Cryptography: From Credential to Verifiable Nullifier	38
7.4	Smart Contract Registry Design	39
7.5	Proof-of-Uniqueness Lifecycle	40

7.6	Implemented Prototype	42
7.7	Summary	45
8	Evaluation	46
8.1	Experimental Setup and Methodology	46
8.2	Experiment 1: Client-Side Cryptographic Performance	46
8.3	Experiment 2: Smart Contract Gas Analysis	47
8.4	System Requirements Evaluation	47
8.5	Summary	48
9	Security Analysis	49
9.1	Threat Model and Trust Boundaries	49
9.2	Sybil Attack Resistance and Privacy	50
9.3	Zero-Knowledge Guarantees	50
9.4	Replay Resistance and Permissionless Operations	50
10	Discussion & Limitations	52
10.1	OPRF Trust Assumptions and Key Rotation	52
10.2	Wallet-Bound Authorization, Recovery, and Key Management	52
10.3	Trusted Issuer Governance and Real-World Credential Infrastructure	53
10.4	Ethereum State Bloat and Maintenance Scalability	53
11	Conclusion	55
	Bibliography	56
A	Contents of Electronic Appendix	60

List of Figures

2.1	Structure of the blockchain blocks. The block consists of the header and the body. The header contains metadata such as the nonce, difficulty, and root hash, while the body contains the transactions organized in the Merkle tree. Source: [18].	8
2.2	Visualization of the Ethereum Modified Merkle-Patricia-Trie. It shows how the world state is stored using different types of nodes, like extension, branch, and leaf nodes. Source: [23].	12
3.1	Verifiable Credential ecosystem showing how the Issuer, Holder, Verifier, and Verifiable Data Registry interact during credential issuance, presentation, and verification. Source: [8].	14
3.2	Comparison of Identity Approaches.	16
3.3	Structure of a Decentralized Identifier (DID), consisting of the DID scheme, DID method, and method-specific identifier. Source: [16].	17
3.4	Example W3C Verifiable Credential 2.0	18
4.1	Ali Baba's Cave illustration. Source: [13].	20
4.2	Comparison of Zero-Knowledge Systems.	22
6.1	Comparison of extracted data payloads between an ePassport (left) and a standard government eID Certificate (right). Personal information has been replaced with placeholder values.	31
6.2	Sequence diagram of the on-chain storage architecture. The user sends the zk-SNARK identity proof directly to the smart contract, which stores the ID hash.	32
6.3	Sequence diagram of the off-chain storage architecture. The smart contract only emits events, and a decentralized prover network keeps the Merkle tree off-chain. The provers generate a second zk-SNARK proof to update the single on-chain root.	35
6.4	Diagram of the TACEO:OPRF nullifier generation from a blinded user secret and a distributed TACEO network secret. Source: [35].	36
7.1	Diagram of the proposed protocol enrollment phase.	41
7.2	Diagram of the proposed protocol authorization phase.	41
7.3	Example of the Verifiable Credentials data model used by the implemented Proof-of-Uniqueness prototype.	43
8.1	Client-side benchmark results for Noir/Barretenberg v0.36.0 proof generation and verification under Node.js v25.5.0.	47

8.2	Foundry gas profiling results for the <code>IdentityRegistry</code> contract and generated Ultra verifier path.	47
-----	---	----

Chapter 1

Introduction

In today’s digital world, more communication, finance, and social interaction happen online. Many online systems need to know not only that a user controls an account but also that a single real person cannot create multiple accounts and pretend to be many different people. This is especially important in decentralized systems, where there is no single company or administrator that everyone agrees to trust.

This problem is difficult because simple solutions are not sufficient. One option is to ask a central authority to check every user, but then that authority becomes very powerful and may collect sensitive personal data. Another option is to publish identity-related information directly in the decentralized system, but that can harm user privacy. The goal, therefore, is to prove that a user is unique without revealing who the user is or creating a new central point of control.

A motivating example is the Proof-of-Social-Capital (PoSC) consensus protocol, which uses online influence to determine who gains power in a decentralized network. In such a system, real users support content creators, who can then participate in running the network. Without a reliable way to prevent duplicate identities, a malicious person could create many fake accounts, making their influence seem much larger than it really is. This kind of attack is called a Sybil attack. To prevent it, PoSC needs a way to ensure each person registers only once.

This thesis proposes a privacy-preserving Proof-of-Uniqueness system for this purpose. The basic idea is that a user receives a digital credential from a trusted issuer, proves its validity, and registers a unique public identifier that cannot easily be linked to the user’s personal information. The decentralized system can then check whether this value has already been registered, while the user’s private identity data remains hidden. The thesis contributes both the design of this system and a working prototype that demonstrates how it can be built using current tools, albeit with limitations.

1.1 Thesis Organization

The remainder of this thesis is organized as follows:

Chapter 2 introduces blockchains and smart contracts, using Bitcoin and Ethereum as the main examples.

Chapter 3 discusses digital identity and explains different ways online identity can be managed.

Chapter 4 explains the cryptographic tools that enable proving facts without revealing the private data that underlies them.

Chapter 5 introduces the Proof-of-Social-Capital protocol, explains why it needs protection against fake duplicate identities, and identifies the gap that motivates this thesis.

Chapter 6 analyzes practical design options for Proof-of-Uniqueness, including who can issue identity credentials, how registration data can be stored, and how duplicate registrations can be detected privately.

Chapter 7 describes the proposed system design and the implemented prototype.

Chapter 8 evaluates the prototype and measures its practical costs.

Chapter 9 provides a security analysis of the resulting system, focusing on trust assumptions, privacy, resistance to duplicate identities, and replay protection.

Chapter 10 discusses the main practical limitations of the approach and their implications for real-world use.

Finally, Chapter 11 concludes the thesis by summarizing the achieved results and outlining possible future work.

Chapter 2

Introduction to Blockchains and Smart Contracts

This chapter covers the foundational knowledge of decentralized systems, known as blockchains. It first explains what blockchains are and how they fundamentally operate. It then examines Bitcoin as a practical example, illustrating the underlying data structures, such as Merkle Trees. Finally, the chapter explores Ethereum and smart contracts to demonstrate how modern blockchains enable programmable logic and state management.

2.1 Fundamental Concepts of Blockchain Technology

In simple terms, a blockchain is a digital ledger that records information in a way that makes it difficult or impossible to change, hack, or cheat the system. Unlike traditional databases that are controlled by a single central authority (like a bank or a government), a blockchain is distributed across a network of computers called nodes.

The name „blockchain“ comes from how the data is stored. Information is grouped together in sets called „blocks“ (as shown in Figure 2.1). Each block contains a cryptographic hash of the previous block, creating a link between them and forming a chain. This structure ensures immutability: if someone tries to change data in an old block, the block’s hash changes. This breaks the link to the next block, making the tampering obvious to the entire network. This mechanism allows the system to be „trustless,“ meaning participants do not need to trust each other; instead, they rely on the cryptographic design of the system.

2.2 Bitcoin as the First Decentralized Blockchain System

Bitcoin is the first and most well-known implementation of a decentralized blockchain. It was introduced in 2008 by the unknown Satoshi Nakamoto [27]. While Bitcoin was designed primarily as a digital currency to solve the double-spending problem without a central intermediary, its architecture established the cryptographic primitives used in most modern decentralized systems.

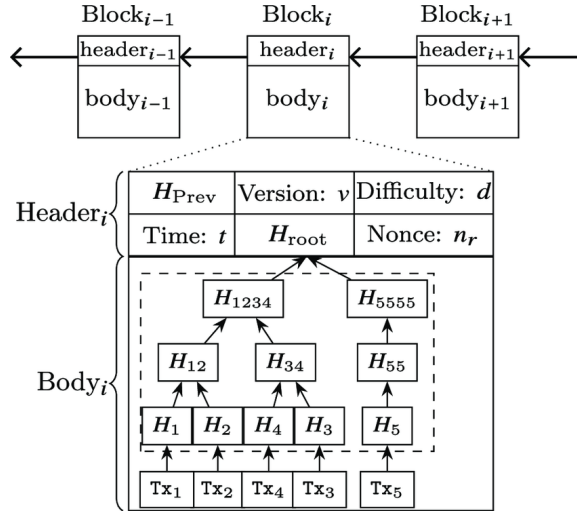


Figure 2.1: Structure of the blockchain blocks. The block consists of the header and the body. The header contains metadata such as the nonce, difficulty, and root hash, while the body contains the transactions organized in the Merkle tree. Source: [18].

Merkle Tree Data Structure

The Merkle Tree is a binary hash tree in which every leaf node represents the hash of a data block (e.g., a transaction), and every non-leaf node is a cryptographic hash of its concatenated child nodes.

The Merkle Root is the single hash at the top of this tree. It acts as a unique summary of all transactions in the block. This architecture allows for Simple Payment Verification [27]. A user can verify that a specific transaction exists in a block by possessing only the Merkle Root and a specific branch of hashes (the Merkle path), rather than searching through all the transactions.

Bitcoin Proof-of-Work Consensus Mechanism

To maintain ledger integrity without a central administrator, Bitcoin uses a consensus protocol called Proof-of-Work (PoW). Network participants, known as miners, compete to solve a computational puzzle derived from the block header. The difficulty of this puzzle ensures that rewriting the ledger would require an economically infeasible amount of computational power, thereby securing the network's history.

Block Structure

A Bitcoin block is composed of two primary sections: the block header and the block body [36], as shown in Figure 2.1.

- **Block Header:** The 80-byte header is the most critical part of the block for verification and mining. It contains the following six fields:
 - **Version:** Indicates the protocol version number. This allows nodes to determine which validation rules to apply.

- **Previous Block Hash:** The SHA-256 hash of the previous block’s header. This field explicitly links the new block to its predecessor, forming the immutable „chain“.
 - **Merkle Root:** A single hash derived from all transactions in the block. It ensures that any alteration to a transaction invalidates the proof-of-work in a block.
 - **Timestamp:** A Unix timestamp recording when the miner started hashing the header. It is used to adjust the network difficulty over time.
 - **Bits:** A compact representation of the „Difficulty Target.“ The block’s hash must be numerically lower than this target to be valid.
 - **Nonce:** An arbitrary number that miners change repeatedly. The goal is to find a nonce that produces a block hash that satisfies the difficulty target.
- **Block Body:** The raw list of transactions, starting with the Coinbase transaction (which rewards the miner), followed by user transactions.

2.3 Ethereum and Programmable Blockchain Platforms

While Bitcoin pioneered the concept of a decentralized ledger, its functionality is intentionally limited to tracking currency ownership. Ethereum, introduced in a 2013 white paper by Vitalik Buterin [2], generalized this architecture to create a „World Computer.“ Rather than just validating currency transfers, Ethereum is designed as a general-purpose, programmable blockchain capable of executing code [30].

Also, unlike Bitcoin, which relies on the energy-intensive Proof-of-Work (PoW), Ethereum currently uses a Proof-of-Stake (PoS) consensus protocol. In this model, network security is not derived from miners solving computational puzzles. Instead, it relies on „validators“ who lock up (stake) their capital in Ether. These validators are randomly selected to propose new blocks and verify others’ work, making the system much more energy-efficient.

The Ethereum Virtual Machine (EVM)

The core distinction between the two systems is the Ethereum Virtual Machine (EVM). If Bitcoin is analogous to a distributed calculator, Ethereum is a distributed state machine [11].

The EVM is a quasi-Turing-complete execution environment that runs on every node in the network. It maintains a data structure known as the „World State,“ which maps addresses to account states. Unlike Bitcoin’s simple transaction history, the Ethereum state changes not just through balance transfers, but also through the execution of smart contract code. This state is cryptographically secured using a Merkle Patricia Trie (as shown in Figure 2.2), a more complex variant of the Merkle Tree described in the previous section, defined formally in the Ethereum Yellow Paper [39].

Gas and the Halting Problem

Because the EVM allows for loops and complex logic, it faces the „Halting Problem“: the impossibility of determining whether a program will run forever or eventually terminate. To prevent malicious or accidental infinite loops from freezing the network, Ethereum introduced the concept of Gas [11].

Every operation in the EVM is assigned a specific cost in Gas units. Transaction senders must pay for this Gas using Ether. If a transaction runs out of the allocated Gas before completing, the EVM halts execution and reverts all state changes, ensuring the network remains live and responsive regardless of the code being executed.

Account Types

Within the Ethereum state, there are two distinct types of accounts:

- **Externally Owned Accounts (EOAs):** Controlled by private keys (users). They can initiate transactions but contain no code.
- **Contract Accounts:** Controlled by their code (smart contracts). They cannot initiate transactions on their own; they only run when triggered by transactions from an EOA or another contract.

Ethereum Transactions

Transactions are the cryptographically signed instructions initiated by Externally Owned Accounts (EOAs). They are the only mechanism by which the state of the Ethereum network can be updated [2]. A standard Ethereum transaction contains the following key fields:

- **Nonce:** A counter indicating the number of transactions sent from the account. This prevents replay attacks, ensuring that a transaction cannot be intercepted and resubmitted maliciously.
- **To:** The destination address. This can be an EOA (for value transfer) or a Contract Account (to trigger code execution).
- **Value:** The amount of Ether (in Wei) to transfer to the destination address.
- **Gas Limit & Fee Parameters:** Parameters that determine the maximum computational cost the sender is willing to pay for the transaction execution.
- **Data:** A variable-length hex byte array. For simple Ether transfers, this is typically empty. However, for smart contract interactions, this field is critical: it contains the Function Selector (which identifies the function to call) followed by the encoded arguments required by that function.
- **Signature:** The cryptographic signature generated by the sender's private key, proving authorization and ensuring the transaction data has not been tampered with.

Once constructed, a transaction is broadcast to the network, validated by nodes, and eventually included in a block. Upon inclusion, the EVM executes the logic defined in the Data field (if targeting a contract) or simply updates balances (if a simple transfer).

2.4 What are Ethereum Smart Contracts?

Smart contracts are self-executing programs that run on the Ethereum blockchain, facilitating the automated, transparent, and tamper-proof execution of predefined rules [11]. Once deployed, their code is immutable, meaning it cannot be altered even by its creator, and its execution is guaranteed by the network's consensus protocol.

Lifecycle and Deployment

Unlike standard user accounts, smart contracts are not controlled by a private key but by their own code. To deploy a smart contract, a user (EOA) sends a special transaction to the network [11]:

- The **To** field is left empty.
- The **Data** field contains the compiled bytecode of the contract.

Upon successful execution, the network assigns the contract a unique address. This address is deterministically computed from the sender's address and their account Nonce (transaction count).

Interacting with Contracts

Interaction with a smart contract occurs through „calls.“ These can be categorized into two types:

- **Read-Only Calls:** These execute a function that retrieves data without modifying the state. They run locally on a node and do not incur Gas costs.
- **State-Changing Transactions:** These execute functions that modify the blockchain state (e.g., updating a registry). These require a signed transaction and the payment of Gas.

To invoke a specific function, the transaction's Data field must start with a Function Selector. This is the first 4 bytes of the Keccak-256 hash of the function signature. The EVM uses this selector to jump to the correct portion of the bytecode during execution.

Solidity

While the EVM executes low-level bytecode, developers typically write smart contracts in high-level languages, the most prominent of which is Solidity [33]. Solidity is an object-oriented, statically typed language influenced by C++, Python, and JavaScript. It supports inheritance, complex user-defined types, and libraries, enabling developers to build sophisticated decentralized applications, such as the identity registry proposed in this thesis.

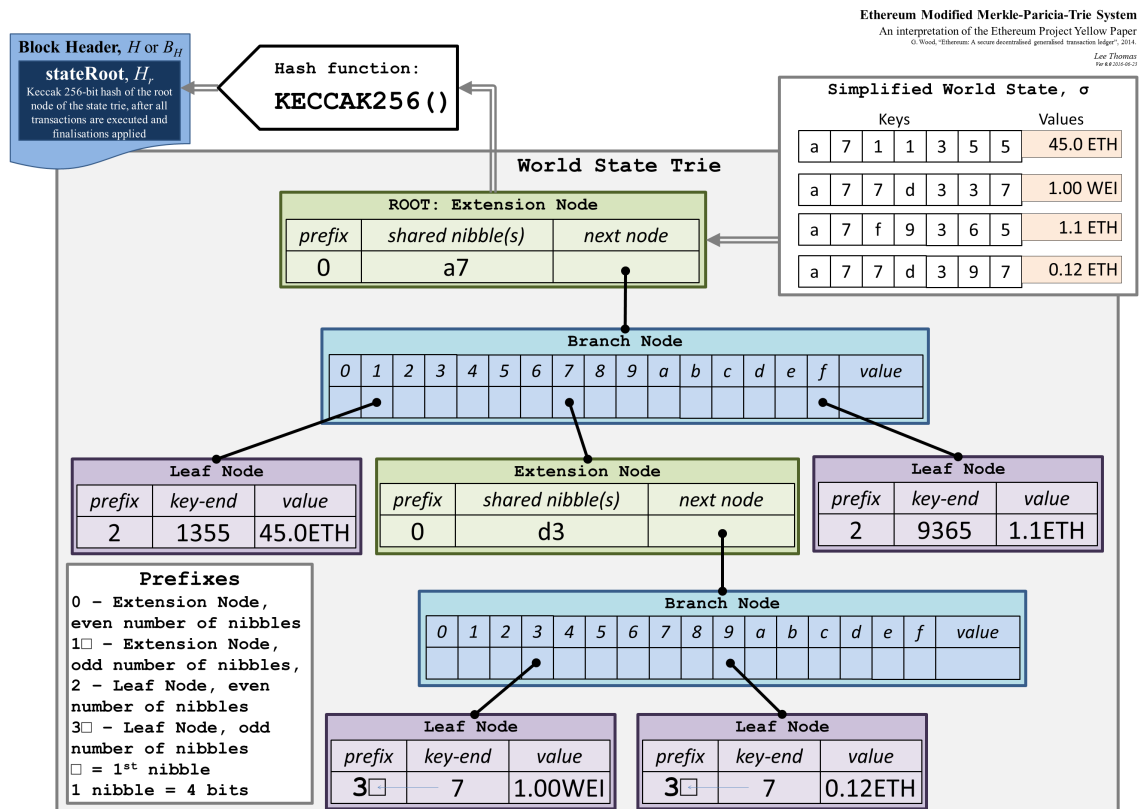


Figure 2.2: Visualization of the Ethereum Modified Merkle-Patricia-Trie. It shows how the world state is stored using different types of nodes, like extension, branch, and leaf nodes. Source: [23].

Chapter 3

Identity Management Approaches

This chapter presents the theoretical background of digital identity systems. It defines the fundamental concepts of digital identity and Self-Sovereign Identity. It also describes the “Trust Triangle” architecture, which underlies modern decentralized systems. The chapter then examines different identity management approaches, including centralized, federated, and decentralized models. Finally, it discusses the core technical standards defined by the World Wide Web Consortium.

3.1 What is Digital Identity and Self-Sovereign Identity?

At its most fundamental level, a Digital Identity is a set of electronically captured attributes, credentials, or claims that represent an entity, such as a person, organization, or device, in the digital sphere. Unlike a physical ID card, which is a static object, a digital identity is a collection of data points (e.g., usernames, purchase history, biometric data) used to verify an entity’s legitimacy and authorization to access services.

Self-Sovereign Identity

Self-Sovereign Identity (SSI) represents a paradigm shift in how these digital identities are managed. It is a model that empowers individuals to own and control their identity without relying on administrative authorities.

In an SSI system, the user is the ultimate authority over their data. They store their credentials on their own devices (typically in a digital wallet) and share them only with specific peers [21].

The Trust Triangle

The architecture of SSI is commonly described using the „Trust Triangle“, a model that separates the roles of issuance, holding, and verification to ensure trust without a central intermediary, as illustrated in Figure 3.1 [21].

- **Issuer:** An entity (e.g., a university, government, or bank) that asserts claims about a subject by creating and cryptographically signing a digital credential. The Issuer is responsible for the validity of the data at the time of issuance.
- **Holder:** The user or entity that requests and receives the credential from the Issuer. The Holder stores this credential in a secure digital wallet and is the only party that

interacts directly with the Verifier, ensuring they remain in control of who sees their data.

- **Verifier:** An entity (e.g., an employer or website) that requests proof of a claim from the Holder. The Verifier checks the Issuer’s digital signature to authenticate the data, often using a blockchain or distributed ledger as a „trust anchor“ to retrieve the Issuer’s public keys.

3.2 Centralized, Federated, and Existing Decentralized Identity Models

To fully appreciate the necessity of SSI, it is important to understand the evolution of the identity management models that preceded it.

Centralized Identity

In the early stages of the internet, identity management was strictly centralized. Each service provider (e.g., an email service, forum, or bank) acted as its own identity authority, issuing unique credentials (usernames and passwords) to its users. This „siloed“ approach forces users to manage dozens of different accounts, leading to „password fatigue“ and security vulnerabilities. Crucially, the service provider retains absolute control and ownership over the identity, with the power to revoke access or track user activity at will [20].

Federated Identity

To address the usability challenges of silos, the Federated Identity model emerged. In this system, a trusted third party known as an Identity Provider (IdP) vouches for the user’s identity across multiple independent services. Common examples include Single Sign-On (SSO) solutions like „Login with Google“ or „Login with Facebook“.

While this model significantly improves user convenience, it creates a critical privacy bottleneck: the IdP gains full visibility into every service the user accesses. Furthermore, the user’s digital existence becomes dependent on the IdP; if the central account is banned or compromised, the user loses access to all linked services [20].

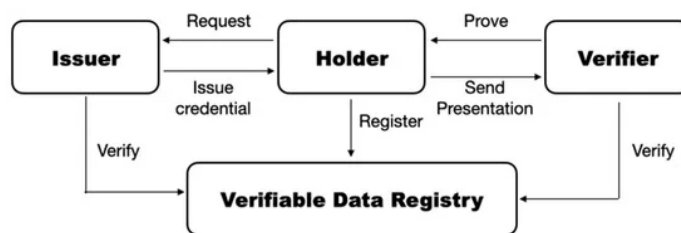


Figure 3.1: Verifiable Credential ecosystem showing how the Issuer, Holder, Verifier, and Verifiable Data Registry interact during credential issuance, presentation, and verification. Source: [8].

Decentralized Identity

Decentralized Identity represents the implementation of the Self-Sovereign Identity (SSI) philosophy. It fundamentally shifts control from external administrators to the individual user. By leveraging blockchain technology and asymmetric cryptography, users can generate and manage their own identifiers (DIDs) without permission from a central authority. In this model, the „root of trust“ shifts from administrative (trust in a company’s database) to cryptographic (trust in mathematical proofs) [20].

However, achieving decentralized identity requires a mechanism to verify that a digital identifier corresponds to a unique human without relying on a central registry. Several approaches have emerged to solve this „Proof of Personhood“ (PoP) problem, each with distinct strengths and weaknesses.

Soulbound Tokens (SBTs)

Proposed by Vitalik Buterin, Soulbound Tokens are non-transferable Non-Fungible Tokens (NFTs) held in a user’s wallet that represent permanent traits or reputations, such as a university degree or credit score. The primary advantage of this approach is its compatibility with the existing Ethereum infrastructure, which enables a publicly verifiable account history [28].

However, because SBTs reside on a public ledger, they offer zero privacy. Any observer can view the entire history of the wallet, making this method unsuitable for sensitive identity data, such as medical records or government IDs, which require confidentiality.

Social Verification (Web of Trust and Social Graphs)

Another major category relies on social relationships to establish identity. The Web of Trust model, historically associated with PGP (Pretty Good Privacy), relies on a subjective, peer-to-peer verification process where users explicitly „sign“ the keys of people they know and trust. In contrast, modern Social Graph protocols (such as BrightID or Proof of Humanity) utilize the network topology itself. These systems analyze the graph of user connections to algorithmically identify clusters of genuine humans versus „Sybil“ clusters of fake accounts [1].

While these social approaches are highly decentralized and require no specialized hardware, they face significant scalability and privacy challenges. To verify a user, the system often requires mapping their social connections, which can expose their real-world relationships to the public. Furthermore, these systems remain vulnerable to sophisticated attacks where a malicious actor cultivates a large number of fake trusted connections over time.

Biometric Verification

To address the uncertainty of social proofs, protocols like World ID utilize biometric verification. These systems use distinct physical characteristics, such as iris scans, to ensure that a digital identifier is bound to a unique human. This method offers the highest resistance to Sybil attacks, as it is physically impossible for a single person to generate multiple biometric signatures [1].

However, this security comes at the cost of decentralization. The reliance on specialized hardware (such as the „Orb“) introduces a physical supply chain dependency, and the col-

Mechanism	Sybil Resistance	Decentralization	Privacy	Inclusivity
Soulbound Tokens (SBTs)	Low	High	Low	High
Social Verification	Medium	High	Low	Medium
Biometric (e.g., World ID)	High	Medium	High	Low
Gov-Backed ZK-ID	High	Medium	High	Low

Figure 3.2: Comparison of Identity Approaches.

lection of biometric data raises concerns about the potential centralization of the database and the long-term privacy of users’ biological traits.

Government-Backed Cryptographic Identity

An alternative approach leverages government-issued credentials, such as NFC-enabled e-passports. Protocols like Self reuse the cryptographic signatures embedded by issuing authorities, allowing users to scan their passports with a smartphone and generate proof of holding a valid, unique document without revealing personal data [22]. This provides strong Sybil resistance because forging physical passports is difficult.

However, reliance on specific physical documents limits inclusivity, as many individuals do not possess NFC-enabled passports. Additionally, this solution is not fraud-resistant, and trust remains anchored in state issuers, making the resulting identity state-derived rather than strictly self-sovereign.

The main trade-offs between these approaches are summarized in Figure 3.2.

3.3 W3C Standards for Decentralized Identity

To make decentralized identity interoperable across different systems, the World Wide Web Consortium (W3C) has defined two critical standards: Decentralized Identifiers (DIDs) and Verifiable Credentials (VCs). Together, these standards replace the reliance on centralized databases with a cryptographic verification model.

Decentralized Identifiers

A Decentralized Identifier (DID) is a globally unique identifier that enables an entity to generate and control its identity without requiring a centralized registration authority [16]. Unlike traditional identifiers such as email addresses or domain names, which are rented from service providers and can be revoked at any time, a DID is created and owned entirely by the user.

Technically, a DID is a string typically formatted as a Uniform Resource Identifier (e.g., *did:method:unique-id*), see Figure 3.3. Its primary function is to resolve to a DID Document, a JSON-LD object that contains the subject’s public cryptographic keys and service endpoints. This document serves as the „root of trust“ for the identity. By retrieving the DID Document from a verifiable data registry (such as a blockchain), a verifier can obtain the public key needed to validate the user’s digital signatures without contacting any central administrator or identity provider.



Figure 3.3: Structure of a Decentralized Identifier (DID), consisting of the DID scheme, DID method, and method-specific identifier. Source: [16].

Verifiable Credentials

While a DID acts as an anchor for identity, a Verifiable Credential (VC) functions as the digital equivalent of a physical document, such as a passport, university degree, or driver’s license. It is a change-resistant data container that allows an Issuer to assert specific claims about a Holder. A standard VC consists of three primary components (as shown in Figure 3.4): metadata describing the issuer and issuance date, the actual claims (e.g., „Age: Over 18“ or „Degree: Computer Science“), and a cryptographic proof [17].

The proof, typically a digital signature generated by the Issuer’s private key, allows any third party to verify the authenticity of the credential. When a Verifier receives a VC, they can cryptographically check the signature against the Issuer’s public DID Document. This mechanism makes trust portable; the Verifier does not need to contact the issuing university to validate a degree, nor do they need to trust the Holder’s word. They simply rely on the mathematical validity of the Issuer’s signature.

The Privacy Gap

Despite their robust security architecture, standard VCs often lack granular privacy controls. By default, presenting a VC typically reveals all its data. For example, to prove one is over 18 using a digital passport credential, a user might be forced to share their full name, exact birthdate, and document number. This „all-or-nothing“ disclosure model sets the stage for integrating Zero-Knowledge Proofs, which enable the verification of specific claims without revealing sensitive data.

```

{
  "@context": [
    "https://www.w3.org/ns/credentials/v2",
    "https://www.w3.org/ns/credentials/examples/v2"
  ],
  "id": "http://university.example/credentials/3732",
  "type": [
    "VerifiableCredential",
    "ExampleDegreeCredential"
  ],
  "issuer": "https://university.example/issuers/565049",
  "validFrom": "2010-01-01T00:00:00Z",
  "credentialSubject": {
    "id": "did:example:ebfeb1f712ebc6f1c276e12ec21",
    "degree": {
      "type": "ExampleBachelorDegree",
      "name": "Bachelor of Science and Arts"
    }
  },
  "proof": {
    "type": "DataIntegrityProof",
    "created": "2025-04-27T17:58:33Z",
    "verificationMethod": "did:key:zDnaebSRtPnW6YCpxAh...",
    "cryptosuite": "ecdsa-rdfc-2019",
    "proofPurpose": "assertionMethod",
    "proofValue": "z5WHRyhjLd2H5RFcSqW..."
  }
}

```

Figure 3.4: Example W3C Verifiable Credential 2.0

[17]

Chapter 4

Zero-Knowledge Cryptography and zk-SNARKs

This chapter covers the cryptographic foundation of privacy-preserving identity systems. It begins by explaining the fundamental concepts of Zero-Knowledge Proofs. We then look into the process of transforming a computer program into a mathematical proof. Finally, we provide a comparison of different ZK systems, concluding with a deep dive into zk-SNARKs.

4.1 Introduction to Zero-Knowledge Proofs

Zero-Knowledge Proofs (ZKPs) are cryptographic protocols that allow one party (the Prover) to convince another party (the Verifier) that a specific statement is true without revealing any information beyond the validity of the statement itself.

In digital identity, this capability is very important. It allows a user to prove a claim such as „I am over 18“ without revealing their birth date or any other private data [5].

Core Roles and Definitions

A standard Zero-Knowledge protocol involves three key components:

- **The Prover:** The entity that holds a secret (often called the witness, denoted as w) and wishes to prove knowledge of it without disclosing it.
- **The Verifier:** The entity that validates the proof to ensure the Prover is telling the truth.
- **The Statement:** The public assertion being proven (e.g., „The hash of w equals x “).

Fundamental Properties

For a protocol to be considered a valid Zero-Knowledge Proof, it must satisfy three mathematical properties [5]:

1. **Completeness:** If the statement is true and both the Prover and Verifier follow the protocol honestly, the Verifier will always be convinced.
2. **Soundness:** If the statement is false, a cheating Prover cannot convince the Verifier, except with negligible probability. This prevents malicious actors from forging proofs.

3. **Zero-Knowledge:** If the statement is true, the Verifier learns nothing other than the fact that the statement is true. They learn nothing about the secret witness itself.

Conceptual Example: Ali Baba's Cave

To understand how this is conceptually possible, cryptographers often use the „Ali Baba's Cave“ analogy (also comparable to the „Where's Waldo“ puzzle often cited in ZK literature).

Imagine a cave shaped like a ring, with a magic door blocking the path on the far side (as shown in Figure 4.1). The door only opens for someone who knows the secret magic word.

- **The Scenario:** Alice (Prover) wants to prove to Bob (Verifier) that she knows the magic word, but she does not want to tell it to him.
- **The Protocol:** Bob waits outside while Alice enters the cave and randomly chooses to stand on side A or side B. Bob then walks to the entrance and shouts, „Come out from side B!“
- **The Proof:** If Alice knows the magic word, she can open the door and exit from side B, regardless of which side she originally entered. If she doesn't know the word, she can only exit from side B if she's already standing there (a 50% chance).

By repeating this test n times, the probability of Alice successfully cheating drops to $1/2^n$. If she succeeds 20 times in a row, Bob can be statistically certain she knows the magic word, yet Bob never hears the word itself.

4.2 From Code to Proof: How it Works

To use Zero-Knowledge proofs in a real application, we cannot simply run a standard computer program (like Python or Java). The logic must be translated into a specific mathematical format that cryptographic protocols can understand. This translation process is often referred to as the „ZK pipeline“ [3].

The Transformation Pipeline

The process of turning a high-level goal (e.g., „check if age > 18“) into a proof involves four distinct steps:

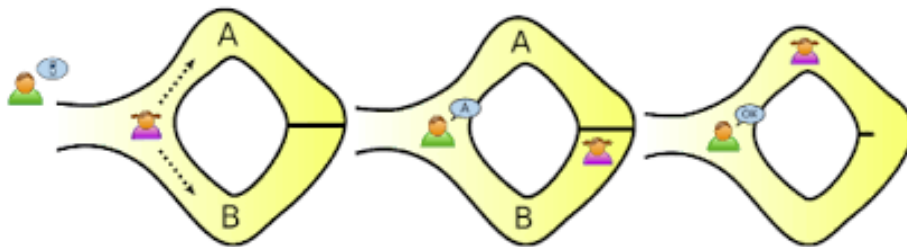


Figure 4.1: Ali Baba's Cave illustration. Source: [13].

1. **Computation (Flattening):** The first step is to break down the high-level code into its simplest possible operations, consisting only of additions and multiplications. For example, a complex equation like $x^3 + x + 5 = 35$ is „flattened“ into a series of basic gates:

$$a = x \times x \tag{4.1}$$

$$b = a \times x \tag{4.2}$$

$$y = b + x + 5 \tag{4.3}$$

This sequence is called an Arithmetic Circuit.

2. **Rank-1 Constraint System (R1CS):** The Arithmetic Circuit is then converted into a system of vector equations called R1CS. This system enforces that for every step of the computation, the inputs and outputs match correctly. It acts as a rigid „spreadsheet“ where every row represents a constraint (e.g., „value in column A times value in column B must equal value in column C“) [3].
3. **Quadratic Arithmetic Program (QAP):** This is the most critical mathematical transformation. The R1CS constraints are converted into Polynomials. Instead of checking thousands of individual equations, the Verifier only needs to check that two specific polynomials divide cleanly into each other ($P(x)/T(x)$ has no remainder). If they do, it proves that all the constraints in the circuit were satisfied simultaneously [3].
4. **Elliptic Curve Commitment:** Finally, because the polynomials are too large to send over the network, the Prover uses Elliptic Curve Cryptography to „commit“ to them. They send only a few small points on a curve (the proof) rather than the entire dataset [3].

Interactive vs. Non-Interactive

In the „Ali Baba“ example, Bob had to shout challenges to Alice. In the mathematical context of polynomials described above, this corresponds to the Verifier sending a random number to the Prover to check if the equations hold true at that specific coordinate. However, in a digital system, this back-and-forth communication is slow and impractical, especially for blockchains.

To solve this, modern systems use the Fiat-Shamir heuristic. Instead of waiting for a Verifier to send a random challenge, the Prover simulates the challenge itself by hashing their own data (e.g., $challenge = \text{Hash}(\text{Previous Data})$). Because a cryptographic hash function is unpredictable, the result serves as a „random“ challenge that the Prover could not have anticipated in advance. This turns the protocol into a Non-Interactive Zero-Knowledge proof, where the Prover generates a single packet of data that anyone can verify offline [4].

4.3 Comparison of Modern Zero-Knowledge Proof Systems

While the theoretical concept of Zero-Knowledge is universal, its practical implementation varies significantly. There is no single „best“ ZK protocol; rather, there is a family of

proving schemes, each with different trade-offs regarding proof size, verification speed, and trust assumptions. The three most prominent categories in the current landscape are zk-SNARKs, zk-STARKs, and Bulletproofs [31].

zk-SNARKs (Succinct Non-Interactive Arguments of Knowledge)

zk-SNARKs are currently the industry standard for blockchain applications, primarily due to their efficiency. They produce the smallest proofs (often just a few hundred bytes) and offer constant-time verification, meaning it takes the same few milliseconds to verify a proof regardless of the complexity of the underlying computation. This efficiency makes them ideal for on-chain identity verification, where gas costs are a critical constraint. However, this performance comes with notable drawbacks. Most SNARK constructions require a „Trusted Setup“ (see 4.3), and they typically rely on elliptic curve cryptography, which makes them vulnerable to theoretical future attacks by quantum computers [26].

zk-STARKs (Scalable Transparent Arguments of Knowledge)

Designed to address the limitations of SNARKs, STARKs rely on hash functions rather than elliptic curves. Their primary advantage is that they are „Transparent,“ meaning they require no trusted setup ceremony to initialize the system. Furthermore, because they do not rely on the discrete logarithm problem, they are considered „Post-Quantum Secure,“ making them resistant to future threats from quantum computing. The significant trade-off, however, is proof size. A STARK proof can be hundreds of times larger (45kB+) than an SNARK proof, making them significantly more expensive to store and verify on public blockchains like Ethereum [26].

Bulletproofs

Bulletproofs are a specialized form of range proofs often utilized in privacy-focused cryptocurrencies like Monero. Their main strength is that they require no trusted setup and produce relatively short proofs. However, unlike SNARKs, their verification time is linear rather than constant. As the complexity of the claim grows, the time required to verify it increases significantly. This scaling issue makes it less suitable for complex identity logic [31].

The practical differences between these proof systems are summarized in Figure 4.2.

System	Proof Size	Verification Speed	Trusted Setup	Quantum Secure
zk-SNARK	Very Small (< 1kB)	Very Fast (< 10ms)	Yes (Usually)	No
zk-STARK	Large (> 45kB)	Fast	No	Yes
Bulletproofs	Small (1.5kB)	Slow (Linear)	No	No

Figure 4.2: Comparison of Zero-Knowledge Systems.
[31]

4.4 Deep Dive: zk-SNARKs

Among the various Zero-Knowledge implementations, zk-SNARKs are the most widely adopted for blockchain applications due to their succinctness and verification speed. While alternatives like STARKs eliminate trusted setups, their large proofs make on-chain verification prohibitively expensive. To understand why zk-SNARKs are the optimal choice, we must first break down the acronym and then address their most significant architectural challenge: the Trusted Setup.

Defining the Acronym

The term zk-SNARK stands for Zero-Knowledge Succinct Non-Interactive Argument of Knowledge. Each component of this name represents a critical technical property:

- **Succinct:** The proof size is incredibly small and can be verified in constant time, regardless of the size.
- **Non-Interactive:** The Prover sends a single message to the Verifier. There is no back-and-forth communication required.
- **Argument:** This distinguishes it from a mathematical „Proof“ (which is absolute). An „Argument“ is secure because breaking the cryptography would require an impossible amount of computing power. While not theoretically impossible, it is computationally infeasible for any real-world adversary.
- **Of Knowledge:** The Prover does not just demonstrate that a solution exists; they demonstrate that they know the solution (the witness) at the time of proving.

The „Trusted Setup“ and Toxic Waste

The primary trade-off for the efficiency of SNARKs is the requirement for a Trusted Setup Ceremony. Before the system can be used, a set of global parameters, the Common Reference String (CRS), must be generated.

This process involves creating cryptographic secrets, often referred to as „Toxic Waste“ [7]. These secrets are used to generate the public proving and verification keys. Crucially, if this toxic waste is not destroyed after the setup, a malicious actor who retains it could generate fake proofs.

To mitigate this risk, protocols use Multi-Party Computation (MPC) ceremonies. In an MPC ceremony, widely known from the Zcash launch, dozens or hundreds of participants contribute randomness to the key generation. As long as at least one participant acts honestly and destroys their portion of the toxic waste, the entire system is secure [7].

What are proving Schemes? (Groth16, PLONK, and Honk)

While the general zk-SNARK pipeline is similar across tools, different „proving schemes“ handle the constraints’ mathematics differently. The choice of scheme affects how the trusted setup works and how fast the proof is generated.

- **Groth16:** This is the most famous proving scheme and the default for tools like Circom. It produces the smallest possible proofs and is very fast to verify. However,

its biggest downside is that it requires a circuit-specific trusted setup. If you change a single line of your code, you must run a new setup ceremony from scratch [15].

- **PLONK:** Used by modern frameworks like Noir, PLONK introduced a „Universal“ trusted setup. This means the ceremony only needs to be done once for the entire ecosystem. As long as your program stays below a certain size, you can change your code as much as you want without needing a new ceremony [12].
- **Honk:** This is the latest generation of proving schemes. Honk improves upon PLONK by using „sum-check“ protocols and better lookup arguments. It is designed to be significantly faster for the Prover while maintaining small proof sizes, making it the current state of the art for browser-based proving [38].

Practical zk-SNARK Tooling

In practice, developers do not write zk-SNARKs directly as polynomial equations. Instead, they use specialized toolchains that compile higher-level circuit code into arithmetic constraints and then generate proofs from concrete witness data.

One widely used language is Circom, a domain-specific language for describing arithmetic circuits. It is especially common in the Ethereum ecosystem and is often paired with tools such as `snarkjs` [25]. Another modern alternative is Noir [34], which provides a syntax closer to general-purpose programming languages while still compiling programs into representations suitable for zero-knowledge proving.

It is also useful to distinguish between a circuit language and a proving backend. A circuit language describes the computation and constraints that must hold. A proving backend performs the cryptographic work required to generate and verify proofs from those constraints.

Barretenberg is one example of such a proving backend and cryptographic library [19]. In practice, developers often interact with it through a command-line tool that compiles artifacts, generates proving and verification keys, creates proofs, and exports verifier components. In short, tools such as Circom or Noir specify what should be proved, while tools such as Barretenberg provide the machinery that enables proof generation and verification.

Chapter 5

Proof-of-Social-Capital

This chapter introduces Proof-of-Social-Capital (PoSC), a novel consensus protocol proposed by Juraj Mariani and Ivan Homoliak in 2025 [24]. It outlines its core motivation: replacing traditional consensus protocols with social influence to ensure a fairer distribution of consensus power. Furthermore, we describe the system’s architecture and economic model. Finally, we analyze the protocol’s reliance on privacy-preserving sybil resistance and identify the specific implementation gap this thesis aims to resolve.

5.1 System Overview and Motivation

The evolution of blockchain consensus has largely been driven by the need to secure networks without relying on centralized authorities. The first generation, Proof-of-Work, established decentralized security but introduced significant drawbacks: it consumes vast amounts of energy and tends towards centralization due to the high cost of specialized mining hardware, creating a steep entry barrier for ordinary users [24].

Its successor, Proof-of-Stake (PoS), resolved the energy consumption issue but replaced hardware costs with financial capital requirements. In PoS systems like Ethereum, influence is directly proportional to wealth; participants with substantial financial resources can consolidate control, gaining an edge over those with limited capital. This effectively recreates the centralization of wealth found in traditional finance [24].

To address these limitations, the PoSC protocol proposes a fundamental shift in how consensus power is allocated. Instead of relying on scarce physical or financial resources, PoSC leverages „social capital“ the value derived from human attention and influence. In this model, consensus nodes (content creators) secure the network not by buying tokens or hardware, but by earning their followers’ support. This approach aims to lower the entry barrier and ensure that the security of the blockchain is fueled by a non-tradable resource that is arguably more democratic than raw capital [24].

5.2 System Design and Economic Model

The PoSC protocol functions similarly to Ethereum’s Proof-of-Stake but fundamentally alters the resource used for security. The system differentiates between two primary network entities: Consensus Nodes (Content Creators) and Light Clients (Followers).

- **Consensus Nodes (Content Creators):** These are users with high social influence who wish to participate in block validation. They do not stake financial assets;

instead, they „stake“ the social capital they have accumulated through endorsements from their followers. Their probability of being elected as a block leader is proportional to their support [24].

- **Light Clients (Followers):** These are standard users who consume content. Each unique user is assigned a limited, non-tradable budget of „Passive Social Capital.“ This budget has no value unless it is endorsed (allocated) to a Content Creator, at which point it becomes „Active Social Capital“ for that creator [24].

Consensus Mechanics and Scaling

A critical innovation of PoSC is its approach to fairness. In traditional social media, influence often follows a „power law“ distribution, where a tiny fraction of creators (e.g., the top 1%) controls the vast majority of attention. To prevent this inequality from centralizing the blockchain consensus, the protocol introduces a scaling mechanism.

The system distinguishes between three states of social capital:

1. **Passive Social Capital:** The unspent budget held by followers.
2. **Active Social Capital:** The raw sum of endorsements received by a creator.
3. **Effective Social Capital:** The final value used for consensus power, calculated by applying a non-linear scaling function (such as a square root or logarithmic function) to the active capital.

By using a scaling function (e.g., $Effective = \sqrt{Active}$), the protocol lowers the benefit of massive follower counts. Experiments in the original paper demonstrated that this mechanism reduced the Gini coefficient (a measure of inequality) from ≈ 0.78 to ≈ 0.29 , ensuring a much more decentralized validator set than traditional social media platforms allow [24].

Incentive Structure

The protocol employs a dual-incentive model to sustain engagement. Content Creators are rewarded with native tokens for successfully proposing and attesting blocks, similar to Ethereum validators. Uniquely, Followers are also incentivized; creators can distribute a portion of their revenue back to their supporters as a reward for engaging with content (e.g., viewing ads or campaigns). This creates a circular economy where attention is monetized and shared rather than extracted [24].

5.3 Sybil Attack Resistance Mechanism

The most critical security challenge for the PoSC protocol is preventing Sybil attacks. In traditional Proof-of-Stake systems, the scarce resource (money) is expensive to acquire, making it costly for an attacker to create thousands of fake identities. However, in PoSC, „Passive Social Capital“ is assigned to every user by default. Without a strict uniqueness check, a malicious actor could generate thousands of sybil accounts, endorse themselves, and artificially inflate their consensus power to dominate the network [24].

Therefore, the protocol relies on a „one-person-one-account“ policy, enforced through a privacy-preserving identity layer.

Verifiable Credentials and zkSNARKs

To ensure that every account corresponds to a unique human being without creating a centralized database of personal data, the protocol utilizes W3C Verifiable Credentials issued by trusted authorities (such as government eID providers).

The verification process occurs entirely on the user’s side (client-side) to protect privacy. Instead of uploading their ID document to the blockchain, the user generates a zkSNARK proof. This zero-knowledge proof confirms two facts simultaneously [24]:

1. **Validity:** The user possesses a valid credential signed by a recognized, trusted issuer.
2. **Ownership:** The user knows the private key associated with that credential.

Uniqueness Enforcement (The ID-Hash Nullifier)

While the zkSNARK proves that the document is real, it does not inherently prevent a user from using the same document twice to create two different accounts. To solve this, the protocol introduces the concept of a „Nullifier“ or ID-hash.

During the proof generation, the user hashes a subset of their static identity fields (e.g., $\text{Hash}(\text{Name} + \text{BirthDate} + \text{Nationality} + \dots)$). This resulting ID hash is unique to the individual but has sufficient entropy to be computationally impossible to reverse-engineer into the original personal data. This hash is sent to the blockchain along with the zkSNARK.

The smart contract stores all registered ID-hashes. If a user attempts to register again, the contract detects that the ID-hash already exists in the state and rejects the transaction. This ensures that while the system remains permissionless and private, it is mathematically impossible for a single human to control more than one allocation of social capital [24].

5.4 Limitations and Thesis Contribution

The original PoSC implementation was designed primarily to validate the consensus logic (leader election, social capital scaling, and block proposal). Consequently, the identity verification component remained a „Proof of Concept“ with several practical limitations that this thesis aims to explore.

First, the reliance on simplified cryptography needs to be addressed. The prototype used standard ECDSA signatures as placeholders for nullifiers, avoiding the complexity of zero-knowledge proofs. This thesis introduces the implementation of fully functional zk-SNARK circuits, attempting to define the arithmetic constraints required to verify signatures within the proof and generate an ID-Hash with sufficient entropy [24].

Second, this thesis aims to investigate the integration of real-world trust anchors. The original system assumed ideal „Trusted Issuers,“ abstracting away the complexity of the current infrastructure. The goal is to bridge this gap by exploring potential trust anchors and investigating approaches to identify suitable identity sources that can be integrated into the protocol to replace theoretical assumptions.

Finally, scalable storage needs to be explored and tested. The original design proposed storing uniqueness constraints directly in the on-chain Merkle Patricia Trie (MPT), which introduces prohibitive overhead for possibly millions of users [24]. We analyze alternative storage architectures to design a mechanism that handles high-volume identity registration without causing excessive state bloat associated with on-chain MPT storage.

Chapter 6

Options for Proof-of-Uniqueness

This chapter examines practical approaches to implementing the Proof-of-Uniqueness (PoU) system. It begins by defining the core requirements of a PoU system. It then analyzes potential models for trusted identity issuers. Subsequently, it evaluates two storage architectures to manage identity commitments and mitigate on-chain state bloat. Finally, it proposes integrating a Verifiable Threshold Oblivious Pseudorandom Function (vOPRF) to ensure deduplication without exposing users to the vulnerabilities of single-secret privacy architectures.

6.1 Definition and System Requirements

At a high level, several building blocks are required for an effective PoU system. A functional system consists of three different actors and the data they exchange to verify a user, as mentioned in Section 3.1. Certain interactions between these actors are very important building blocks for a functional PoU system. We investigate these core components below [1].

Deduplication

For a PoU to be useful, it must have a strict notion of uniqueness. If the proof can be acquired multiple times and transferred to fraudulent actors or bots, the system fails to serve its purpose. Therefore, the mechanism must accurately deduplicate across users who are issued a credential [1].

Authentication

To make credentials useful, they need to be hard to transfer to others, preventing fraud. When issuing credentials, issuers only need to verify that a person is unique, without requiring additional personal information. However, each credential needs to be securely tied to a specific person. Because wallets and smartphones can be transferred, it is crucial to authenticate the user as the rightful owner of the credential to prevent unauthorized use [1].

Recovery

If the user loses their smartphone or their credentials are compromised, we need effective recovery mechanisms. In setups where users manage their own keys, this is a significant challenge.

Revocation

We cannot assume that all participants act with perfect integrity. When an issuer is found to be compromised or malicious, the impact can be mitigated by removing the affected credentials from the list of accepted proofs. Also, the user should be able to revoke their own credential if they wish.

Expiry

The efficacy of security mechanisms degrades over time as new cryptographic attacks are continually developed. As a result, many identity systems add a predefined expiry date to credentials at issuance, similar to physical passports. Although expiry is not strictly required, it should be an integral part of the system design, as it strengthens the overall integrity and long-term robustness of the PoU system [1].

Inclusivity

A global PoU should be as inclusive as possible, meaning it is available to everyone. People should be able to participate regardless of their nationality, race, gender, or economic means. To ensure accessibility and support millions of users, the solution should not require expensive or specialized hardware and must function efficiently on standard, widely available consumer devices.

Decentralization and Privacy

The issuance of the credentials should not be controlled by a single entity to maximize resilience. Additionally, since it operates on blockchain, the system must avoid storing sensitive data or allowing users to be traced, ensuring individuals remain anonymous and fully in control of their information [1]. To protect users' privacy, the unique identifier ID-Hash must be derived from values with high, unpredictable entropy. If the entropy of the input fields is too low, an attacker could use brute-force attacks to easily guess the user's private data [24].

6.2 Options for Identity Issuers

The original Proof-of-Social-Capital (PoSC) protocol abstracts the identity layer by assuming the existence of ideal “trusted issuers” [24]. In this section, we investigate practical options for these issuers to make the system functional in reality.

Electronic Passports (ePassports via NFC)

One investigated approach is to use the cryptographic NFC chip embedded inside modern electronic passports. The user can extract verified identity data directly to their smartphone using NFC.

The main advantage of this approach is its high entropy. Passports contain a high-resolution biometric facial photograph. This provides excellent cryptographic entropy, ideal for generating a secure, unpredictable ID-Hash that resists brute-force deanonymization attacks.

However, after researching, there are two major limitations to this option:

- **Inclusivity:** Not all citizens in the world possess an NFC-readable passport, and even fewer own smartphones with the necessary NFC capabilities to read them. This violates the protocol’s strict inclusivity requirement [29].
- **Cryptographic Overhead:** This is the most critical bottleneck. The vast majority of ePassports rely on RSA signatures (specifically RSA-2048) for their data authenticity. Verifying an RSA-2048 signature inside a zk-SNARK circuit requires a prohibitively large number of arithmetic constraints. This makes zero-knowledge proof generation too computationally heavy and highly inefficient to run on standard consumer mobile devices [32].

National ID Certificates (eID)

Another approach is to leverage standard government-issued digital identity smart cards. I specifically conducted an experimental investigation using the Czech national ID card (*eObčanka*) to test this feasibility.

During the exploration, I identified significant limitations regarding this approach:

- **Low Information Entropy:** These digital certificates typically contain a very limited set of personal data fields. Because of this lack of data, it is mathematically difficult to generate an ID-Hash with sufficient entropy to protect user privacy against brute-force attacks.
- **The Sybil Vulnerability:** To artificially increase the low entropy, one might try to incorporate the certificate’s unique serial code into the ID-Hash. However, this would critically break the system’s Sybil resistance. A user can easily report their physical card as lost, pay a small administrative fee, and receive a newly issued certificate with a new serial code. This allows a single human to easily register multiple valid identities in the system.

Future Outlook: The European Digital Identity (EUDI) Wallet

The current limitations of heavy, legacy RSA cryptography and the low entropy in physical smart cards have also been investigated for the near future. The European Union is preparing the European Digital Identity (EUDI) Wallet framework, mandated by the eIDAS 2.0 regulation, which aims to provide every citizen with a secure digital wallet by 2026 [10].

The important technological shift is that the EUDI Wallet is being designed with native support for Verifiable Credentials and Zero-Knowledge Proofs, enabling privacy-preserving features such as selective disclosure. This development could directly address the cryptographic bottlenecks identified in the research by eliminating the need to verify heavy RSA-2048 signatures in custom mobile circuits. It also promises to provide standardized attributes with sufficient entropy. Concluding that this initiative demonstrates that the PoU identity layer will become fully feasible in the near future [10].

The difference between the data exposed by an ePassport and a national eID certificate is illustrated in Figure 6.1.

eID Certificate (X.509)

```
Certificate:
Data:
  Version: 3 (0x2)
  Serial Number: 23918185 (0x16cf669)
  Signature Algorithm: sha256WithRSAEncryption
  Issuer: C=UT, O=State Services, CN=...
  Validity
    Not Before: Jan 1 11:00:00 2025 GMT
    Not After : Jan 1 11:00:00 2030 GMT
  Subject: C=UT, CN=John Doe, SN=Doe,
    GN=John, serialNumber=ID9876543
  Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
    Public-Key: (2048 bit)
    Modulus:
      00:ad:33:b6:6b:c7:1e:bd:bb:24:...
      ... [Truncated for brevity] ...
      10:55:ea:4a:a9:cb
    Exponent: 65537 (0x10001)
  X509v3 extensions:
    X509v3 Key Usage: critical
      Non Repudiation
  Signature Algorithm: sha256WithRSAEncryption
  Signature Value:
    39:09:99:2d:83:90:70:83:a0:3a:f2:57:...
    ... [Truncated for brevity] ...
    e4:0a:07:f6:36:5a:c0:e3
```

Figure 6.1: Comparison of extracted data payloads between an ePassport (left) and a standard government eID Certificate (right). Personal information has been replaced with placeholder values.

6.3 Storage Architectures for Identity Commitments

Next, the challenge of identity storage must be addressed. A global PoU system must be able to register millions of users. Every registered user generates a unique ID-Hash that must be stored along with some other metadata to prevent them from registering twice. This section investigates two architectures for securely storing this massive amount of data.

On-Chain State Mapping

The most straightforward approach that was investigated is direct on-chain storage. In this architecture, the user generates a zk-SNARK proof of their identity and sends it directly to the smart contract. The smart contract verifies the proof and stores the resulting ID-Hash in its internal state, usually using a standard Solidity `mapping`.

The main advantage of this approach is that it is structurally very simple and relies entirely on the Layer 1 blockchain for security. However, storing millions of hashes directly in the Ethereum global state creates massive state bloat. Because every node in the network must permanently store this data, the gas costs for users writing to this mapping could become prohibitively expensive. This makes the direct approach very difficult to scale for a globally inclusive system.

The sequence of this direct on-chain architecture is shown in Figure 6.2.

On-Chain Storage Architecture

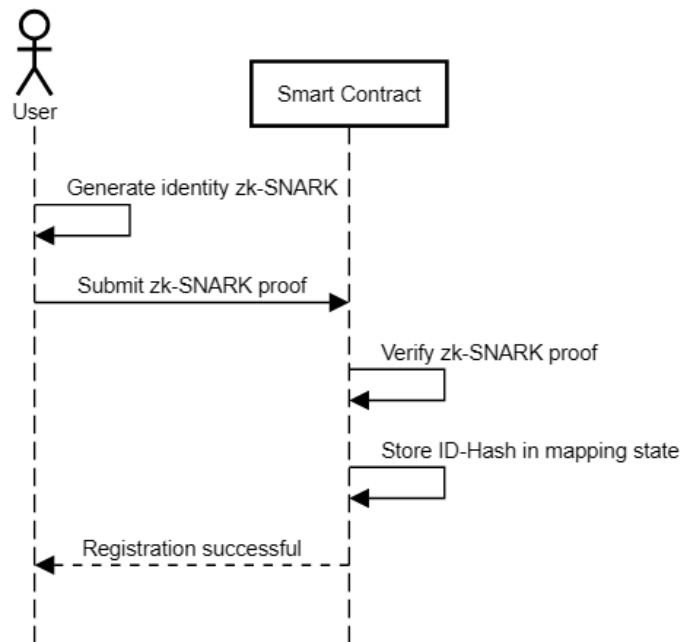


Figure 6.2: Sequence diagram of the on-chain storage architecture. The user sends the zk-SNARK identity proof directly to the smart contract, which stores the ID hash.

Off-Chain Merkle Tree with Incentivized Provers

To address state bloat, I investigated a more scalable architecture that leverages off-chain storage. Instead of storing every ID-Hash, the smart contract stores only a single 32-byte cryptographic hash representing the root of a Merkle Tree.

In this approach, the user submits their identity zk-SNARK to the smart contract, attaching a fixed amount of ETH as a reward bounty. The smart contract verifies the proof, temporarily holds the ETH, and emits a „claim“ event containing the user’s ID hash.

A decentralized network of independent provers listens to these events. They keep the entire Merkle Tree stored off-chain. When a prover sees the event, they generate a second zk-SNARK proof. This second proof mathematically guarantees two things: that the user’s ID-Hash is not already in the tree (proving non-inclusion) and that the new Merkle root is calculated correctly.

The prover submits this second proof to the smart contract. The contract verifies it, updates the single root, emits an insert event, and releases the ETH bounty to the prover. The insert event is very important because it allows any new prover to reconstruct the off-chain tree from zero just by reading the blockchain history (the entire sequence is shown in Figure 6.3).

While this approach successfully reduces the heavy storage burden on the blockchain and minimizes gas costs, coordinating state changes through decentralized off-chain provers introduces significant system complexity. Specifically, managing the commitments in the case of credential expiry, revocation, or authentication becomes highly difficult when relying on external provers to constantly update the tree for every lifecycle event.

6.4 Oblivious Pseudorandom Function for Privacy-Preserving Deduplication

In the original design, systems often relied on submitting an identity hash directly derived from the user’s credentials. But modern privacy systems that use zero-knowledge proofs have a significant weakness: if the single secret held by the user is ever leaked or stolen, their protection is permanently lost. An attacker could use this leaked secret to compute all past actions and link the person’s history together. This makes it very hard to build long-term identity systems in which pseudonyms must remain private for months or years.

To address this problem, this section introduces an Oblivious Pseudorandom Function (OPRF). An OPRF is a two-party cryptographic protocol in which one party holds a secret key and the other provides an input. Together, they compute a pseudorandom output corresponding to that input and key. The input-holding party obtains the final result, while the key-holding party learns nothing about the input. At the same time, the input holder gains no information about the secret key beyond what can be inferred from the output.

Specifically, the TACEO’s solution is considered here, which provides a Verifiable Threshold OPRF service TACEO:OPRF [35]. TACEO uses a distributed network of servers rather than a single central server. The secret key is divided among these nodes using Multi-Party Computation (MPC). This means the public uniqueness identifier (the nullifier) is derived from a joint calculation between the user’s hidden identity secret and the TACEO network’s threshold secret. This prevents single points of failure and ensures deduplication does not rely on a single user secret.

The TACEO:OPRF Protocol Flow

This protocol has three very useful properties for Proof-of-Uniqueness. First, determinism means the same ID-Hash always produces the same output under the same key. Second, unlinkability ensures the blinded query hides the real input. Third, verifiability means the client can check that the evaluation used the correct distributed key [35].

Let the private identity input be denoted as H_{id} . In the OPRF flow, the client first blinds H_{id} with a random scalar β :

$$A = \beta \cdot H_{id} \tag{6.1}$$

Assume the network secret key is shared among n parties, and a threshold subset S with $|S| = t$ participates in the evaluation. Each party $i \in S$ holds a secret share k_i and returns a partial evaluation

$$B_i = k_i \cdot A. \tag{6.2}$$

The client then aggregates the threshold responses using Lagrange coefficients λ_i [6]:

$$B = \sum_{i \in S} \lambda_i B_i = k \cdot A, \tag{6.3}$$

where k is the effective joint OPRF secret reconstructed only in the exponent. Finally, the client removes the blinding factor and derives the final deterministic output:

$$nullifier = \beta^{-1} \cdot B. \tag{6.4}$$

This means that the same hidden identity input always produces the same nullifier under the same OPRF key, while neither the OPRF nodes nor outside observers learn the original ID-Hash [35].

In our context, the protocol could work as follows:

1. **Authorization and Blinding:** The user locally generates a deterministic ID-Hash from their Verifiable Credential. They then blind this input with a random scalar so that the query sent to the TACEO:OPRF network reveals no usable information about the original identity input. Before the evaluation is accepted, the user must also prove they are authorized to use the service, for example, by presenting a zero-knowledge proof of credential ownership together with a live signature from the VC holder.
2. **Threshold Server Computation:** The user sends the blinded query to the decentralized threshold network of OPRF nodes. Each MPC node applies its secret share to that blinded input. No single node ever learns the full secret, and the network learns nothing about the user's input.
3. **Unblinding and Nullifier Generation:** The nodes return their evaluated shares. The user combines them, removes the blinding factor locally, and derives the final nullifier. This value is deterministic for the same ID-Hash and OPRF key, but it remains computationally unlinkable to the original credential contents.

Because TACEO:OPRF is a verifiable OPRF, the client can additionally verify that the evaluation was performed under the expected distributed OPRF public key, typically via a threshold discrete-log equality proof [9, 35]. This strengthens trust in the result: the client does not merely receive a pseudorandom-looking value, but one that can be checked against the intended network key. As a consequence, one effective identity input produces one deterministic nullifier for a given OPRF key, which is exactly the property needed for deduplication in a Proof-of-Uniqueness system.

Figure 6.4 shows how the blinded user secret and the distributed TACEO network secret produce the final nullifier.

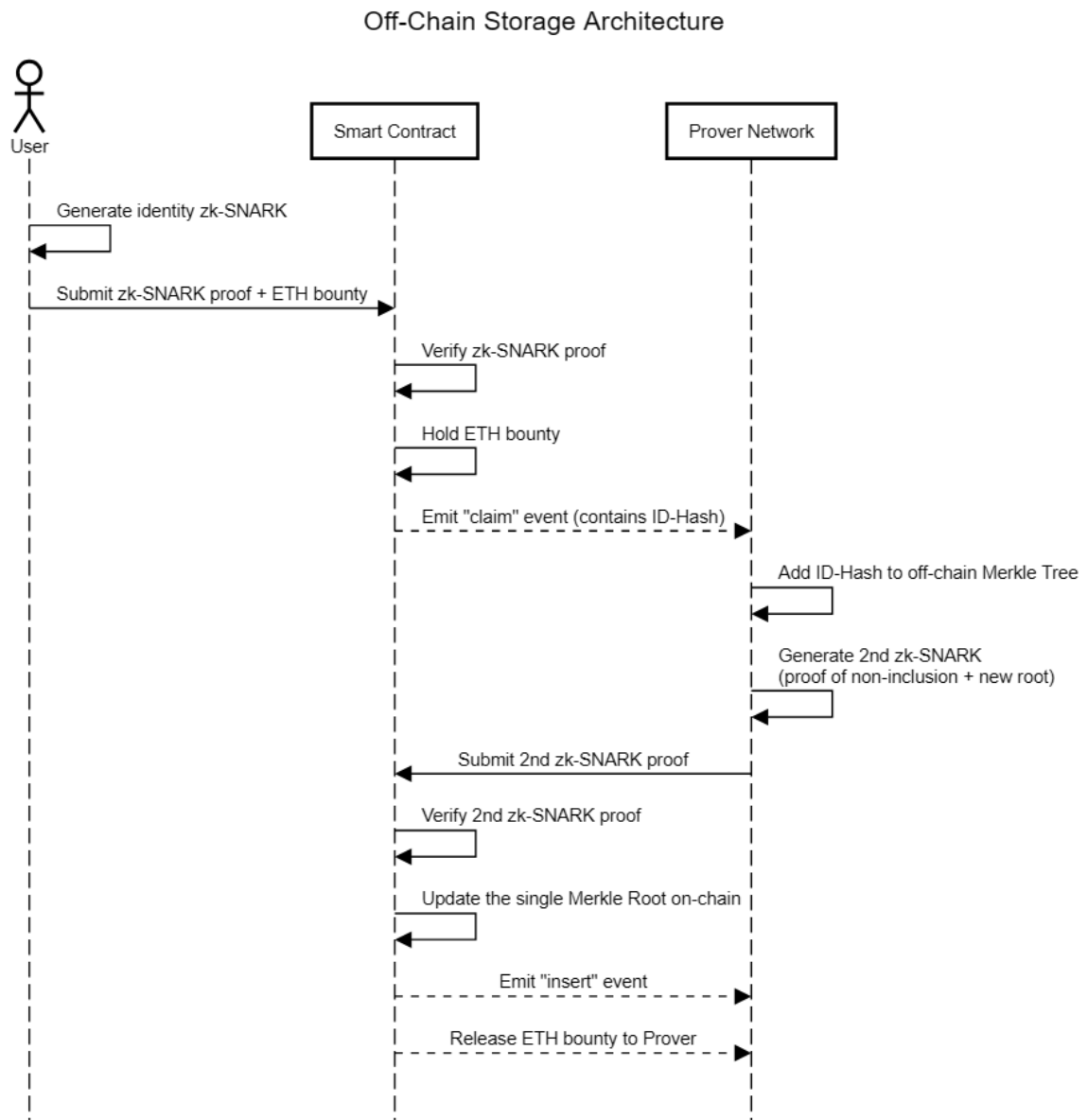


Figure 6.3: Sequence diagram of the off-chain storage architecture. The smart contract only emits events, and a decentralized prover network keeps the Merkle tree off-chain. The provers generate a second zk-SNARK proof to update the single on-chain root.

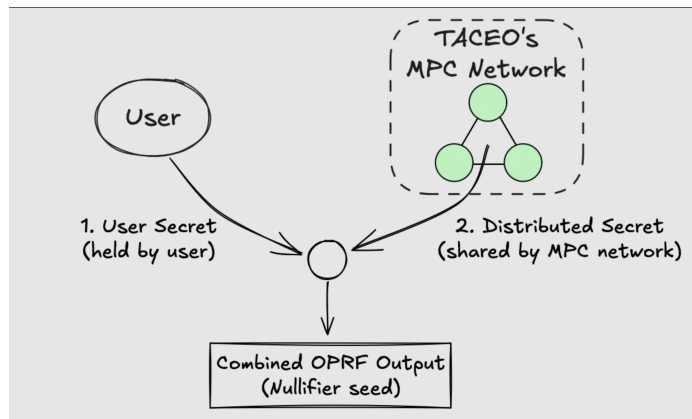


Figure 6.4: Diagram of the TACEO:OPRF nullifier generation from a blinded user secret and a distributed TACEO network secret. Source: [35].

Chapter 7

Proposed Approach

This chapter defines the proposed approach for the privacy-preserving Proof-of-Uniqueness system. First, it introduces the overall design and the main actors in the protocol. It then describes the credential and holder-binding model, the cryptographic construction, the TACEO:OPRF integration, the smart contract registry design, and the complete protocol lifecycle. Finally, it presents the implemented prototype and the concrete technologies used to validate the design.

7.1 Design Overview

At a high level, the proposed architecture connects four actors: the issuer, the holder, the TACEO:OPRF network, and the smart contract registry. First, the trusted issuer verifies a person's identity in the real world and issues them a Verifiable Credential (VC). The holder then uses this credential locally to create two separate zk-SNARKs. The first proof is used to authorize a request to the TACEO:OPRF network, which helps transform the user's hidden identity into a unique, deterministic nullifier. The second proof is then used to safely enroll this nullifier in the smart contract registry.

This design strictly separates private identity data from public uniqueness data. The private layer keeps the VC, the holder's private key, and the hidden identity input completely safe on the user's device. The public layer only contains the generated nullifier, basic issuer metadata, and a connected wallet address. The smart contract registry never sees the private credential attributes. It only verifies the zk-SNARK proofs and the public metadata.

Each part of the system has a specific, limited job. The issuer is responsible for verifying in the real world. The two zk-SNARKs ensure that the data is consistent and that every action is authorized without leaking secrets. The TACEO:OPRF network adds a threshold secret to disconnect the nullifier from user-sensitive information. Finally, the smart contract registry enforces the rules for uniqueness, expiry, and revocation without needing to know who the users actually are.

7.2 Credential and Holder Binding Design

The design assumes an issuer that correctly performs the off-chain real-world verification before issuing a credential. However, as mentioned in Chapter 6, a perfect issuer does not yet exist. A zero-knowledge proof can verify that a credential was signed by the claimed

issuer and that its fields were not modified, but it cannot prove that the issuer’s real-world verification process was honest. Because we do not have this ideal issuer today, we must carefully outline the strict requirements that any future credential system needs to meet to work with our proposed architecture.

The first major requirement is zk-friendliness. As discussed in Chapter 6, traditional cryptography, like heavy RSA signatures, is not practical for consumer device zero-knowledge proofs. For the system to be inclusive and efficient, the issuer must use modern signature schemes (e.g., EdDSA Baby Jubjub) that are easy and fast to verify within a zk-SNARK circuit [37].

The second requirement is the data structure. To properly bind the user, the credential must contain four categories of information:

- **Semantic metadata:** Information such as the credential type, issuer identifier, issuance time, and expiry time.
- **Identity attributes:** A set of stable personal fields that together provide enough high entropy to safely identify a person.
- **Holder public key:** A key that strongly binds the use of the credential to a private key controlled only by the holder’s device.
- **Issuer signature:** A zk-friendly cryptographic signature over all the credential contents to prove authenticity.

The third requirement is key separation. Using the credential’s embedded holder public key directly on the blockchain for later authorization would leak the user’s identity and make the credential unusable elsewhere. To prevent this privacy loss, the user must use a separate standard blockchain wallet. The holder key is only used locally to prove control over the credential. The separate wallet is used for all on-chain interactions, such as enrollment, revocation, and future authorization. A valid enrollment safely connects these two: the holder key authorizes the hidden credential, and the wallet authorizes the public identity record.

7.3 Protocol Cryptography: From Credential to Verifiable Nullifier

Once the user has valid credentials and a separate blockchain wallet, the system must process this data without leaking any private information. The core principle of the cryptographic design is that the credential is never published. Instead, the user only uses it locally as private input to create zk-SNARKs, which are then passed between individual actors.

To make these proofs run fast on standard consumer devices, the system must use zk-friendly cryptography. A very important part of this is the hash function. When the user extracts specific fields from their credential to create their private identity input (ID-Hash), the system uses the Poseidon hash function. Poseidon is specifically designed to be highly efficient within zk-SNARK circuits because it imposes far fewer mathematical constraints than traditional hash functions like SHA-256 [14].

The private identity input can differ between designs; however, for example, it can be calculated using Poseidon as follows:

$$H_{id} = \text{Poseidon}(\text{credentialSubjectId}, \text{name}, \text{dateOfBirth}, \text{placeOfBirth}, \text{sex}, \text{nationality}, \text{validFrom}) \quad (7.1)$$

This H_{id} value is deterministic, meaning it is always the same for a given Verifiable Credential, but it is kept completely secret. The whole process of securely turning this secret H_{id} into a public uniqueness record happens in three main steps:

Step 1: The Authorization zk-SNARK

Before the TACEO network serves the user, it needs to know that the user is allowed to make a request. But the user cannot send their real credentials. So, the user creates the first authorization zk-SNARK.

In this step, the circuit validates the issuer’s signature and the holder’s signature, then computes the H_{id} and blinds it with a random number. This means that the zero-knowledge proof shows that the blinded query is derived from a valid, issuer-signed credential and that the caller owns the private key associated with the VC. The network only sees the blinded query and the proof, so no identity data is leaked.

Step 2: The TACEO:OPRF Evaluation

After creating the first proof, the user sends the blinded query and the authorization zk-SNARK to the TACEO:OPRF network. Because the query is blinded, the network’s nodes do not learn the user’s true H_{id} .

The network first checks the zk-SNARK. If it is valid, it means the user is authorized. Then, the nodes apply their secret threshold shares to the blinded query and return the partial results. The user combines these results and removes the random blinding factor locally. This reveals the final public nullifier. Because of how OPRF works, this nullifier is perfectly deterministic: the same H_{id} always creates the exact same nullifier.

Step 3: The Enrollment zk-SNARK

Now that the user has the final nullifier, they must register it on the smart contract registry. To do this privately, the user creates the second zk-SNARK.

This proof brings everything together. It proves that the user has a valid credential and that the blinded input sent to the TACEO:OPRF network was correctly generated from their secret H_{id} . It also proves that the OPRF transcript is mathematically correct and evaluated under the trusted OPRF public key.

Additionally, the user includes their blockchain wallet address as a public input to the zk-SNARK. This binds the zero-knowledge proof itself to that specific wallet address to prevent replay attacks. Because the proof is tied to a single wallet, a malicious actor or front-runner cannot steal it and submit it to the smart contract using a different wallet. Once this zk-SNARK is forged, the user can submit it to the smart contract registry.

7.4 Smart Contract Registry Design

The smart contract registry is the public source of truth for enrolled identities. It first verifies the submitted zero-knowledge enrollment proof. If the proof is mathematically

correct, the contract stores the new identity record along with its public metadata and enforces the lifecycle rules.

The registry stores the active enrollment state indexed by the nullifier. Each active record needs only the metadata required to evaluate future validity: expiry, issuer reference, and bound wallet address. The registry also maintains trust anchors for accepted issuers and the accepted OPRF public key. If an issuer is removed or a credential expires, the associated identity is no longer considered valid.

This design deliberately favors a direct on-chain active-state registry over an off-chain commitment tree. The off-chain Merkle-tree option discussed in Chapter 6 would reduce the direct chain state, but it would make revocation, expiry, issuer removal, and third-party validity checks dependent on external tree maintainers, thereby adding complexity. For the proposed architecture, the registry remains the single synchronization point for live identity status. The cost is that active records still occupy chain state. A partial solution to this problem is presented in Section 7.5.

7.5 Proof-of-Uniqueness Lifecycle

The complete protocol consists of enrollment, authorization, revocation, cleanup, and governance. These phases are connected by the same state object: the enrolled nullifier and the wallet address bound to it.

Enrollment Phase

Enrollment begins after the holder receives a valid credential. The holder locally derives the private identity input and creates a blinded OPRF query. The holder then proves that this blinded query is connected to a valid credential and an authorization from the credential's holder key. If the OPRF network accepts the proof, it evaluates the blinded query and returns threshold responses.

The holder verifies and unblinds the OPRF response to obtain the nullifier. The holder then creates an enrollment proof showing that the credential, the OPRF transcript, the nullifier, and the selected wallet address all correspond to the same hidden identity input. The wallet also authorizes the enrollment, so a proof generated for one wallet cannot be submitted as an enrollment for another wallet.

The registry accepts the enrollment only if the proof is valid, the issuer is trusted, the credential has not expired, the OPRF public key matches the accepted trust anchor, the wallet binding is correct, and the nullifier has not already been enrolled. If these checks pass, the registry records the nullifier as an active identity.

The full enrollment phase is shown in Figure 7.1.

Authorization Phase

After enrollment, a third-party service can use the registry as a proof-of-uniqueness oracle. The service asks the user for a fresh challenge signature. The user responds with the enrolled nullifier and a signature from the wallet bound to that nullifier.

The third party checks the registry state for the nullifier and obtains the associated wallet address if the identity is still valid. If the challenge signature recovers the same wallet address, the service learns that the user controls the wallet associated with a unique active identity.

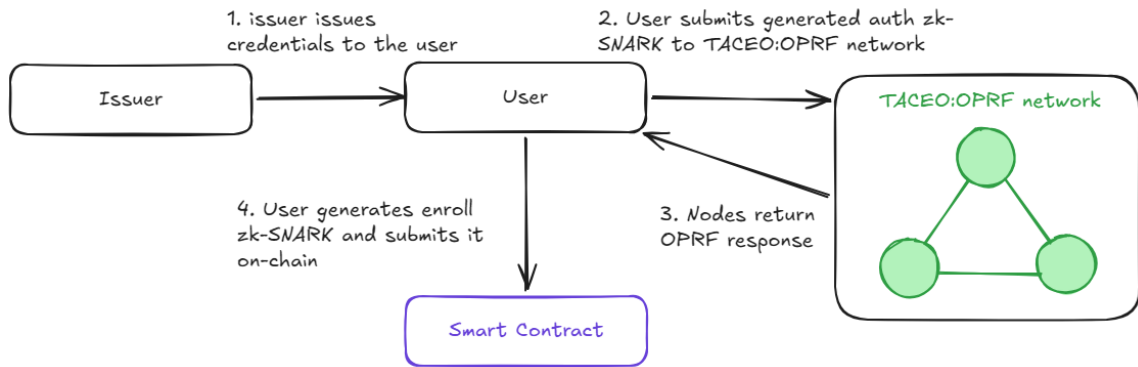


Figure 7.1: Diagram of the proposed protocol enrollment phase.

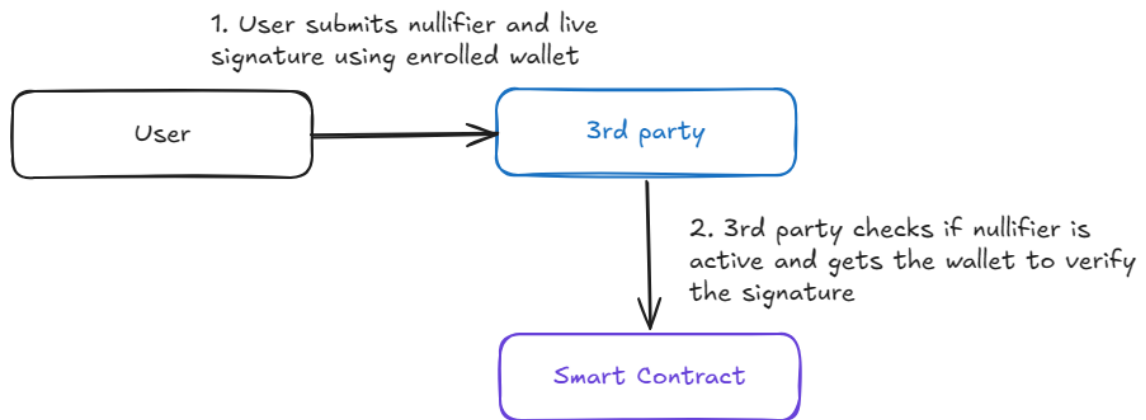


Figure 7.2: Diagram of the proposed protocol authorization phase.

The authorization phase is shown in Figure 7.2.

Revocation Phase

An enrolled user must be able to remove their identity from the registry. In the normal case, revocation is authorized by the wallet bound during enrollment. The user signs a revocation message for the enrolled nullifier, and the registry deletes the active record if the signature is valid.

If the bound wallet is lost, the same design can support a recovery path based on credential possession. A dedicated recovery proof would recompute the same hidden identity input and nullifier from the original credential, verify holder-key control, and authorize revocation or re-binding without revealing the credential contents. This recovery proof is a possible protocol extension rather than part of the minimal on-chain lifecycle.

Cleanup and Governance

A long-running smart contract registry needs a way to clean up old or invalid records. In this system, an active identity record is removed under two conditions: either the user's original credential has expired, or the issuer who signed the credential is no longer trusted by the system.

To keep the registry clean, the design uses a permissionless cleanup mechanism. This means anyone on the network can call a maintenance function. When this happens, the smart contract checks the records against the current rules. If a record has expired or is associated with an untrusted issuer key, the smart contract safely deletes it.

Governance is the process of managing these trust rules. The owner of the smart contract can add or remove trusted issuer public keys and update the accepted TACEO:OPRF public key.

7.6 Implemented Prototype

The implemented prototype proves that the proposed architecture can be built and works in practice. It consists of a web client, zero-knowledge circuits, a TACEO:OPRF testnet, and an Ethereum smart contract. It is important to note that this entire prototype is currently runnable only locally. This section explains the details of how these parts are built and connected together.

Technology Stack

The user interface is a web client built with React and TypeScript using Vite. For generating zero-knowledge proofs in the browser, it uses the Noir JavaScript library and the Barretenberg backend. To communicate with the OPRF network, it uses the TACEO:OPRF client and core libraries. To interact with the blockchain and manage the user's wallet, it uses the Wagmi and Viem libraries. Finally, it uses the Noble cryptography libraries for basic cryptographic math.

The zero-knowledge circuits are written in the Noir language. The smart contract is written in Solidity and tested with the Foundry framework. For local testing, the prototype runs an Anvil local Ethereum node and a local TACEO testnet network with three OPRF nodes.

Credential Implementation

The prototype uses the W3C Verifiable Credential 2.0 standard with a Data Integrity proof, as shown in Figure 7.3. This is important because it maintains the standard credential format while replacing the usual web signatures with cryptographic primitives that are easy to verify within a zero-knowledge circuit. The prototype uses the BabyJubJub elliptic curve for keys and signatures, and the Poseidon hash function. These credentials are mocks generated in the web client.

The credential has two main parts. The metadata fields (such as `issuer` and `validUntil`) indicate who issued it and when it expires. The `credentialSubject` fields belong to the user. They contain personal data (like name, birth date, and nationality) and the holder's public key. In the code, the system puts these fields into a Poseidon Merkle tree with 16 leaves. The issuer signs the root of this tree, and the signature is saved in the `proofValue` field.

Circuit Implementation

The prototype has two zero-knowledge circuits written in Noir:

```

{
  "@context": [
    "https://www.w3.org/ns/credentials/v2",
    "https://w3id.org/security/data-integrity/v2"
  ],
  "id": "urn:uuid:f656befd-ba14-4b17-a5e2-002c52bba66a",
  "type": [
    "VerifiableCredential",
    "UniquenessIdentityCredential"
  ],
  "issuer": {
    "id": "did:babyjubjub:zuikR1bMc5uJSR5GL2...",
    "name": "Example Authority"
  },
  "validFrom": "2026-03-28T15:44:36.409Z",
  "validUntil": "2031-03-28T15:44:36.409Z",
  "credentialSubject": {
    "id": "urn:person:n8e0onox7ot69wpdcv",
    "name": "Jan Novak",
    "dateOfBirth": "1990-06-15",
    "placeOfBirth": "Prague",
    "nationality": "Czech",
    "sex": "male",
    "holderPublicKey": {
      "type": "BabyJubJubPublicKey",
      "x": "1386023521535053718401658...",
      "y": "8524295514129179254289310..."
    }
  },
  "proof": {
    "type": "DataIntegrityProof",
    "cryptosuite": "eddsa-babyjubjub-poseidon-2024",
    "created": "2026-03-28T15:44:36.409Z",
    "verificationMethod": "did:babyjubjub:zuikR1bMc5uJS...Y3k#key-1",
    "proofPurpose": "assertionMethod",
    "proofValue": "zCH7dn7MJPCjLQMD9Y3DDWGzj..."
  }
}

```

Figure 7.3: Example of the Verifiable Credentials data model used by the implemented Proof-of-Uniqueness prototype.

- **vc_blinded_query_auth_proof:** This circuit checks the credential and the issuer's signature. It securely calculates the private ID-Hash and creates the blinded query for the OPRF network. It proves the user owns the credential.
- **vc_oprf_enrollment_proof:** This circuit checks the credential again. Then, it verifies that the OPRF network gave a correct mathematical answer. Finally, it binds the final nullifier to the user's selected blockchain wallet.

The enrollment circuit is compiled into a Solidity verifier contract. This verifier exposes five public signals to the smart contract: the OPRF public key, the expiry date, the issuer's public key, the wallet address, and the nullifier. The React client and the smart contract both use this exact same order of signals.

TACEO:OPRF Testnet Integration

The project includes a forked Rust implementation of the TACEO:OPRF testnet. To do this, their open source repository¹ was forked, extended with a custom authorization module, and runs only locally on a setup with three nodes. The TACEO:OPRF testnet is the reason why the implemented prototype runs only locally. Adding a custom module to the real TACEO:OPRF testnet requires official registration, which necessitates consultation and cooperation with TACEO.

Before a node computes an answer for the user, the custom module first checks the user's `vc_blinded_query_auth_proof`. If the proof is valid, the nodes send back their partial answers. The web client collects these answers, verifies them, unblinds the result, and obtains the final nullifier. This final result is then used to generate the enrollment circuit.

Smart Contract Implementation

The smart contract is named `IdentityRegistry`. It is deployed together with the Noir verifier contract. The contract saves the active users in a mapping called `identitiesByNullifier`. Every record saves the `validUntil` date, the issuer's key, the connected `walletAddress`, and an `exists` flag. The smart contract also has a `trustedIssuers` mapping to keep track of which issuers are allowed.

The contract has these main functions:

- **enroll:** Verifies the second zk-SNARK proof. It checks the wallet signature, ensures the issuer is trusted, checks the expiry date, and blocks duplicate nullifiers. If everything is correct, it saves the user.
- **revoke:** Deletes a user record if the connected wallet signs a valid revocation message.
- **getIdentity** and **isIdentityValid:** Read-only functions that let other apps check if a user is currently valid.
- **purgeInvalidRecords:** Scans the active records and deletes records that are expired or belong to a removed issuer.
- **Admin functions:** Functions like `addTrustedIssuer` and `setTrustedOprfPublicKey` let the contract owner update the system rules.

¹<https://github.com/TaceoLabs/oprf-testnet>

Client Flow

The React application brings all these pieces together into one user flow. It generates the user's credentials and prepares the data. It generates the initial authorization proof and contacts the local TACEO:OPRF nodes to obtain the nullifier. Then, it builds the second enrollment proof. It asks the user to sign a message with their standard wallet and sends the final transaction to the smart contract. After the user is enrolled, the client can also revoke the identity. The latter authorization phase for third-party services is not implemented in this prototype. This step simply relies on standard wallet challenge-response signatures. Because this is a very common practice, it does not add novel scientific value to the prototype and was therefore left out.

7.7 Summary

This chapter presents the complete design and implementation of the privacy-preserving Proof-of-Uniqueness system. The architecture successfully connects the trusted issuer, the user, the TACEO:OPRF network, and the smart contract registry.

The main achievement of this design is the strict separation of private identity data from the public. By using zk-friendly cryptography, two zero-knowledge proofs, and a threshold network, a user can safely register a unique identifier without exposing their personal attributes. Finally, the built prototype demonstrates that this complex architecture is practical and functions effectively using common tools such as Noir, React, and Solidity.

Chapter 8

Evaluation

This chapter presents the evaluation of the proposed Proof-of-Uniqueness (PoU) system. First, it defines the experimental environment and methodology. Second, it provides performance tests for client-side cryptography and smart contract gas usage. Finally, it evaluates whether the system meets the core requirements defined in Chapter 6.

8.1 Experimental Setup and Methodology

Before presenting the results, it is important to define the technical environment. This ensures the results are interpreted correctly.

Tooling and Tech Stack The system was tested using specific versions of the cryptographic stack. The zero-knowledge circuits were written in Noir (v0.36.0) and compiled using the Barretenberg backend (v0.36.0). The proving scheme used is UltraHonk.

The client-side environment runs on Node.js v25.5.0. The smart contracts were written in Solidity (0.8.13) and tested using the Foundry framework.

Benchmark Methodology

To measure the true cryptographic cost, the benchmarks isolate zero-knowledge operations from network delays:

- **Client-Side:** The tests use standard Node.js scripts. The Barretenberg backend is configured with a large Wasm memory limit (65536 pages) on a single thread. To ensure we only measure CPU performance, the network connection to OPRF nodes is bypassed by synthesizing the OPRF transcript locally.
- **Smart Contract:** Gas analysis was performed using Foundry’s local Anvil node. Because the generated verifier contract is very large (over 11,000 bytes), Anvil is configured with a code-size limit override. The test uses a real, mathematically valid proof fixture exported from the client benchmark to ensure the gas results are accurate for the Ethereum mainnet.

8.2 Experiment 1: Client-Side Cryptographic Performance

The speed of the client application mostly depends on Noir proof generation. Basic cryptographic steps are very fast: Poseidon hashing took 0.154 ms, BabyJubJub signing took 14.021 ms, and VC preprocessing took 1.305 ms.

The main bottleneck is the generation of zero-knowledge proofs. The results for the two core circuits are shown in Figure 8.1. The results confirm that while proving takes several seconds, verification is near-instant, which is the primary benefit of the zk-SNARK architecture.

Operation	Witness	Proving	Verification	Proof Size
OPRF authorization proof	619.57 ms	9.60 s	73.51 ms	2144 B
Enrollment proof	1.25 s	17.96 s	on-chain	2144 B

Figure 8.1: Client-side benchmark results for Noir/Barretenberg v0.36.0 proof generation and verification under Node.js v25.5.0.

8.3 Experiment 2: Smart Contract Gas Analysis

The smart contract evaluation focuses on the cost of the enrollment path. The results are summarized in Figure 8.2.

The full enrollment path costs 614,514 gas. The zero-knowledge verifier accounts for 376,502 gas, which is the largest cost component. Revocation is much cheaper because it uses a standard EIP-712 wallet signature instead of a zk-proof. The purge function uses a „swap-and-pop“ method to remove invalid records from the active list, preventing revoked or deleted records from incurring recurring costs in future scans.

Measurement	Gas / Size
Enrollment verifier deployment	2,489,530 gas
Registry deployment	1,535,752 gas
Enrollment verifier runtime size	11,053 bytes
Registry runtime size	6,479 bytes
VcOprfEnrollmentUltraVerifier.verify runtime	376,502 gas
IdentityRegistry.enroll (Full with real verifier)	614,514 gas
IdentityRegistry.revoke	42,162 gas
purgeInvalidRecords (active scan cost)	1,248 gas / record
purgeInvalidRecords (removal overhead)	4,276 gas / record

Figure 8.2: Foundry gas profiling results for the IdentityRegistry contract and generated Ultra verifier path.

8.4 System Requirements Evaluation

The implemented prototype satisfies the requirements with the following results:

1. **Deduplication:** The system stops duplicate users by storing a deterministic nullifier. Using the same ID-Hash and OPRF key always yields the same nullifier, which the contract rejects if it already exists.

2. **Authentication and Non-Transferability:** To enroll, the user must prove VC ownership and bind it to a wallet. While a user could technically sell their private keys, future integration with the EUDI wallet makes this unlikely, as selling such a wallet would compromise the user’s entire digital life.
3. **Recovery:** Users can delete their identity via wallet signature. If the wallet is lost, a zk-SNARK proof of VC possession allows the user to recompute their nullifier and reset their identity.
4. **Revocation:** Users can revoke their own record, and the system owner can remove untrusted issuers. The purge function then cleans invalid records.
5. **Expiry:** Enrollment enforces the `validUntil` field. Expired records can be purged from the registry permissionlessly.
6. **Inclusivity:** No special biometric hardware is needed. Any standard device with a browser can participate.
7. **Decentralization and Privacy:** The system is fully decentralized at the registry level. Private data is never stored on-chain or seen by TACEO nodes, ensuring strong user privacy.

8.5 Summary

The evaluation shows that the PoU design is practical. Although client-side proving takes around 18 seconds for enrollment, this is a one-time cost. The same applies to the on-chain enrollment cost of 614k gas, which is feasible for a one-time cost. The system successfully meets all core privacy and security requirements.

Chapter 9

Security Analysis

This chapter provides a security analysis of the Proof-of-Uniqueness design. It examines trust boundaries, Sybil resistance, privacy, zero-knowledge guarantees, replay attacks, and the cleanup of invalid records.

9.1 Threat Model and Trust Boundaries

To effectively analyze the system, we need to define the trust boundaries across five main parts: the Issuer, the Client Device, the TACEO:OPRF Network, the Network Transit, and the Smart Contract. The potential threats and their specific cryptographic defenses are outlined below:

- **The Issuer:** We assume the issuer honestly checks the real person before giving the credential. If an issuer's key is compromised, attackers could forge fake VCs. The defense against this is the on-chain `trustedIssuers` whitelist. The system owner can quickly remove the bad issuer, and anyone can then use the permissionless purge function to delete the invalid records.
- **The TACEO:OPRF Network:** We trust the network to be online and that the nodes will not work together maliciously to cheat. To prevent unauthorized use, the network requires VC ownership proofs and live holder signatures. Furthermore, the smart contract only accepts enrollments that use the anchored trusted TACEO:OPRF public key. If too many nodes collude, user privacy could be weakened, but the use of threshold Multi-Party Computation (MPC) makes this highly unlikely.
- **The Client Device:** The user's phone or browser is the weakest link, facing threats like identity hijacking and key leakage. To defend against this, the system strictly separates three things: the VC data, the holder private key, and the Ethereum wallet key. If an attacker manages to steal all of these at once, they can hijack the identity, but keeping them separate minimizes this risk.
- **Network Transit:** We assume attackers can see the internet traffic between the individual components. Because of this, the system uses OPRF blinding and zk-SNARKs, so that the real identity data is never sent over the internet. However, an attacker could still track the user's IP address when they send the final transaction to the smart contract or when they communicate with the OPRF nodes. This could link the user's real-world identity to their public nullifier. To fix this, users can use

standard network privacy tools like a VPN, Proxy, or Tor, as the protocol itself does not hide the network source.

- **The Smart Contract:** The smart contract is completely trustless and public, but it could face state manipulation or signature abuse. To defend against this, no secret data is ever sent to it. The contract uses strict zk-SNARK verifier checks, a nullifier mapping to prevent duplicates, and deadline-bound revocation signatures to ensure only correct updates are saved on the blockchain.

9.2 Sybil Attack Resistance and Privacy

The main goal of the system is to stop Sybil attacks without losing privacy. We do this by calculating a secret ID-Hash inside the zk-SNARK using the Poseidon hash function. This secret is never put on the blockchain.

Instead, the client only publishes the nullifier made by the OPRF network. This makes brute-force guessing practically impossible. If a user tries to register twice with the same private identity data and the same OPRF key, they get the exact same nullifier, and the smart contract blocks the second attempt.

Privacy also depends on the OPRF network. The smart contract binds accepted enrollments to a specific trusted OPRF public key. If the system owner changes that trusted OPRF key in the future, the same user would generate a different nullifier. Therefore, the system would need a migration policy to prevent users from accidentally registering twice after a key rotation.

9.3 Zero-Knowledge Guarantees

The security of the authentication and enrollment steps relies on the mathematical properties of the zk-SNARK system:

- **Completeness:** If an honest user has a valid credential, uses the right keys, and gets a valid OPRF response, the client will generate a valid proof, and the smart contract will accept it.
- **Soundness:** It is mathematically almost impossible to create a valid proof without having the real private data. The circuit forces the user to prove they followed all rules, including checking the issuer's signature and the OPRF response.
- **Zero-Knowledge:** The verifier only learns the public outputs. For enrollment, this is just the OPRF key, expiry date, issuer key, wallet address, and nullifier. The real credential contents stay completely hidden.

9.4 Replay Resistance and Permissionless Operations

A replay attack occurs when a hacker copies a valid message or transaction and resends it to cause damage. In this system, we must protect both the enrollment and revocation processes from these attacks while ensuring maintenance remains safe.

The first replay point is the TACEO:OPRF authorization step. If an attacker intercepts the authorization proof and the blinded query, they can at most resend the same request to

the OPRF nodes. This does not reveal the private identity input because the nodes only see the blinded query, and the blinding factor remains on the user’s device. The authorization proof also includes the request identifier and the blinded query as public outputs, so it cannot be reused for a different request or for a different blinded input. If the attacker also intercepts the OPRF response or the final nullifier, this still does not allow them to create their own identity as long as they do not discover the blinding factor.

For the enrollment phase, if a hacker intercepts the final zk-SNARK proof package before it reaches the blockchain, they cannot steal the identity. This is because the zero-knowledge proof is mathematically bound to the specific wallet address chosen by the user. If the attacker submits the intercepted proof, the smart contract will simply register the identity to the original user’s wallet. The attacker gains absolutely nothing and only wastes their own money paying for the gas fees.

During the revocation phase, the prototype uses a standard wallet signature rather than a complex zk-SNARK. To stop replays, the wallet signs a message containing two things:

1. **nullifier**: This ties the revocation strictly to one specific identity record.
2. **deadline**: This limits how long the signature is valid.

The smart contract checks this signature. This prevents replay attacks across users because the signature only works for that specific nullifier. It also prevents endless replays due to the deadline. If a hacker intercepts the signature and sends it again before the deadline, it just tries to delete a record that has already been deleted. The contract rejects it, so the replay attack cannot do any damage.

Finally, we consider the security of the purging function. This function is permissionless, meaning anyone can call it. It does not create security risks because the smart contract strictly enforces that it deletes records only when they are already invalid (e.g., because they have expired or the issuer has been removed). An attacker cannot use this function to delete good, active users.

Chapter 10

Discussion & Limitations

This chapter discusses the practical limitations of the proposed Proof-of-Uniqueness system. The prototype successfully achieves privacy and deduplication. However, several challenges remain before it can be used in a real, large-scale production environment. The main problems are OPRF trust rules, finding and governing trusted issuers, managing private keys, and scaling blockchain storage.

10.1 OPRF Trust Assumptions and Key Rotation

Using the threshold OPRF network solves the privacy problem without needing special hardware. However, it entails trade-offs in terms of trust and key management.

First, the privacy depends on the OPRF nodes not working together to cheat. If too many nodes collude, the privacy of the blinded queries could be weakened. However, in reality, this is highly unlikely to happen. As long as the network is run by a few independent operators, the risk of them secretly collaborating to breach user privacy is very low.

Second, deduplication depends on the trusted OPRF public key. If the system owner changes this key in the future, the same user will generate a different nullifier. This is okay for a prototype, but a real system needs a good migration plan so users do not accidentally register twice if a key rotation occurs.

10.2 Wallet-Bound Authorization, Recovery, and Key Management

The current design cryptographically binds enrollment and normal authorization to an external Ethereum wallet. The enrollment proof includes the wallet address as a public output and proves the holder's authorization for that exact wallet, whereas later third-party use relies on challenge signatures from the same wallet. This prevents the reuse of a stolen enrollment proof with a different address and gives verifiers a simple way to check the live control of the enrolled identity.

This design also places a real key-management burden on the user. The VC data, the holder private key, and the Ethereum wallet key are separated for privacy and security, but all three must be protected carefully. If an attacker obtains the credentials and the relevant private keys, they can take over the enrolled identity. A production system, therefore, needs a strong wallet backup, passkey or hardware-backed storage, multi-device recovery, or issuer-assisted credential re-issuance.

Recovery is not limited to the wallet alone. If the wallet is still available, the user can revoke it directly with a wallet signature. If the wallet is lost, a zk-SNARK proof of VC possession can recover authority from the credential itself by proving that the user holds the original issuer-signed VC and controls the holder’s private key embedded in it. However, this recovery path still depends on the user preserving access to the credential and holder key.

10.3 Trusted Issuer Governance and Real-World Credential Infrastructure

A core structural limitation of the current prototype is its reliance on a whitelist of trusted issuers. The smart contract registry explicitly accepts identities only if their issuer public key is present in the trusted issuers mapping. Finding an issuer that can reliably verify real people, issue privacy-preserving credentials, and remain accountable over time is one of the main deployment challenges.

While necessary for bootstrapping, this model remains a central focus. A malicious or compromised administrator could add a fraudulent issuer, thereby weakening the PoU guarantees. The current multi-owner model improves operational flexibility but does not fundamentally decentralize trust. A stronger long-term governance model would require stronger multi-party governance, decentralized voting over issuer inclusion, or a higher-level legal and institutional framework that constrains who may issue accepted credentials.

A particularly promising direction is integration with standardized government-backed verifiable credential infrastructures, such as the European Digital Identity (EUDI) Wallet ecosystem. Such infrastructures could provide real issuer environments, stronger legal accountability, and more interoperable credential formats than the idealized issuer model used in this thesis. If these ecosystems expose privacy-preserving W3C credentials and compatible verification workflows, the PoU architecture proposed in this thesis could evolve from a controlled prototype into a registry layer backed by recognized public-sector identity issuers rather than purely hypothetical trust anchors.

10.4 Ethereum State Bloat and Maintenance Scalability

The prototype uses a direct on-chain mapping (`mapping(uint256 => IdentityRecord)`) to store registered identities. As shown in Chapter 8, this approach keeps enrollment and lookup logic straightforward and efficient at the current scale.

However, writing every identity record to the Ethereum Layer-1 state is inherently difficult to scale to very large populations. If the protocol were to onboard millions of users, storing nullifier-indexed records directly would increase state growth and network operating costs. The current purge mechanism is better than a full-history scan because removed records are deleted from the active list, but it is still linear in the number of active records.

The alternative off-chain commitment-tree architecture discussed in Chapter 6 reduces direct state bloat but introduces substantial operational complexity for data availability and lifecycle management. Perhaps this could be researched further. A more practical near-term direction could also be to move the registry to a Layer-2 rollup while also introducing incremental or batched maintenance strategies for expired or de-trusted records. Lastly,

exploring an external storage or indexing service for the blockchain could be a feasible option.

Chapter 11

Conclusion

The goal of this thesis was to design, build, and test a privacy-preserving Proof-of-Uniqueness system. This system is needed to stop Sybil attacks in protocols like Proof-of-Social-Capital. By connecting off-chain Verifiable Credentials to on-chain Ethereum smart contracts using zero-knowledge proofs and the TACEO:OPRF network, this work demonstrates that we can enforce uniqueness without exposing private user data.

The final architecture uses two separate zk-SNARK proofs and a minimal smart contract registry. The prototype uses a zk-friendly credential format with BabyJubJub curves and Poseidon hashing. The user generates proofs directly in their browser using Noir, communicates with the TACEO threshold network to obtain a public nullifier, and then registers the nullifier on the blockchain.

The evaluation proves that the design works in practice while also highlighting the main deployment limitations. The system still relies on a centralized list of trusted issuers and a stable OPRF public key. It also places responsibility on the user to protect the VC, the holder private key, and the connected wallet key. Finally, storing every user record directly on the blockchain will be hard to scale for millions of people.

Overall, this thesis successfully provides a working prototype that solves the main identity challenge for the PoSC vision. It proves that we can stop duplicate accounts while keeping real-world identity data completely private. For future real-world deployment, the system will need improved storage scaling, stronger private-key recovery and protection, decentralized governance for trusted issuers, and integration with real government digital identity infrastructures.

Bibliography

- [1] ALTMAN, S.; NOVENDSTERN, M. and BLANIA, A. *Building the Real Human Network* online. 2023. Available at: <https://whitepaper.world.org/>. [cit. 2025-12-18].
- [2] BUTERIN, V. *Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform* online. 2014. Available at: https://ethereum.org/content/whitepaper/whitepaper-pdf/Ethereum_Whitepaper_-_Buterin_2014.pdf. [cit. 2025-11-13].
- [3] BUTERIN, V. *Quadratic Arithmetic Programs: From Zero to Hero* online. 2016. Available at: <https://medium.com/@VitalikButerin/quadratic-arithmetic-programs-from-zero-to-hero-f6d558cea649>. [cit. 2026-01-04].
- [4] BUTERIN, V. *An Approximate Introduction to How zk-SNARKs Are Possible* online. 2021. Available at: <https://vitalik.eth.limo/general/2021/01/26/snarks.html>. [cit. 2026-01-04].
- [5] CHAINLINK. *What Is a Zero-Knowledge Proof?* online. 2024. Available at: <https://chain.link/education/zero-knowledge-proof-zkp>. [cit. 2026-01-04].
- [6] CHATTOPADHYAY, A.; SAHA, S.; NAG, A. and NANDI, S. Secret Sharing: A Comprehensive Survey, Taxonomy and Applications. *Computer Science Review*. Volume 51th ed., 2024, vol. 51, no. 51, p. 100608. Available at: <https://doi.org/10.1016/j.cosrev.2023.100608>.
- [7] COMPANY, E. C. *How To Generate SNARK Parameters Securely* online. 2016. Available at: <https://electriccoin.co/blog/snark-parameters/>. [cit. 2026-01-05].
- [8] CURA, J. *Self Sovereign Identity (SSI)* online. 2024. Available at: <https://extrimian.io/wikis/self-sovereign-identity-ssi/>. [cit. 2025-12-17].
- [9] DAVIDSON, A.; FAZ HERNANDEZ, A.; SULLIVAN, N. and WOOD, C. A. *Oblivious Pseudorandom Functions (OPRFs) Using Prime-Order Groups*. RFC 9497. RFC Editor, 2023. Available at: <https://www.rfc-editor.org/info/rfc9497>.
- [10] ELFORS, S.; ALTMANN, P. L. and SEDLMEIR, J. *Electronic Signatures and Trust Infrastructures (ESI); Selective Disclosure and Zero-Knowledge Proofs Applied to Electronic Attestation of Attributes; Part 1: Feasibility Study*. ETSI Technical Report ETSI TR 119 476-1 V1.3.1. European Telecommunications Standards Institute, 2025. Available at: https://www.etsi.org/deliver/etsi_tr/119400_119499/11947601/01.03.01_60/tr_11947601v010301p.pdf. [cit. 2026-02-22].

- [11] FOUNDATION, E. *Ethereum Development Documentation*. Ethereum Foundation, 2025. Available at: <https://ethereum.org/developers/docs>. [cit. 2025-11-13].
- [12] GABIZON, A.; WILLIAMSON, Z. J. and CIOBOTARU, O. *PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge* Cryptology ePrint Archive, Paper 2019/953. 2019. Available at: <https://eprint.iacr.org/2019/953>.
- [13] GALASSI, D. *Journey to Zero-Knowledge* online. 2023. Available at: <https://datawok.net/companions/zk-proofs/journey-to-zero-knowledge.pdf>. [cit. 2026-01-04].
- [14] GRASSI, L.; KHOVRATOVICH, D.; RECHBERGER, C.; ROY, A. and SCHOFNEGGER, M. Poseidon: A New Hash Function for Zero-Knowledge Proof Systems. In: USENIX Association. *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 2021, p. 519–535. ISBN 978-1-939133-24-3. Available at: <https://www.usenix.org/conference/usenixsecurity21/presentation/grassi>.
- [15] GROTH, J. On the Size of Pairing-based Non-interactive Arguments. In: FISCHLIN, M. and CORON, J.-S., ed. *Advances in Cryptology – EUROCRYPT 2016*. Springer Berlin Heidelberg, 2016, vol. 9666, p. 305–326. Lecture Notes in Computer Science. ISBN 978-3-662-49895-8.
- [16] GROUP, W. D. I. W. *Decentralized Identifiers (DIDs) v1.1*. W3C Candidate Recommendation Snapshot. World Wide Web Consortium, 2026. Available at: <https://www.w3.org/TR/did-1.1/>. [cit. 2025-12-19].
- [17] GROUP, W. V. C. W. *Verifiable Credentials Data Model v2.0*. W3C Recommendation. World Wide Web Consortium, 2025. Available at: <https://www.w3.org/TR/vc-data-model-2.0/>. [cit. 2025-12-19].
- [18] HU, B.; ZHANG, Z.; LIU, J.; LIU, Y.; YIN, J. et al. A Comprehensive Survey on Smart Contract Construction and Execution: Paradigms, Tools, and Systems. *Patterns*. Volume 2th ed., 2021, vol. 2, no. 2, p. 100179. Available at: <https://doi.org/10.1016/j.patter.2020.100179>.
- [19] LABS, A. *Barretenberg Documentation*. Aztec Labs, 2026. Available at: <https://barretenberg.aztec.network/docs/>.
- [20] LABS, D. *Centralized ID, Federated ID, Decentralized ID: What’s the Difference?* online. 2025. Available at: <https://www.dock.io/post/centralized-id-federated-id-decentralized-id-whats-the-difference>. [cit. 2025-12-17].
- [21] LABS, D. *Self-Sovereign Identity: The Ultimate Guide 2025* online. 2025. Available at: <https://www.dock.io/post/self-sovereign-identity>. [cit. 2025-12-17].
- [22] LABS, S. *Self Protocol*. Self Labs, 2026. Available at: <https://docs.self.xyz/>. [cit. 2025-12-18].
- [23] LEE, T. *ELI5: How Does a Merkle-Patricia-Trie Tree Work?* online. 2016. Available at: <https://ethereum.stackexchange.com/questions/6415/eli5-how-does-a-merkle-patricia-trie-tree-work>. [cit. 2025-12-15].

- [24] MARIANI, J. and HOMOLIAK, I. *Proof-of-Social-Capital: A Consensus Protocol Replacing Stake for Social Capital* arXiv preprint. 2025. Available at: <https://arxiv.org/abs/2505.12144>. [cit. 2026-01-08].
- [25] MUNOZ TAPIA, J. L.; BELLES, M.; ISABEL, M.; RUBIO, A. and BAYLINA, J. *CIRCOM: A Robust and Scalable Language for Building Complex Zero-Knowledge Circuits* TechRxiv preprint. 2022. Available at: <https://doi.org/10.36227/techrxiv.19374986.v1>.
- [26] NAINWAL, A.; KAMBLE, A. and AWATHARE, N. *A Comparative Analysis of zk-SNARKs and zk-STARKs: Theory and Practice* arXiv preprint. 2025. Available at: <https://arxiv.org/abs/2512.10020>. [cit. 2026-01-05].
- [27] NAKAMOTO, S. *Bitcoin: A Peer-to-Peer Electronic Cash System* online. 2008. Available at: <https://bitcoin.org/bitcoin.pdf>. [cit. 2025-11-11].
- [28] OHLHAVER, P.; WEYL, E. G. and BUTERIN, V. *Decentralized Society: Finding Web3's Soul* SSRN Electronic Journal. 2022. Available at: <https://doi.org/10.2139/ssrn.4105763>. [cit. 2025-12-18].
- [29] ORGANIZATION, I. C. A. *EPassport Basics* online. 2026. Available at: <https://www.icao.int/icao-pkd/epassport-basics>. [cit. 2026-05-06].
- [30] REIFF, N. *Bitcoin vs. Ethereum: What's the Difference?* online. 2025. Available at: <https://www.investopedia.com/articles/investing/031416/bitcoin-vs-ethereum-driven-different-purposes.asp>. [cit. 2025-11-13].
- [31] ROELINK, B. O.; EL HAJJ, M. and SARMAH, D. K. Systematic Review: Comparing zk-SNARK, zk-STARK, and Bulletproof Protocols for Privacy-Preserving Authentication. *Security and Privacy*. Volume 7th ed., 2024, vol. 7, no. 5, p. e401. Available at: <https://doi.org/10.1002/spy2.401>.
- [32] ROSENBERG, M.; WHITE, J.; GARMAN, C. and MIERS, I. Zk-creds: Flexible Anonymous Credentials from zkSNARKs and Existing Identity Infrastructure. In: IEEE. *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2023, p. 1882–1900. ISBN 979-8-3503-9330-8. Available at: <https://eprint.iacr.org/2022/878>.
- [33] TEAM, S. *Solidity Documentation*. Solidity Team, 2025. Available at: <https://docs.soliditylang.org/en/v0.8.33/>. [cit. 2025-12-15].
- [34] TEAM, T. N. *Noir Documentation*. The Noir Team, 2026. Available at: <https://noir-lang.org/docs/>.
- [35] WALCH, R. and GAUCHER, Q. *Introducing TACEO:OPRF* online. 2026. Available at: <https://core.taceo.io/articles/taceo-oprf/>.
- [36] WALKER, G. *Block* online. 2024. Available at: <https://learnmeabitcoin.com/technical/block/>. [cit. 2025-12-11].
- [37] WHITEHAT, B.; BELLES, M. and BAYLINA, J. *ERC-2494: Baby Jubjub Elliptic Curve*. ERC 2494. Ethereum Improvement Proposals, 2020. Available at: <https://eips.ethereum.org/EIPS/eip-2494>.

- [38] WILLIAMSON, Z. J.; GABIZON, A. and AZTEC LABS. *The Honk Proving System: Protocol Specification*. Technical Report. Aztec Protocol, 2023. Available at: <https://docs.aztec.network/aztec/cryptography/honk>.
- [39] WOOD, G. *Ethereum: A Secure Decentralised Generalised Transaction Ledger*. Ethereum Foundation, 2014. Available at: <https://ethereum.github.io/yellowpaper/paper.pdf>. [cit. 2025-11-13].

Appendix A

Contents of Electronic Appendix

- **src/** - source code of the implemented prototype
 - **circuits/** - Noir circuits and generated verifier artifacts
 - **client/** - React web client; client-side benchmark scripts are in **client/benchmarks/**
 - **smart-contracts/** - Foundry project with the **IdentityRegistry** contract; gas and scaling tests are in **smart-contracts/test/**
 - **oprftestnet/** - local TACEO:OPRF testnet fork used by the prototype; the custom authentication module is in **oprftestnet-authentication/**
- **README.md** - main setup guide and project documentation
- **Makefile** - helper commands for running the local prototype
- **docs/** - the source code that generates this PDF file, including the Makefile.
 - **xvozdaa00.pdf** - the pdf file of this thesis

The **src/oprftestnet/** directory is not originally my project. It is an extended version of the TACEO:OPRF testnet open source implementation ¹. The list of added files is in **src/oprftestnet/README.md**.

¹<https://github.com/TaceoLabs/oprftestnet>