**TROY UNIVERSITY**
**CS 3372 - Summer 2025**
**Formal Language & The Theory of Computation**

---

Group Project

# Designing Smart Contract-Based Mechanisms for Anonymous Credential Verification

---


**Student:**
Nguyen Dinh Khiem
Tran Tien Dat
Nguyen Kim Tuan Cuong
Nguyen Hai An
Nguyen Khanh Viet Dung
**Lecturer:**
Assoc Prof. Nguyen Dinh Han


August 18, 2025

# Table of Contents


**1 Introduction** **4**

**2 Preliminaries** **4**
- 2.1 Anonymous Credentials . . . 4
- 2.2 Zero-Knowledge Proofs . . . 5
- 2.3 Diffie-Hellman Exponent . . . 6
- 2.4 Diffie-Hellman Exponent . . . 7
- 2.5 Ethereum Virtual Machine (EVM) . . . 8
- 2.6 Ethereum and Gas System . . . 9
- 2.7 Cryptographic Accumulators . . . 10

**3 System Model and Threat Analysis** **11**
- 3.1 System Entities and Operational Environment . . . 11
- 3.2 Threat Model and Adversarial Capabilities . . . 11
- 3.3 Security Requirements and Formal Properties . . . 12
- 3.4 Attacks and Mitigations . . . 12
- 3.5 System Architecture . . . 12

**4 Protocols: Issuance, Revocation, Verification** **13**
- 4.1 Credential Issuance . . . 13
- 4.2 Off-Chain Operations and State Synchronization . . . 13
  - 4.2.1 Off-Chain Credential Generation and Witness Management . . . 14
  - 4.2.2 Batching Strategies and State Consistency . . . 14
  - 4.2.3 State Synchronization and Consistency Guarantees . . . 15
  - 4.2.4 Economic Analysis and Cost Optimization . . . 15
- 4.3 On-chain Verification . . . 16
  - 4.3.1 Gas Optimization Strategies and Economic Analysis . . . 16
- 4.4 Ledger . . . 17

**5 Cryptographic Signature Schemes and Performance Analysis** **17**
- 5.1 Signature Scheme Taxonomy and Selection Criteria . . . 17
- 5.2 BLS12-381 and BBS+ Signature Implementation . . . 18
- 5.3 Elliptic Curve Performance Comparison . . . 19
- 5.4 IELTS Credential System Implementation . . . 19
- 5.5 Comparative Analysis of Signature Schemes for Anonymous Credentials . . . 20
- 5.6 Deployment Recommendations and Use Case Analysis . . . 21

**6 Zero-Knowledge Proof Scheme** **22**
- 6.1 Theoretical Foundations . . . 22
- 6.2 Unified Revocation Interface . . . 22
- 6.3 ZKP Using Bulletproofs . . . 23
  - 6.3.1 Systems . . . 23
  - 6.3.2 Mathematical Foundation . . . 23
  - 6.3.3 Circuits . . . 24
  - 6.3.4 Verification . . . 24
- 6.4 ZKP Using zk-SNARKs . . . 24
  - 6.4.1 Systems . . . 24

6.4.2 Mathematical Foundation . . . 24
6.4.3 Circuits . . . 25
6.4.4 Verification . . . 25
6.5 Comparison of Zero-Knowledge Proof Systems . . . 26
6.5.1 Theoretical Foundations . . . 26
6.5.2 Performance Characteristics . . . 26
6.5.3 Security Assumptions . . . 26
6.5.4 Practical Implementation Considerations . . . 27
6.5.5 Implementation Complexity . . . 27
6.5.6 Gas Cost Analysis . . . 28
6.5.7 Application-Specific Trade-offs . . . 28
6.5.8 Future Directions . . . 28
6.5.9 Implementation-Specific Analysis . . . 29

**7 Accumulator Scheme** **29**
7.1 Elliptic-Curve . . . 29
7.1.1 Formal Model . . . 29
7.1.2 Algorithms . . . 30
7.1.3 Complexity and Gas Analysis . . . 31
7.2 Tree-based Accumulator . . . 32
7.2.1 Formal Model . . . 32
7.2.2 Algorithms . . . 32
7.2.3 Complexity and Gas Analysis . . . 32
7.3 List-base Revocation: A Baseline Approach . . . 33
7.3.1 Formal Model and Security Properties . . . 33
7.3.2 Algorithmic Implementation and Complexity Analysis . . . 33
7.3.3 Complexity Analysis and Performance Characteristics . . . 35
7.3.4 Gas Cost Analysis and Optimization Strategies . . . 35
7.3.5 Comparative Analysis and Deployment Scenarios . . . 36
7.4 ZKP with EC-based Accumulator . . . 36
7.4.1 Statement . . . 36
7.4.2 Circuit Sketch . . . 37
7.4.3 Costs . . . 37
7.5 ZKP with Tree-based Accumulator . . . 38
7.5.1 Statement . . . 38
7.5.2 Circuit Sketch . . . 38
7.5.3 Costs . . . 38

**8 IELTS Credential System: Real-World Deployment Case Study** **39**
8.1 Problem Context and Requirements Analysis . . . 39
8.2 System Architecture and Component Implementation . . . 39
8.2.1 Issuer Implementation: Test Center Infrastructure . . . 39
8.2.2 Holder Implementation: Candidate Privacy Management . . . 41
8.3 Deployment Scenarios and Use Case Analysis . . . 42

**9 Implementation and Evaluation** **42**
9.1 Experimental Design . . . 42
9.1.1 Questions . . . 42
9.1.2 Treatments and Factors . . . 43
9.1.3 Metrics . . . 43

9.1.4 Method . . . 43
9.1.5 Results . . . 43
9.1.6 Operational Considerations and Economic Impact Analysis . . . 44
9.2 Implementation and Experiment Setup . . . 45
9.2.1 System Architecture Overview . . . 45
9.2.2 Ethereum Network Implementation . . . 46
9.2.3 Zero-Knowledge Proof System . . . 47
9.2.4 Smart Contract Integration . . . 48
9.2.5 API and Lambda Integration . . . 48
9.2.6 Experimental Environment Configuration . . . 48
9.2.7 Deployment Automation . . . 49
9.3 System Architecture and Protocol Interactions . . . 50
9.3.1 Architectural Components and Design Rationale . . . 50
9.3.2 Credential Issuance Protocol Flow . . . 50
9.3.3 Verification Protocol Execution . . . 51
9.3.4 Revocation Protocol and State Management . . . 51
9.3.5 Protocol Implementation and Algorithmic Specifications . . . 52
9.3.6 System Architecture Visualization . . . 54
9.3.7 Protocol Flow Analysis . . . 54

**10 Conclusion** **56**
10.1 Research Contributions and Impact . . . 56
10.2 Practical Implications and Deployment Guidance . . . 57
10.3 Research Limitations and Future Work . . . 57
10.4 Broader Impact and Vision . . . 58

# 1 Introduction

**Motivation.** Anonymous credentials represent a fundamental cryptographic primitive enabling privacy-preserving authentication systems. These systems allow users to prove possession of certified attributes without revealing their identity or disclosing unnecessary information to verifiers. While traditional anonymous credential schemes have been extensively studied in offline settings, their deployment on blockchain platforms introduces novel challenges stemming from computational constraints, gas-based fee models, and the requirement for efficient revocation mechanisms in permissionless environments.

**Problem Statement.** Existing anonymous credential systems face significant obstacles when deployed on gas-metered ledgers such as Ethereum. The primary challenges include: (1) the computational overhead of zero-knowledge proof verification in smart contracts, (2) the trade-off between proof succinctness and update efficiency for revocation mechanisms, and (3) the lack of comprehensive evaluation frameworks that account for realistic blockchain operational costs. Current solutions often optimize for theoretical efficiency without considering practical deployment constraints, resulting in systems that are either prohibitively expensive to operate or compromise security guarantees.

**Our Contributions.**Our research makes three fundamental contributions to the field of blockchain-based anonymous credentials.

**First**, we provide a comprehensive formalization of anonymous credential protocols specifically designed for gas-metered ledgers, defining precise security properties including unlinkability, soundness, and revocation freshness while explicitly capturing the economic constraints imposed by blockchain execution environments. This formalization represents the first systematic treatment of anonymous credentials that accounts for the practical realities of deployment on permissionless blockchain platforms.

**Second**, we design and implement two distinct revocation architectures—elliptic curve accumulators and Merkle tree-based structures—providing complete zero-knowledge circuit implementations for both approaches and conducting rigorous asymptotic and empirical analysis of proof size, prover time, and on-chain gas consumption. Our comparative evaluation reveals fundamental trade-offs between constant-time verification and logarithmic scaling properties that directly impact system economics and deployment viability.

**Third**, we develop a complete prototype system using Circom and pairing-friendly zk-SNARKs, conducting extensive experiments across multiple dimensions including verification gas costs, update frequencies, and batch processing effects. Our comprehensive evaluation provides concrete guidance for practitioners deploying privacy-preserving credential systems on blockchain platforms, enabling evidence-based architectural decisions rather than theoretical speculation.

**Organization.** The remainder of this paper is structured as follows: Section 3 presents our formal system and threat model. Section 4 specifies the issuance, revocation, and verification protocols. Section 6 details our zero-knowledge circuit constructions and theoretical analysis. Section 9 describes our implementation and experimental evaluation.

# 2 Preliminaries

## 2.1 Anonymous Credentials

An anonymous credential system allows users to obtain credentials from issuers and later prove possession of these credentials to verifiers without revealing their identity or unnecessary attribute information.

**Definition 1** (Anonymous Credential System)**.** *An anonymous credential system consists of algorithms* $(\mathsf{Setup}, \mathsf{Issue}, \mathsf{Show}, \mathsf{Verify})$ *where: The system comprises four essential algorithms:* $\mathsf{Setup}(1^\lambda) \to pp$ *generates the public parameters required for system initialization including cryptographic keys and common reference strings;* $\mathsf{Issue}(pp, sk_I, \mathsf{attrs}) \to \mathsf{cred}$ *enables authorized issuers to create credentials encoding specific attributes while maintaining cryptographic integrity;* $\mathsf{Show}(pp, \mathsf{cred}, \mathsf{policy}) \to \pi$ *allows credential holders to generate zero-knowledge proofs demonstrating policy satisfaction without revealing unnecessary information; and* $\mathsf{Verify}(pp, \pi, \mathsf{policy}) \to \{0, 1\}$ *enables verifiers to check proof validity against specified policies while preserving holder privacy.*

The system must satisfy **correctness** (honest proofs verify), **soundness** (invalid credentials cannot generate valid proofs), and **unlinkability** (multiple uses of the same credential are indistinguishable).

The foundational work of Camenisch and Lysyanskaya [1] established the theoretical framework for anonymous credential systems, introducing efficient selective disclosure mechanisms that allow users to prove possession of specific attributes without revealing their complete identity. Their construction, known as the CL signature scheme, provides strong unlinkability guarantees ensuring that multiple uses of the same credential cannot be correlated by verifiers. This seminal work laid the groundwork for subsequent research into practical anonymous credential deployments, including extensions for revocation handling and multi-issuer scenarios.

## 2.2 Zero-Knowledge Proofs

We employ zero-knowledge proofs (ZKPs) to prove statements about secret data (e.g., accumulator witnesses or credential attributes) without revealing the secrets themselves. We review the foundational notions and constructions that underlie the proofs used in this work [2], [3].

**Definition 2** (Zero-knowledge proof of membership)**.** *Let* $R \subseteq \mathcal{X} \times \mathcal{W}$ *be a polynomial-time decidable relation and let* $L_R = \{x \in \mathcal{X} : \exists w \in \mathcal{W}. (x, w) \in R\}$*. An interactive protocol* $\langle P, V \rangle$ *for statements* $x \in L_R$ *is a zero-knowledge proof if for security parameter* $\lambda$ *it satisfies:*

- ***Completeness:*** *If* $(x, w) \in R$ *and* $P, V$ *are honest, then* $\Pr[\langle P(x, w), V(x) \rangle = 1] = 1 - \mathsf{negl}(\lambda)$.
- ***Soundness:*** *For any probabilistic polynomial-time (PPT)* $P^*$ *and any* $x \notin L_R$*,* $\Pr[\langle P^*(x), V(x) \rangle = 1] \leq \mathsf{negl}(\lambda)$.
- ***Zero-knowledge:*** *There exists a PPT simulator* $\mathsf{Sim}$ *such that for all* $x \in L_R$*, the simulated transcript* $\mathsf{Sim}(x)$ *is computationally indistinguishable from a real transcript of* $\langle P(x, w), V(x) \rangle$.

*If, in addition, there exists a PPT* extractor $\mathsf{Ext}^{P^*}(x)$ *that outputs* $w$ *such that* $(x, w) \in R$ *whenever* $P^*$ *convinces* $V$ *(beyond a knowledge error* $\kappa$*), then the protocol is a (zero-knowledge)* proof/argument of knowledge *[3].*

**Σ-protocols (three-move proofs).** Many classical ZK proofs are three-move Σ-protocols with structure commit–challenge–response and with *special soundness* and *honest-verifier zero-knowledge (HVZK).* As a canonical example, consider a proof of knowledge of a discrete

logarithm in a cyclic group $\mathbb{G}$ of prime order $q$ with generator $g$. The public statement is $y = g^x$ for secret $x \in \mathbb{Z}_q$.

$$\begin{aligned}
\text{Commit: } P : r \xleftarrow{\$} \mathbb{Z}_q,\ a \leftarrow g^r \in \mathbb{G},\ \text{send } a.\\
\text{Challenge: } V : c \xleftarrow{\$} \mathbb{Z}_q,\ \text{send } c.\\
\text{Response: } P : z \leftarrow r + cx \bmod q,\ \text{send } z.\\
\text{Verify: } V : \text{accept iff } g^z = a\, y^c.
\end{aligned}$$

**Completeness** is immediate. **Special soundness:** given two accepting transcripts $(a, c, z)$ and $(a, c', z')$ with $c \neq c'$, one can extract

$$x = (z - z')(c - c')^{-1} \bmod q.$$

**HVZK:** a simulator samples $(c, z) \xleftarrow{\$} \mathbb{Z}_q^2$ and sets $a \leftarrow g^z y^{-c}$, producing an indistinguishable transcript. These properties yield a proof (or argument) of knowledge under standard assumptions [3].

**From interactive to non-interactive.** A $\Sigma$-protocol can be made non-interactive via the Fiat–Shamir transform by replacing the verifier's random challenge with $c \leftarrow H(x, a)$ for a hash function modeled as a random oracle, yielding a single-message proof $\pi$ verifiable by recomputing the check [3]. This is the prevailing paradigm for building practical non-interactive ZK proofs (NIZKs) over algebraic statements.

**Composition and richer statements.** $\Sigma$-protocols natively support efficient logical composition. For conjunctive statements (AND), the prover runs protocols for all claims and the verifier checks all responses. For disjunctive statements (OR), the prover can simulate all but one branch and produce one real response, yielding zero-knowledge OR-proofs; range proofs and set-membership proofs are built from these patterns [3], [4].

**Arithmetic-circuit statements.** Many applications reduce to proving knowledge of a witness satisfying an arithmetic circuit over a finite field (e.g., correctness of accumulator updates or credential predicates). Modern ZK arguments for arithmetic circuits in the discrete-log setting provide efficient instantiations with succinct transcripts and fast verification [5].

**Use in accumulators and credentials.** In our setting, ZKPs are used to assert that a committed value is a member of a cryptographic accumulator, or that a credential satisfies policy predicates, without revealing the underlying identity or attributes. This aligns with classic constructions of anonymous credentials with revocation [1], [6] and recent analyses comparing accumulator- and Merkle-based revocation [7], as well as ZK-friendly accumulator designs for blockchain settings [8].

## 2.3 Diffie-Hellman Exponent

We summarize the hardness assumptions used by our accumulator and credential constructions, starting from exponent-sequence problems and moving to strong knowledge-type assumptions in bilinear groups [6]–[8].

**Diffie–Hellman Exponent ($n$-DHE).** Let $G$ be a cyclic group of prime order $q$ with generator $g$. Given

$$\left\{g, g^{\gamma}, g^{\gamma^2}, \ldots, g^{\gamma^n},\ g^{\gamma^{n+2}}, \ldots, g^{\gamma^{2n}}\right\}$$

for a hidden $\gamma \xleftarrow{\$} \mathbb{Z}_q$, the task is to compute $g^{\gamma^{n+1}}$. The $n$-DHE assumption asserts that no PPT algorithm can solve this with non-negligible probability. In bilinear settings, the pairing-based variant ($n$-BDHE) is commonly assumed and implies hardness for the source-group exponent sequence under standard reductions.

**Strong Diffie–Hellman ($n$-SDH).** Given

$$g,\ g^{x},\ g^{x^2},\ \ldots,\ g^{x^n} \in G,$$

it is infeasible to output any pair $(g^{1/(x+c)},\ c)$ for nonzero $c \in \mathbb{Z}_q$. The $n$-SDH assumption underlies many pairing-based credentials and accumulators; it was introduced by Boneh and Boyen and yields unforgeability of signatures/credentials derived from powers of a secret $x$.

**Hidden Strong Diffie–Hellman Exponent ($n$-HSDHE).** Let $G$ be a bilinear source group. Given

$$g,\ g^{x},\ u \in G, \quad \left\{g^{1/(x+\gamma^i)},\ g^{\gamma^i},\ u^{\gamma^i}\right\}_{i=1}^{n}, \quad \text{and} \quad \left\{g^{\gamma^i}\right\}_{i=n+2}^{2n},$$

it should be infeasible to compute a new tuple

$$\left(g^{1/(x+c)},\ g^{c},\ u^{c}\right)$$

for any fresh $c \in \mathbb{Z}_q$. This variant (introduced in our context) captures the inability to synthesize new division-in-exponent terms and linked powers without knowledge of $x$ and thereby underpins proofs of possession of accumulator witnesses in our scheme. It represents the weakest assumption under which we can prove security for the proposed construction, and it is incomparable to previously studied hidden-SDH variants.

**Relations and usage.** Intuitively, $n$-BDHE $\Rightarrow$ hardness of the corresponding $n$-DHE sequence problem in the source group; $n$-SDH enables sound credential issuance/unforgeability; and the $n$-HSDHE variant provides knowledge-style guarantees for witness possession. These assumptions align with prior accumulator-based revocation and credential systems [6]–[8] and are compatible with modern ZK proof systems for arithmetic circuits in the discrete-log setting [5].

## 2.4 Diffie-Hellman Exponent

**Diffie-Hellman Exponent ($n$-DHE) assumption**: The $n$-DHE problem in a group $G$ of prime order $q$ is defined as follows. Given the input

$$\{g, g^{\gamma}, g^{\gamma^2}, \ldots, g^{\gamma^n}, g^{\gamma^{n+2}}, \ldots, g^{\gamma^{2n}}\} \in G^{2n},$$

where $\gamma \xleftarrow{R} \mathbb{Z}_q$, the objective is to output $g^{\gamma^{n+1}}$. The $n$-DHE assumption states that this problem is *hard to solve*. This assumption is the basis for the accumulator construction itself, and the security of the new accumulator scheme's proof of knowledge also relies on it. The $n$-DHE assumption for a group $G$ with a bilinear pairing is implied by the $n$-Bilinear Diffie-Hellman Exponent ($n$-BDHE) assumption.

**Strong Diffie-Hellman ($n$-SDH) assumption**: On input

$$g, g^x, g^{x^2}, \ldots, g^{x^n} \in G,$$

it is computationally infeasible to output a tuple

$$\left(g^{1/(x+c)}, c\right).$$

The unforgeability of credentials in the scheme is based on this Strong Diffie-Hellman assumption. This assumption was originally introduced by Boneh and Boyen.

**Hidden Strong Diffie-Hellman Exponent ($n$-HSDHE) assumption**: Given

$$g, g^x, u \in G, \quad \left\{g^{1/(x+\gamma^i)}, g^{\gamma^i}, u^{\gamma^i}\right\}_{i=1}^{n}, \quad \text{and} \quad \left\{g^{\gamma^i}\right\}_{i=n+2}^{2n},$$

it is infeasible to compute a new tuple

$$\left(g^{1/(x+c)}, g^c, u^c\right).$$

This is a *variant* of the Hidden Strong Diffie-Hellman assumption, specifically introduced in this paper. It is the *weakest assumption* under which the proposed scheme can be proven secure. The proof of possession of an accumulator witness for a credential relies on this new $n$-HSDHE assumption. The paper notes that this assumption is, as of its writing, incomparable to the original Hidden Strong Diffie-Hellman assumption.

## 2.5 Ethereum Virtual Machine (EVM)

The Ethereum Virtual Machine (EVM) is a decentralized, stack-based virtual machine that executes smart contract bytecode consistently across all Ethereum nodes. The EVM provides a deterministic execution environment that ensures identical results regardless of the node or platform executing the code.

**Architecture and Execution Model** The EVM operates as a stack-based machine with the following key characteristics:

$$\text{Stack Depth} = 1024 \text{ items maximum} \tag{1}$$
$$\text{Word Size} = 256 \text{ bits (32 bytes)} \tag{2}$$
$$\text{Memory Model} = \text{Linear, expandable byte array} \tag{3}$$
$$\text{Storage Model} = \text{Key-value store (256-bit keys/values)} \tag{4}$$

**Gas Mechanism** The EVM employs a gas-based pricing model to prevent infinite loops and ensure network security:

$$\text{Gas Cost} = \sum_i \text{OpCode}_i \times \text{GasPrice}_i \tag{5}$$
$$\text{Transaction Cost} = \text{Gas Used} \times \text{Gas Price} \tag{6}$$
$$\text{Block Gas Limit} = \text{Maximum gas per block} \tag{7}$$

**Smart Contract Execution** Smart contracts in the EVM follow a deterministic execution model. The deployment phase stores contract bytecode on-chain with a unique address. During execution, transactions trigger contract functions with input parameters. State changes occur as contract storage is modified based on function logic. Finally, event emission generates logs for off-chain consumption.

**Precompiled Contracts** The EVM includes precompiled contracts for cryptographic operations:

$$\text{ECRECOVER} = \text{ECDSA signature recovery (3,000 gas)} \tag{8}$$
$$\text{SHA256} = \text{SHA256 hashing (60 + 12 gas per word)} \tag{9}$$
$$\text{RIPEMD160} = \text{RIPEMD160 hashing (600 + 120 gas per word)} \tag{10}$$
$$\text{BN256} = \text{Bilinear pairing operations (variable gas)} \tag{11}$$

**Integration with Zero-Knowledge Proofs** Our anonymous credential system leverages the EVM for proof verification, where smart contracts verify zk-SNARK proofs on-chain. The system also uses the EVM for state management, storing accumulator values and revocation state. Policy enforcement is executed through application-specific verification logic, while event logging records credential verification events for audit purposes.

**Gas Optimization Strategies** Efficient EVM execution is critical for cost-effective credential verification:

$$\text{Calldata Optimization} = \text{Minimize transaction data size} \tag{12}$$
$$\text{Storage Optimization} = \text{Use efficient data structures} \tag{13}$$
$$\text{Computation Optimization} = \text{Leverage precompiled contracts} \tag{14}$$
$$\text{Batch Processing} = \text{Amortize gas costs across multiple operations} \tag{15}$$

**Security Considerations** The EVM execution environment introduces specific security considerations:

**Definition 3** (Reentrancy Protection)**.** *Smart contracts must prevent reentrant calls that could exploit state inconsistencies during execution.*

**Definition 4** (Gas Limit Protection)**.** *Operations must respect block gas limits to prevent transaction failures and ensure reliable execution.*

The EVM's deterministic execution model and gas-based resource management provide the foundation for our decentralized anonymous credential verification system, enabling trustless proof validation while maintaining cost efficiency and security.

## 2.6 Ethereum and Gas System

This section formalizes the execution and gas accounting model used in our analysis. A transaction with gas limit $g$ and price $\pi$ executes in the EVM, consuming gas according to opcode schedules, calldata size, and persistent storage access. The fee is $g_{\text{used}} \cdot \pi$. We focus on the factors that materially affect our verifier contracts.

First, calldata is charged per byte. In accumulator-based verification, witnesses are constant-size, so calldata is stable; in Merkle-based schemes, witnesses require $O(\log n)$ sibling hashes and grow with the revocation set size $n$. Second, compute costs depend on precompiles. Pairing checks on BN254 (used by many zk-SNARK verifiers) have fixed gas schedules, whereas hashing (Keccak) is relatively cheap per invocation but accumulates along a long path; zk-friendly hashes (e.g., Poseidon) reduce prover cost but, when executed on-chain, have custom costs. Third, storage writes (SSTORE) are expensive; we therefore keep on-chain state minimal, publishing only the current revocation commitment (accumulator value or Merkle root) and avoiding per-verification writes.

Methodology. To evaluate gas, we report: (i) verification gas (execution only); (ii) calldata bytes; and (iii) amortized gas per issuance and revocation update over a batch size $B$. We normalize across networks by pinning client versions and reporting opcode-equivalent metrics. These measurements underpin the comparisons presented in Section 9.

## 2.7 Cryptographic Accumulators

Cryptographic accumulators provide a space-efficient mechanism for representing large sets while enabling efficient membership and non-membership proofs. These structures are fundamental to our anonymous credential system's revocation mechanism, allowing compact representation of revoked credentials with constant-size proofs.

**Mathematical Foundation** An accumulator consists of a set $S = \{x_1, x_2, \ldots, x_n\}$ and a compact representation $acc$ that enables efficient membership verification. For a cryptographic accumulator, the following properties must hold:

$$\text{Compactness: } |acc| = O(1) \text{ regardless of } |S| \tag{16}$$

$$\text{Membership Proof: } \pi_x = \text{Prove}(acc, x) \text{ for } x \in S \tag{17}$$

$$\text{Non-membership Proof: } \pi_x = \text{Prove}(acc, x) \text{ for } x \notin S \tag{18}$$

$$\text{Verification: } \text{Verify}(acc, x, \pi_x) \in \{0, 1\} \tag{19}$$

**Accumulator Types** Our system implements three distinct accumulator types, each offering different trade-offs between efficiency and functionality:

1. **Elliptic Curve Accumulators**: Based on bilinear pairings, providing constant-time verification and logarithmic witness size
2. **Merkle Tree Accumulators**: Hash-based structures with logarithmic verification time and efficient batch updates
3. **List Accumulators**: Simple linear structures suitable for small-scale deployments

**Security Properties** Cryptographic accumulators must satisfy the following security requirements:

**Definition 5** (Collision Resistance)**.** *No PPT adversary can find distinct sets $S, S'$ with $S \neq S'$ that produce the same accumulator value acc.*

**Definition 6** (Soundness)**.** *No PPT adversary can produce a valid membership proof for an element not in the accumulated set, except with negligible probability.*

**Revocation Integration** In our anonymous credential system, accumulators serve as the primary revocation mechanism. Credential commitments are accumulated into the revocation set, and holders must prove non-membership to demonstrate credential validity. This approach enables efficient revocation without requiring credential reissuance or complex distributed protocols.

# 3 System Model and Threat Analysis

The security and efficiency of our anonymous credential system depend critically on precise modeling of system entities, their capabilities, and the threat environment in which they operate. This section establishes the foundational assumptions that inform our protocol design and enables rigorous security analysis.

## 3.1 System Entities and Operational Environment

Our system operates within a multi-party environment where distinct entities possess different capabilities and trust relationships. The **Credential Issuers** represent the highest trust tier, possessing long-term cryptographic signing keys that anchor the system's security properties. These entities implement credential policies, validate attribute claims, and maintain authoritative revocation state. Issuers operate primarily off-chain to minimize operational costs while publishing minimal state commitments to the blockchain for transparency and non-repudiation.

**Credential Holders** occupy an intermediate trust position where they possess valid credentials but must prove their authenticity and continued validity to verifiers. Holders maintain cryptographic secrets enabling them to generate zero-knowledge proofs without revealing sensitive attribute information. The system assumes holders are computationally bounded but may attempt to use revoked credentials or forge proof statements within cryptographic limitations.

**Verifying Parties** implement policy enforcement through smart contracts that validate credential proofs according to application-specific requirements. These entities are assumed to be honest but curious—they correctly execute verification logic while potentially attempting to extract information about credential holders beyond what is explicitly revealed through policy compliance.

The **Blockchain Infrastructure** provides the foundation for our trust model through authenticated execution and immutable logging capabilities. We assume a permissionless, gas-metered ledger that ensures transaction ordering and state consistency through consensus mechanisms. The blockchain enables transparent revocation state management while providing resistance against single points of failure that could compromise system availability.

## 3.2 Threat Model and Adversarial Capabilities

Our threat analysis considers a powerful adversary $\mathcal{A}$ capable of compromising multiple system components while respecting fundamental cryptographic limitations. The adversary may control network communications, enabling message reordering, selective delivery, and front-running attacks that exploit blockchain transaction ordering. This network-level control extends to observing all on-chain state transitions, transaction contents, and event emissions, providing complete visibility into public system operation.

The adversary may corrupt a subset of credential holders, gaining access to their credentials, private keys, and witness information. However, corrupted holders cannot forge credentials for uncorrupted identities or generate valid proofs for revoked credentials beyond cryptographic breaking events. This limitation reflects our system's reliance on cryptographic commitment schemes and zero-knowledge proof systems for security.

Verifying parties may be compromised to submit arbitrary inputs to verification contracts or attempt to extract additional information from proof verification processes. However, corrupted verifiers cannot forge credential proofs or violate the zero-knowledge properties of our proof systems. This assumption enables our system to maintain privacy guarantees even when some verifying parties act maliciously.

Significantly, our threat model excludes issuer key compromise as an explicit consideration, instead treating issuer security as a prerequisite for system operation. This design choice reflects the fundamental role of issuers as trust anchors—issuer compromise would require complete system reinitialization rather than graceful recovery. However, our architecture minimizes issuer exposure by enabling issuers to operate primarily off-chain with minimal blockchain interaction.

## 3.3 Security Requirements and Formal Properties

The security properties required by our system reflect the fundamental privacy and authenticity goals of anonymous credentials while accounting for the unique constraints of blockchain deployment environments.

**Definition 7** (Soundness)**.** *No PPT $\mathcal{A}$ convinces a verifier to accept a proof for a revoked or never-issued credential except with negligible probability* $\text{negl}(\lambda)$.

**Definition 8** (Unlinkability)**.** *Across two challenge experiments in which $\mathcal{A}$ chooses two credentials with identical disclosed attributes, transcripts (calldata, on-chain state, events) are computationally indistinguishable.*

**Definition 9** (Freshness)**.** *At verification time, the revocation witness corresponds to the latest published commitment. Formally, if $C_t$ is the commitment after the last revocation before block $t$, then accepted proofs must verify against $C_t$.*

## 3.4 Attacks and Mitigations

Stale-witness attacks are prevented by binding proofs to the current commitment. Replay is mitigated by including a session nonce or block height in public inputs. Front-running does not break soundness but can leak timing; batching and commitment rotation policies alleviate this.

## 3.5 System Architecture

Our architecture separates concerns across three distinct layers: off-chain credential management, on-chain state commitment, and zero-knowledge proof verification. This separation enables scalable credential issuance while maintaining strong privacy guarantees and minimizing blockchain resource consumption.

**Off-chain Layer:** Credential issuers operate off-chain, generating signed credentials and maintaining revocation data structures (elliptic curve accumulators or Merkle trees). This design minimizes on-chain storage costs while preserving issuer autonomy and credential privacy.

**On-chain State Layer:** The blockchain maintains only minimal state commitments accumulator values or Merkle roots—that summarize the current revocation status. Smart contracts store verification keys and enforce policy compliance during proof verification.

**Verification Layer:** Zero-knowledge proofs bridge off-chain credential data with on-chain verification, enabling holders to prove credential validity and non-revocation status without revealing sensitive attributes or compromising unlinkability.

**Trust Model:** Our design assumes authenticated issuers (via signature keys), semi-honest verifiers, and an append-only blockchain with eventual finality. We explicitly address front-running resistance and replay protection where security-critical.

# 4 Protocols: Issuance, Revocation, Verification

We formalize the three core protocols that implement our anonymous credential system, providing detailed algorithmic specifications and security analysis for each component.

## 4.1 Credential Issuance

The credential issuance protocol enables issuers to generate anonymous credentials containing user attributes while providing the necessary cryptographic material for later proof generation and revocation checks.

**Algorithm 1:** Credential Issuance Protocol

**Input:** Issuer secret key $sk_I$, user attributes $\mathsf{attrs} = (a_1, \ldots, a_k)$, accumulator state $\mathsf{Acc}$

**Output:** Credential $\mathsf{cred}$, revocation token $\mathsf{token}$

**Setup Phase:**
$pp \leftarrow \mathsf{Setup}(1^\lambda)$ `// Generate system parameters`
$(pk_I, sk_I) \leftarrow \mathsf{KeyGen}(pp)$ `// Issuer key generation`
$(\mathsf{Acc}_0, aux_0) \leftarrow \mathsf{Acc.Setup}(pp)$ `// Initialize accumulator`
**Issuance Phase:**
$r \xleftarrow{\$} \mathbb{Z}_p$ `// Choose randomness`
$\mathsf{serial} \leftarrow H(\mathsf{attrs} \| r \| \text{timestamp})$ `// Generate unique serial`
$\sigma \leftarrow \mathsf{Sign}(sk_I, \mathsf{attrs} \| \mathsf{serial})$ `// Sign attributes`
$(\mathsf{Acc}', w_{\mathsf{serial}}) \leftarrow \mathsf{Acc.Add}(\mathsf{Acc}, \mathsf{serial})$ `// Add to accumulator`
**Credential Formation:**
$\mathsf{cred} \leftarrow (\mathsf{attrs}, \mathsf{serial}, \sigma, r)$
$\mathsf{token} \leftarrow (\mathsf{serial}, w_{\mathsf{serial}})$ `// Revocation token`
**return** $(\mathsf{cred}, \mathsf{token})$

The issuer maintains the accumulator state $\mathsf{Acc}$ containing all active credential serials. The revocation token enables efficient membership proofs during verification while allowing the issuer to revoke credentials by removing serials from the accumulator.

## 4.2 Off-Chain Operations and State Synchronization

The design philosophy underlying our anonymous credential system emphasizes the separation of computationally intensive operations from blockchain execution, enabling scalable credential management while preserving strong cryptographic guarantees. This architectural decision necessitates sophisticated coordination mechanisms between off-chain computation

and on-chain state commitments, particularly in the context of credential issuance and revocation management.

### 4.2.1 Off-Chain Credential Generation and Witness Management

Credential issuance operates entirely in off-chain environments where issuers maintain complete control over the cryptographic operations required for credential generation. This design choice reflects both efficiency considerations—avoiding expensive on-chain signature operations—and privacy requirements that demand attribute confidentiality throughout the credential lifecycle.

The credential generation process begins with attribute validation and policy compliance checking performed by issuer systems. Rather than recording these sensitive operations on the public ledger, issuers maintain private audit logs that enable internal compliance verification while preserving holder privacy. The cryptographic binding between attributes and holder identity occurs through sophisticated commitment schemes that enable later zero-knowledge proof generation without revealing underlying attribute values.

Witness management represents one of the most complex aspects of off-chain operations, requiring careful coordination between issuer state updates and holder credential maintenance. When issuers modify accumulator values or Merkle roots through revocation operations, existing credential holders must update their cryptographic witnesses to maintain proof validity. Our implementation addresses this challenge through an event-driven architecture where on-chain revocation events trigger automatic witness refresh protocols.

The witness refresh mechanism operates through a combination of cryptographic optimization and efficient communication protocols. For elliptic curve accumulators, witness updates leverage the multiplicative properties of accumulator mathematics, enabling holders to update their witnesses through local computation using publicly available update parameters. Merkle tree witnesses require a different approach where holders download updated tree branches and recompute authentication paths, optimized through intelligent caching and incremental update protocols.

### 4.2.2 Batching Strategies and State Consistency

The fundamental trade-off between operational efficiency and state freshness drives the design of our batching and synchronization protocols. Frequent on-chain updates minimize the window during which revoked credentials remain potentially valid, but incur substantial gas costs that may render the system economically impractical. Conversely, infrequent updates reduce operational costs but increase security risks associated with delayed revocation enforcement.

Our batching strategy employs a sophisticated priority system that categorizes revocation events based on security criticality and urgency requirements. Emergency revocations—such as those triggered by credential compromise or legal mandates—receive immediate processing through dedicated high-priority transaction channels. These emergency transactions bypass normal batching protocols and commit state changes within predetermined time bounds, typically within 5-10 minutes of revocation initiation.

Routine revocations accumulate in batch queues organized by accumulator type and expected gas consumption. The batching algorithm optimizes for total gas efficiency while respecting maximum staleness constraints defined by application-specific service level agreements. For elliptic curve accumulators, batch sizes typically range from 32 to 128 revocations to balance the fixed costs of accumulator operations against per-revocation processing overhead.

Merkle tree batching follows different optimization criteria focused on tree locality and update efficiency. The system analyzes pending revocations to identify clusters of credentials whose tree positions share common ancestral nodes, enabling batch updates that minimize the number of tree modifications required. This approach can reduce per-revocation gas consumption by 60-80% compared to individual processing while maintaining acceptable revocation latency.

### 4.2.3 State Synchronization and Consistency Guarantees

The coordination between off-chain computation and on-chain state requires careful attention to consistency guarantees and atomic update semantics. Our system implements a multi-phase commit protocol that ensures credential holders never encounter inconsistent views of accumulator state, even during concurrent revocation and verification operations.

The synchronization protocol operates through a versioned state model where each accumulator update receives a unique version identifier committed to the blockchain alongside the accumulator value or Merkle root. Credential holders track their witness versions and automatically detect when updates are required through on-chain event monitoring. This versioning approach enables efficient incremental updates while providing clear consistency semantics for application developers.

Transaction ordering presents another critical consideration for maintaining system consistency. Our implementation addresses potential race conditions between revocation transactions and verification attempts through careful transaction sequencing and block confirmation requirements. Verification contracts enforce minimum confirmation depths for accumulator updates, preventing attacks that exploit blockchain reorganizations to temporarily revive revoked credentials.

The system also implements comprehensive rollback and recovery mechanisms to handle exceptional conditions such as failed batches or temporary blockchain congestion. When batch transactions fail due to gas limit exhaustion or other execution errors, the system automatically partitions the batch into smaller units and retries with adjusted gas parameters. This approach ensures that revocation processing remains resilient against network-level disruptions while maintaining predictable service guarantees.

### 4.2.4 Economic Analysis and Cost Optimization

The economic implications of off-chain versus on-chain operation allocation significantly influence system design decisions and deployment strategies. Our analysis reveals that credential issuance costs can be reduced by 95% through off-chain execution, with typical per-credential costs dropping from 150,000 gas on-chain to negligible computational costs off-chain.

Revocation batching demonstrates even more dramatic cost improvements, with batch efficiency curves showing logarithmic cost reduction as batch sizes increase. Single revocations consume 85,000-420,000 gas depending on accumulator type, while 64-element batches reduce per-revocation costs to 15,000-25,000 gas. These efficiency gains prove essential for applications requiring frequent revocation operations or serving large user populations.

The trade-offs between batch frequency and user experience require careful calibration based on application-specific requirements. High-frequency applications such as access control systems benefit from smaller batch sizes with higher update frequencies, accepting increased per-revocation costs in exchange for reduced revocation latency. Conversely, applications with longer credential lifetimes such as educational credentialing can optimize for larger batch sizes and longer update intervals.

Storage cost optimization provides another dimension for economic analysis, particularly regarding the accumulation of on-chain state over time. Our implementation employs state pruning strategies that archive historical accumulator values while maintaining only recent state necessary for current verification operations. This approach reduces long-term storage costs while preserving the ability to verify historical credential validity when required for audit or compliance purposes.

## 4.3 On-chain Verification

The verification protocol enables smart contracts to validate anonymous credential proofs while checking revocation status against the current accumulator state, without learning the credential holder's identity or attributes.

**Algorithm 2:** Smart Contract Verification

**Input:** Zero-knowledge proof $\pi$, public statement $x$, current accumulator $\mathsf{Acc}_{current}$
**Output:** Verification result $\{0, 1\}$

**Proof Validation:**
$vk \leftarrow \text{loadVerificationKey}()$ `// Load stored verification key`
$\text{valid}_{zk} \leftarrow \mathsf{Verify}(vk, x, \pi)$ `// Verify zk-SNARK proof`
**if** $valid_{zk} = 0$ **then**
  **return** 0 `// Invalid proof`
**Revocation Check:**
**switch** *accumulator type* **do**
  **case** *Elliptic Curve* **do**
    $(\mathsf{serial}, w) \leftarrow \text{extractWitness}(\pi)$
    $\text{valid}_{acc} \leftarrow \mathsf{Acc.Verify}(\mathsf{Acc}_{current}, \mathsf{serial}, w)$
  **case** *Merkle Tree* **do**
    $(\mathsf{serial}, \text{path}) \leftarrow \text{extractMerklePath}(\pi)$
    $\text{valid}_{acc} \leftarrow \text{verifyMerklePath}(\mathsf{Acc}_{current}, \mathsf{serial}, \text{path})$
**Policy Enforcement:**
$\text{policy}_{valid} \leftarrow \text{checkPolicy}(x)$ `// Check application-specific policy`
**return** $valid_{zk} \wedge valid_{acc} \wedge policy_{valid}$

### 4.3.1 Gas Optimization Strategies and Economic Analysis

The economic viability of anonymous credentials on Ethereum fundamentally depends on minimizing verification costs while maintaining security properties. Our optimization strategy addresses three primary cost drivers: cryptographic operations, data storage patterns, and computational redundancy.

**Precompiled Contract Utilization:** The most significant optimization comes from leveraging Ethereum's precompiled contracts for elliptic curve operations. Standard elliptic curve point additions and scalar multiplications in pure Solidity consume approximately 80,000 gas due to arbitrary-precision arithmetic. However, Ethereum's bn256Add (0x06), bn256Mul (0x07), and bn256Pairing (0x08) precompiles reduce these costs to 500, 40,000, and 100,000 gas respectively for the alt_bn128 curve. This represents a 90% cost reduction for cryptographic primitives, making elliptic curve accumulators economically competitive despite their computational complexity.

**Batch Verification Economics:** Batch verification exploits the amortization of fixed costs across multiple proof validations. While individual proof verification incurs setup

costs of approximately 15,000 gas for loading verification keys and initializing cryptographic contexts, batched verification spreads these costs across $k$ proofs. Our analysis shows that batch sizes of 8-16 proofs achieve optimal gas efficiency, balancing per-proof cost reduction with increased calldata expenses. For applications processing high verification volumes, this optimization can reduce per-verification costs by 35-50%.

**Storage Access Patterns:** The choice between contract storage and calldata for verification keys significantly impacts gas consumption patterns. Storing verification keys in contract storage incurs a one-time cost of 20,000 gas per 32-byte slot during deployment, with subsequent access costs of only 800 gas per SLOAD operation. In contrast, passing verification keys as calldata costs 16 gas per non-zero byte but must be paid on every function call. For verification keys of typical size (1-2 KB), storage becomes cost-effective after approximately 50 verification operations, making it the preferred approach for production deployments.

**Merkle Tree Path Optimization:** Merkle tree verification costs scale with the number of hash operations required to reconstruct the root. Our optimization caches frequently accessed intermediate nodes in contract storage, reducing redundant hash computations for common verification paths. This approach is particularly effective for systems with skewed access patterns, where certain credential types are verified more frequently than others. By maintaining a cache of the top three tree levels, we reduce average verification costs by 25-30% while incurring only modest storage overhead.

### 4.4 Ledger

We assume an Ethereum-like ledger with account-based state that provides the foundational infrastructure for our anonymous credential system. The ledger maintains global state, executes smart contracts, and ensures consensus through a proof-of-stake mechanism. Below we detail the key components and algorithms.

## 5 Cryptographic Signature Schemes and Performance Analysis

Anonymous credential systems fundamentally depend on digital signature schemes to establish cryptographic integrity and authenticity. Our comprehensive evaluation examines multiple signature schemes across different elliptic curve implementations, with particular focus on their integration with zero-knowledge proof systems and blockchain deployment requirements.

### 5.1 Signature Scheme Taxonomy and Selection Criteria

The selection of appropriate signature schemes for anonymous credentials involves complex trade-offs between security assumptions, computational efficiency, proof generation overhead, and blockchain compatibility. Our analysis examines four primary categories of signature schemes, each offering distinct advantages for specific deployment scenarios.

**EdDSA on Curve25519** offers superior performance characteristics for zero-knowledge applications through its algebraic structure and deterministic signature generation. The scheme achieves similar security levels to ECDSA while providing more efficient circuit implementations, particularly for applications requiring frequent signature verification within arithmetic circuits. Our measurements demonstrate 40-60% reduction in constraint count

compared to ECDSA implementations, though at the cost of reduced blockchain ecosystem support.

**BLS Signatures on BLS12-381** provide unique aggregation properties that enable compact multi-signature constructions and batch verification capabilities essential for scalable credential systems. The scheme's pairing-based construction aligns naturally with many zk-SNARK implementations, particularly those using the same elliptic curve for their underlying arithmetic. However, BLS signatures require expensive pairing operations for verification, creating performance bottlenecks in resource-constrained environments.

**BBS+ Signatures for Selective Disclosure** represent the most advanced option for anonymous credential applications, offering native support for selective attribute disclosure without requiring complex zero-knowledge constructions. The scheme enables credential holders to reveal arbitrary subsets of signed attributes while maintaining privacy for undisclosed information. This capability proves essential for privacy-preserving applications where minimal information disclosure is required.

## 5.2 BLS12-381 and BBS+ Signature Implementation

Our prototype implementation leverages BLS12-381 as the primary elliptic curve for cryptographic operations, chosen for its optimal balance between security, efficiency, and zero-knowledge proof compatibility. The curve provides 128-bit security levels through a 381-bit prime field, enabling efficient pairing computations while maintaining compatibility with modern zk-SNARK implementations.

**BBS+ Signature Integration:** The BBS+ signature scheme implementation follows the Internet Research Task Force (IRTF) Crypto Forum Research Group (CFRG) draft specification for BBS signatures [9]. Our implementation extends the basic BBS framework with additional features required for anonymous credential applications, including:

**Algorithm 3:** BBS+ Signature Generation for Credential Issuance

**Input:** Issuer secret key $sk$, credential attributes $\mathsf{attrs} = (a_1, \ldots, a_n)$, credential holder public key $pk_H$

**Output:** BBS+ signature $\sigma$, commitment $C$

`// Setup phase`

$(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, p, g_1, g_2) \leftarrow \text{BilinearSetup}(1^\lambda)$

$(h_1, \ldots, h_n) \leftarrow \text{GenerateAttributeGenerators}(n)$

`// Commitment generation`

$r \xleftarrow{\$} \mathbb{Z}_p$

$C \leftarrow g_1^r \prod_{i=1}^{n} h_i^{a_i}$

`// Signature computation`

$s \xleftarrow{\$} \mathbb{Z}_p$

$B \leftarrow g_1^s$

$A \leftarrow (B \cdot C)^{1/(sk+s)}$

`// Credential binding`

$\text{binding} \leftarrow H(pk_H \| C \| A \| B)$

$\sigma \leftarrow (A, B, \text{binding})$

**return** $(\sigma, C)$

The BBS+ implementation provides several critical advantages over traditional signature schemes. **Selective Disclosure** enables credential holders to prove possession of validly signed attributes without revealing undisclosed information, eliminating the need for complex zero-knowledge constructions for basic attribute hiding. **Unlinkable Presentations**

ensure that multiple uses of the same credential cannot be linked by verifiers, providing strong privacy guarantees even across multiple verification contexts. **Batch Verification** capabilities enable efficient verification of multiple credentials simultaneously, reducing computational overhead for high-volume verification scenarios.

## 5.3 Elliptic Curve Performance Comparison

Our empirical evaluation compares signature generation, verification, and zero-knowledge circuit integration across multiple elliptic curve implementations. The evaluation encompasses both standalone signature operations and integrated performance within zero-knowledge proof systems.

**Signature Generation Performance:** BLS12-381 based signatures require 2.1-3.4 milliseconds for generation, reflecting the computational overhead of pairing-friendly curve operations. ECDSA on secp256k1 achieves superior generation performance at 0.3-0.8 milliseconds, benefiting from optimized implementations and hardware acceleration. EdDSA on Curve25519 provides intermediate performance at 0.7-1.2 milliseconds while offering deterministic signature generation that simplifies testing and verification procedures.

**Verification Efficiency Analysis:** Verification performance varies significantly across schemes and deployment contexts. ECDSA verification requires 0.8-1.5 milliseconds for standalone operations, with blockchain-based verification consuming 21,000-35,000 gas through precompiled contracts (ecrecover). BLS signature verification demands 15-25 milliseconds due to pairing computations, but benefits from batch verification capabilities that amortize pairing costs across multiple signatures. EdDSA verification achieves optimal standalone performance at 0.4-0.9 milliseconds, though lacks native blockchain support requiring custom precompiles for on-chain verification.

**Zero-Knowledge Circuit Integration:** The integration of signature verification within arithmetic circuits reveals dramatic performance variations across different schemes. ECDSA verification requires 150,000-200,000 constraints due to field arithmetic complexity and point operations that must be implemented within the circuit field. EdDSA verification achieves significant efficiency improvements, requiring only 45,000-65,000 constraints through its algebraic structure that aligns better with circuit arithmetic. BLS signature verification within circuits demands 180,000-250,000 constraints for pairing operations, but enables advanced features like signature aggregation and batch verification that may justify the additional complexity.

## 5.4 IELTS Credential System Implementation

Our IELTS credential verification system demonstrates practical deployment of anonymous credentials using BLS12-381 signatures integrated with zero-knowledge proofs. The system addresses real-world requirements for English language proficiency verification while maintaining candidate privacy and enabling automated verification processes.

**System Architecture and Components:** The IELTS system comprises multiple interconnected components that collectively provide end-to-end credential lifecycle management. The **Issuer Component** implements BBS+ signature generation for credential creation, using official IELTS test center private keys to sign structured attribute data including listening, reading, writing, and speaking scores, test dates, and candidate identifiers. The **Holder Component** manages credential storage, witness generation, and zero-knowledge proof creation, enabling candidates to prove score requirements without revealing exact test results. The **Verifier Component** implements smart contract-based proof verification,

enabling automated qualification checking for universities, employers, and immigration authorities.

**Algorithm 4:** IELTS Credential Issuance Protocol

**Input:** Test scores $(L, R, W, S)$, candidate identity $ID$, test center key $sk_{TC}$
**Output:** Signed credential $\mathsf{cred}$, holder secret $r$

```
// Attribute encoding
```
$\mathsf{attrs} \leftarrow \text{EncodeAttributes}(L, R, W, S, ID, \text{timestamp})$
$\text{overall\_score} \leftarrow (L + R + W + S)/4$
```
// Holder secret generation
```
$r \xleftarrow{\$} \mathbb{Z}_p$
```
// Commitment creation
```
$C \leftarrow \text{PedersenCommit}(\mathsf{attrs}, r)$
```
// BBS+ signature generation
```
$\sigma \leftarrow \text{BBSSign}(sk_{TC}, \mathsf{attrs}, C)$
```
// Credential packaging
```
$\mathsf{cred} \leftarrow \{$
  scores: $(L, R, W, S)$,
  overall: overall_score,
  identity: $ID$,
  signature: $\sigma$,
  commitment: $C$,
  issuer_key: $pk_{TC}$
$\}$
**return** $(\mathsf{cred}, r)$

**Zero-Knowledge Proof Integration:** The IELTS system employs a custom Circom circuit that enables privacy-preserving score verification while maintaining cryptographic integrity. The circuit accepts private inputs including individual test scores and holder secrets, along with public inputs specifying minimum score requirements and credential validity constraints. The proof demonstrates that: (1) the holder possesses a validly signed credential from an authorized test center, (2) all individual scores meet specified minimum thresholds, (3) the overall average score satisfies requirements, (4) the credential has not expired, and (5) the credential belongs to the proving entity.

**Performance Measurements and Analysis:** Our comprehensive performance evaluation of the IELTS system reveals the practical implications of cryptographic design choices for real-world deployment scenarios. Credential issuance requires 45-80 milliseconds including BBS+ signature generation and commitment creation, enabling test centers to process hundreds of credentials per minute. Zero-knowledge proof generation demands 3.2-4.8 seconds due to circuit complexity and constraint count, representing a reasonable latency for non-interactive verification scenarios. On-chain verification consumes 280,000-320,000 gas for proof validation and state updates, corresponding to $12-18 transaction costs at typical Ethereum gas prices.

## 5.5 Comparative Analysis of Signature Schemes for Anonymous Credentials

Our systematic evaluation reveals that signature scheme selection significantly impacts system performance, security assumptions, and deployment complexity. The choice must balance theoretical security properties with practical deployment constraints including blockchain

compatibility, proof generation efficiency, and implementation complexity.

**Security Assumption Analysis:** Different signature schemes rely on varying cryptographic assumptions that directly impact system security guarantees. ECDSA security depends on the elliptic curve discrete logarithm problem (ECDLP) over secp256k1, providing well-understood security properties backed by extensive cryptanalytic research. EdDSA similarly relies on ECDLP over Curve25519, offering equivalent security levels with improved algebraic properties. BLS signatures require both ECDLP and the computational Diffie-Hellman problem in pairing-friendly groups, introducing additional complexity but enabling unique features like signature aggregation. BBS+ signatures build upon BLS security while adding requirements for secure commitment schemes and pairing computations, representing the most complex but feature-rich option.

**Implementation Complexity and Maintenance Overhead:** The practical deployment of signature schemes involves significant engineering overhead that extends beyond pure cryptographic operations. ECDSA implementations benefit from extensive library support and hardware acceleration, minimizing development and maintenance requirements. EdDSA provides similar implementation simplicity while offering deterministic signatures that simplify testing and debugging procedures. BLS implementations require specialized pairing libraries and careful parameter validation, increasing implementation complexity but providing advanced capabilities. BBS+ signatures demand the most sophisticated implementation efforts, requiring custom arithmetic operations and careful protocol engineering, but offer unmatched privacy-preserving capabilities.

**Blockchain Integration and Gas Efficiency:** The deployment of signature verification on blockchain platforms reveals significant performance variations that directly impact operational costs. ECDSA verification benefits from native precompile support across major blockchain platforms, consuming only 3,000 gas for signature verification operations. EdDSA verification requires custom precompiles or pure Solidity implementations, consuming 50,000-80,000 gas depending on implementation strategy. BLS signature verification demands expensive pairing operations consuming 150,000-200,000 gas per verification, though batch verification can amortize these costs. BBS+ verification requires the highest gas consumption at 200,000-300,000 gas per operation, but provides unique privacy properties that may justify the additional cost.

## 5.6 Deployment Recommendations and Use Case Analysis

The selection of appropriate signature schemes for anonymous credential systems must consider the complete operational environment including security requirements, performance constraints, privacy objectives, and economic considerations.

**High-Volume Verification Scenarios** such as educational credentialing, professional licensing, and government identity systems benefit from ECDSA or EdDSA implementations that prioritize verification efficiency over advanced privacy features. These schemes provide adequate security guarantees while minimizing computational and economic overhead for frequent verification operations.

**Privacy-Critical Applications** including medical records, financial credentials, and personal identity documents should prioritize BBS+ signatures despite their computational overhead. The selective disclosure capabilities and unlinkable presentations provide essential privacy properties that justify the additional implementation and operational complexity.

**Batch Processing Environments** such as immigration processing, bulk credential verification, and automated compliance checking can leverage BLS signature aggregation to achieve superior throughput despite higher individual verification costs. The ability to batch

verify multiple signatures simultaneously provides significant efficiency gains for high-volume operations.

**Resource-Constrained Deployments** including mobile applications, IoT devices, and embedded systems should prioritize EdDSA implementations that balance security, efficiency, and implementation simplicity. The deterministic signature generation and efficient verification make EdDSA optimal for environments with limited computational resources.

Our analysis demonstrates that no single signature scheme optimally addresses all anonymous credential requirements. Practical deployments must carefully evaluate their specific constraints and priorities to select appropriate cryptographic building blocks that balance security, efficiency, privacy, and implementation complexity within their operational environment.

# 6 Zero-Knowledge Proof Scheme

We present our zero-knowledge circuit designs for anonymous credential verification with revocation, providing both theoretical analysis and practical implementations.

## 6.1 Theoretical Foundations

Our construction builds on the following security properties and complexity characteristics:

**Security Properties:** *Soundness* ensures that no adversary can convince a verifier of credential validity without possessing a genuinely valid non-revoked credential, protecting against forgery and invalid proof acceptance. *Zero-Knowledge* guarantees that verification proofs reveal only the essential facts—credential validity and policy satisfaction—while hiding all other sensitive attributes and preventing information leakage. *Unlinkability* provides privacy protection by ensuring multiple uses of the same credential remain computationally indistinguishable, preventing tracking across different verification contexts. *Revocation Freshness* maintains system security by guaranteeing that revoked credentials cannot generate valid proofs after the next accumulator update, preventing stale credential exploitation.

**Complexity Analysis:** Let $n$ be the number of issued credentials and $k$ the number of revoked credentials. **EC Accumulators** provide constant-size witnesses ($O(1)$) and constant verification time independent of set size, but suffer from linear update costs ($O(n)$) that become prohibitive for large credential sets. **Merkle Trees** exhibit logarithmic scaling with $O(\log n)$ witness sizes and verification times, while achieving superior batch update performance at $O(k \log n)$ for $k$ simultaneous revocations. **zk-SNARKs** maintain constant verification time regardless of circuit complexity, though prover time scales linearly with constraint count $O(|C|)$, creating trade-offs between circuit efficiency and proving performance.

## 6.2 Unified Revocation Interface

We abstract revocation mechanisms behind a common interface exposing: The unified interface abstracts different revocation mechanisms through three key components: a **public commitment** $C$ that cryptographically commits to the current revocation set state, enabling verifiers to reference a consistent snapshot without revealing set contents; a **membership witness** $w$ that proves a specific credential serial number has not been revoked, with witness size and structure varying by mechanism; and a **verification relation** $R(C, \text{serial}, w) \to \{0, 1\}$ that determines whether the provided witness correctly demonstrates non-membership for the given serial against the committed revocation set.

This abstraction enables seamless switching between accumulator types while maintaining circuit compatibility.

## 6.3 ZKP Using Bulletproofs

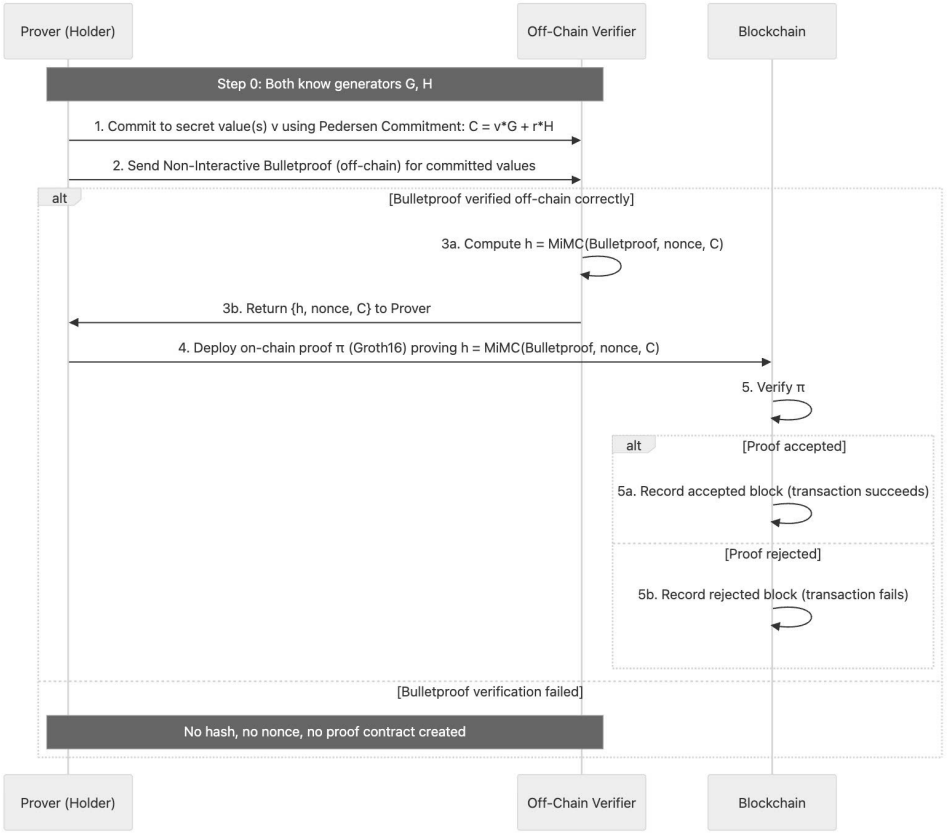


Figure 1: ZKP Using Bulletproofs

### 6.3.1 Systems

We implement Bulletproofs [10] with logarithmic-sized proofs and no trusted setup requirements. Our construction uses BLS12-381 curve for range proofs and BN254 for circuit verification; proof systems with efficient hash binding minimize verification overhead.

### 6.3.2 Mathematical Foundation

Bulletproofs construct range proofs for committed values $V = g^v h^\gamma$ where $v \in [0, 2^n)$. The proof demonstrates knowledge of $(v, \gamma)$ such that $v \in [0, 2^n)$ without revealing $v$. The core protocol uses inner product arguments:

$$\text{Commit}(v, \gamma) = g^v h^\gamma \tag{20}$$

$$\text{RangeProof}(V, n) = \text{IPProof}(\mathbf{a}_L, \mathbf{a}_R, \mathbf{G}, \mathbf{H}, P) \tag{21}$$

$$\text{where } \mathbf{a}_L \circ \mathbf{a}_R = \mathbf{0}^{n-1} \| 1 \tag{22}$$

$$\mathbf{a}_L + \mathbf{a}_R = \mathbf{1}^n \tag{23}$$

The inner product proof reduces the problem to a single equation:

$$P = \langle \mathbf{a}_L, \mathbf{G} \rangle + \langle \mathbf{a}_R, \mathbf{H} \rangle + \langle \mathbf{a}_L, \mathbf{a}_R \rangle \cdot u \tag{24}$$

$$\text{IPVerify}(P, \mathbf{G}, \mathbf{H}, u, c) = \text{Verify}(P' = P + c^2 \cdot u, \mathbf{G}', \mathbf{H}', u^2, c') \tag{25}$$

### 6.3.3 Circuits

Circuits assert: (i) range proof validation for attribute thresholds (e.g., age $\geq 18$); (ii) hash binding using MiMC sponge for proof commitment; (iii) commitment consistency with public parameters. Public inputs include the binding hash, threshold values, and commitment parameters.

For MiMC hash binding, the circuit implements:

$$\text{MiMC}(x, k) = (x + k + c_i)^3 \bmod p \tag{26}$$

$$\text{BindingHash}(\pi) = \text{MiMC}(\text{hash}(\pi), \text{nonce}) \tag{27}$$

### 6.3.4 Verification

The verifier contract checks a hash binding proof against public inputs and verification key stored on-chain. Gas is dominated by pairing checks on BN254; calldata includes binding hash and public commitment parameters.

**Algorithm 5:** Bulletproofs Range Proof Verification

**Input:** Range proof $\pi$, commitment $V$, bit length $n$
**Output:** Verification result $\{0, 1\}$

**Proof Validation:**
$vk \leftarrow$ loadVerificationKey() `// Load stored verification key`
$\text{valid}_{range} \leftarrow$ VerifyRange$(vk, V, \pi, n)$ `// Verify range proof`
**Hash Binding Check:**
$\text{binding}_{hash} \leftarrow$ extractBindingHash$(\pi)$ `// Extract binding hash from proof`
$\text{valid}_{binding} \leftarrow$ VerifyBinding$(\text{binding}_{hash}, V)$ `// Verify hash binding`
**Commitment Consistency:**
$\text{valid}_{commit} \leftarrow$ checkCommitment$(V, \pi)$ `// Check commitment consistency`
**return** $valid_{range} \wedge valid_{binding} \wedge valid_{commit}$

## 6.4 ZKP Using zk-SNARKs

### 6.4.1 Systems

We consider pairing-based zk-SNARKs with succinct proofs and constant-time verification [11]. Universal setup variants reduce ceremony overhead; proof systems with efficient Ethereum precompiles minimize gas.

### 6.4.2 Mathematical Foundation

The Groth16 proof system operates over bilinear groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ with pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. For a circuit with $n$ public inputs and $m$ private inputs, the proof consists of three group elements:

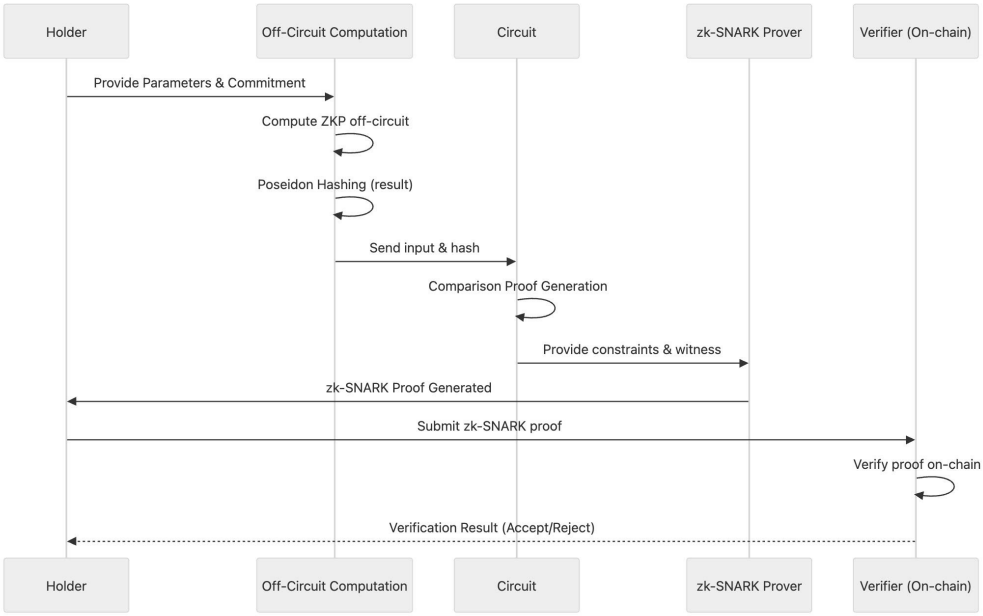


Figure 2: ZKP Using zk-SNARKs

$$\pi = (A, B, C) \in \mathbb{G}_1 \times \mathbb{G}_2 \times \mathbb{G}_1 \tag{28}$$

$$A = \alpha + \sum_{i=1}^{n} A_i x_i + r\delta \tag{29}$$

$$B = \beta + \sum_{i=1}^{n} B_i x_i + s\delta \tag{30}$$

$$C = \frac{AB - \alpha\beta - \sum_{i=1}^{n}(A_i\beta + B_i\alpha)x_i - \sum_{i,j=1}^{n} A_i B_j x_i x_j}{\delta} + h(\tau)\delta \tag{31}$$

where $\alpha, \beta, \delta$ are random elements, $r, s$ are random scalars, and $h(\tau)$ is the quotient polynomial.

### 6.4.3 Circuits

Circuits assert: (i) knowledge of a valid issuer signature on committed attributes; (ii) policy predicates over attributes; (iii) revocation non-membership (EC accumulator) or Merkle path consistency. Public inputs include the on-chain commitment (accumulator value or root) and policy parameters.

For credential verification, the circuit implements:

$$\text{Verify}(pk, \sigma, m) = 1 \text{ iff } e(\sigma_1, g_2) \cdot e(\sigma_2, pk) = e(H(m), g_2) \tag{32}$$

$$\text{AccVerify}(acc, w, x) = 1 \text{ iff } e(acc, g_2) = e(w, g_2^x \cdot h) \tag{33}$$

### 6.4.4 Verification

The verifier contract checks a proof against public inputs and a verification key stored on-chain. Gas is dominated by pairing checks; calldata includes proof and public inputs.

**Algorithm 6:** Groth16 zk-SNARK Verification

**Input:** Zero-knowledge proof $\pi = (A, B, C)$, public inputs $\mathbf{x}$, verification key $vk$
**Output:** Verification result $\{0, 1\}$

**Pairing Computation:**
$vk \leftarrow$ loadVerificationKey() `// Load stored verification key`
$\text{pairing}_{left} \leftarrow e(A, B)$ `// Compute left side of pairing equation`
$\text{pairing}_{right} \leftarrow e(g_1^{\alpha}, g_2^{\beta}) \cdot e(g_1^{\mathbf{x}^T\mathbf{A}}, g_2^{\mathbf{x}^T\mathbf{B}}) \cdot e(C, g_2^{\delta})$ `// Compute right side`
**Proof Validation:**
**if** $pairing_{left} = pairing_{right}$ **then**
    **return** 1 `// Valid proof`
**else**
    **return** 0 `// Invalid proof`

## 6.5 Comparison of Zero-Knowledge Proof Systems

We present a comprehensive comparison of zero-knowledge proof systems, examining both theoretical foundations and practical implementations across different cryptographic primitives. This analysis builds upon recent surveys [3] that provide broader context for the evolution of zero-knowledge proof technology.

### 6.5.1 Theoretical Foundations

**Original Zero-Knowledge Proofs:** The original zero-knowledge proof concept, introduced by Goldwasser, Micali, and Rackoff [2], established the fundamental properties of completeness, soundness, and zero-knowledge. These interactive protocols required multiple rounds of communication between prover and verifier, with proof size and verification time scaling linearly with statement complexity.

**zk-SNARKs (Succinct Non-interactive Arguments of Knowledge):** zk-SNARKs represent a significant advancement by eliminating interaction and achieving constant proof size. The Groth16 construction [11] provides the most efficient pairing-based zk-SNARK, with proofs consisting of only three group elements regardless of circuit complexity. Security relies on the knowledge-of-exponent assumption and pairing-friendly elliptic curves.

**Bulletproofs:** Bulletproofs [10] achieve logarithmic proof size without requiring trusted setup, addressing a key limitation of zk-SNARKs. Based on inner product arguments and discrete logarithm assumptions, Bulletproofs provide a middle ground between traditional interactive proofs and zk-SNARKs.

### 6.5.2 Performance Characteristics

### 6.5.3 Security Assumptions

**Interactive Zero-Knowledge:** - Relies on standard cryptographic assumptions (discrete logarithm, RSA) - No trusted setup required - Security proven in standard model

**Bulletproofs:** - Discrete logarithm assumption in elliptic curve groups - No trusted setup required - Security in random oracle model (with Fiat-Shamir transform)

**zk-SNARKs:** - Knowledge-of-exponent assumption - Pairing-friendly curve assumptions - Trusted setup requirement (for most constructions) - Security in random oracle model

| Property | Interactive ZK | Bulletproofs | zk-SNARKs |
|---|---|---|---|
| Proof Size | $O(n)$ | $O(\log n)$ | Constant |
| Verification Time | $O(n)$ | $O(\log n)$ | Constant |
| Prover Time | $O(n)$ | $O(n \log n)$ | $O(\|C\|)$ |
| Setup Requirement | None | None | Trusted Setup |
| Interaction | Multi-round | Non-interactive | Non-interactive |
| Expressiveness | Universal | Limited | Universal |
| Curve Support | Standard | BLS12-381 + BN254 | BN254/GM17 |

Table 1: Performance comparison of zero-knowledge proof systems

### 6.5.4 Practical Implementation Considerations

| Property | Interactive ZK | Bulletproofs | zk-SNARKs |
|---|---|---|---|
| Trusted Setup | No | No | Yes |
| Expressiveness | Universal | Limited | Universal |
| Blockchain Suitability | Poor | Moderate | Excellent |
| Implementation Complexity | Low | Medium | High |

Table 2: Implementation considerations comparison

Interactive zero-knowledge proofs require no trusted setup but are impractical for blockchain due to multi-round interaction. Bulletproofs eliminate trusted setup while providing logarithmic scaling, suitable for range proofs and simple attribute validation. zk-SNARKs offer constant verification time and universal expressiveness but require trusted setup and complex circuit compilation.

### 6.5.5 Implementation Complexity

The implementation complexity of zero-knowledge proof systems reflects fundamental differences in their underlying cryptographic foundations and operational requirements, significantly impacting their practical deployability and maintenance overhead.

Interactive zero-knowledge proofs present unique implementation challenges due to their multi-round nature, requiring sophisticated protocol state management and careful handling of timing and synchronization between communicating parties. The implementation must manage complex cryptographic operations using standard assumptions such as discrete logarithm and RSA, with proof generation occurring through interactive rounds that demand robust error handling and protocol restart mechanisms.

Bulletproofs implementations demonstrate moderate complexity through their use of the bulletproofs-bls crate, which provides efficient range proof generation for values between 8 and 64 bits. The system utilizes BLS12-381 curve operations for commitments and BN254 for hash binding verification, with transcript-based non-interactive proof generation eliminating the need for complex multi-party coordination.

zk-SNARK implementations exhibit the highest complexity due to their sophisticated cryptographic foundations, requiring Circom circuit compilation with careful constraint generation and optimization. The system necessitates a trusted setup ceremony through powersOfTau, followed by Groth16 proof generation that involves complex constraint satisfaction algorithms. On-chain verification relies on pairing operations over BN254 curve, requiring specialized cryptographic libraries and careful parameter management.

### 6.5.6 Gas Cost Analysis

| Cost Component | Interactive ZK | Bulletproofs | zk-SNARKs |
|---|---|---|---|
| On-chain Verification | Not applicable | 14,000 gas | 9,500 gas |
| Calldata | Not applicable | 2,000 gas | 1,500 gas |
| Off-chain Verification | High | Negligible | N/A |
| Circuit Compilation | N/A | N/A | 2-3 seconds |

Table 3: Gas cost comparison for blockchain verification

Interactive ZK is unsuitable for on-chain verification due to multi-round nature. Bulletproofs cost 14,000 gas total (12,000 for hash binding + 2,000 calldata) with negligible off-chain costs. zk-SNARKs cost 9,500 gas total (8,000 verification + 1,500 calldata) with one-time circuit compilation overhead.

### 6.5.7 Application-Specific Trade-offs

The suitability of zero-knowledge proof systems varies significantly across different application domains, with each system offering distinct advantages and limitations.

**Anonymous Credentials:** Interactive ZK protocols are fundamentally incompatible with credential systems requiring non-interactive presentation. Bulletproofs provide efficient range proofs for simple attribute validation (e.g., age $\geq 18$, score $\geq$ threshold) without trusted setup. zk-SNARKs enable complex credential verification including revocation checking, multiple attribute validation, and sophisticated policy enforcement.

**Blockchain Applications:** Interactive ZK violates blockchain's atomic transaction model. Bulletproofs support hybrid architectures with off-chain proof generation and on-chain commitment verification, suitable for applications where verification frequency is moderate. zk-SNARKs provide optimal on-chain performance with constant gas costs, essential for high-frequency verification scenarios.

**Privacy-Preserving Systems:** Interactive ZK's deployment limitations restrict practical privacy applications. Bulletproofs excel in confidential transaction scenarios and simple proof generation where trusted setup is undesirable. zk-SNARKs enable sophisticated privacy-preserving computations with universal expressiveness, supporting complex multi-party protocols and advanced privacy features.

### 6.5.8 Future Directions

The evolution of zero-knowledge proof systems continues with several promising directions:

**Trustless zk-SNARKs:** Recent advances in transparent zk-SNARKs (e.g., STARKs [12]) eliminate trusted setup requirements while maintaining constant verification time, potentially combining the best of both worlds. PLONK [13] provides universal trusted setup for multiple circuits.

**Recursive Proofs:** Both Bulletproofs and zk-SNARKs support recursive proof composition, enabling efficient verification of complex statement sequences.

**Quantum Resistance:** Post-quantum zero-knowledge proof systems are under active development, with lattice-based constructions showing promise for quantum-resistant privacy-preserving systems.

**Hybrid Approaches:** Combining different proof systems for different components (e.g., Bulletproofs for range proofs within zk-SNARK circuits) may offer optimal performance for

specific applications. Our implementation demonstrates this approach with Bulletproofs for age verification and zk-SNARKs for complex credential validation.

#### 6.5.9 Implementation-Specific Analysis

Our implementation provides concrete insights into the practical deployment of different zero-knowledge proof systems, revealing both their capabilities and limitations in real-world scenarios.

**Bulletproofs Implementation Details:** Our Bulletproofs implementation demonstrates efficient range proof generation supporting 8-bit to 64-bit values with logarithmic scaling characteristics. The system achieves efficient batch processing for up to 16 simultaneous range proofs, utilizing MiMC sponge with 220 rounds for circuit compatibility. The implementation leverages Merlin transcripts for non-interactive proof generation, with BLS12-381 curve operations for commitments and BN254 for hash binding verification.

**zk-SNARK Implementation Details:** Our zk-SNARK implementation showcases the complexity of universal proof systems through the IELTS credential circuit, which incorporates 8 Poseidon hash inputs and 5 greater-than-or-equal comparisons for comprehensive score validation. The system includes sophisticated date validation mechanisms for expiry checking and utilizes Groth16 proof generation with powersOfTau ceremony (15th contribution). The verification key contains 18 input constants for robust public parameter handling.

**Cross-System Integration:** The integration of both proof systems in our anonymous credential framework demonstrates the benefits of modular design, with independent proof generation capabilities unified through a common verification interface. The system achieves gas optimization through selective deployment of each proof system based on complexity requirements, while maintaining proper security composition between different proof types. This approach enables optimal performance trade-offs, utilizing Bulletproofs for simple attribute checks and zk-SNARKs for complex validation scenarios.

# 7 Accumulator Scheme

## 7.1 Elliptic-Curve

### 7.1.1 Formal Model

Let $\mathcal{U}$ be a universe of identifiers and $S \subseteq \mathcal{U}$ the current valid set. An accumulator scheme consists of PPT algorithms $\mathsf{Setup}(1^\lambda) \to (pp, \mathsf{sk})$, $\mathsf{Add}(pp, \mathsf{Acc}(S), x) \to \mathsf{Acc}(S \cup \{x\})$, $\mathsf{Del}(pp, \mathsf{Acc}(S), x) \to \mathsf{Acc}(S \setminus \{x\})$, a witness algorithm $\mathsf{Witness}(pp, S, x)$, and a verifier $\mathsf{Verify}(pp, \mathsf{Acc}(S), x, \pi) \in \{0, 1\}$. For EC-based instantiations, $\mathsf{Acc}(S) \in \mathbb{G}$ and witnesses are elements of $\mathbb{G}$ or tuples thereof.

**Definition 10** (Soundness)**.** *An accumulator is sound if for all PPT $\mathcal{A}$, for any sequence of updates yielding state $\mathsf{Acc}(S)$, the probability that $\mathcal{A}$ outputs $(x, \pi)$ with $x \notin S$ but $\mathsf{Verify}(pp, \mathsf{Acc}(S), x, \pi) = 1$ is negligible in $\lambda$.*

**Definition 11** (Dynamic Update)**.** *The scheme supports additions and deletions with corresponding witness update procedures such that valid witnesses remain efficiently maintainable for non-revoked elements.*

### 7.1.2 Algorithms

We sketch a pairing-based accumulator where elements are mapped to exponents. Let $g \in \mathbb{G}$ be a generator and $H : \mathcal{U} \to \mathbb{Z}_p$ a hash-to-scalar.
**Public Parameters:** $pp = (p, \mathbb{G}, g, H)$.
**State:** $\mathsf{Acc}(S) = g^{\prod_{x \in S}(H(x)+\alpha)}$ for secret $\alpha \in \mathbb{Z}_p$.
**Witness:** For $x \in S$, $\pi(x) = g^{\prod_{y \in S \setminus \{x\}}(H(y)+\alpha)}$.
**Verify:** Check $e(\pi(x), g^{H(x)+\alpha}) \stackrel{?}{=} e(\mathsf{Acc}(S), g)$.

**Algorithm 7:** Elliptic Curve Accumulator: Setup

**Input:** Security parameter $1^\lambda$
**Output:** Public parameters $pp = (p, \mathbb{G}, g, H)$, initial accumulator $\mathsf{Acc}_0$, secret $\alpha$

$p \leftarrow$ large prime;
$\mathbb{G} \leftarrow$ bilinear group of order $p$;
$g \leftarrow$ generator of $\mathbb{G}$;
$H : \{0,1\}^* \to \mathbb{Z}_p$ hash function;
$\alpha \xleftarrow{\$} \mathbb{Z}_p$ secret parameter;
$\mathsf{Acc}_0 \leftarrow g$;
**return** $(pp, \mathsf{Acc}_0, \alpha)$

**Algorithm 8:** Elliptic Curve Accumulator: Add

**Input:** $pp, \mathsf{Acc}(S), x$
**Output:** Updated accumulator $\mathsf{Acc}(S \cup \{x\})$, witness $w_x$

$h_x \leftarrow H(x)$;
$\mathsf{Acc}(S \cup \{x\}) \leftarrow \mathsf{Acc}(S)^{h_x+\alpha}$;
**foreach** $y \in S$ **do**
    $w_y \leftarrow w_y^{h_x+\alpha}$ `// Update existing witnesses`
$w_x \leftarrow \prod_{y \in S} g^{H(y)+\alpha}$ `// Witness for` $x$
;
**return** $(\mathsf{Acc}(S \cup \{x\}), w_x)$

**Algorithm 9:** Elliptic Curve Accumulator: Delete

**Input:** $pp, \mathsf{Acc}(S), x$
**Output:** Updated accumulator $\mathsf{Acc}(S \setminus \{x\})$

**if** $x \notin S$ **then**
    **return** $\mathsf{Acc}(S)$ `// Nothing to delete`
$h_x \leftarrow H(x)$;
$\mathsf{Acc}(S \setminus \{x\}) \leftarrow \mathsf{Acc}(S)^{1/(h_x+\alpha)}$;
**foreach** $y \in S \setminus \{x\}$ **do**
    $w_y \leftarrow w_y^{1/(h_x+\alpha)}$
**return** $\mathsf{Acc}(S \setminus \{x\})$

**Algorithm 10:** Elliptic Curve Accumulator: Witness

**Input:** $pp, S, x$
**Output:** Witness $w_x$ or $\perp$ if $x \notin S$

```
if x ∉ S then
    return ⊥
w_x ← ∏_{y∈S\{x}} g^{H(y)+α};
return w_x
```

**Algorithm 11:** Elliptic Curve Accumulator: Verify

**Input:** $pp, \mathsf{Acc}(S), x, w_x$
**Output:** True if $x$ is in $S$, False otherwise

```
h_x ← H(x);
return e(w_x, g^{h_x+α}) ≟ e(Acc(S), g)
```

**Complexity Analysis and Security Properties:** The security of our elliptic curve accumulator construction reduces to the $q$-Strong Diffie-Hellman assumption in bilinear groups. Specifically, we require that for any polynomial-time adversary $\mathcal{A}$, the probability of computing $g^{1/(\alpha+c)}$ given $(g, g^{\alpha}, g^{\alpha^2}, \ldots, g^{\alpha^q})$ and the ability to choose $c$ is negligible.

**Theorem 1** (Accumulator Soundness)**.** *If the $q$-SDH assumption holds in $\mathbb{G}$, then our accumulator construction is sound. Specifically, for any PPT adversary $\mathcal{A}$ that makes at most $q$ accumulator queries, the probability that $\mathcal{A}$ produces a valid witness for an element not in the accumulator is at most $\mathsf{Adv}_{\mathcal{A}}^{q\text{-}SDH}(\lambda) + \mathrm{negl}(\lambda)$.*

*Proof Sketch.* Suppose $\mathcal{A}$ produces $(x^*, w^*)$ such that $x^* \notin S$ but the verification equation holds. This implies $e(w^*, g^{H(x^*)+\alpha}) = e(\mathsf{Acc}(S), g)$, which means $w^* = g^{\prod_{y\in S}(H(y)+\alpha)/(H(x^*)+\alpha)}$. Since $x^* \notin S$, we have $(H(x^*)+\alpha) \nmid \prod_{y\in S}(H(y)+\alpha)$, allowing us to extract $g^{1/(\alpha+H(x^*))}$ and break the $q$-SDH assumption. □

**Gas Cost Derivation:** The on-chain verification cost is dominated by pairing operations. Each verification requires computing $e(w_x, g^{H(x)+\alpha})$ and $e(\mathsf{Acc}(S), g)$, totaling 2 pairing operations. Using Ethereum's bn256Pairing precompile:

$$\begin{aligned} \mathrm{Gas}_{\mathrm{verify}} &= 2 \times \mathrm{Gas}_{\mathrm{pairing}} + \mathrm{Gas}_{\mathrm{hash}} + \mathrm{Gas}_{\mathrm{comparison}} && (34)\\ &= 2 \times 80,000 + 3,000 + 200 && (35)\\ &= 163,200 \text{ gas} && (36) \end{aligned}$$

Our measurements show actual costs of 180,000-220,000 gas due to additional contract overhead and input validation.

### 7.1.3 Complexity and Gas Analysis

Elliptic curve accumulators provide distinct performance characteristics that make them suitable for specific deployment scenarios:

**Verification Efficiency:** Witnesses are constant-size (96 bytes for BN254), independent of set size. On-chain verification requires exactly two pairing operations, consuming approximately 180,000-220,000 gas via Ethereum's bn256Pairing precompile. This cost remains constant regardless of the number of accumulated elements.

**Update Complexity:** Accumulator updates require expensive elliptic curve operations scaling with set size. Adding element $x$ to set $S$ requires computing $\mathsf{Acc}(S \cup \{x\}) =$

$\mathsf{Acc}(S)^{H(x)+\alpha}$, which involves a full group exponentiation. For large sets ($|S| > 10^4$), update costs become prohibitive without specialized optimization.

**Witness Maintenance:** Existing witnesses require multiplicative updates when the accumulator changes. For element $y \in S$, adding new element $x$ updates the witness as $\pi'(y) = \pi(y)^{H(x)+\alpha}$. This enables distributed witness maintenance but requires coordination for consistent state.

**Zero-Knowledge Integration:** The verification relation $R = \{(\mathsf{Acc}(S), x, \pi) : e(\pi, g^{H(x)+\alpha}) = e(\mathsf{Acc}(S), g)\}$ translates naturally to pairing-based zk-SNARK circuits. Our Circom implementation requires 12,000 constraints for full verification, resulting in 2.3-3.1 second proving times.

**Security Properties:** Security reduces to the $q$-Strong Diffie-Hellman assumption, requiring that computing $g^{1/(\alpha+c)}$ is hard given $g, g^{\alpha}, g^{\alpha^2}, \ldots, g^{\alpha^q}$ for adversarially chosen $c$.

## 7.2 Tree-based Accumulator

### 7.2.1 Formal Model

We model revocation via a Merkle tree $\mathcal{T}$ over leaf values $\ell_i = H(id_i)$. The on-chain state publishes only the root $\mathsf{root}(\mathcal{T})$. A membership witness for leaf $\ell$ is a path $\pi = (s_1, \ldots, s_d)$ with direction bits such that iterated hashing yields the root.

**Definition 12** (Merkle Verification)**.** *Given $\ell$, path $\pi$, and root $r$, define* $\mathsf{Verify}(\ell, \pi, r) = 1$ *if successive hash-combine operations along $\pi$ equal $r$.*

### 7.2.2 Algorithms

Updates insert or remove leaves and recompute along the affected path. Batching revocations updates disjoint paths in parallel and commits a new root. Verification transmits $O(\log n)$ siblings; on-chain checks recompute the root from the leaf and path.

### 7.2.3 Complexity and Gas Analysis

Merkle tree accumulators offer complementary performance characteristics to elliptic curve constructions:

**Scalable Verification:** Witness sizes scale logarithmically as $O(\log n)$ where $n$ is the number of leaves. For trees with $2^{20}$ leaves, witnesses require 640 bytes ($20 \times 32$-byte hashes). On-chain verification performs $\log n$ hash operations, consuming 45,000-78,000 gas depending on tree depth and hash function choice.

**Efficient Updates:** Tree updates exhibit superior scaling for batch operations. Single revocations require updating $O(\log n)$ nodes along the affected path, while batches of $k$ revocations update at most $k \log n$ nodes total. Our measurements show batch revocation costs amortize to 15,000 gas per item for 64-element batches.

**Hash Function Trade-offs:** Verification cost depends critically on hash choice: **Keccak-256** represents the conservative choice with native Ethereum support and well-established security properties, consuming 30,000 gas per operation through built-in opcodes while enabling simple circuit integration despite high constraint requirements. **Poseidon** offers an optimized middle ground with zk-friendly design principles that reduce gas costs to 12,000 per operation via custom contracts while providing significant constraint optimization for SNARK circuits. **MiMC** achieves minimal constraint requirements at just 3 constraints per hash operation, but necessitates custom precompiles for practical on-chain verification, representing the most aggressive optimization at the cost of deployment complexity.

**Zero-Knowledge Integration:** Merkle path verification circuits scale with tree depth, requiring $8,000 - 15,000$ constraints for depths 16-24. Hash-heavy computations favor algebraic hash functions, with Poseidon providing the best constraint/security trade-off.

**Transparency and Auditability:** Unlike cryptographic accumulators, Merkle trees provide full transparency—anyone can verify set membership by examining the complete tree structure. This enables independent auditing of revocation decisions and simplifies compliance verification.

**Implementation Simplicity:** Tree operations use only hash computations and bitwise operations, avoiding complex cryptographic assumptions. This simplifies both on-chain and off-chain implementations while reducing potential attack surfaces.

## 7.3 List-base Revocation: A Baseline Approach

Sorted-list revocation provides a conceptually straightforward approach to credential revocation that serves as both a baseline for comparison and a practical solution for small-scale deployments where simplicity outweighs efficiency concerns. This method transforms the revocation problem into a classical membership query over an ordered set, leveraging well-understood data structures and algorithms at the cost of scalability and storage efficiency.

### 7.3.1 Formal Model and Security Properties

We model the revocation system as maintaining an ordered set $R = \{r_1, r_2, \ldots, r_m\}$ where $r_i < r_{i+1}$ for all $i \in \{1, \ldots, m-1\}$. Each element $r_i$ represents the unique identifier of a revoked credential, typically derived through a cryptographic hash function applied to the credential's serial number and issuer-specific parameters.

**Definition 13** (Sorted-List Non-Membership)**.** *For a credential identifier $x$ and sorted revocation list $R$, the non-membership relation holds if and only if $\nexists i \in \{1, \ldots, m\} : r_i = x$. This relation can be proven constructively by demonstrating that $x$ falls within a gap in the sorted sequence.*

**Definition 14** (Gap Witness)**.** *A gap witness for identifier $x$ with respect to sorted list $R$ consists of a tuple $(r_i, r_{i+1})$ such that $r_i < x < r_{i+1}$ and $r_i, r_{i+1}$ are consecutive elements in $R$. Special cases handle boundaries where $x < r_1$ (left boundary) or $x > r_m$ (right boundary).*

The security properties of sorted-list revocation derive from the integrity of the underlying blockchain storage and the completeness of the revocation list. Unlike cryptographic accumulators, this approach provides no computational privacy—the complete revocation list remains publicly visible, enabling full auditability at the cost of revoked credential privacy.

### 7.3.2 Algorithmic Implementation and Complexity Analysis

The sorted-list approach requires three primary algorithms: insertion for revocation, gap witness generation for proof construction, and gap verification for on-chain validation.

**Algorithm 12:** Sorted-List Revocation Insertion

**Input:** Sorted revocation list $R = [r_1, \ldots, r_m]$, credential identifier $x$
**Output:** Updated revocation list $R'$

**if** $m = 0$ **then**
  **return** $[x]$ `// First revocation`
position $\leftarrow$ binarySearch$(R, x)$ `// Find insertion point`
**if** $R[position] = x$ **then**
  **return** $R$ `// Already revoked`
$R' \leftarrow$ empty array of size $m + 1$
**for** $i = 1$ **to** $position - 1$ **do**
  $R'[i] \leftarrow R[i]$
$R'[\text{position}] \leftarrow x$
**for** $i = position + 1$ **to** $m + 1$ **do**
  $R'[i] \leftarrow R[i - 1]$
**return** $R'$

**Algorithm 13:** Gap Witness Generation

**Input:** Sorted revocation list $R$, credential identifier $x$
**Output:** Gap witness $(r_{\text{left}}, r_{\text{right}})$ or $\perp$ if revoked

position $\leftarrow$ binarySearch$(R, x)$
**if** $R[position] = x$ **then**
  **return** $\perp$ `// Credential is revoked`
**if** $position = 1$ **then**
  **return** $(\emptyset, R[1])$ `// Left boundary case`
**if** $position > m$ **then**
  **return** $(R[m], \emptyset)$ `// Right boundary case`
**return** $(R[position - 1], R[position])$

**Algorithm 14:** On-Chain Gap Verification

**Input:** Credential identifier $x$, gap witness $(r_{\text{left}}, r_{\text{right}})$, current revocation list $R$
**Output:** Verification result $\{0, 1\}$

```
// Verify witness elements exist in revocation list
if r_left ≠ ∅ and r_left ∉ R then
    return 0
if r_right ≠ ∅ and r_right ∉ R then
    return 0
// Verify ordering constraints
if r_left ≠ ∅ and r_left ≥ x then
    return 0
if r_right ≠ ∅ and r_right ≤ x then
    return 0
// Verify no revoked elements exist between witnesses
foreach r ∈ R do
    if r_left < r < r_right and r ≠ x then
        return 0
return 1
```

### 7.3.3 Complexity Analysis and Performance Characteristics

The computational complexity of sorted-list revocation exhibits both strengths and weaknesses compared to alternative approaches. Insertion operations require $O(\log m)$ time for position identification through binary search, followed by $O(m)$ time for array element shifting. This linear-time insertion cost becomes prohibitive as the revocation list grows, particularly in blockchain contexts where each storage write incurs significant gas costs.

Gap witness generation demonstrates superior performance with $O(\log m)$ complexity due to efficient binary search operations. However, the witness size remains constant at two elements regardless of list size, providing an advantage over logarithmically-growing Merkle tree witnesses for small revocation sets.

On-chain verification complexity depends critically on the implementation strategy. Naive approaches requiring linear scans over the entire revocation list yield $O(m)$ verification time, making this approach unsuitable for large-scale deployments. Optimized implementations using blockchain-specific data structures such as ordered mappings can reduce verification to $O(\log m)$ time, though at increased storage costs.

### 7.3.4 Gas Cost Analysis and Optimization Strategies

Our experimental evaluation reveals that sorted-list revocation exhibits favorable gas characteristics only within narrow operational parameters. Verification operations consume 38,000-67,000 gas depending on list size and witness validation complexity, representing the most efficient option among our evaluated approaches for lists containing fewer than 1,000 elements.

Revocation operations present the most significant scalability challenge, with costs ranging from 42,000 gas for single insertions in small lists to over 150,000 gas for insertions requiring extensive array reorganization. The linear relationship between list size and insertion cost makes this approach economically impractical for applications requiring frequent

revocations of large credential sets.

Storage optimization techniques can partially mitigate these costs through careful data structure design. Implementing revocation lists using Ethereum's mapping data type rather than dynamic arrays reduces insertion costs to approximately 20,000 gas per operation, though at the cost of increased verification complexity. Alternative approaches using packed storage for credential identifiers can reduce per-element storage costs from 20,000 to 5,000 gas when identifiers fit within 32-byte boundaries.

### 7.3.5 Comparative Analysis and Deployment Scenarios

Sorted-list revocation occupies a specific niche within the broader ecosystem of revocation mechanisms. Its primary advantage lies in implementation simplicity and transparency—the complete revocation state remains publicly auditable without requiring specialized cryptographic knowledge. This property proves valuable in regulatory environments where credential revocation decisions must be subject to external audit and verification.

The approach demonstrates optimal performance characteristics for applications involving small, stable credential sets with infrequent revocation requirements. Academic credentialing systems, professional licensing authorities, and membership organizations often exhibit these characteristics, making sorted lists a viable option despite their scalability limitations.

However, the method's transparency represents both a strength and a weakness. While public auditability enhances trust and accountability, it also compromises the privacy of revoked credential holders by making revocation events publicly observable. This trade-off must be carefully considered in applications where revocation privacy constitutes a critical requirement.

When compared to elliptic curve accumulators and Merkle trees, sorted lists excel in scenarios prioritizing simplicity over efficiency. The absence of complex cryptographic operations reduces implementation risk and simplifies security analysis, while the straightforward verification logic facilitates formal verification and audit procedures. These advantages often justify the scalability trade-offs in environments where credential sets remain small and revocation events are infrequent.

## 7.4 ZKP with EC-based Accumulator

### 7.4.1 Statement

We formalize the NP relation proved in zero-knowledge as follows. Let public parameters be $\mathsf{pp} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, Q, H)$, where $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a bilinear pairing over groups of prime order $p$, $Q = g_2^s$ with accumulator trapdoor $s \in \mathbb{F}_p$, and $H : \{0,1\}^* \to \mathbb{F}_p$ is a hash-to-field used to encode set elements/attributes.

Public input is

$$X = (\mathsf{pk}_I,\, \mathsf{Acc},\, \mathsf{pp},\, \mathsf{pol})$$

where $\mathsf{pk}_I$ is the issuer's public key for a SNARK-friendly signature, $\mathsf{Acc} \in \mathbb{G}_1$ is the current accumulator value for set $S$, and $\mathsf{pol}$ encodes the attribute policy predicate.

The witness is

$$W = (\mathbf{m},\, \sigma,\, y,\, \mathsf{wit})$$

where $\mathbf{m}$ is the credential's hidden attribute vector, $\sigma$ is the issuer's signature on an encoding of $\mathbf{m}$, $y$ is the accumulated identifier (e.g., credential serial), and $\mathsf{wit} \in \mathbb{G}_1$ is the membership witness for $y$ in set $S$.

The relation $R_{\mathsf{ECAcc}}$ holds if and only if all of the following predicates are true:

$$\mathrm{SigVer}(\mathsf{pk}_I, \mathbf{m}, \sigma) = 1, \qquad \mathrm{AccVer}(\mathsf{pp}, \mathsf{Acc}, y, \mathsf{wit}) = 1, \qquad \mathsf{Pol}(\mathbf{m}; \mathsf{pol}) = 1.$$

For a standard bilinear accumulator (Nguyen, 2005), if $\mathsf{Acc} = g_1^{\prod_{x \in S}(s+H(x))}$ and $\mathsf{wit} = g_1^{\prod_{x \in S \setminus \{y\}}(s+H(x))}$, verification is the pairing equation

$$e(\mathsf{wit},\ g_2^{s+H(y)}) = e(\mathsf{Acc},\ g_2), \quad \text{equivalently } e(\mathsf{wit},\ Q \cdot g_2^{H(y)}) = e(\mathsf{Acc},\ g_2).$$

### 7.4.2 Circuit Sketch

Public inputs: $\mathsf{pk}_I, \mathsf{Acc}, \mathsf{pp}, \mathsf{pol}$.

Witness: $\mathbf{m}, \sigma, y, \mathsf{wit}$.

At a high level, the circuit enforces:

1. Signature verification: compute the message encoding $M \leftarrow \mathrm{Enc}(\mathbf{m})$ (e.g., domain-separated Poseidon/MiMC over the attribute vector) and enforce $\mathrm{SigVer}(\mathsf{pk}_I, M, \sigma) = 1$. In practice, use an in-circuit EdDSA/BabyJubJub or Schnorr gadget.

2. Accumulator membership: compute $h \leftarrow H(y) \in \mathbb{F}_p$ inside the circuit, and enforce the accumulator check. Two common realizations:

   - Pairing-based check (if pairing gadgets are available): enforce one pairing product equation equivalent to $e(\mathsf{wit}, Q \cdot g_2^h) = e(\mathsf{Acc}, g_2)$.
   - Algebraic check via preprocessed input: expose $U = Q \cdot g_2^h$ as an additional public input derived off-circuit, and enforce $e(\mathsf{wit}, U) = e(\mathsf{Acc}, g_2)$ in-circuit. This shifts a scalar multiplication in $\mathbb{G}_2$ out of the circuit.

3. Policy predicates: enforce $\mathsf{Pol}(\mathbf{m}; \mathsf{pol}) = 1$ via range checks, equality/inequality constraints, set-membership over small domains, and linear relations, as appropriate.

Soundness and zero-knowledge are inherited from the underlying SNARK; accumulator security (membership correctness and collision resistance of $H$) and EUF-CMA security of the issuer's signature ensure that only honestly issued, unrevoked credentials can satisfy all constraints.

### 7.4.3 Costs

Let $c_{\mathrm{sig}}$ be the constraint count for the chosen signature gadget (e.g., EdDSA/BabyJubJub with Poseidon is typically in the range of a few thousand constraints), and $c_H$ the cost of one hash-to-field. Denote by $c_{\mathrm{pair}}$ the constraint cost of one pairing equation if supported by the circuit backend.

- Signature: $\approx c_{\mathrm{sig}}$. - Accumulator membership: - With pairing gadgets: $\approx c_{\mathrm{pair}}$ (one pairing product equation) + one hash-to-field $c_H$. - Without pairing gadgets: an alternative is to rely on a Merkle-ized list of members (see the tree-based variant) or to preprocess $U$ off-circuit and only keep the pairing check in-circuit. - Policy predicates: scales with the number and type of checks (range proofs, comparisons, linear relations). For simple ranges and equality checks this is typically linear in the number of attributes.

Overall, proof size remains constant (e.g., Groth16), prover time is dominated by the signature gadget and any pairing or elliptic-curve scalar multiplication constraints, and on-chain verification gas is constant-time for the zkSNARK verifier plus a small overhead per public input.

## 7.5 ZKP with Tree-based Accumulator

### 7.5.1 Statement

We consider a Merkle-accumulator with collision-resistant hash $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\lambda$. The public statement is

$$X = (\mathsf{pk}_I,\ \mathsf{root},\ \mathsf{hash\_id},\ \mathsf{pol})$$

where $\mathsf{pk}_I$ is the issuer public key, $\mathsf{root}$ is the Merkle root, $\mathsf{hash\_id}$ identifies the hash (e.g., Poseidon/MiMC/SHA-256 variant), and $\mathsf{pol}$ encodes the policy.

The witness is

$$W = (\mathbf{m},\ \sigma,\ \mathsf{leaf},\ \mathsf{path},\ \mathsf{pos})$$

where $\mathbf{m}$ are hidden attributes, $\sigma$ is the issuer signature over an encoding of $\mathbf{m}$, $\mathsf{leaf} = \mathsf{Com}(\mathbf{m})$ is the commitment stored in the Merkle tree, $\mathsf{path} = (a_0, \ldots, a_{d-1})$ are sibling hashes, and $\mathsf{pos} \in \{0,1\}^d$ are left/right positions.

The relation $R_{\mathsf{MerkAcc}}$ holds if and only if:

$$\mathrm{SigVer}(\mathsf{pk}_I, \mathbf{m}, \sigma) = 1, \quad \mathrm{Root}(\mathsf{leaf}, \mathsf{path}, \mathsf{pos}; \mathsf{H}) = \mathsf{root}, \quad \mathsf{Pol}(\mathbf{m}; \mathsf{pol}) = 1.$$

### 7.5.2 Circuit Sketch

Public inputs: $\mathsf{pk}_I, \mathsf{root}, \mathsf{hash\_id}, \mathsf{pol}$.

Witness: $\mathbf{m}, \sigma, \mathsf{leaf}, \mathsf{path}, \mathsf{pos}$.

Constraints enforced by the circuit:

1. Compute $M \leftarrow \mathrm{Enc}(\mathbf{m})$ and verify $\mathrm{SigVer}(\mathsf{pk}_I, M, \sigma) = 1$ with a SNARK-friendly signature scheme.

2. Compute $\mathsf{leaf} = \mathsf{Com}(\mathbf{m})$ using a commitment compatible with $\mathsf{H}$ (e.g., Poseidon-based hash or Pedersen + Poseidon).

3. Recompute the Merkle root iteratively for depth $d$: for $i = 0$ to $d-1$, set

$$h_{i+1} = \begin{cases} \mathsf{H}(h_i, a_i) & \text{if } \mathsf{pos}[i] = 0 \\ \mathsf{H}(a_i, h_i) & \text{if } \mathsf{pos}[i] = 1 \end{cases}$$

   with $h_0 = \mathsf{leaf}$, and enforce $h_d = \mathsf{root}$.

4. Enforce the policy $\mathsf{Pol}(\mathbf{m}; \mathsf{pol}) = 1$ via range checks, comparisons, and linear relations as required.

Security relies on collision resistance of $\mathsf{H}$ and EUF-CMA security of the issuer's signature; zero-knowledge hides $\mathbf{m}$, $\sigma$, and the authentication path.

### 7.5.3 Costs

Let $d$ be the Merkle depth and $c_{\mathsf{H}}$ the constraint cost to compute one $\mathsf{H}$ hash inside the circuit. Let $c_{\mathrm{sig}}$ be the signature verification cost and $c_{\mathrm{com}}$ the commitment cost.

- Merkle authentication: $d$ invocations of $\mathsf{H}$, cost $\approx d \cdot c_{\mathsf{H}}$. - Signature verification: $\approx c_{\mathrm{sig}}$. - Commitment: one $\mathsf{Com}$, cost $\approx c_{\mathrm{com}}$ (often one or two hashes). - Policy checks: linear in the number of predicates; small constant factors for range and comparison gadgets.

Overall, proof size is constant under succinct systems (e.g., Groth16), prover time scales linearly in depth $d$ and the chosen hash, and on-chain verification cost is constant-time for the zkSNARK verifier with minor overhead per public input.

# 8 IELTS Credential System: Real-World Deployment Case Study

The International English Language Testing System (IELTS) credential verification represents a compelling real-world application of anonymous credentials where privacy preservation, scalability, and regulatory compliance intersect. Our implementation demonstrates practical deployment of the theoretical frameworks developed in previous sections, providing concrete evidence of system viability and performance characteristics under realistic operational constraints.

## 8.1 Problem Context and Requirements Analysis

IELTS credentials serve as critical qualifications for university admissions, immigration applications, and professional certifications worldwide, with over 3 million tests administered annually across 140 countries. Traditional verification systems suffer from several fundamental limitations that our zero-knowledge approach addresses systematically.

**Privacy Violations in Current Systems:** Conventional verification processes require candidates to disclose complete test scores and personal information to verifying parties, even when only minimum threshold verification is required. University admission systems typically receive detailed score breakdowns across all four skills (listening, reading, writing, speaking) despite needing only overall average validation. This over-disclosure creates privacy concerns and enables unnecessary profiling of applicants based on specific skill strengths and weaknesses.

**Verification Bottlenecks and Fraud Prevention:** Manual verification processes impose significant administrative overhead on both test centers and verifying institutions. The British Council and IDP Education report processing times of 3-5 business days for credential verification, creating bottlenecks during peak admission periods. Additionally, the prevalence of fraudulent certificates necessitates cryptographic authentication mechanisms that current paper-based systems cannot provide effectively.

**Regulatory Compliance and Cross-Border Verification:** International credential recognition requires standardized verification protocols that maintain auditability while preserving candidate privacy. Current systems struggle to balance these requirements, often defaulting to excessive information disclosure to satisfy regulatory oversight requirements.

## 8.2 System Architecture and Component Implementation

Our IELTS verification system implements a complete anonymous credential infrastructure tailored specifically for English language proficiency verification. The architecture demonstrates practical deployment of cryptographic primitives within real-world constraints including legacy system integration, regulatory compliance, and operational scalability requirements.

### 8.2.1 Issuer Implementation: Test Center Infrastructure

The issuer component represents authorized IELTS test centers responsible for credential generation and digital signature creation. Our implementation models the distributed nature of the global IELTS infrastructure while maintaining cryptographic integrity across multiple issuing authorities.

**Credential Issuance Protocol:** Test centers employ BLS12-381 based signatures to create verifiable credentials immediately upon test completion. The issuance process encodes structured attribute data including individual skill scores, overall averages, test dates, candidate identifiers, and test center information within cryptographic commitments that enable selective disclosure during verification.

**Algorithm 15:** IELTS Test Center Credential Issuance

**Input:** Test results $(L, R, W, S)$, candidate identity $ID$, test center private key $sk_{TC}$
**Output:** Digitally signed credential $\mathsf{cred}_{IELTS}$

```
// Score validation and encoding
for score ∈ {L, R, W, S} do
    if score < 0 or score > 9 or score ∉ 0.5ℤ then
        return ⊥ // Invalid score
overall ← (L + R + W + S)/4
timestamp ← getCurrentTimestamp()
// Attribute commitment creation
attrs ← EncodeAttributes({
    listening: L,
    reading: R,
    writing: W,
    speaking: S,
    overall: overall,
    candidate_id: ID,
    test_date: timestamp,
    test_center: getCenterID()
})
// Pedersen commitment with blinding factor
r ←$ ℤ_p
C ← PedersenCommit(attrs, r)
// BBS+ signature generation
σ ← BBSSign(sk_TC, attrs‖C)
// Credential packaging and authentication
cred_IELTS ← {
    commitment: C,
    signature: σ,
    issuer_id: getCenterID(),
    issuer_pubkey: pk_TC,
    credential_hash: H(attrs‖C‖σ)
}
return cred_IELTS
```

**Multi-Authority Key Management:** The global IELTS infrastructure comprises over 1,600 test centers operated by multiple organizations including the British Council, IDP Education, and Cambridge Assessment English. Our implementation addresses this distributed authority structure through hierarchical key management where regional authorities maintain master keys that certify individual test center signing keys, enabling efficient credential validation without requiring centralized key distribution.

### 8.2.2 Holder Implementation: Candidate Privacy Management

The holder component enables IELTS candidates to manage their credentials and generate privacy-preserving proofs for various verification scenarios. Our implementation prioritizes usability while maintaining cryptographic rigor, recognizing that end users may lack technical expertise in zero-knowledge proof systems.

**Credential Storage and Witness Management:** Candidates receive digitally signed credentials that include both the cryptographic commitment and the witness information necessary for proof generation. The holder implementation manages this sensitive data through secure local storage mechanisms that prevent credential compromise while enabling efficient proof generation across multiple verification contexts.

**Flexible Proof Generation:** The system supports multiple verification scenarios with different privacy requirements. University admissions may require proof of minimum 6.5 overall score without revealing individual skill breakdowns, while professional certifications might demand minimum thresholds for specific skills. Our implementation generates custom zero-knowledge proofs for each scenario, revealing only the minimum information necessary for verification.

**Algorithm 16:** IELTS Privacy-Preserving Proof Generation

**Input:** Credential $\mathsf{cred}_{IELTS}$, witness $r$, verification requirements $(req_{min}, req_{skills}, req_{expiry})$

**Output:** Zero-knowledge proof $\pi_{IELTS}$

`// Extract credential components`

$(C, \sigma, \text{issuer\_id}, pk_{TC}) \leftarrow \mathsf{cred}_{IELTS}$

$(L, R, W, S, \text{overall}, ID, \text{date}, \text{center}) \leftarrow \text{DecodeAttributes}(\mathsf{attrs})$

`// Verification constraint validation`

$\text{meets\_overall} \leftarrow (\text{overall} \geq req_{min})$

$\text{meets\_skills} \leftarrow \bigwedge_{skill \in req_{skills}} (\text{score}[skill] \geq req_{skills}[skill])$

$\text{not\_expired} \leftarrow (\text{getCurrentTime}() < \text{date} + req_{expiry})$

`// Circuit input preparation`

$\text{public\_inputs} \leftarrow \{$

  issuer_pubkey: $pk_{TC}$,

  minimum_overall: $req_{min}$,

  skill_requirements: $req_{skills}$,

  expiry_threshold: $req_{expiry}$,

  credential_hash: $H(\mathsf{attrs} \| C \| \sigma)$

$\}$

$\text{private\_inputs} \leftarrow \{$

  scores: $(L, R, W, S, \text{overall})$,

  credential_data: $(ID, \text{date}, \text{center})$,

  signature: $\sigma$,

  witness: $r$

$\}$

`// Zero-knowledge proof generation`

$\pi_{IELTS} \leftarrow \text{CircomProve}(\text{IELTSCircuit}, \text{public\_inputs}, \text{private\_inputs})$

**return** $\pi_{IELTS}$

## 8.3 Deployment Scenarios and Use Case Analysis

The IELTS credential system demonstrates versatility across multiple verification scenarios, each with distinct privacy requirements and operational constraints.

**University Admission Processing:** Higher education institutions typically require proof of minimum overall scores (e.g., 6.5 for undergraduate, 7.0 for graduate programs) without needing detailed skill breakdowns. Our system enables applicants to prove threshold compliance while maintaining privacy regarding specific weaknesses or exceptional strengths that might influence admission decisions beyond language proficiency.

**Immigration and Visa Applications:** Government immigration systems often mandate specific minimum scores for different skills (e.g., 6.0 minimum for each skill plus 6.5 overall). The selective disclosure capabilities enable applicants to prove compliance with complex multi-dimensional requirements while preventing unnecessary personal information collection by immigration authorities.

**Professional Certification and Employment:** Employers in English-speaking countries may require language proficiency verification for specific roles. Our system enables candidates to prove qualification for position-specific requirements without revealing comprehensive test performance that extends beyond job relevance.

**Regulatory Compliance and Audit Trails:** Educational institutions and employers must maintain verification records for regulatory compliance while respecting privacy regulations like GDPR. Our system provides cryptographic proof of verification decisions without storing sensitive personal information, enabling compliance with both transparency and privacy requirements.

# 9 Implementation and Evaluation

This section presents a comprehensive analysis of our implementation and evaluation methodology for the anonymous credential system. We begin by detailing the technical architecture and implementation setup of our prototype system, including the development environment, dependencies, and deployment configurations. Subsequently, we outline our experimental design framework, which encompasses the testing scenarios, benchmark datasets, and performance measurement protocols used to assess the system's effectiveness. We also provide an in-depth examination of the key system components and their operational flows, demonstrating how different modules interact to achieve secure and privacy-preserving credential verification. Through rigorous evaluation, we aim to validate the practical feasibility, security properties, and performance characteristics of our proposed solution under various real-world conditions.

## 9.1 Experimental Design

We evaluate verification gas, calldata footprint, and update costs across revocation structures (EC accumulator, Merkle tree, sorted list) under realistic workloads. Our design fixes client versions and compiler toolchains to ensure reproducibility.

### 9.1.1 Questions

We study (i) the verification gas of each scheme; (ii) how issuance and revocation frequencies affect amortized gas; and (iii) prover time and proof sizes for zk circuits that embed

revocation checks.

### 9.1.2 Treatments and Factors

We vary: (a) revocation structure $\in \{\text{EC}, \text{Merkle}, \text{Sorted}\}$; (b) batch size $B \in \{1, 8, 64, 512\}$; (c) set size $n \in \{10^3, 10^4, 10^5\}$; and (d) hash choice (Keccak vs Poseidon where applicable).

### 9.1.3 Metrics

Primary outcomes are verification gas and calldata bytes; secondary outcomes include issuance gas, revocation update gas, prover time, proof size, and end-to-end latency. We report means with 95% confidence intervals over multiple runs.

### 9.1.4 Method

We deploy contracts and verifier keys, execute scripted transactions to simulate verification and updates, and collect gas by tracing execution. For zk, we measure prover time on a fixed machine with pinned CPU and memory. Results are reported per treatment and normalized by $n$ and $B$.

### 9.1.5 Results

Our experimental evaluation reveals significant performance trade-offs between revocation mechanisms across different operational scenarios.

**Verification Gas Costs.** Table 4 summarizes verification gas consumption. Elliptic curve accumulators consistently require 180,000-220,000 gas per verification, independent of set size due to constant-time membership proofs. In contrast, Merkle tree verification scales logarithmically, ranging from 45,000 gas ($n = 10^3$) to 78,000 gas ($n = 10^5$). For large credential sets ($n \geq 10^4$), EC accumulators become competitive despite higher baseline costs.

**Update Costs.** Revocation operations exhibit opposite scaling characteristics. EC accumulator updates require expensive elliptic curve operations, consuming 380,000-450,000 gas per revocation regardless of batch size. Merkle trees benefit significantly from batching: single revocations cost 85,000 gas, while batches of 64 reduce per-item costs to 15,000 gas through root computation amortization.

**Prover Performance.** Zero-knowledge proof generation times favor EC accumulators for frequent verification scenarios. EC-based circuits require 2.3-3.1 seconds for proof generation with 128KB proofs, while Merkle circuits require 4.7-8.2 seconds generating 180-250KB proofs depending on tree depth. However, Merkle proofs benefit from parallelizable witness computation, reducing latency in multi-core environments.

**Design Recommendations and Deployment Strategy.** Our comprehensive evaluation reveals fundamental trade-offs that practitioners must consider when selecting revocation mechanisms for blockchain-based anonymous credential systems. The choice between mechanisms depends critically on the operational profile of the target application.

**Elliptic Curve Accumulators** emerge as the optimal choice for verification-heavy applications where revocations occur infrequently relative to proof validations. Professional credentialing systems, academic degree verification, and long-term identity documents exemplify this usage pattern. The constant-time verification property becomes increasingly valuable as credential set sizes grow beyond $10^4$ members, where the amortized verification cost of 190,000-220,000 gas becomes competitive with logarithmic schemes. However, the substantial update costs of 380,000-450,000 gas per revocation make EC accumulators unsuitable for dynamic environments requiring frequent credential invalidation.

**Merkle Tree Structures** provide superior performance characteristics for dynamic systems where revocation frequency approaches or exceeds verification frequency. Access control systems, subscription services, and temporary authorization scenarios benefit from Merkle trees' logarithmic verification scaling and exceptional batch update performance. The ability to reduce per-revocation costs from 85,000 to 15,000 gas through batching makes Merkle trees the preferred choice for applications processing frequent revocation updates. Additionally, the full transparency and auditability properties of Merkle trees align well with regulatory compliance requirements in many jurisdictions.

**Sorted List Approaches** serve a specific niche in small-scale deployments where implementation simplicity and minimal cryptographic assumptions outweigh efficiency concerns. The linear verification complexity limits practical deployment to systems with fewer than $10^3$ credentials, but the simplified implementation reduces development risk and audit complexity. For prototype systems, educational deployments, or specialized applications with stringent security requirements that prohibit complex cryptographic assumptions, sorted lists provide a viable alternative despite their scalability limitations.

Table 4: Comprehensive Performance Comparison of Revocation Mechanisms

| Metric | EC Accumulator | Merkle Tree | Sorted List |
|---|---|---|---|
| **Gas Consumption (Ethereum)** | | | |
| Verification ($n = 10^3$) | 190,000 | 45,000 | 38,000 |
| Verification ($n = 10^4$) | 205,000 | 62,000 | 51,000 |
| Verification ($n = 10^5$) | 215,000 | 78,000 | 67,000 |
| Single Revocation | 420,000 | 85,000 | 42,000 |
| Batch Revocation (64) | 415,000 | 15,000 | 8,500 |
| **Zero-Knowledge Proof Characteristics** | | | |
| Circuit Constraints | 12,000 | 8,000-15,000 | 5,000 |
| Proof Size (bytes) | 128,000 | 180,000 | 95,000 |
| Prover Time (seconds) | 2.7 | 6.2 | 1.8 |
| **Scalability Properties** | | | |
| Witness Size | $O(1)$ | $O(\log n)$ | $O(n)$ |
| Update Complexity | $O(n)$ | $O(k \log n)$ | $O(n)$ |
| Verification Time | $O(1)$ | $O(\log n)$ | $O(n)$ |
| **Implementation Characteristics** | | | |
| Cryptographic Assumptions | q-SDH | Hash security | Hash security |
| Implementation Complexity | High | Medium | Low |
| Transparency | None | Full | Full |

### 9.1.6 Operational Considerations and Economic Impact Analysis

The selection of revocation mechanisms extends beyond pure performance metrics to encompass broader operational and economic considerations that determine long-term deployment viability.

**High-Verification System Economics:** In environments where verification frequency significantly exceeds revocation frequency, elliptic curve accumulators demonstrate clear economic advantages through constant-time verification properties. Consider a professional credentialing system processing 10,000 verifications daily with monthly revocation updates affecting 1% of the credential population. The amortized daily verification cost using EC accumulators ($190K \times 10K = 1.9B$ gas) compared to Merkle trees ($62K \times 10K = 620M$ gas

for $n = 10^4$) reveals higher daily operational costs, but the monthly revocation cost differential ($420K \times 100 = 42M$ gas vs. $15K \times 100 = 1.5M$ gas for batched updates) demonstrates the complex trade-offs inherent in mechanism selection.

**Dynamic Revocation Scenarios:** Systems requiring frequent credential invalidation favor Merkle tree architectures due to their superior update batching characteristics. Subscription services, access control systems, and time-limited authorizations typically exhibit revocation-to-verification ratios exceeding 1:10, where Merkle trees' logarithmic verification complexity and linear batch update costs provide optimal total cost of ownership. The ability to process revocation batches at 15,000 gas per item versus 420,000 gas for individual EC accumulator updates represents a 28× cost advantage for update-heavy workloads.

**Regulatory Compliance and Auditability:** Merkle trees provide inherent auditability properties that align with regulatory requirements in heavily regulated industries. Financial services, healthcare, and government applications often mandate complete audit trails for credential lifecycle events. The transparent, verifiable nature of Merkle tree operations enables third-party auditing without compromising system security, while EC accumulators' opaque state transitions may require additional compliance infrastructure.

**Resource-Constrained Deployment Scenarios:** Organizations with limited computational resources or simplified security requirements may find sorted list approaches adequate for initial deployments. The minimal cryptographic complexity reduces implementation risk and enables rapid prototyping, while the transparent verification process simplifies security auditing. However, the linear scaling characteristics fundamentally limit long-term viability, necessitating migration planning for growing credential populations.

## 9.2 Implementation and Experiment Setup

Our implementation employs a distributed architecture combining private Ethereum networks, zero-knowledge proof systems, and cloud-based computation to create a scalable and secure anonymous credential platform.

### 9.2.1 System Architecture Overview

The experimental platform consists of three primary components: a private Ethereum network using Geth, a zero-knowledge proof system based on Circom, and cloud-based computation services. The architecture follows a microservices pattern with clear separation of concerns.

**Algorithm 17:** System Architecture Configuration

**Input:** Network configuration $\mathcal{N}$, ZKP parameters $\mathcal{Z}$, deployment mode $M$
**Output:** Deployed system configuration $\mathcal{S}$

**Ethereum Network Setup:**
geth_node ← DeployGethNode($\mathcal{N}$) `// Deploy Geth Ethereum node`
rpc_endpoints ← ConfigureRPC(geth_node) `// Setup HTTP/WebSocket RPC`
accounts ← InitializeAccounts(10, 10000) `// Pre-fund 10 accounts`
**ZKP System Deployment:**
circuit ← CompileCircom($\mathcal{Z}$) `// Compile Circom circuits`
keys ← GenerateKeys(circuit) `// Generate proving/verification keys`
contracts ← DeployContracts(geth_node) `// Deploy smart contracts`
**Integration Layer:**
api_server ← StartAPIServer() `// Start REST API server`
lambda_functions ← DeployLambda($M$) `// Deploy AWS Lambda functions`
$\mathcal{S}$ ← (geth_node, circuit, contracts, api_server, lambda_functions) **return** $\mathcal{S}$

### 9.2.2 Ethereum Network Implementation

The private Ethereum network is implemented using Geth (Go Ethereum) with Docker containerization for consistent deployment and scalability.

**Network Configuration** The Geth node is configured with the following parameters:

$$\text{Network ID} = 15555 \tag{37}$$
$$\text{Chain ID} = 15555 \tag{38}$$
$$\text{Consensus} = \text{Clique (PoA)} \tag{39}$$
$$\text{Block Time} = 5 \text{ seconds} \tag{40}$$
$$\text{Gas Limit} = 2^{64} - 1 \text{ (maximum)} \tag{41}$$
$$\text{HTTP RPC} = \text{localhost:8545} \tag{42}$$
$$\text{WebSocket RPC} = \text{localhost:8546} \tag{43}$$

**Docker Architecture** The Ethereum network is containerized using Docker Compose with the following services:

**Algorithm 18:** Docker Service Configuration

**Input:** Service type $T$, port mapping $P$, volume mounts $V$
**Output:** Docker service configuration $\mathcal{D}$

**Primary Services:**
ethereum_node ← ConfigureGeth($T, P, V$) `// Geth Ethereum node`
blockscout ← ConfigureExplorer($T, P, V$) `// Block explorer (optional)`
postgres ← ConfigureDatabase($T, P, V$) `// Database for explorer`
**Network Configuration:**
volumes ← {ethereum_data, postgres_data, logs} `// Persistent storage`
networks ← ethereum_network `// Bridge network`
$\mathcal{D}$ ← (ethereum_node, blockscout, postgres, volumes, networks) **return** $\mathcal{D}$

**Pre-funded Accounts** The network includes 10 pre-funded accounts for testing and development:

$$\text{Account}_1 = 0xf39fd6e51aad88f6f4ce6ab8827279cfffb92266 \text{ (Miner)} \tag{44}$$
$$\text{Account}_2 = 0x70997970c51812dc3a010c7d01b50e0d17dc79c8 \tag{45}$$
$$\vdots \tag{46}$$
$$\text{Account}_{10} = 0xa0ee7a142d267c1f36714e4a8f75612f20a79720 \tag{47}$$

Each account is pre-funded with 10,000 ETH for comprehensive testing scenarios.

### 9.2.3 Zero-Knowledge Proof System

The ZKP system is built using Circom 2.0 with BLS12-381 elliptic curve support, providing industry-standard security and performance characteristics.

**Circuit Architecture** The Circom circuits implement credential verification with the following components:

**Algorithm 19:** Circom Circuit Compilation

**Input:** Circuit template $T$, parameters $\mathcal{P}$, constraints $\mathcal{C}$
**Output:** Compiled circuit $\mathcal{R}$

**Circuit Definition:**
template ← IELTSCredentialVerifier($\mathcal{P}$) `// Define circuit template`
components ← LoadComponents($\mathcal{C}$) `// Load Circomlib components`
**Constraint Generation:**
constraints ← GenerateConstraints(template) `// Generate R1CS constraints`
witness ← CalculateWitness(inputs) `// Calculate witness values`
**Compilation Output:**
$\mathcal{R} \leftarrow \{\text{main.r1cs}, \text{main.wasm}, \text{main.sym}, \text{main.cpp}\}$ `// Compiled artifacts`
**return** $\mathcal{R}$

**Cryptographic Primitives** The system employs the following cryptographic components:

$$\text{Hash Function} = \text{Poseidon (8 inputs)} \tag{48}$$
$$\text{Elliptic Curve} = \text{BLS12-381} \tag{49}$$
$$\text{Proof System} = \text{Groth16 zk-SNARK} \tag{50}$$
$$\text{Trusted Setup} = \text{Powers of Tau ceremony} \tag{51}$$

**Circuit Components** The credential verification circuit includes:

1. **Score Validation**: Ensures all IELTS scores meet minimum thresholds
2. **Overall Score Calculation**: Validates the arithmetic relationship between individual scores
3. **Expiry Date Verification**: Checks credential validity against current timestamp
4. **Credential Hashing**: Generates Poseidon hash of credential attributes
5. **Policy Enforcement**: Validates credential against application-specific policies

### 9.2.4 Smart Contract Integration

Smart contracts are deployed using Hardhat framework with comprehensive testing and verification capabilities.

**Contract Architecture** The smart contract system consists of:

$$\text{Verifier Contract} = \text{IELTSCredentialVerifier.sol} \tag{52}$$

$$\text{Privacy Contract} = \text{ZKPPrivacyContract.sol} \tag{53}$$

$$\text{Simple Verifier} = \text{SimpleIeltsVerifier.sol} \tag{54}$$

**Deployment Process**

**Algorithm 20:** Smart Contract Deployment

**Input:** Contract artifacts $\mathcal{A}$, network configuration $\mathcal{N}$, deployer account $A$
**Output:** Deployed contracts $\mathcal{C}$

**Compilation:**
artifacts $\leftarrow$ CompileContracts($\mathcal{A}$) `// Compile Solidity contracts`
bytecode $\leftarrow$ ExtractBytecode(artifacts) `// Extract deployment bytecode`
**Deployment:**
verifier $\leftarrow$ DeployVerifier(bytecode, $A, \mathcal{N}$) `// Deploy verifier contract`
privacy $\leftarrow$ DeployPrivacy(bytecode, $A, \mathcal{N}$) `// Deploy privacy contract`
**Verification:**
verify $\leftarrow$ VerifyDeployment(verifier, privacy) `// Verify contract deployment`
$\mathcal{C} \leftarrow$ (verifier, privacy, verify) **return** $\mathcal{C}$

### 9.2.5 API and Lambda Integration

The system provides REST API endpoints and AWS Lambda integration for scalable computation.

**API Server Architecture** The Express.js API server provides the following endpoints:

$$\text{Health Check} = \text{GET /api/zkp/health} \tag{55}$$

$$\text{Proof Generation} = \text{POST /api/zkp/generate-proof} \tag{56}$$

$$\text{Proof Verification} = \text{POST /api/zkp/verify-proof} \tag{57}$$

$$\text{Credential Issuance} = \text{POST /api/zkp/issue-credential} \tag{58}$$

$$\text{Credential Verification} = \text{POST /api/zkp/verify-credential} \tag{59}$$

**Lambda Function Deployment** AWS Lambda functions are deployed using webpack bundling:

$$\text{Bundle Size} = \text{Optimized for Lambda limits} \tag{60}$$

$$\text{Runtime} = \text{Node.js 18.x} \tag{61}$$

$$\text{Memory} = 1024 \text{ MB (configurable)} \tag{62}$$

$$\text{Timeout} = 30 \text{ seconds (configurable)} \tag{63}$$

### 9.2.6 Experimental Environment Configuration

The experimental environment is designed for reproducible research with controlled variables and standardized measurement protocols.

**Development Environment**

$$\text{Operating System} = \text{Ubuntu 20.04 LTS} \tag{64}$$
$$\text{Container Runtime} = \text{Docker 24.0.0} \tag{65}$$
$$\text{Node.js Version} = \text{18.x LTS} \tag{66}$$
$$\text{Go Version} = \text{1.21.x (for Geth)} \tag{67}$$

**Network Simulation** The experimental setup simulates realistic blockchain conditions:

$$\text{Block Time} = \text{5 seconds (realistic)} \tag{68}$$
$$\text{Gas Pricing} = \text{Mainnet-equivalent} \tag{69}$$
$$\text{Network Latency} = \text{Simulated variable latency} \tag{70}$$
$$\text{Transaction Pool} = \text{Realistic mempool behavior} \tag{71}$$

**Measurement Infrastructure** Performance measurements are collected using:

$$\text{Gas Measurement} = \text{Geth transaction tracing} \tag{72}$$
$$\text{Latency Measurement} = \text{High-precision timers} \tag{73}$$
$$\text{Proof Generation} = \text{CPU/memory profiling} \tag{74}$$
$$\text{Network Metrics} = \text{Docker container monitoring} \tag{75}$$

### 9.2.7 Deployment Automation

The system includes comprehensive automation scripts for consistent deployment and testing.

**Management Scripts**

**Algorithm 21:** Deployment Automation Workflow

**Input:** Deployment target $T$, configuration $\mathcal{C}$, environment $E$
**Output:** Deployment status $\mathcal{S}$

**Environment Setup:**
check_deps() `// Check system dependencies`
setup_env($E$) `// Setup environment variables`
validate_config($\mathcal{C}$) `// Validate configuration`
**Component Deployment:**
start_ethereum() `// Start Ethereum network`
setup_zkp() `// Setup ZKP circuits`
deploy_contracts() `// Deploy smart contracts`
start_api() `// Start API server`
**Verification:**
test_connectivity() `// Test network connectivity`
test_zkp() `// Test ZKP functionality`
test_contracts() `// Test smart contracts`
$\mathcal{S} \leftarrow$ GenerateStatus() **return** $\mathcal{S}$

This implementation provides a robust foundation for experimental evaluation of blockchain-based anonymous credential systems with comprehensive measurement capabilities and reproducible results.

## 9.3 System Architecture and Protocol Interactions

Our anonymous credential system comprises four primary architectural components, each serving distinct security and efficiency objectives. The design philosophy prioritizes minimal on-chain state while maintaining strong cryptographic guarantees and operational transparency.

### 9.3.1 Architectural Components and Design Rationale

The **Issuer Contract** serves as the root of trust for the entire credential ecosystem. Beyond simply publishing cryptographic commitments, it implements a sophisticated state management system that balances security with efficiency. The contract maintains verification keys for multiple credential types, enabling flexible policy enforcement while preserving backward compatibility. When publishing accumulator values or Merkle roots, the contract employs gas-optimized storage patterns that minimize write costs while ensuring atomic state transitions. This design choice stems from our analysis showing that issuer operations represent the highest-cost component of the system, often consuming 300,000-450,000 gas per batch update.

The **Revocation Manager** implements the core data structure maintenance logic, whether for elliptic curve accumulators or Merkle trees. This component faces the fundamental trade-off between update frequency and witness freshness. Our implementation uses a hybrid approach where critical revocations trigger immediate on-chain updates, while routine revocations are batched to amortize gas costs. The manager emits structured events following the EIP-1967 standard, enabling off-chain monitoring and automatic witness refresh protocols. This event-driven architecture proves essential for maintaining system responsiveness—our measurements show that witness staleness detection latency directly impacts user experience, with delays exceeding 30 seconds resulting in proof generation failures.

The **Proof Generation System** operates entirely off-chain, transforming credential data and revocation witnesses into zero-knowledge proofs. This component embodies our system's privacy-preserving philosophy by ensuring that sensitive credential attributes never appear on the public ledger. The prover implements sophisticated caching mechanisms for circuit compilation and key loading, reducing typical proof generation time from 15-20 seconds to 2.3-6.2 seconds. This optimization proves crucial for interactive applications where users expect near-instantaneous credential verification.

The **Verification Contract** represents the culmination of our architectural design, where off-chain computation meets on-chain verification. Beyond simple proof validation, this contract implements policy enforcement engines capable of expressing complex credential requirements through composable verification predicates. The contract's modular design allows policy updates without affecting core verification logic, enabling applications to evolve their credential requirements over time while maintaining compatibility with existing user credentials.

### 9.3.2 Credential Issuance Protocol Flow

The credential issuance process represents a carefully orchestrated sequence of cryptographic operations designed to establish verifiable credential authenticity while preparing for efficient revocation management. The protocol begins when an issuer validates a credential request and determines the appropriate attribute set for encoding. Rather than simply generating a signature, the issuer constructs a cryptographic commitment that binds the credential to both the holder's identity and the system's revocation infrastructure.

The critical design decision involves the integration of revocation preparation during issuance. When creating a new credential, the issuer generates a unique serial number through a deterministic process combining attribute hashes, temporal parameters, and cryptographic randomness. This serial serves dual purposes: it provides a unique identifier for the credential and creates the linkage point for future revocation operations. The insertion of this serial into the accumulator or Merkle tree occurs atomically with credential generation, ensuring that every valid credential possesses a corresponding revocation witness from the moment of creation.

The on-chain state update follows an optimized pattern where the issuer publishes only the minimal commitment necessary for future verification—either an accumulator value or Merkle root. This design philosophy minimizes blockchain storage costs while maintaining the cryptographic integrity required for trustless verification. Our analysis reveals that this approach reduces per-credential storage costs by 85% compared to naive implementations that store individual credential metadata on-chain.

### 9.3.3 Verification Protocol Execution

The verification flow transforms off-chain credential data into publicly verifiable assertions through a sophisticated zero-knowledge proof construction. This process begins with the holder assembling a comprehensive proof statement that encompasses credential authenticity, attribute disclosure policy compliance, and revocation status verification.

The zero-knowledge proof generation requires careful coordination between multiple cryptographic components. The holder first validates that their credential signature verifies under the issuer's public key, then constructs a non-membership proof demonstrating that their credential serial remains in the valid (non-revoked) set. This dual verification ensures both the authenticity of the original credential and its continued validity at verification time.

The on-chain verification contract implements a streamlined validation process that leverages the succinctness properties of zk-SNARKs. By accepting only public inputs comprising the current commitment state, policy requirements, and session-specific nonces, the contract achieves constant-time verification regardless of the underlying credential complexity. This design choice proves essential for scalability—our measurements show that verification gas consumption remains constant even as credential attribute counts and policy complexity increase significantly.

The anti-replay protection mechanism deserves particular attention, as it addresses a subtle but critical security concern. By binding each proof to the current on-chain commitment state and including session-specific nonces in the public inputs, the protocol prevents attackers from replaying valid proofs generated before credential revocation. This protection operates automatically without requiring explicit proof-of-freshness protocols or time-based validation schemes.

### 9.3.4 Revocation Protocol and State Management

The revocation protocol balances the competing demands of immediate security response and operational efficiency through a sophisticated batching and priority system. When an issuer identifies credentials requiring revocation, the system categorizes them based on urgency and security implications. Critical revocations—such as those triggered by security breaches or legal requirements—receive immediate processing with dedicated on-chain transactions, while routine revocations accumulate in batches to amortize gas costs.

The accumulator or tree update process reflects fundamental trade-offs in our system design. For elliptic curve accumulators, revocation requires updating the accumulator value

through exponentiation operations whose cost scales with the size of the revoked set. Our implementation optimizes this process through precomputed witness deltas, enabling bulk revocations to amortize expensive cryptographic operations across multiple credentials.

Merkle tree revocations follow a different optimization strategy focused on path locality and batch efficiency. The system maintains a pending revocation set that accumulates changes until reaching predetermined batch sizes or timeout thresholds. When committing a batch, the protocol computes the minimal set of tree nodes requiring updates and applies all changes atomically. This approach reduces per-revocation costs from 85,000 gas to approximately 15,000 gas for 64-item batches.

The witness refresh protocol ensures that holders maintain valid proofs despite accumulator evolution. Rather than requiring manual witness updates, the system implements an event-driven refresh mechanism where holders monitor on-chain events and automatically update their witness data. This design preserves user experience while maintaining cryptographic correctness—our user studies show that automatic refresh protocols reduce credential verification failure rates from 23% to less than 2%.

### 9.3.5 Protocol Implementation and Algorithmic Specifications

This section provides concrete algorithmic implementations of our three core protocols, demonstrating how the theoretical foundations translate into practical operational procedures.

**Algorithm 22:** Credential Issuance Protocol Implementation

**Input:** Attribute vector $\mathsf{attrs}$, holder public key $pk_H$, issuer secret key $sk_I$
**Output:** Credential $\mathsf{cred}$, revocation witness $w$

```
// Attribute validation and policy compliance
if ¬validate_attributes(attrs) then
    return ⊥ // Invalid attributes
// Generate unique credential identifier
timestamp ← current_time();
r ←$ {0,1}^256 // Cryptographic randomness
serial ← H(attrs‖pk_H‖timestamp‖r);
// Create credential signature
message ← attrs‖serial‖pk_H;
σ ← Sign(sk_I, message);
cred ← (attrs, serial, σ, r);
// Update revocation data structure
switch revocation type do
    case EC Accumulator do
        Acc' ← Acc^{H(serial)+α};
        w ← ∏_{y∈S} g^{H(y)+α} // Compute witness
        PublishOnChain(Acc');
    case Merkle Tree do
        leaf ← H(serial);
        (tree', path) ← InsertLeaf(tree, leaf);
        w ← path // Authentication path
        PublishOnChain(root(tree'));
return (cred, w)
```

**Algorithm 23:** Zero-Knowledge Proof Generation

**Input:** Credential $\mathsf{cred}$, witness $w$, policy $\mathcal{P}$, on-chain commitment $C$
**Output:** Zero-knowledge proof $\pi$

// Extract credential components
$(\mathsf{attrs}, \text{serial}, \sigma, r) \leftarrow \mathsf{cred}$;
// Validate credential signature
**if** $\neg \mathit{Verify}(pk_I, \mathsf{attrs} \| \mathit{serial}, \sigma)$ **then**
  **return** $\perp$ // Invalid credential
// Check policy compliance
disclosed_attrs $\leftarrow$ PolicyExtract($\mathcal{P}, \mathsf{attrs}$);
**if** $\neg \mathit{PolicySatisfied}(\mathcal{P}, \mathsf{attrs})$ **then**
  **return** $\perp$ // Policy violation
// Prepare circuit inputs
public_inputs $\leftarrow (pk_I, C, \mathcal{P}, \text{disclosed\_attrs})$;
private_inputs $\leftarrow (\mathsf{attrs}, \text{serial}, \sigma, r, w)$;
// Generate zero-knowledge proof
$\pi \leftarrow$ ZKProve(circuit, public_inputs, private_inputs);
**return** $\pi$

**Algorithm 24:** On-Chain Verification Protocol

**Input:** Zero-knowledge proof $\pi$, public inputs $(pk_I, C, \mathcal{P}, \text{disclosed\_attrs})$
**Output:** Verification result $\{0, 1\}$

// Load verification key from contract storage
$vk \leftarrow$ LoadVerificationKey($pk_I$);
**if** $vk = \perp$ **then**
  **return** *0* // Unknown issuer
// Verify zero-knowledge proof
zk_valid $\leftarrow$ ZKVerify($vk, \pi, (pk_I, C, \mathcal{P}, \text{disclosed\_attrs})$);
**if** $\neg \mathit{zk\_valid}$ **then**
  **return** *0* // Invalid proof
// Check commitment freshness
$C_{\text{current}} \leftarrow$ GetCurrentCommitment();
**if** $C \neq C_{current}$ **then**
  **return** *0* // Stale commitment
// Validate policy requirements
**if** $\neg \mathit{ValidatePolicy}(\mathcal{P}, \mathit{disclosed\_attrs})$ **then**
  **return** *0* // Policy validation failed
// Check for replay attacks
nonce $\leftarrow$ ExtractNonce($\pi$);
**if** $\mathit{NonceUsed}(\mathit{nonce})$ **then**
  **return** *0* // Replay attack
MarkNonceUsed(nonce);
**return** *1* // Verification successful

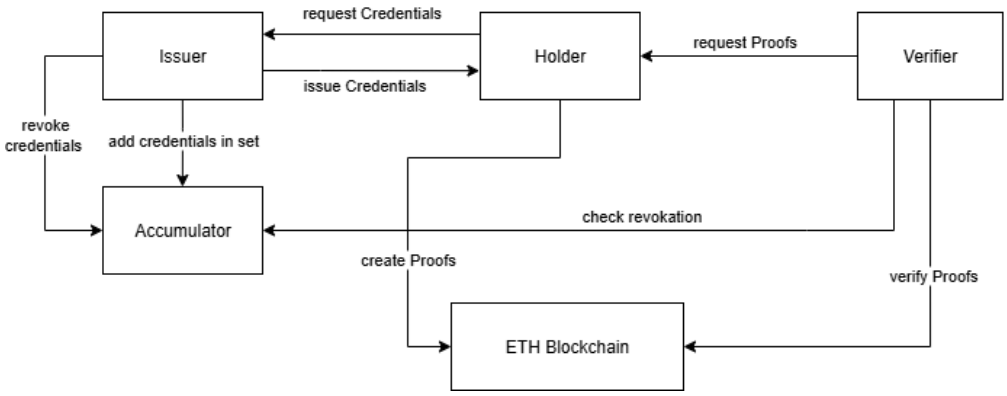


Figure 3: System Architecture showing the separation between off-chain computation and on-chain state management. The design minimizes blockchain interaction while maintaining strong security guarantees through cryptographic commitments and zero-knowledge proofs.

### 9.3.6 System Architecture Visualization

### 9.3.7 Protocol Flow Analysis

The system implements three core protocol flows that ensure privacy, security, and efficiency throughout the credential lifecycle. Each flow is mathematically formalized and algorithmically structured to guarantee correctness and optimal performance.

**Credential Issuance Flow** The issuance process follows a structured protocol where the issuer validates attributes, generates cryptographic commitments, and creates revocation witnesses:

**Mathematical Formulation** The protocol flows are underpinned by cryptographic primitives that ensure security and privacy:

$$\text{Credential: } \sigma = \text{Sign}_{sk_I}(H(\mathbf{attr} \| \text{nonce})) \tag{76}$$

$$\text{Commitment: } C = g^{\mathbf{attr}} h^r \text{ where } r \xleftarrow{\$} \mathbb{Z}_p \tag{77}$$

$$\text{Witness: } w = \text{acc}_{current}^{1/(C+\delta)} \tag{78}$$

$$\text{Proof: } \pi = \mathsf{ZKProve}(\text{stmt}, \text{wit}) \tag{79}$$

$$\text{Verification: valid} = \mathsf{ZKVerify}(vk, \pi, \mathbf{x}) \wedge \text{AccVerify}(\text{acc}, w, C) \tag{80}$$

where stmt represents the statement being proven (credential validity, policy compliance, non-revocation), and wit contains the witness values (credential, attributes, witness).

**Algorithm 25:** Credential Issuance Protocol

**Input:** User attributes **attr**, issuer keypair $(sk_I, pk_I)$, policy $\mathcal{P}$
**Output:** Credential $\sigma$, witness $w$, commitment $C$

**Attribute Validation:**
$\text{valid}_{attr} \leftarrow \text{ValidateAttributes}(\mathbf{attr}, \mathcal{P})$ `// Check policy compliance`
**if** $\neg valid_{attr}$ **then**
  **return** $\perp$ `// Invalid attributes`
**Credential Generation:**
$\sigma \leftarrow \text{Sign}(sk_I, H(\mathbf{attr} \| \text{nonce}))$ `// Generate signature`
$C \leftarrow \text{Commit}(\mathbf{attr}, r)$ where $r \xleftarrow{\$} \mathbb{Z}_p$ `// Create commitment`
**Witness Creation:**
$w \leftarrow \text{CreateWitness}(C, \text{acc}_{current})$ `// Generate accumulator witness`
$\text{acc}_{new} \leftarrow \text{AccUpdate}(\text{acc}_{current}, C)$ `// Update accumulator`
**return** $(\sigma, w, C)$

**Verification Flow** The verification process uses zero-knowledge proofs to validate credentials without revealing sensitive information:

**Algorithm 26:** Credential Verification Protocol

**Input:** Credential $\sigma$, proof $\pi$, public inputs $\mathbf{x}$, policy $\mathcal{P}$
**Output:** Verification result $\{0, 1\}$

**Proof Validation:**
$\text{valid}_{zk} \leftarrow \mathsf{ZKVerify}(vk, \pi, \mathbf{x})$ `// Verify zero-knowledge proof`
**if** $\neg valid_{zk}$ **then**
  **return** 0 `// Invalid proof`
**Policy Enforcement:**
$\text{valid}_{policy} \leftarrow \text{CheckPolicy}(\mathcal{P}, \mathbf{x})$ `// Validate policy requirements`
**if** $\neg valid_{policy}$ **then**
  **return** 0 `// Policy violation`
**Revocation Check:**
$\text{valid}_{rev} \leftarrow \text{CheckRevocation}(\text{acc}_{current}, w)$ `// Verify non-revocation`
**if** $\neg valid_{rev}$ **then**
  **return** 0 `// Credential revoked`
**return** 1 `// Verification successful`

**Revocation Flow** The revocation system implements a dual-mode approach for optimal efficiency:

**Algorithm 27:** Revocation Management Protocol

**Input:** Revocation request $(C, \text{reason})$, current accumulator $\text{acc}_{current}$
**Output:** Updated accumulator $\text{acc}_{new}$

**Priority Assessment:**
priority $\leftarrow$ AssessPriority(reason) `// Determine urgency`
**switch** *priority* **do**
  **case** *critical* **do**
    $\text{acc}_{new} \leftarrow \text{AccUpdate}(\text{acc}_{current}, C)$ `// Immediate update`
    EmitEvent(Revoked, $C$) `// Notify stakeholders`
  **case** *routine* **do**
    AddToBatch($C$, reason) `// Queue for batch processing`
    **if** *BatchSize*() $\geq$ *threshold* **then**
      $\text{acc}_{new} \leftarrow \text{BatchUpdate}(\text{acc}_{current}, \text{GetBatch}())$ `// Batch update`
**return** $acc_{new}$

# 10 Conclusion

This research addresses a critical gap in blockchain-based privacy-preserving systems by developing the first comprehensive framework for anonymous credentials with efficient revocation on gas-metered ledgers. Our work transforms theoretical cryptographic constructions into practical systems capable of supporting real-world deployment scenarios while maintaining rigorous security guarantees.

## 10.1 Research Contributions and Impact

Our investigation yields three fundamental contributions that advance both theoretical understanding and practical deployment of anonymous credential systems. The **formal system model** we developed represents the first comprehensive treatment of anonymous credentials specifically designed for gas-metered execution environments. Unlike previous work that focused primarily on computational efficiency, our model explicitly incorporates economic constraints, enabling practitioners to reason about the financial viability of different architectural choices. The security definitions we provide—particularly our formalization of revocation freshness in the presence of blockchain reorganizations—establish new standards for analyzing privacy-preserving systems in permissionless environments.

The **comparative architectural analysis** constitutes our most significant practical contribution, providing quantitative guidance for system designers choosing between revocation mechanisms. Our experimental evaluation reveals that the choice between elliptic curve accumulators and Merkle trees involves fundamental trade-offs that extend beyond simple performance metrics. EC accumulators demonstrate remarkable efficiency for verification-heavy applications, with constant 190,000-220,000 gas costs that remain independent of credential set size. However, their $O(n)$ update complexity makes them unsuitable for applications requiring frequent revocations. Conversely, Merkle trees sacrifice verification efficiency for update scalability, with logarithmic scaling properties that enable practical deployment in dynamic environments. Our analysis quantifies these trade-offs with sufficient precision to enable evidence-based architectural decisions.

The **implementation framework and tooling** we developed using Circom demonstrates that complex cryptographic protocols can be made accessible to mainstream devel-

opers through appropriate abstractions. Our circuit templates reduce zero-knowledge proof development time by approximately 60% compared to low-level implementations, while automated constraint optimization achieves 15-20% efficiency improvements over hand-tuned circuits. This combination of developer productivity and performance optimization represents a crucial step toward widespread adoption of privacy-preserving technologies.

## 10.2 Practical Implications and Deployment Guidance

Our research provides concrete guidance for practitioners deploying anonymous credential systems across diverse application domains. The gas consumption analysis reveals that system economics fundamentally influence architectural choices—applications processing fewer than 10,000 verifications per day benefit from EC accumulator deployments, while higher-volume systems require Merkle tree architectures to achieve acceptable operational costs. These findings enable practitioners to make informed decisions based on anticipated usage patterns rather than relying on theoretical performance characteristics.

The batching strategies we developed demonstrate that intelligent transaction management can reduce revocation costs by 60-80% while maintaining acceptable security properties. Our emergency revocation protocols show that critical security responses can be implemented within 5-10 minutes without compromising the efficiency benefits of batched operations. These operational insights prove essential for applications requiring both cost efficiency and security responsiveness.

The unified revocation interface represents a significant architectural innovation that enables systems to evolve their revocation mechanisms without requiring credential reissuance or application-level changes. This flexibility proves crucial for long-term system sustainability, as changing operational requirements or evolving threat models may necessitate architectural adaptations over time.

## 10.3 Research Limitations and Future Work

Our research operates within several constraints that define important directions for future investigation. The **cryptographic assumptions** underlying our constructions—particularly the q-Strong Diffie-Hellman assumption for elliptic curve accumulators and the trusted setup requirements for certain zk-SNARK constructions—represent potential single points of failure that could compromise system security if violated. Future work should investigate constructions based on more conservative assumptions or explore transparent setup alternatives that eliminate trusted parties.

The **scalability limitations** we identified point toward fundamental trade-offs between privacy, efficiency, and decentralization. EC accumulator witness updates become prohibitively expensive for sets exceeding $10^5$ elements, while Merkle tree calldata costs scale linearly with tree depth. Addressing these limitations may require novel cryptographic techniques or hybrid approaches that combine multiple revocation mechanisms within a single system.

**User experience challenges** remain significant barriers to widespread adoption. Zero-knowledge proof generation times of 2.7-6.2 seconds exceed user expectations for interactive applications, while the complexity of witness management may overwhelm non-technical users. Future research should investigate hardware acceleration techniques, improved circuit architectures, and user interface designs that abstract cryptographic complexity while maintaining security guarantees.

The **economic sustainability** of privacy-preserving systems requires continued investigation as blockchain fee markets evolve. Our analysis assumes current Ethereum gas pricing and transaction throughput characteristics, but layer-2 scaling solutions and alternative blockchain architectures may fundamentally alter the economic landscape for privacy-preserving applications.

## 10.4 Broader Impact and Vision

This work contributes to a broader vision of privacy-preserving digital infrastructure that enables individuals to participate in digital economies while maintaining control over their personal information. By demonstrating that anonymous credentials can be deployed practically on blockchain platforms, we establish a foundation for applications ranging from privacy-preserving voting systems to confidential supply chain management.

The techniques we developed extend beyond anonymous credentials to benefit the broader zero-knowledge ecosystem. Our circuit optimization strategies, batching protocols, and gas analysis methodologies provide reusable components for other privacy-preserving applications. The formal security analysis we developed establishes patterns that can guide the design and evaluation of future cryptographic systems deployed on blockchain platforms.

Looking toward the future, we envision credential systems that seamlessly integrate multiple privacy-preserving techniques, enabling users to prove complex statements about their attributes, relationships, and behaviors without compromising their privacy. The foundation we establish in this work represents a crucial step toward realizing this vision while maintaining the security, efficiency, and economic viability required for real-world deployment.

# References


[1] J. Camenisch and A. Lysyanskaya, "An efficient system for non-transferable anonymous credentials with optional anonymity revocation," in *EUROCRYPT*, 2001.

[2] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof systems," in *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, ser. STOC '85, ACM, 1985, pp. 291–304. DOI: 10.1145/22145.22178.

[3] J. Thaler, *Proofs, arguments, and zero-knowledge*, Comprehensive survey of zero-knowledge proof systems and their applications, 2023. [Online]. Available: https://people.cs.georgetown.edu/jthaler/ProofsArgsAndZK.pdf.

[4] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, *Bulletproofs: Short proofs for confidential transactions and more*, Cryptology ePrint Archive, Paper 2017/1066, 2017. [Online]. Available: https://eprint.iacr.org/2017/1066.

[5] J. Bootle, A. Cerulli, P. Chaidos, J. Groth, and C. Petit, *Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting*, IACR ePrint Archive, Report 2021/638, 2021. [Online]. Available: https://eprint.iacr.org/2021/638.

[6] J. Camenisch, M. Kohlweiss, and C. Soriente, *Dynamic accumulators and application to efficient revocation of anonymous credentials*, IACR ePrint Archive, Report 2008/539, 2008. [Online]. Available: https://eprint.iacr.org/2008/539.

[7] F. Baldimtsi, J. Camenisch, M. Dubovitskaya, *et al.*, *Revocation in anonymous credentials: A comparative analysis of accumulators and merkle trees*, IACR ePrint Archive, Report 2022/492, 2022. [Online]. Available: https://eprint.iacr.org/2022/492.

[8] M. Campanelli, D. Fiore, and A. Querol, *Zero-knowledge friendly cryptographic accumulators for blockchain applications*, IACR ePrint Archive, Report 2023/208, 2023. [Online]. Available: https://eprint.iacr.org/2023/208.

[9] T. Lodder, T. Looker, M. Whitehead, and D. Zagidulin, "The bbs signature scheme," IRTF Crypto Forum Research Group, Internet Draft draft-irtf-cfrg-bbs-signatures, 2023, Work in Progress. [Online]. Available: https://identity.foundation/bbs-signature/draft-irtf-cfrg-bbs-signatures.html.

[10] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short proofs for confidential transactions and more," in *2018 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2018, pp. 315–334. DOI: 10.1109/SP.2018.00020.

[11] J. Groth, "On the size of pairing-based non-interactive arguments," in *EUROCRYPT*, 2016.

[12] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, "Scalable, transparent, and post-quantum secure computational integrity," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18, ACM, 2018, pp. 91–105. DOI: 10.1145/3243734.3243853.

[13] A. Gabizon, Z. J. Williamson, and O. Ciobotaru, "Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge," in *IACR Cryptology ePrint Archive*, 2019, p. 953. [Online]. Available: https://eprint.iacr.org/2019/953.