A Privacy-Preserving and Transparent Certification System for Digital Credentials

Rodrigo Q. Saramago
University of Stavanger, Norway

Hein Meling
University of Stavanger, Norway

Leander N. Jehl
University of Stavanger, Norway

Abstract

A certification system is responsible for issuing digital credentials, which attest claims about a subject, e.g., an academic diploma. Such credentials are valuable for individuals and society, and widespread adoption requires a trusted certification system. Trust can be gained by being transparent when issuing and verifying digital credentials. However, there is a fundamental tradeoff between privacy and transparency. For instance, admitting a student to an academic program must preserve the student's privacy, i.e., the student's grades must not be revealed to unauthorized parties. At the same time, other applicants may demand transparency to ensure fairness in the admission process. Thus, building a certification system with the right balance between privacy and transparency is challenging.

This paper proposes a novel design for a certification system that provides sufficient transparency and preserves privacy through selective disclosure of claims such that authorized parties can verify them. Moreover, unauthorized parties can also verify the correctness of the certification process without compromising privacy. We achieve this using an incremental Merkle tree of cryptographic commitments to users' credentials. The commitments are added to the tree based on verifying zero-knowledge issuance proofs. Users store credentials off-chain and can prove the ownership and authenticity of credentials without revealing their commitments. Further, our approach enables users to prove statements about the credential's claims in zero-knowledge. Our design offers a cost-efficient solution, reducing the amount of linkable on-chain data by up to 79 % per credential compared to prior work, while maintaining transparency.

2012 ACM Subject Classification Security and privacy → Privacy-preserving protocols; Security and privacy → Pseudonymity, anonymity and untraceability; Information systems → Extraction, transformation and loading

Keywords and phrases verifiable credentials, privacy-preserving, zero-knowledge, blockchain

Digital Object Identifier 10.4230/LIPIcs.OPODIS.2022.9

Supplementary Material *Software (Source Code)*: `https://github.com/r0qs/zkcertree` archived at `swh:1:dir:491b29c170a47c1c5697bfe504075848d19a27bf`

Funding This work is partially funded by the BBChain and Credence projects under grants 274451 and 288126 from the Research Council of Norway.

1 Introduction

Smart contract-based issuing of digital credentials can increase transparency and thus help to detect fraud. This is especially important for academic credentials, which should be the result of a long learning and evaluation process. Fraud has been shown within educational institutions [13], but more importantly, fraudulent organizations, known as degree mills which give out credentials with no or dubious processes [2, 15]. Transparency standards of the evaluation and issuance process could significantly simplify the detection and blacklisting

of such organizations. Further, unlike purely signature-based approaches, blockchain-based credentials allow long-term validity and revocation. However, in existing solutions [9, 29, 36], transparency of the issuance process is achieved only at the cost of users' privacy.

Saramago et al. [29] propose to issue credentials for individual certification steps, such as courses completed towards a degree. Their protocol allows authenticity checks of these individual credentials through metadata logged to a blockchain. Presenting individual certification steps provides transparency to the issuance process and enables verification of the issuance process, e.g., the time frame of the credentials' creation. Such properties guarantee that the issued credentials represent achievements acquired over a specific trusted time frame. Additionally, the metadata logged on-chain gives transparency of the issuers' overall processes, e.g., how many credentials are created during which periods. This transparency of the overall issuing process is useful for accreditation purposes and simplifies the detection of degree mills. They present an implementation of the above process using smart contracts running in an EVM-compatible blockchain. However, this implementation allows one to track all users' credentials and related activities based on the on-chain data. This is possible since the user's address (hash of its public key) is contained in the metadata for all its credentials. In the case of academic diplomas, this traceability may, for example, expose a user's failed or unfinished courses, even though irrelevant for their final certification.

This paper presents zkCert, a novel certification system design that allows transparency for an issuer's process while maintaining the users' privacy. zkCert enables users to commit to a set of credentials issued to them and prove the ownership and authenticity of such credentials in zero knowledge. Moreover, users can disclose a partial set of credentials to authorized parties for verification. Our approach reduces the amount of traceable information published on-chain while preserving the transparent and accountable certification process.

Besides the privacy and scalability concerns, a key challenge with implementing a certification system on an EVM-based platform is the high deployment, transaction, and storage costs. This demands that contracts are carefully designed to manage their resource use. Specifically, a contract must be deployed sparingly, it must utilize its storage efficiently, and transactions must be carefully managed.

We have implemented zkCert as a smart contract, called Notary, that acts on behalf of an issuer. Additionally, we implemented several applications based on zkSNARK circuits in the Circom [21] language to enable users to proof statements about their Notary issued credentials in zero-knowledge.

Our Notary contract allows educational institutions to automate a large portion of their issuing processes of digital credentials. National authorization agencies responsible for institutions can conduct audits without violating users' privacy. Users can apply to study programs or jobs by selectively disclosing only relevant credentials to the educational institution or employer, which would be designated as verifiers. Hence, a verifier could rank candidates based on their grades without actually knowing the grades. These are just some applications that zkCert can support via zkSNARK circuits and Notary issued credentials.

Our contributions include: (1) zkCert, a novel certification system design, combining an incremental Merkle tree for scalability with zkSNARKs for privacy, (2) a Notary smart contract implementation of zkCert, (3) a set of circuits demonstrating the capabilities of Notary issued credentials, and (4) our evaluation shows that zkCert is cost-efficient, scales well, and outperforms prior work despite stronger privacy guarantees.

2 Background

Credentials are certificates that attest a statement about a subject. The W3C data model defines a verifiable credential as a digital representation of a physical certificate in a cryptographically secure and machine-verifiable form [12]. For example, an academic diploma is a credential attesting the proficiency of a student and the authenticity of its digital representation can be verified through the use of digital signatures.

Compared to paper-based certificates, digital certificates offer a more secure and scalable alternative, being easy to create, revoke and verify, and further enabling a certification system that is less prone to human errors. The use of digital signatures, however, requires that a verifier can check if specific cryptographic keys belong to issuing authorities and individuals [10]. The security of the system relies on keeping the signing keys secret and compromised keys of issuing authorities may allow arbitrary creation of authentic credentials.

2.1 Blockchain Data Registries

Certificate authorities have a long history of security and privacy issues affecting trust in such systems, limiting mass adoption [18, 22]. On the other hand, blockchain technologies provide a decentralized and transparent database infrastructure that can improve how certificates are issued and managed.

Data written on the blockchain is timestamped and tamper-resistant, forming a secure, consistent, and append-only log of transactions shared across peers in the blockchain network. Further, some blockchains can also execute code as part of a state transition, i.e., smart contracts. However, this level of transparency, where anyone can verify transactions, limits its use for applications that require a higher degree of privacy. For instance, popular blockchain technologies commonly offer pseudo-anonymity, where entities are represented by pseudonyms corresponding to public keys. Thus, naively using built-in transaction methods leaves a trail of all entities' activities. Consequently, the knowledge of identities with whom the pseudonyms are associated puts privacy at risk.

Hence, a fundamental tension exists between transparency and privacy in a blockchain. However, zero-knowledge proofs applied to blockchain state transitions [8, 20] offer a promising solution to balance this tradeoff.

2.2 zkSNARKs

Zero-knowledge succinct, non-interactive arguments of knowledge (zkSNARK) [5] is a cryptographic proof construction where one can prove the knowledge of a secret while only revealing its validity and no other information. The proof satisfies an NP relation (nondeterministic polynomial time), and some zkSNARKs can prove any relation within a bounded-size arithmetic circuit [6, 24]. Anyone can verify a proof, and the proof length and the verification time are sublinear in the circuit size. However, proof generation is expensive; typically, orders of magnitude slower than checking the relation directly [16, 24].

To prove a program's computation using zkSNARKs, the program must be expressed as a set of quadratic constraints. Thus, in simple terms, proving a computation requires converting the program to a polynomial and evaluating it for a set of inputs. The first step is transforming a program from a domain-specific language to an arithmetic circuit [21].

An arithmetic circuit is a directed acyclic graph over a finite prime field such that the vertices are called gates, and the edges are called wires [6]. Wires can represent inputs or outputs. Input wires can be public or private, and outputs are always public. Private inputs

compose the witness, or the secret data, that must not be revealed to ensure zero knowledge. Gates take two inputs and perform multiplication or addition in the field [24]. Arithmetic circuits are closely related to circuits of logical gates but with arithmetic gates instead.

The circuit is then converted into a system of equations that express the arithmetic circuit satisfiability, i.e., a constraint system. The rank-1 constraint system (R1CS) is one such system and is used in many state-of-the-art zero-knowledge-proof systems [24]. However, since the number of constraints can be huge and, thus, inefficient for real programs, the R1CS representation is usually encoded as a single polynomial in the form of a Quadratic Arithmetic Program (QAP) [19]. QAP is a way to efficiently compute arithmetic programs over large finite fields using polynomials to represent the circuit constraints rather than numbers.

To be non-interactive, zero-knowledge protocols usually require a *trusted setup*. A trusted setup defines the parameters used to construct and verify proofs. These parameters are a set of random numbers that generate the proving and verifying keys in the system. Thus, if a single party knows all the parameters, that party could produce fraudulent proofs. One approach to avoid this problem is establishing a distributed ceremony with multiple unknown parties that derive their own private random data and collaborate through a Multi-Party Computation protocol to combine their shares for the final set of parameters [24]. As a result, the setup can be trusted, assuming at least one of the participating parties deletes their private data, as all parties would need to collude to put the construction at risk.

Some zkSNARK constructions require a trusted setup for each new program, i.e., the setup requires application-specific circuits [3]. However, recent works introduced the concept of *universal setup*. PLONK [16] is one such construction that allows multiple programs to reuse a single trusted setup within a bounded circuit size.

2.3 Incremental Merkle Tree

A Merkle tree is a (binary) tree data structure where leaf nodes are labeled with the hash of the data, and non-leaf nodes are the result of the hash of their children [23]. Merkle trees provide an efficient way to verify the membership of a leaf without requiring the verifier to know all the leaves by using *Merkle proofs*.

A Merkle proof consists of the leaf data to be proven, the tree's root hash, and a branch consisting of all siblings of the nodes in the path from the data to the root. For instance, suppose that a large database is stored as a Merkle tree and that the root is publicly known and trusted, e.g., digitally signed. A verifier can perform a lookup in such a database by verifying a Merkle proof of the data without requiring access to the whole database. Thus, Merkle trees can be used as tamper-resistant and authenticated data structures, allowing public audits.

However, naively updating a Merkle tree, e.g., adding a leaf, incur recomputing all intermediary nodes up to the new root. Since the size of Merkle trees can be huge, it is often impractical to reconstruct the whole tree every time a new leaf is added.

An *incremental Merkle tree* is an optimization that reduces time and space complexity by initializing the tree with empty nodes (filled with zeros) and keeping a partial Merkle tree for data updates [25]. A leaf node in such a tree is incrementally added (from left to right), updating the partial Merkle tree by replacing an empty node. Therefore, only the hashes of the nodes on the path from the new leaf to the root need to be recomputed.

2.4 Cryptographic Commitments

Cryptographic commitments allow one to commit to a value while keeping it secret from others and subsequently making the commitment public. Once committed, the scheme ensures that the value was not changed before disclosure [26].

Commitment schemes have two main properties, namely hiding and binding. Hiding ensures that no one can learn the value until it is revealed, even with access to the commitment, e.g., the hash of a preimage. When a commitment is opened, e.g., the preimage is revealed, the binding assures that it is infeasible to replace the value with something else.

Poseidon hash [17] is an elliptic curve hashing algorithm that compresses values into points on the curve, which can be produced and verified efficiently within arithmetic circuits, requiring fewer constraints per round. Poseidon is tailored to work on finite fields used in zero-knowledge proof systems and is currently one of the most efficient hashing functions for zero-knowledge applications. Poseidon can also be used for signature and commitment schemes where the knowledge of the committed value is proven in zero knowledge and is faster than previous methods like Pedersen commitments [17, 26].

2.5 Related Works

Semaphore [33] is a zero-knowledge protocol that allows users to vote by casting a signal, and only one, as a member of some group of entities without revealing their identity. Users in their system can participate in anonymous polls and Decentralized Autonomous Organizations (DAOs) and prove their participation on-chain. Their system provides a high level of privacy but was designed for a different use case, i.e., private voting, despite sharing similarities with the commitment-reveal scheme used in our work.

Tornado Cash [34] is another application that makes use of zero-knowledge proofs to provide anonymous payments through a mixing service. Users of tornado cash make deposits by committing to some amount of a cryptocurrency, e.g., ETH, and can later withdraw the deposited amount by providing a zkSNARK proof of the commitment within a Merkle tree and a nullifier. The nullifier is revealed in the withdrawal and used to prevent double-spending on a commitment. Tornado cash enables anonymity by grouping all deposits to the tornado contract in an *anonymity set*. The smaller the anonymity set, the worst the privacy of the system since it becomes easy to analyze the on-chain data and infer relations between deposits and withdraws. zkCert was also inspired by Tornado Cash adapting their solution to the verifiable credentials scenario.

Iden3 [31] is a Self-Sovereign Identity (SSI) platform that allows users to provide zero-knowledge proofs of membership and non-membership of claims on-chain. Their solution for verifiable credentials also allows on-chain revocations. In their system, identities sign the root of a claim tree of other identity creating a chain of signed trees that resembles web of trust. Each identity act as an issuer and claims are cryptographically proved and verified.

Certree [29] is a certification protocol that provides a transparent log of the issuer's actions by establishing an on-chain tree data structure of smart contracts and credentials. Each credential hash is stored on the blockchain as leaves of the tree-like structure and is identified by the subject's unique address. Their tree of smart contracts models the issuer's certification processes and defines authorization rules and scopes of each credential created by the issuer. However, all operations and identifiable metadata of credentials can be linked to the subject's address. This is a consequence of the proposed progressive construction of their credential tree. We propose zkCert, providing the same level of transparency as Certree, but significantly improve privacy due to our use of zero-knowledge proofs. zkCert

uses PLONK and provides a progressive certification process in the form of an incremental Merkle tree, enabling improved on-chain storage utilization and lower monetary costs to produce credentials.

3 System Model

We aim to design a certification system that delivers strong privacy guarantees to subjects while providing transparency of the entire process. As such, we assume a certification ecosystem consisting of four main parties defined by the W3C verifiable credentials data model [12]: issuers, subjects, verifiers, and a verifiable data registry. For convenience, we assume the subject to be the same as the holder defined by the W3C specification.

The purpose of a certification system is to issue credentials and provide sufficient information to third-party entities to verify the authenticity of credentials. A credential is often an official document that attests to a particular fact. In our model, a credential is a digital document that stores data as a claim, i.e., a statement about a subject.

Subjects, in a nutshell, are entities about which claims are made. Issuers are responsible for executing the certification process and asserting claims about subjects in the form of credentials. They are typically represented by a group of individuals that act on behalf of the issuer organization. A verifier is any entity that evaluates whether credential claims are authentic and meet specific criteria. Last but not least, a verifiable data registry facilitates the creation and validation of keys, identifiers, and other pertinent data [12].

3.1 System Properties

The goal of this work is to provide a certification system with the following properties.

Transparency. It should be able to demonstrate the process by which the issuer used to arrive at the conclusions stated in the claims [18].

Privacy. It should minimize exposure of linkable information to preserve users' privacy.

Integrity. It should prevent manipulation of a credential's content, once issued.

Trusted Timestamping. It should prevent manipulation of an issued credential's issuing time.

Authenticity. Is should allow anyone to verify the authenticity of a presented credential.

Data and Service Availability. Subjects should have ownership of their credentials' data and should not depend on the issuer for verification. Credentials should remain valid and be verifiable even though the issuer becomes unavailable, e.g., goes out of business.

Revocability. Issuers and subjects should be able to revoke previously issued credentials, and anyone should be able to verify the change in a credential's status.

Agreement. Subjects can receive credentials from any issuer and must be able to accept or deny the issuance of a credential to him.

In Section 4.4 we briefly discuss how our system implements those properties.

3.2 System Assumptions

We assume that one or more cryptographic key pairs represent the entities in the system and that cryptographic primitives cannot be easily circumvented. System entities should be able to exchange sensitive data over a secure communication channel.

We assume that issuers can confirm the identity of subjects and that verifiers can identify issuers using mechanisms like decentralized identifiers [11]. Thus, we consider the existence of a registry for verifying the system entities' public keys, e.g., DPKI [1].

Moreover, we assume the existence of a blockchain-based verifiable data registry, with support for smart contracts, that gives write access to issuers and subjects and read-only access to verifiers. Additionally, the entities in the system are not able to trivially bypass the blockchain's security, timestamping, safety, and append-only properties.

3.3 Threat Model

Our system contains issuers, subjects, and verifiers, each of which may attack the system in different ways. A corrupt issuer may try to create or gain control over credentials with a correct timeframe. Creating credentials with a correct timeframe may take several years. Corrupted issuers may therefore attempt to alter or gain control over previously issued credentials. A corrupt subject may try to receive (from the issuer) or present (to a verifier) false credentials. The subject may also try to prevent revocation or falsely present revoked credentials as valid. Finally, a corrupt verifier may attempt to extract additional information about a subject's credentials from public information.

4 Privacy-Preserving Credentials

In this section, we describe our protocol, zkCert, for a privacy-preserving certification system. zkCert builds on a blockchain that supports smart contracts. The blockchain acts as a verifiable data registry for certification processes with strong timestamping properties. The certification processes are managed by a set of smart contracts, collectively named *Notary*, that encode the certification logic.

The contracts keep track of all issued credentials using an efficient Merkle tree data structure that can be augmented incrementally. The Merkle tree stores cryptographic commitments to subjects' credentials. Subjects can prove ownership of commitments in the tree without revealing which commitments they own. We cover this mechanism in detail in Section 4.2. First, however, we introduce the credential document format we use to prove, in zero knowledge, the existence of individual fields of a credential. This format allows individual fields to be selectively disclosed. Lastly, in Section 4.3 we describe how credentials can be verified and give some example use cases.

4.1 Credential Document File Format

An issuer creates a credential by grouping claims about a subject in a document. In zkCert, this document is a JSON file following an issuer-defined schema. The schema may be aligned with the W3C verifiable credentials data model [12], which specifies mechanisms to express secure and machine-verifiable digital credentials.

According to the W3C specification, a verifiable credential is composed of three core elements:

(W3C1) Claims;
(W3C2) Credential's metadata;
(W3C3) Proof mechanisms.

Claims are a sequence of assertions made by an issuer about a subject. A credential's metadata describe properties that can be used for verification, as well as expiration dates and revocation mechanisms. Lastly, the proof mechanisms are used to verify a credential's authorship, prove its claims, and detect tampering.

The specification does not dictate a particular proof mechanism or file format. To facilitate a transparent and privacy-preserving certification process, we decouple the three elements of a verifiable credential into on-chain and off-chain data. We explain the rationale for this decoupling in the rest of the paper.

4.1.1 Merkle Tree of Claims

In our protocol, the claims (W3C1) are represented as fields of a JSON document. When issuing a credential, the document is augmented with a Merkle tree called *credential tree*. This tree is constructed deterministically from individual fields of the original document. The credential tree enables subjects to selectively disclose parts of the document and to prove that these parts belong to the original document using Merkle proofs. For instance, a student could reveal the field corresponding to his grades for a set of credentials without revealing other information about the credential besides the grades.

Our certification system requires that issuers use a predefined schema to create credentials. Otherwise, claims about specific fields may be deceiving. For instance, a credential could have fields for approval and revocation dates. In this scenario, a Merkle proof for the approval date alone would be insufficient to determine if the approval has been revoked.

The credential documents' fields form the leaves of the credential tree, constructed using Equation (1), where *key* is the SHA256 hash of the field's canonical property name modulo the SNARK prime field. The leaf F is the Poseidon hash of the concatenation of the *key*, the field's *value*, and a random *salt*.

$$F = \text{Poseidon}(key \| value \| salt) \tag{1}$$

As each field expresses a claim, the combination of fields can be used to compose an information graph about the subject. This, in turn, can be used to prove statements about the subject's credentials in zero knowledge. For example, the subject's grade for a course is greater than a given value. Figure 1 shows an example document and its credential tree.

The credential tree encoding enables more sophisticated computations over a credential's fields without revealing them. For example, we can prove that a set of credentials were issued in a specific period, and we can compute the weighted sum of a set of fields. These are relevant use cases for proving the duration of a bachelor's degree and a student's GPA. Appendix A provide details of how these examples are implemented in zkCert.

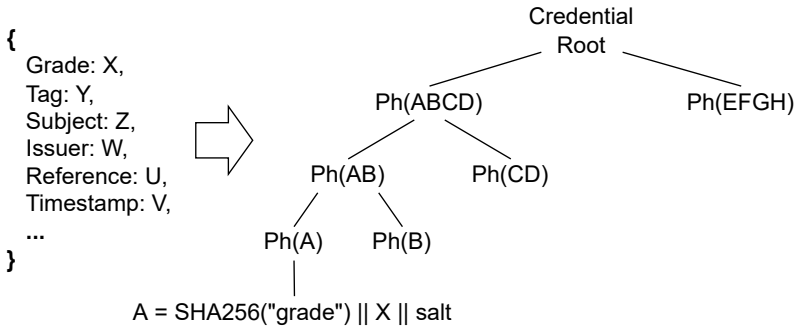

Figure 1 The encoding of a credential into a credential tree. Ph is the Poseidon hash function. Each field of the credential document is encoded as a leaf of the credential tree using Equation (1) over the field's key, value, and salt. In the example, A represents the encoding for the property "grade" and value "X".

4.2 Certification Tree

In this section we explain the certification tree, which is used to store credential commitments for all users. The certification tree is dynamically extended and keeps track of the issuer's actions for transparency.

In our protocol, credentials hold claims data (W3C1) in the form of a credential tree, and is stored off-chain by the subject. Proofs (W3C3) are generated on-demand by the subject, which can be verified by on-chain and off-chain mechanisms during certification. Finally, the credentials' metadata (W3C2) is stored on-chain in the Notary's state.

The metadata is mainly used to verify and manage the credentials created by the Notary in the data registry. For instance, the metadata of a credential contains its current status, i.e., issued, revoked, or expired. Thus, the status can be changed or verified when the issuer revokes a credential without relying on the issuer's servers.

Storing metadata on-chain increases the availability of verification services and can help to prevent censorship. However, it also imposes a challenge since it requires using a unique public identifier per credential to verify its metadata. Moreover, such identifiers could potentially be used to track credential owners.

To overcome this problem, we have designed a mechanism to prove the credential's ownership in zero-knowledge. Our mechanism effectively breaks the link between subjects and their credentials, while maintaining the credential's on-chain state for public verifiability and uses techniques that are already well-known in privacy preserving solutions using blockchains [20, 31, 33, 34].

The process of issuing credentials comprises two phases: Registration and Approval. A credential is considered valid if, and only if, registered by the issuer, approved by the subject, and it is not revoked or expired. Both phases use the PLONK [16] SNARK construction to generate the zkSNARK proofs in zkCert. Due to its upgradable and universal setup process, PLONK allows the issuer to create custom circuits up to a bounded size defined during trusted setup, enabling verification of a variety of credential's information when used in conjunction with the credential and certification tree. We describe these phases in Sections 4.2.2 and 4.2.3. But first we explain the basic structure of the certification tree.

4.2.1 On-chain Credential Registry

When creating a credential, the issuer records a cryptographic commitment to the credential in the Notary contract. Our protocol never reveals the preimage of the commitment. This is achieved by proving the commitment binding within the zero-knowledge circuit.

Commitments are created using the Poseidon hash function, which is efficient in zero-knowledge circuits [17]. Thus, the commitment C in Equation (2) is simply a point on the *bn128* elliptic curve. The *bn128* curve is natively supported on the EVM, enabling contracts to perform efficient zkSNARKs verification as well [28]. The *secret* is a random number, e.g., the subject's EdDSA private key, and *sub* is the hash of the subject's EdDSA public key.

$$C = \text{Poseidon}(cdr, secret, sub) \tag{2}$$

The credential root *cdr* is obtained by formatting the credential document as a Merkle tree, as described in Section 4.1, and retrieving the tree's root. Subjects only disclose the hash of their credential root H_{cdr} when claiming ownership of credentials, i.e., approving the credential issuance, as described in Section 4.2.3. Such hash represents a unique ID for every issued credential.

The commitments are stored in the leaves of an incremental Merkle tree in the issuer's Notary contract. We call this a *certification tree*, and its construction is shown in Figure 2. The tree is initialized with a fixed size in the Notary deployment, and all leaves are initially filled with zeros. Consequently, considering a perfect binary Merkle tree, the maximum number of credentials a Notary can register is 2^h, where h is the tree's height. This tree construction results in a Merkle tree whose hash values are in the *bn128* elliptic curve, allowing efficient computation in the circuit [34].

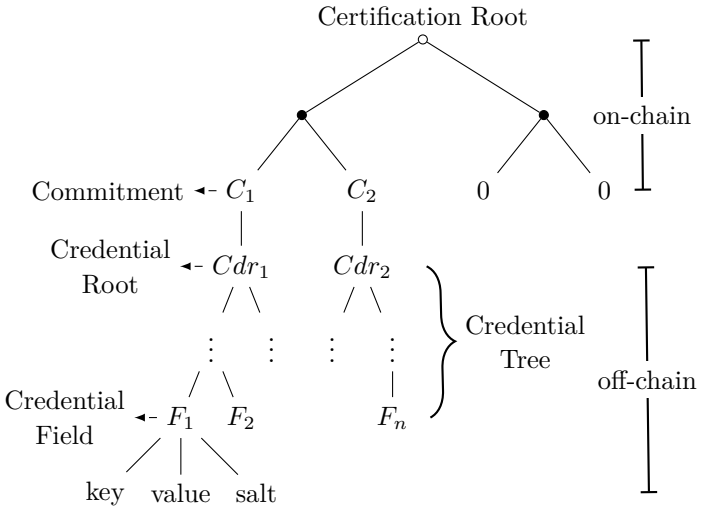

Figure 2 Incremental certification tree and its credentials' trees.

The incremental certification tree is responsible for keeping track of all credentials issued, preserving a progressive issuance process and a cryptographically immutable log of issuers' actions while increasing privacy and optimizing the on-chain storage and execution cost.

For instance, a naive Merkle tree algorithm would require $O(2^h)$ time and space complexity to add new leaves. The incremental Merkle Tree, on the other hand, reduces the computational cost when adding a new leaf, i.e., replacing a zero leaf, requiring only $O(h)$ time and space complexity to update the Merkle tree's root [25]. Leaves are added in the tree from the leftmost to the rightmost leaf, replacing the zero data with the credential commitment and updating the Merkle path to the new root.

Before creating any credential in zkCert, a trusted setup must be performed for the PLONK zkSNARK construction. The parties involved in the trusted setup in zkCert are arbitrary and defined by the issuer and trusted third parties, e.g., universities and national accreditation bodies. The subjects and verifiers do not need to participate in such a setup. Protocols like the scalable two-phase multi-party computation protocol MMORPG proposed by [7] can be used to generate the setup parameters. These parameters create the proving and verification keys used by zkCert to generate and verify proofs.

4.2.2 Registration Phase

To create a credential, an issuer first formats the claims about a subject in the form of a *credential tree*, as defined in Section 4.1. The issuer then sends the credential document to the subject over an encrypted channel. This phase is visualized in Figure 3. After verifying the credential's claims, the subject computes the credential root and generates an *issuance*

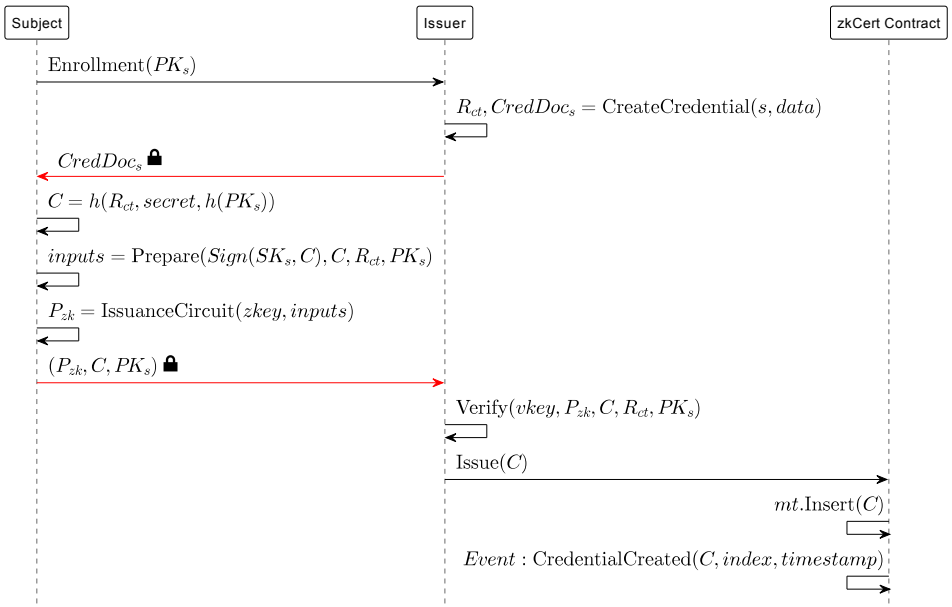

Figure 3 Registration phase: An issuer creates a credential to a subject and appends the commitment C to such credential in the Notary certification tree.

proof in zero knowledge using Circuit 1. The issuance proof attests to the issuer that the commitment is well-formed, i.e., used the expected credential root, and was created by the correct subject, i.e., the owner of the enrolled public key representing the subject's identity. This proof is used to justify the inclusion of the commitment into the Notary certification tree by the issuer and can be used for internal audits.

Circuit 1 Issuance proof.

```
1  IssuanceCircuit():
2      Public Inputs:
3          comm                          ▷ The credential commitment
4          cdr                                ▷ The credential root
5          pk                      ▷ The subject's EdDSA public key
6      Private Inputs::
7          secret                             ▷ The subject's secret
8          sig                       ▷ The commitment signature
9      sub ← Subject(pk)
10     Assert: comm ≠ 0
11     c ← Commitment(cdr, sub, secret)
12     Verify: c = comm
13     v ← EdDSAPoseidonVerifier(c, pk, sig)
14     Verify: v = true
```

The subject then sends, through an off-chain channel, the issuance proof and the commitment to the issuer. Upon receiving the subject's message, the issuer verifies the proof and, if valid, adds the commitment to the certification tree. We assume that the issuer can verify the subject's identity using its public key, e.g., through a PKI.

It is important to note that the issuer is the one who creates a credential and consequently shares management rights, e.g., revocation. As a result, the issuer should already know the subject's identity, the content of the credential, and its commitment. On the other hand, the subject creates the commitment using Equation (2) and is the credential owner. Consequently, nobody other than the subject can demonstrate knowledge of the preimage to such commitment without knowing the subject's secret inputs. Nevertheless, as explained in Section 4.3, the commitments are not revealed during presentations to third parties, and the subject can selectively disclose the credential's data as needed.

4.2.3 Approval Phase

The approval phase ensures that whoever was the subject of a registered credential agreed with its claims and could provide proof claiming its ownership. The phase represents a public consent of a subject within the credential without exposing the subject's identity and enables verification by third parties of the authenticity of future credential presentations.

This phase is visualized in Figure 4. After the issuer registers a commitment, the correspondent subject can, at any later point in time, call the Notary contract to approve the credential by revealing the H_{cdr}.

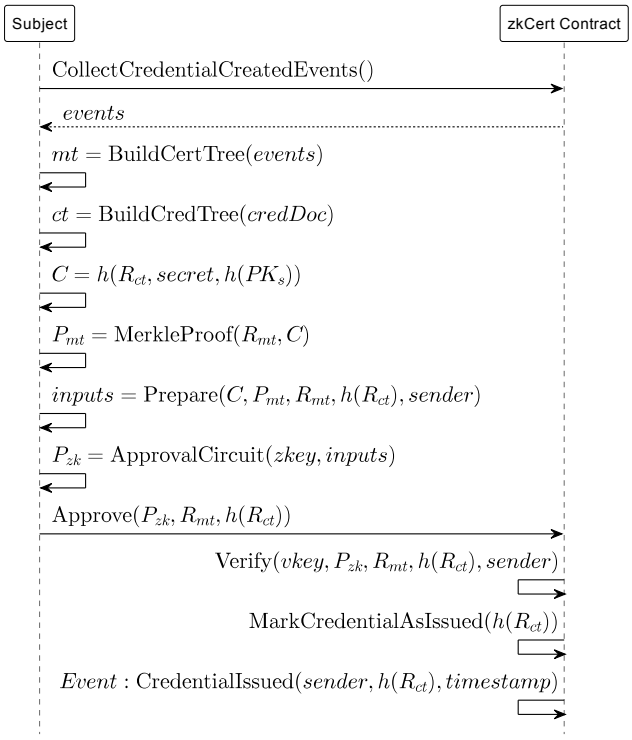

Figure 4 Approval phase: The subject reconstructs the certification tree locally based on the Notary events and generates a zkSNARK proof to approve a credential. The proof is then submitted to the contract by the sender, i.e., the contract's caller, and upon verification, updates the credential's state.

To generate an authenticity proof, a subject first retrieves the last certification root from the Notary contract and queries the necessary *CredentialCreated* events emitted by the contract in order to construct the Merkle proof of his commitments within the tree. This

process is needed since the certification tree is constantly being updated, and to be able to produce proofs for the most recent root, new data need to be retrieved from the blockchain. Of course, the data can be cached, and the issuer or third parties can provide services to speed up the data retrieval.

The subject generates an approval proof by executing Circuit 2, verified on-chain by the Notary contract. It attests that the subject not only has knowledge of a preimage to a commitment within the tree, but also knows the Merkle path to such commitment without revealing it. This attestation breaks the link between a credential's commitment from its presentation to an external verifier, which only needs to know the H_{cdr} to verify the status of the credential on-chain.

Circuit 2 Approval proof.

```
1  ApprovalCircuit():
2      Public Inputs:
3          ctr                                  ▷ The certification tree root
4          H_cdr                          ▷ The hash of the credential root
5      Private Inputs::
6          cdr                                        ▷ The credential root
7          subject                                         ▷ The subject ID
8          secret                                   ▷ The subject's secret
9          MtProof_comm                   ▷ The commitment's merkle proof
10     c ← Commitment(cdr, subject, secret)
11     Verify: Poseidon(cdr) = H_cdr
12     r ← VerifyMerkleProof(c, MtProof_comm)
13     Verify: r = ctr
```

The contract records the H_{cdr} hashes already approved, so a credential cannot be issued twice by the same Notary contract. The existence of the H_{cdr} on-chain also eases the management of issued credentials. Further, the proof contains a *sender* address of an entity authorized by the subject to call the contract in his behalf. This allows subjects to delegate the approval to third parties, which could also be the issuer. Note, however, that the issuer or any delegated third party *cannot* produce a valid proof on behalf of the subject, since only the subject knows the entire preimage data and his private key.

4.3 Verifying Credentials

The W3C verifiable credentials data model [12] defines a verifiable presentation as a mechanism to express data derived from credentials in which the authorship can be verifiable. In our protocol, a subject can prove the authorship of a credential without revealing all credentials they own from a specific issuer. Combined with the credential tree format described in Section 4.1, it is also possible to prove the existence of a field in the credential within the certification tree root and, consequently, the existence of a commitment to such credential on-chain. As writing data on-chain requires signing a transaction, such proofs enable authenticity checks of issued credentials while keeping the underlying data private.

As mentioned in Section 4.2, zkCert relies on the PLONK zkSNARK construction. Thus after the issuer has performed the setup, it can make available the proving and verification keys, and the supported circuits, to users and developers. We provide a set of circuits for the use cases covered in this paper, along with our companion source code[1].

Our design combines on-chain authenticity and validity checks, performed by Notary contracts, with off-chain claims verification based on zero-knowledge proofs provided during the presentation of credentials. A third-party verifier can, without any interaction with the issuer, check:

(a) The authenticity of credentials and its current status (e.g. revocation, expiration date);
(b) If the values in the credential's fields met certain conditional criteria (e.g. all grades are greater than B);
(c) The result of an arithmetic computation over a set of credentials (e.g. the GPA of a student);
(d) The time-frame in which the presented credentials were issued (e.g. if all credentials of a subject were issued in a three years period);

Note that the items listed above do not constitute an exhaustive list, and all can be verified in zero-knowledge, thus without exposing any underlying private values.

Item (a) is verified by requesting zero-knowledge proof for the credentials being presented and cross-checking the hashes H_{cdr} of these credentials in the Notary regarding their current status. It is important to note that such verification reveals the H_{cdr} to the verifier since it is required to check the credential status, e.g., revocation, but an external observer to the blockchain cannot track such activity. Nevertheless, the correspondent commitment and credential's data remain secret to the verifier, and it is infeasible to recover such information from a given H_{cdr}. However, it may be possible to de-anonymize users with statistical analysis of on-chain transaction patterns. The complete flow of this process is included in Appendix A.

Further, due to the way that our credentials are built (see Figure 2), Item (a) can also be extended to verify the authenticity of specific fields of credentials or perform simple arithmetic computation over it. In Appendix A we give an example of a circuit that computes the weighted sum of a set of authenticated integers using our approach.

Subjects can also select specific fields of a set of credential trees they own to present to a verifier in zero-knowledge, this enables checks as described in Item (b). However, a naive implementation would require that the subject create one Merkle proof for each field to be proven.

To reduce the number of Merkle proofs required to proof Items (b)-(d), we added support for Merkle multi-proofs [27], which reduces the number of Merkle proofs required to prove the inclusion of multiple leaves to a single proof. This improvement allows the creation of Merkle multi-proofs for multiple credentials and multiple fields of credentials. We also provide a circuit implementation in Circom [21] of the Merkle multi-proof algorithm, enabling membership proofs in zero knowledge.

4.4 Design Properties Analysis

As mentioned in Section 4, zkCert uses the blockchain to guarantee a trusted timestamping of issued credentials. Timestamp information is added to the credential JSON document and consequently to its credential tree representation. This information can be verified off-chain

[1] `https://github.com/r0qs/zkcertree`

at the circuit level, as exemplified in Appendix A.4, and cross-checked against its on-chain registration and approval metadata in the certification tree, considering a time threshold. This characteristic can help to detect fraudulent patterns in the issuance process.

The blockchain also ensures integrity, authenticity, and service availability while guaranteeing that the metadata stored on-chain are valid and always accessible for verification. Although we do not address the off-chain storage of credentials in this work but assume its existence, we acknowledge that it is an essential part of a certification system. The use of content-addressing storage like IPFS [4] or Swarm [35] can improve censorship resistance and availability of proving data as well, i.e., credential documents.

The certification tree provides transparency in the certification process through its incremental tamper-resistant event log, and combined with identity mechanisms like DIDs [11] and DPKI [1], can ensure the authenticity of an issuer's actions and issued credentials. It can further provide non-repudiation through subject approvals, helping auditing processes.

The approval phase in Section 4.2.3 may not be suitable for all use cases, e.g., issuance of criminal records. However, it is a feature of our protocol to register a public agreement of all signing parties of a certificate, i.e., the issuer and the subject. The approval phase can safely be omitted when the registration phase is sufficient to meet the use case's requirements.

The certification tree also enables the revocation of credentials without relying on a central server. The revocation permissions can be configured through an access control list of authorized issuer's personnel.

Finally, using zkSNARK to prove statements about credentials adds a privacy layer to our certification system, preventing unauthorized entities monitoring blockchain events from tracking user activities, while preserving the transparency properties of the protocol.

5 Evaluation

In this section, we present our evaluation of zkCert. We first evaluate the monetary cost of issuing diplomas on various blockchains. Then we assess the performance of proving and verification time of the zkSNARK construction.

For the monetary cost analysis, we compare the cost of the operations performed by zkCert with Certree's costs [29]. The cost is expressed in dollars instead of gas units [14] since it reflects the storage and computational cost of smart contracts operations in the analyzed blockchains. We also analyzed the on-chain storage usage of both approaches. Our results are shown in Table 1 and include updated costs for Certree based on recent exchange rates for three mainstream blockchains: Ethereum [14], Avalanche [30] and Polygon [32].

An *Exam cert* in Table 1 represents one issued credential and does not consider the cost or storage space of the contract deployed. Other lines show complete cost for a certificate composed of multiple credentials. For instance, a course certificate (*Course cert*) in our example is composed of five exam certificates issued to the same subject. As in [29], we assume that courses correspond to 10 ECTS and a semester to 30 ECTS.

The table includes both costs of issuing and approving credentials. In Certree [29], the approval cost must be borne by the subject, the credential's owner. In zkCert, however, this cost can be offloaded to a third party, e.g., the issuer. This is possible since approval is not linked to the subject's on-chain identity; instead, the subject's approval is linked via a zero-knowledge proof. Further, in Certree, a contract must be deployed for each course, adding progressive deployment costs, while in zkCert, the Notary contract is only deployed once. Thus, zkCert adds a fixed initial storage cost of 8.39 Kb for initializing the incremental certification tree as part of the Notary contract. As described in Section 4.2, issuers can update the Notary contract incrementally as new credentials are issued. It is

Table 1 Monetary cost and on-chain storage utilization comparison of zkCert and Certree for a hypothetical scenario issuing Bachelor's and Master's credentials for one student.

Protocol		ECTS	Cred. Issued	Contracts Deployed	On-chain Storage	Cost ETH/ USD*	Cost AVAX/ USD◁	Cost MATIC/ USD†
Certree	Exam cert		1	0	321 b	20.01	0.22	0.01
	Course cert	10	5	1	1.60 Kb	297.82	3.27	0.15
	Semester cert	30	16	3	5.33 Kb	1106.23	12.15	0.56
	Bachelor's diploma	180	97	18	32.67 Kb	6881.09	75.59	3.49
	Master's diploma	120	65	12	21.89 Kb	4658.32	51.17	2.36
zkCert	Exam cert		1	0	66 b	39.18	0.43	0.02
	Course cert	10	5	1	8.72 Kb	195.92	2.15	0.10
	Semester cert	30	16	1	9.38 Kb	587.77	6.46	0.30
	Bachelor's diploma	180	97	1	22.72 Kb	3618.83	39.75	1.84
	Master's diploma	120	65	1	20.74 Kb	2443.30	26.84	1.24

* Gas price = 33 Gwei and ETH 1 = 1634.53 USD at 2022-08-20.

◁ Gas price = 26 nAVAX and AVAX = 22.79 USD at 2022-08-20.

† Gas price = 34.4 Gwei and MATIC = 0.796 USD at 2022-08-20.

also worth noting that the cost of zkCert considers a partial binary Merkle tree of height up to 31. Table 1 shows the combined cost of deploying contracts and creating and approving credentials.

Table 1 shows that the cost of issuing a single credential increased by almost 49 % with zkCert compared to Certree. This increase is due to adding a new leaf in the Notary's tree during registration and on-chain verification of a zkSNARK proof during approval. However, this initial cost is amortized as more credentials are created since no new contracts must be deployed. In Certree, however, each new leaf or intermediary node in their tree adds a fixed contract deployment cost. For instance, to issue a Bachelor's diploma over three years, there is an aggregate cost reduction of 47 % compared to Certree. Further, using zero-knowledge proofs, we reduced the on-chain space requirement per credential by more than 79 %.

We next evaluate the computation time to generate an approval proof and the corresponding time to verify an approval. The approval circuit has 3379 constraints. The experiment run in a machine with Archlinux installed with 16 GB RAM, processor Intel Core i7-8550U 1.80 GHz and 500 GB SSD.

As shown in Figure 5, the time to generate an approval proof follows a step-function over the tree's height. The tree's maximum size, decided during deployment, dictates the time it takes to generate an approval. We observe, however, that the generation and verification time is unaffected by the number of credentials already issued. Thus, the proving time using a tree of height 21, which can hold 2^{21} elements, will be around 30 seconds, irrespective of the number of elements in the tree. This is because generating a Merkle proof for an incremental Merkle tree has linear time complexity [25]. The verification time of the approval proof is polylogarithmic in the circuit size [16].

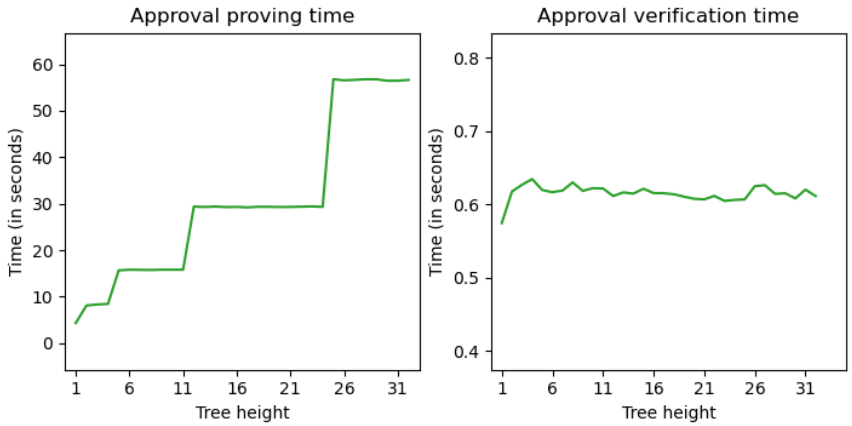

Figure 5 Time to generate and verify approval proofs per the certification tree's height.

6 Conclusion

We have presented zkCert, a certification system for digital credentials that ensures transparency of the issuer's processes and preserves privacy for its users. zkCert is implemented using smart contracts, where an incremental Merkle tree enables scalability, and zkSNARKs provide privacy to users.

Our evaluation shows that zkCert provide significant improvement over prior art, both in on-chain storage utilization and lower monetary costs to produce credentials over time. For deployments that can support up to 2^{31} credentials, it takes less than a minute to generate a credential while maintaining transparency.

Although we focused our work on a specific use case of academic credentials, our design can be used in various use cases that can benefit from a transparent and progressive history of issuance activities while ensuring a high level of privacy for their users, e.g., driver's licenses, supply chain, and fair trade.

References

1 Christopher Allen, Arthur Brock, Vitalik Buterin, Jon Callas, Duke Dorje, Christian Lundkvist, Pavel Kravchenko, Jude Nelson, Drummond Reed, Markus Sabadello, Greg Slepak, Noah Thorp, and Harlan T. Wood. Decentralized public key infrastructure, 2015. URL: `https://www.weboftrust.info/downloads/dpki.pdf`.

2 Philip G. Altbach, Liz Reisberg, and Laura E. Rumbley. Trends in global higher education: Tracking an academic revolution. Technical report, United Nations Educational, Scientific and Cultural Organization (UNESCO), January 2009. URL: `https://www.cep.edu.rs/public/Altbach,_Reisberg,_Rumbley_Tracking_an_Academic_Revolution,_UNESCO_2009.pdf`.

3 Stephanie Bayer and Jens Groth. Efficient zero-knowledge argument for correctness of a shuffle. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, pages 263–280, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

4 Juan Benet. IPFS - content addressed, versioned, P2P file system. *CoRR*, abs/1407.3561, 2014. `arXiv:1407.3561`.

5 Nir Bitansky, Ran Canetti, Alessandro Chiesa, Shafi Goldwasser, Huijia Lin, Aviad Rubinstein, and Eran Tromer. The hunting of the snark. Cryptology ePrint Archive, Paper 2014/580, 2014. URL: `https://eprint.iacr.org/2014/580`.

6 Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016*, pages 327–357, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

7 Sean Bowe, Ariel Gabizon, and Ian Miers. Scalable multi-party computation for zk-snark parameters in the random beacon model. Cryptology ePrint Archive, Paper 2017/1050, 2017. URL: https://eprint.iacr.org/2017/1050.

8 Vitalik Buterin. An approximate introduction to how zk-snarks are possible. Accessed: September 2022. URL: https://vitalik.ca/general/2021/01/26/snarks.html.

9 Guang Chen, Bing Xu, Manli Lu, and Nian-Shing Chen. Exploring blockchain technology and its potential applications for education. *Smart Learning Environments*, 5, December 2018. doi:10.1186/s40561-017-0050-x.

10 Santosh Chokhani, Warwick Ford, Randy V. Sabett, Charles (Chas) R. Merrill, and Stephen S. Wu. Internet x.509 public key infrastructure certificate policy and certification practices framework. RFC 3647, RFC Editor, November 2003. URL: https://www.rfc-editor.org/rfc/rfc3647.

11 World Wide Web Consortium. Decentralized identifiers (dids), 2017. Accessed: August 2022. URL: https://w3c.github.io/did-core/.

12 World Wide Web Consortium. Proposal specification of verifiable credentials, 2019. Accessed: August 2022. URL: https://www.w3.org/TR/vc-data-model.

13 Wikipedia contributors. Varsity Blues scandal, March 2019. Accessed: September 2022. URL: https://en.wikipedia.org/wiki/Varsity_Blues_scandal.

14 Vitalik Buterin et al. A next-generation smart contract and decentralized application platform, 2014. Accessed: August 2022. URL: https://ethereum.org/en/whitepaper.

15 Council for Higher Education Accreditation, Scientific United Nations Educational, and Cultural Organization. Toward effective practice: Discouraging degree mills in higher education, 2009. URL: https://unesdoc.unesco.org/ark:/48223/pf0000183247.

16 Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Paper 2019/953, 2019. URL: https://eprint.iacr.org/2019/953.

17 Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. Cryptology ePrint Archive, Paper 2019/458, 2019. URL: https://eprint.iacr.org/2019/458.

18 Alex Grech and Anthony F. Camilleri. Blockchain in education, 2017. EUR 28778 EN. doi:10.2760/60649.

19 Jens Groth and Mary Maller. Snarky signatures: Minimal signatures of knowledge from simulation-extractable snarks. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017*, pages 581–612, Cham, 2017. Springer International Publishing.

20 Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. Zcash protocol specification, version 2022.3.6, 2022. Accessed: September 2022.

21 Iden3. Circom: Circuit compiler. Accessed: September 2022. URL: https://docs.circom.io/.

22 Ben Laurie and Adam Langley. Certificate transparency, 2013. Accessed: August 2022. URL: https://www.certificate-transparency.org/.

23 Ralph C. Merkle. Protocols for public key cryptosystems. *1980 IEEE Symposium on Security and Privacy*, pages 122–122, 1980.

24 Alex Ozdemir and Dan Boneh. Experimenting with collaborative zk-snarks: Zero-knowledge proofs for distributed secrets. Cryptology ePrint Archive, Paper 2021/1530, 2021. URL: https://eprint.iacr.org/2021/1530.

25 Daejun Park, Yi Zhang, and Grigore Rosu. End-to-end formal verification of ethereum 2.0 deposit smart contract. In Shuvendu K. Lahiri and Chao Wang, editors, *Computer Aided Verification*, pages 151–164, Cham, 2020. Springer International Publishing.

26 Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology — CRYPTO '91*, pages 129–140, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

27 Lum Ramabaja and Arber Avdullahu. Compact merkle multiproofs, 2020. doi:10.48550/arXiv.2002.07648.

28 Christian Reitwiessner. Precompiled contracts for addition and scalar multiplication on the elliptic curve alt-bn128. Accessed: September 2022. URL: https://github.com/ethereum/EIPs/blob/master/EIPS/eip-196.md.

29 Rodrigo Q. Saramago, Leander Jehl, Hein Meling, and Vero Estrada-Galiñanes. A tree-based construction for verifiable diplomas with issuer transparency. In *2021 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)*, pages 101–110, 2021. doi:10.1109/DAPPS52256.2021.00017.

30 Avalanche Team. Avalanche. Accessed: September 2022. URL: https://www.avax.network/.

31 Iden3 Team. Identity protocol. Accessed: September 2022. URL: https://iden3.io/.

32 Polygon Team. Polygon. Accessed: September 2022. URL: https://polygon.technology/.

33 Semaphore Team. Signal anonymously. Accessed: September 2022. URL: https://semaphore.appliedzkp.org/.

34 Tornado Cash Team. Tornado cash documentation. Accessed: September 2022. URL: https://web.archive.org/web/20220624094748/https://docs.tornado.cash/general/readme.

35 Viktor Trón. The book of swarm - storage and communication infrastructure for self-sovereign digital society, 2020. URL: https://swarm-gateways.net/bzz:/latest.bookofswarm.eth/.

36 M. Turkanović, M. Hölbl, K. Košič, M. Heričko, and A. Kamišalić. Eductx: A blockchain-based higher education credit platform. *IEEE Access*, 6:5112–5127, 2018. doi:10.1109/ACCESS.2018.2789929.

A Verifiable Presentations

In this section, we present some potential use cases for our protocol. All verification flows presented in this section do not require that the subject share the entire credential document or reveal the values of the underline credential's fields being proved.

A.1 Authentication Proof

For the verification of Item (a) in Section 4.3, consider Figure 6, that shows how a subject can proof the authenticity of a credential by creating a presentation zkSNARK proof.

Like with the approval mechanism, the subject first reconstructs the latest state of the certification tree locally based on the Notary events on chain. After retrieving the necessary data, the subject generates the proof using the proving key and the private and public inputs, and send it to the verifier with the H_{cdr}, the latest certification root, and his EdDSA public key.

The verifier then checks whether the received certification root exists in the root history of the contract, and if so, verify the proof running the circuit with the received public inputs and the verification key. This verification attests that:

- The subject knows the preimage of a commitment for the presented credential;
- The commitment exists in the contract's merkle tree for the given root;
- The given root is one of the latest roots in the contract;
- The credential was issued to the subject (i.e. he has knowledge of the EdDSA private key used to sign the commitment);
- The subject consent with the claims within the credential (i.e. the credential was approved by the subject);

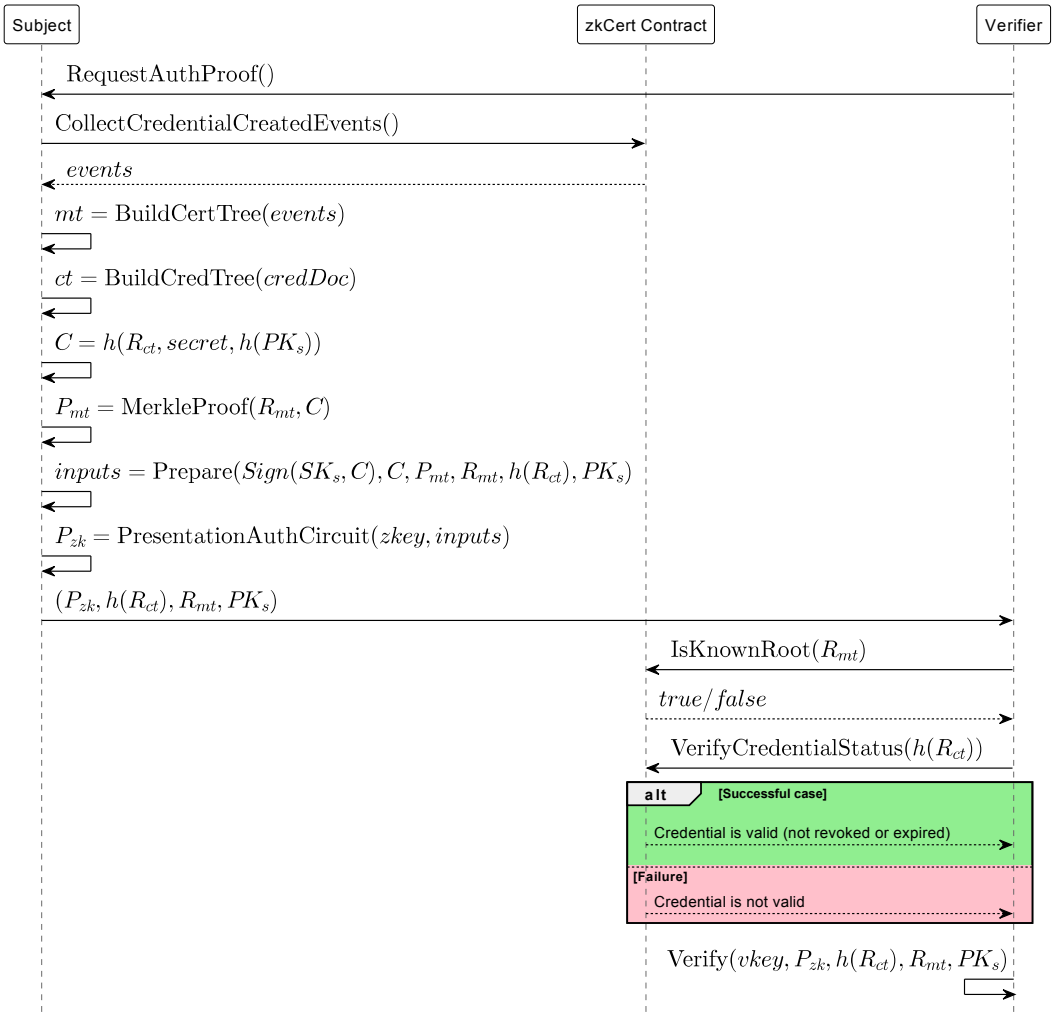

Figure 6 Steps to verify the authenticity of a credential presentation.

A.2 Conditional Proof

For the verification of Item (b) in Section 4.3 consider the case where a student needs to prove that pass in an exam by achieving a grade higher than 60 on a scale of 0 to 100.

The student can choose the specific field of the document by its key, i.e., hash value, to be proven from his credential and create a zkSNARK proof for the field, criterion, and operator challenged by the verifier. The operator is a bitmap representing the conditional operator to be used, e.g., greater or equal. In our example, the criterion is the minimum grade required to pass the exam, i.e., 60. If the key is mismatched, the proof generation will fail. Proof generation also fails if the criterion is not satisfied.

A.3 Score Proof

For the verification of Item (c) in Section 4.3 consider the case where a student would like to apply to a study loan which requires a certain score measured based on the grades of the applicants.

Usually, such a scenario would require that each applicant share all their grades with the loan agency, e.g., by sharing their transcript of records. The loan agency would need to verify the authenticity of each application with different issuers, e.g., universities, and perform some score function to select the best candidates.

Ignoring any personal interview that may be required, the initial triage of the candidates could be automated. However, such automation may not be possible in many current systems. Further, the process may reveal more information than necessary for a triage phase, and the computation of the scores must be performed transparently to represent a fair selection of the candidates.

Thus, a naive solution to the problem may compromise the applicants' privacy. On the other hand, our solution provides an alternative to guarantee the correctness of the computation over authenticated grades while enabling automation of the verification process. Figure 7 illustrate such example in zkCert.

T_i represents a tag or ID for a course that the loan agency considered required to apply to the study program, and W_i is the correspondent weight of such a course. Let's consider that the score function is simply the weighted sum of all grades required. The loan agency would select the candidates with higher scores.

Each candidate can retrieve the information necessary to construct their proof, as described in Section 4.3, compute their score, and send the zkSNARK proof and the public inputs to the verifier.

Further, the verification of the results can be performed by anyone. For instance, a smart contract could check each applicant's zkSNARK proof and determine which of the candidates satisfies the minimum score to pass to the next phase of the admission process. It is important to note that verifying the score, *sc*, does not reveal the student's grades.

A.4 Time-frame Proof

For the verification of Item (d) in Section 4.3 consider the case where a bachelor's student would like to apply for a masters program. The main requirement of such program is that the student has a valid bachelor's degree, which is usually verified by requesting a diploma, i.e., the student's academic credential.

However, when verifying such credential, usually, only the authenticity of the signatures on the diploma is taken in consideration, which does not easily detect fake diplomas produced by Degree Mills [15].

zkCert can be used in such a scenario to prove that a diploma is composed of credentials that fit a specific trustful time range. As the issuer timestamps each credential on creation as well by the Notary contract on approval, neither the student nor the issuer can tamper with the period in which all the credentials that compose a diploma were issued. Figure 8 illustrate such scenario.

Thus, a verifier can require a time-frame proof from the student challenging him to prove that his diploma was issued over a period greater than three years. The proof can be generated and verified using the Circuit 3. The bitmap comparison operator OP is three bits bitmap that defines the currently supported operators in our implementation. For the example above, the operator would be 011, which translates to $\geq$. Our source code[2] contains more details of the currently supported operators.

The verifier can check if the issuance period was indeed greater than three years and the Circuit 3 ensures that a valid proof does not violate the incremental timestamp invariant for the provided credentials without requiring share the credentials documents with the verifier.

[2] `https://github.com/r0qs/zkcertree`

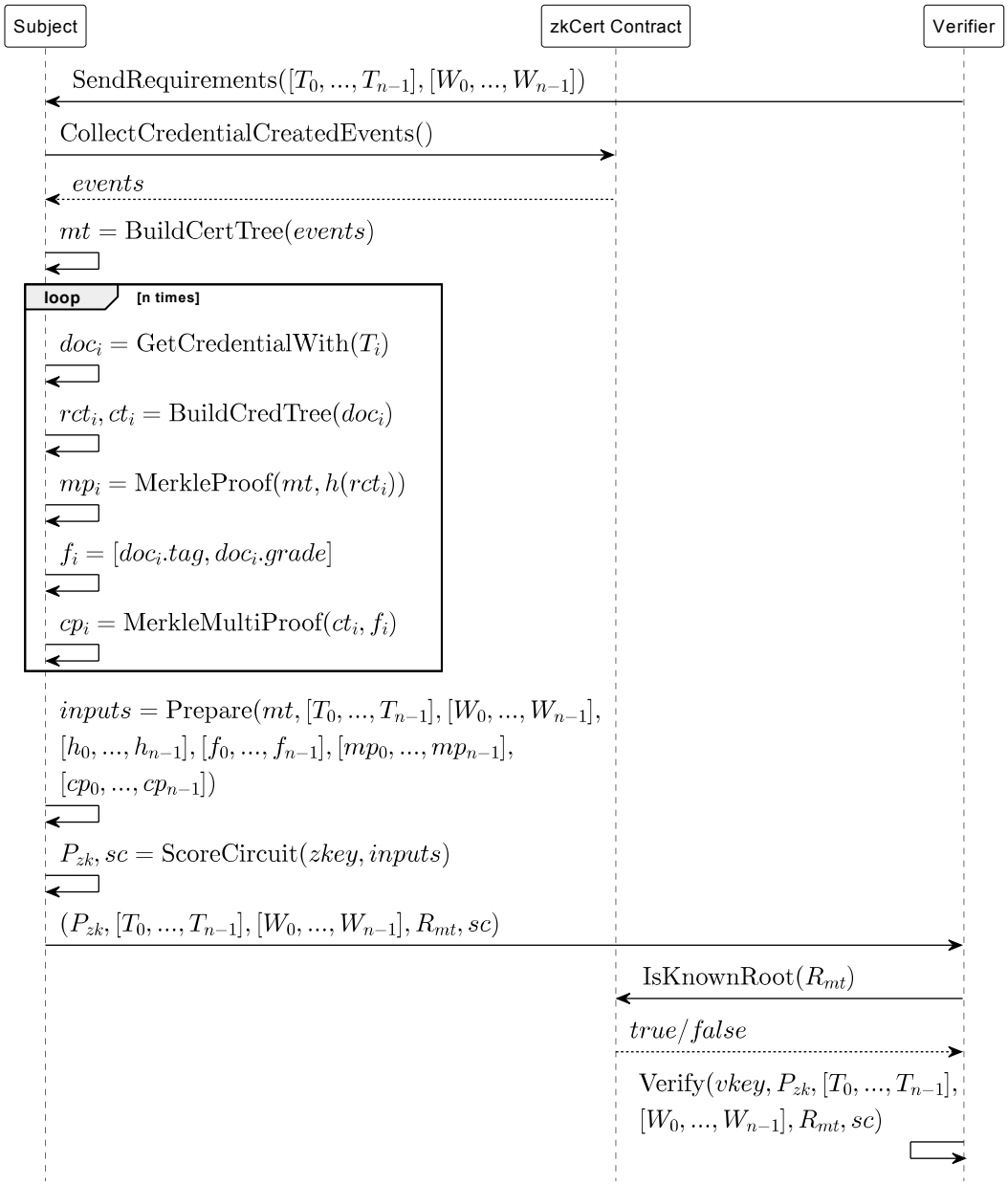

Figure 7 Verify a score function over a set of student's credentials.

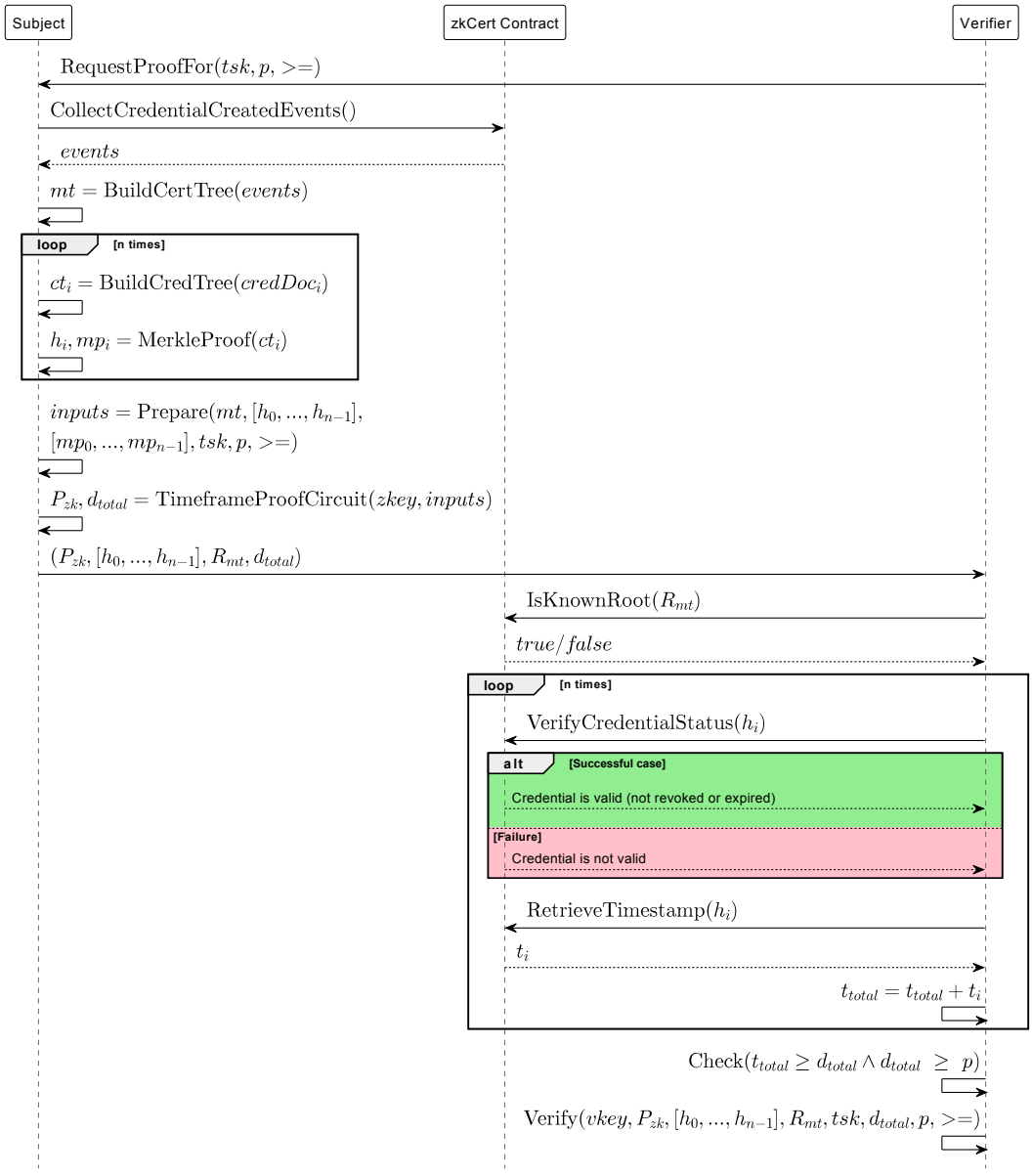

Figure 8 Verify issuance period of a set of credentials.

Circuit 3 Verify issuance period of a set of credentials.

TimeframeProofCircuit(n):
▷ n is the number of credentials in the certree
Public Inputs:
ctr ▷ The latest certree root
$NH[n]$ ▷ List of n nullifier hashes sorted by their credential's timestamp
tsk ▷ The timestamp field key
$period$ ▷ The duration to be checked
OP ▷ The bitmap comparison operator
Private Inputs::
$ts[n][3]$ ▷ Timestamp field of each credential
$MtProof_{fld}[n]$ ▷ Field merkle proofs
$nullifiers[n]$ ▷ The list of credential roots
$subjects[n]$ ▷ The list of subjects
$secrets[n]$ ▷ The list of secrets
$MtProof_{comm}[n]$ ▷ The commitment's merkle proofs
for $i \leftarrow 0$ **to** n **do**
$v \leftarrow VerifyCredentialFieldCircuit(tsk, ts[i], MtProof_{fld}[i], nullifiers[i], subjects[i], secrets[i], MtProof_{comm}[i])$
Verify: $v == true$
end
$d \leftarrow ComputeTotalDuration()$
Assert: $\sum_{i=0}^{n-1}(ts[i+1] - ts[i] = d_i \wedge d_i > 0)$
Verify: $d_{total} == ts[n-1] - ts[0]$
Verify: $(d_{total}\ OP\ period) == true$
$out \leftarrow d_{total}$