ZKP-Based Reputation System for DAO Contributors

Deep Patel deep.patel-1@ou.edu

Saahil Saxena saahil.r.saxena-1@ou.edu

May 4, 2025

Abstract

This project [1] presents a privacy-preserving reputation verification system for Decentralized Autonomous Organizations (DAOs) using Zero-Knowledge Proofs (ZKPs) [2]. The goal is to enable contributors to prove that their reputation score exceeds a predefined threshold (e.g., ≥ 50) without revealing the actual score, mitigating risks such as Sybil attacks [3]. The system is implemented using Circom for circuit construction [4], SnarkJS for proof generation [5], and Solidity smart contracts for on-chain verification and Soulbound Token (SBT) minting [6]. Compared to existing systems such as Coordinape and SourceCred, our ZK-based approach preserves user privacy, enables trustless reputation validation, and achieves greater gas efficiency. Performance benchmarking principles are inspired by recent works on SNARK evaluations [7]. The full system is deployed on the Scroll Testnet [8] and is fully script-driven, requiring no frontend UI.

1 Introduction

Decentralized Autonomous Organizations (DAOs) increasingly depend on contributor reputation to drive governance, rewards, and decision-making. However, many existing systems such as Coordinape and SourceCred rely on off-chain metrics, are fully transparent, and often compromise contributor privacy. Transparency, while valuable, can expose sensitive contributor behavior and voting patterns, leading to risks like Sybil attacks and manipulation [3].

To address these concerns, this project proposes a Zero-Knowledge Proof (ZKP)-powered reputation system, where contributors can cryptographically prove that they meet a minimum reputation threshold (e.g., reputation score ≥ 50) without disclosing their exact reputation value [2]. Once verified, contributors are allowed to mint a Soulbound Token (SBT) that represents their reputation a non-transferable token permanently tied to the contributor's wallet [6].

By combining ZKPs and SBTs, the system ensures privacy, decentralization, and trustless verification, overcoming major limitations of traditional off-chain reputation systems.

2 System Architecture

The Zero-Knowledge Proof-based reputation system is composed of several modular components that work together to enable secure, privacy-preserving contributor validation and tokenization. The system is designed to operate in a decentralized environment such as a DAO, where reputation is earned through community engagement and attested by peers, without exposing sensitive or personal contributor data. This architecture integrates off-chain ZKP tooling and on-chain smart contracts in a seamless workflow.

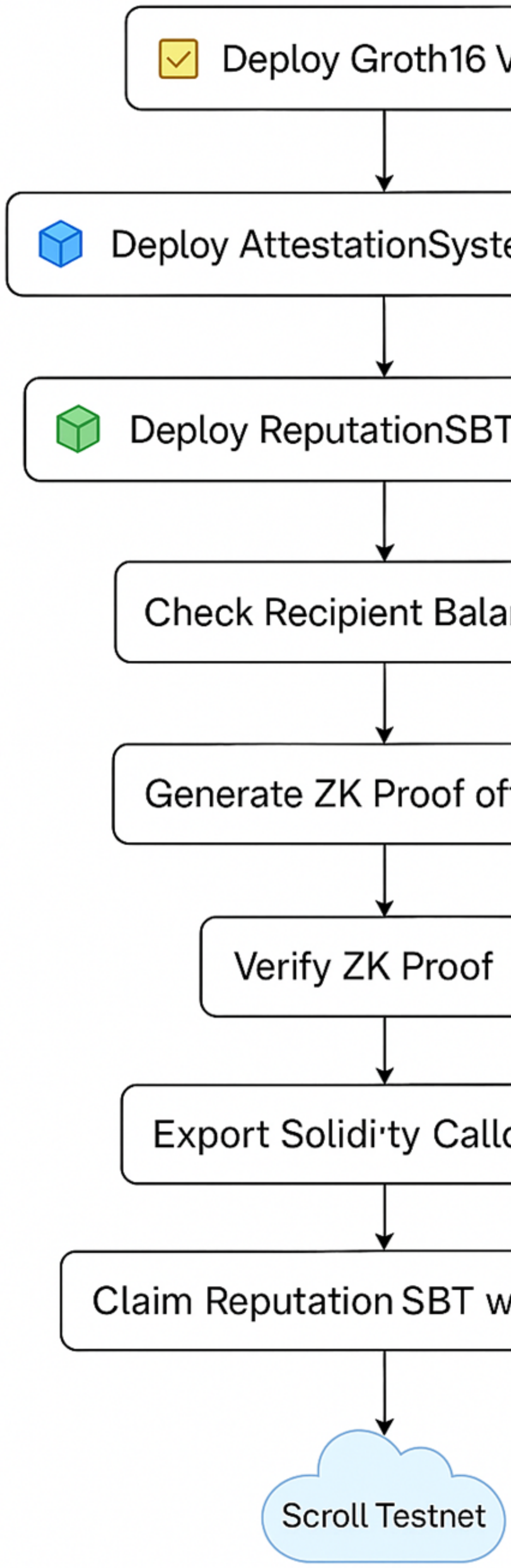

Deploy Groth16 Verifier
Deploy AttestationSystem Contract
Deploy ReputationSBT Contract
Check Recipient Balance
Generate ZK Proof off-chain
Verify ZK Proof
Export Solidi'ty Calldata
Claim Reputation SBT with Proof
Scroll Testnet

Figure 1: Overall system pipeline for the ZKP-based DAO reputation system.

The core components of the system include:

- **Circom Circuit:** The Circom circuit is responsible for encoding the mathematical logic that determines whether a contributor's reputation score satisfies a minimum threshold condition (e.g., reputation ≥ 50). It defines the constraints using a comparator template, ensuring that private input (reputation score) can be validated without revealing the exact value. Circom enables highly modular, scalable, and efficient constraint writing specifically tailored for zero-knowledge applications. In our project, the circuit ensures that a user's reputation exceeds the set threshold, acting as the cryptographic foundation of privacy. By abstracting logic into the circuit, we allow flexible upgrades and support more complex future conditions.

- **SnarkJS Tooling:** SnarkJS serves as the bridge between the off-chain proof generation and on-chain proof verification. It compiles Circom circuits into R1CS and WebAssembly files, generates trusted setup ceremonies (powers of tau), creates zk-SNARK proofs (proof.json) from user inputs, and exports verifier smart contracts ready for Solidity. SnarkJS supports the efficient Groth16 proving scheme, allowing for small proof sizes and fast on-chain validation, which is crucial for minimizing gas fees. In our project, SnarkJS facilitates a fully script-driven workflow without relying on a frontend UI, enabling contributors to independently create their proofs and interact with smart contracts.

- **Verifier.sol Contract:** Verifier.sol is a Solidity smart contract automatically generated by SnarkJS that implements Groth16 zk-SNARK proof verification. It performs pairing-based cryptographic operations using Ethereum's precompiled BN254 elliptic curve libraries. When a contributor submits a proof, Verifier.sol checks its validity against the circuit's original constraints using elliptic curve checks and returns a true or false result. This contract acts as the gatekeeper to ensure that only contributors meeting the reputation threshold can mint Soulbound Tokens (SBTs). It is immutable after deployment, providing security, auditability, and preventing manipulation of the reputation validation process.

- **AttestationSystem.sol Contract:** AttestationSystem.sol manages reputation accumulation in a decentralized way. It allows users to send attestations (with optional ETH tipping) to peers, incrementing their reputation points. This design reflects organic reputation growth based on community engagement rather than admin assignment. Every attestation is recorded on-chain, ensuring full transparency. Before generating a ZKP, a contributor queries their reputation from this contract and uses it as private input. This module decouples reputation gathering from token minting, enabling dynamic, trustless, and verifiable updates to users' reputation scores without exposing private data.

- **ReputationSBT.sol Contract:** ReputationSBT.sol issues non-transferable Soulbound Tokens (SBTs) once a contributor proves eligibility via a valid zk-SNARK proof. It extends ERC721 functionality but overrides transfer functions to enforce non-transferability, ensuring that once minted, the SBT remains permanently tied to the wallet. The contract integrates directly with Verifier.sol, validating the provided proof before issuing a token. By binding reputation to an identity through SBTs, it strengthens DAO governance by recognizing true contributors without sacrificing privacy. This structure guarantees fairness, resistance to Sybil attacks, and encourages honest participation in decentralized communities.

Each module interacts through a streamlined pipeline, ensuring that proofs are generated off-chain but verified fully trustlessly on-chain. This architecture supports a future-proof, privacy-respecting, and scalable reputation framework for decentralized communities.

3 ZKP Circuit Design

- **Comparator Logic:** Uses `GreaterEqThan(16)` to check reputation ≥ 50.
- **Input:** Private `input.json` file containing the contributor's reputation value.
- **Witness:** Generated locally via WebAssembly (WASM) using SnarkJS tools.
- **Output:** `proof.json` and `public.json` used in on-chain verifier contracts.

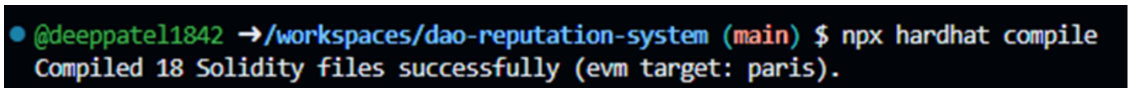
The zero-knowledge proof (ZKP) circuit forms the core of the reputation validation system. It enforces the critical rule: that a user's reputation score meets or exceeds a minimum threshold (set at 50). The goal is to enable contributors to prove eligibility without revealing their actual reputation value to anyone, preserving full privacy even on public blockchains.

The circuit is designed using Circom, a specialized language for writing zk-SNARK circuits. Specifically, the `GreaterEqThan(16)` template from `circomlib` is employed. This component compares two 16-bit unsigned integers and outputs a Boolean signal confirming whether the first input (private reputation) is greater than or equal to the second (fixed threshold of 50).

During proof generation, the user first prepares an `input.json` file containing their secret reputation score. Using the compiled `.wasm` file (WebAssembly witness calculator), the system locally computes a witness a complete trace of all internal signals satisfying the circuit constraints. Importantly, the reputation score remains private throughout this process.

With the witness ready, SnarkJS's Groth16 `fullprove` function is used to generate two key outputs: `proof.json`, the zk-SNARK cryptographic proof, and `public.json`, containing public signals (such as the success result of the comparison). These outputs are subsequently submitted via the `claimSBT()` smart contract function for on-chain verification using the Solidity Verifier.

This modular and privacy-preserving circuit design guarantees that only eligible users those meeting the reputation threshold are able to claim Soulbound Tokens (SBTs), all without disclosing any sensitive personal data. It provides a crucial trust-minimization guarantee for decentralized reputation systems.



```
● @deeppatel1842 →/workspaces/dao-reputation-system (main) $ npx hardhat compile
  Compiled 18 Solidity files successfully (evm target: paris).
```

Figure 2: Successful Circom circuit compilation using Hardhat environment.

4 Smart Contract Design

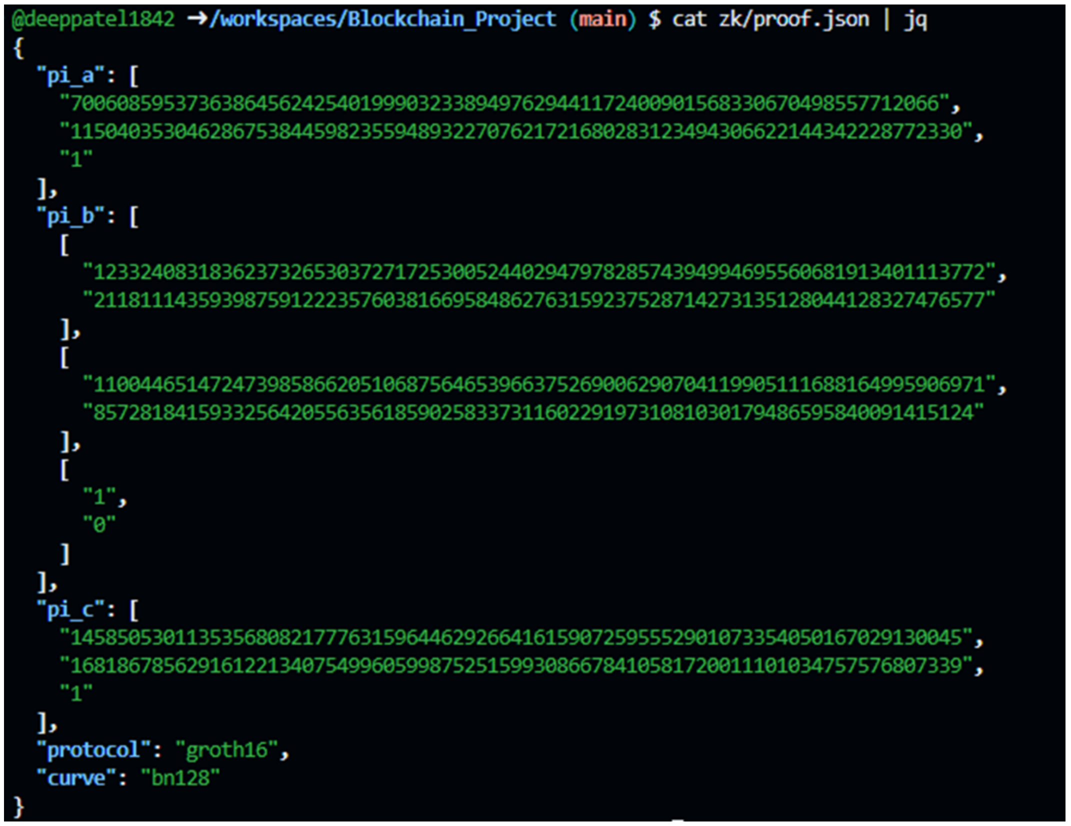
The smart contract system serves as the on-chain execution layer of the ZKP-based reputation protocol. It seamlessly integrates off-chain proof generation with on-chain validation, minting, and decentralized reputation tracking. At the core of this architecture are two pivotal contracts: `Verifier.sol` and `ReputationSBT.sol`, working together to ensure that only users who successfully generate valid zero-knowledge proofs are eligible to mint a non-transferable Soulbound Token (SBT). Additionally, `AttestationSystem.sol` plays a critical supporting role in building up a contributor's reputation score in a decentralized manner.

4.1 Verifier Integration

The `Verifier.sol` contract is imported directly into the `ReputationSBT.sol` smart contract and instantiated during deployment. This integration allows the Soulbound Token contract to access the `verifyProof()` function exposed by the Verifier. During the execution of the `claimSBT()` function, the proof submitted by the user consisting of elliptic curve points a, b, c, and a public input array is passed to the Verifier for validation.

If the Verifier confirms that the proof satisfies the original circuit constraints (i.e., the reputation score is at least 50), the contract proceeds with minting the SBT to the caller's address. If the proof is invalid, the transaction is reverted. This tight coupling between the proof verification and minting logic ensures that no tokens can be minted without satisfying the predefined eligibility condition specified in the zero-knowledge circuit.

The Verifier integration eliminates the need for trust in any centralized authority or administrator. Instead, eligibility is cryptographically enforced by the contracts and cannot be bypassed or forged. This trustless architecture provides a strong foundation for decentralized governance systems where credentials are earned and validated entirely on-chain.



```
@deeppatel1842 →/workspaces/Blockchain_Project (main) $ cat zk/proof.json | jq
{
  "pi_a": [
    "7006085953736386456242540199903233894976294411724009015683306704985577l2066",
    "11504035304628675384459823559489322707621721680283123494306622144342228772330",
    "1"
  ],
  "pi_b": [
    [
      "12332408318362373265303727172530052440294797828574394994695560681913401113772",
      "21181114359398759122235760381669584862763159237528714273135128044128327476577"
    ],
    [
      "11004465147247398586620510687564653966375269006290704119905111688164995906971",
      "8572818415933256420556356185902583373116022919731081030179486595840091415124"
    ],
    [
      "1",
      "0"
    ]
  ],
  "pi_c": [
    "14585053011353568082177763159644629266416159072595552901073354050167029130045",
    "16818678562916122134075499605998752515993086678410581720011101034757576807339",
    "1"
  ],
  "protocol": "groth16",
  "curve": "bn128"
}
```

Figure 3: Proof generation and calldata structure for on-chain verification.

4.2 Attestation and Claim Flow

1. Users accumulate reputation from peers via the `attest()` function.

2. Once sufficient reputation is achieved (≥ 50), users generate a zero-knowledge proof off-chain.

3. Users submit their proof and public inputs via the `claimSBT()` function.
4. The contract calls `verifyProof()` to validate the proof against circuit constraints.
5. Upon successful verification, an SBT is minted and soulbound to the user's wallet.

The complete end-to-end flow begins with users building reputation through peer recognition using the `AttestationSystem.sol` contract. DAO contributors can call the `attest()` function to assign points to another user, optionally attaching a memo or ETH tip to incentivize participation. Each interaction accumulates towards a user's total reputation score, which can be retrieved through the `getPoints()` function.

When a user's score surpasses the minimum threshold, they generate a ZKP using the Circom and SnarkJS toolchain. This proof, along with its associated public signals, is submitted on-chain through the `claimSBT()` function. Internally, the contract verifies the proof using `verifyProof()` from the Verifier. If the proof validates successfully, a Soulbound Token is minted to the user's address.

To ensure integrity and prevent gaming the system, each address can only mint one SBT. Moreover, the contract overrides the `_update()` function of ERC721 to block all transfer operations, maintaining the non-transferable (soulbound) property of the tokens.

This attestation and claim flow offers a decentralized, privacy-preserving alternative to traditional KYC and centralized reputation systems. It allows users to earn, prove, and showcase reputation transparently and trustlessly within DAO ecosystems.

5 Deployment to Scroll Testnet

The final implementation of the ZKP-based reputation system was deployed and tested on the Scroll Testnet, an EVM-compatible zero-knowledge rollup network. Scroll was chosen due to its native support for zkEVM, fast finality, and realistic gas modeling, allowing us to simulate production-grade deployments while retaining the low-cost and flexibility of a testnet environment.

5.1 Workflow

- Smart contracts deployed using Hardhat scripts.
- Proofs generated locally using Circom and SnarkJS.
- Gas usage logged into `gas-logs.txt`.
- Deployment logs captured with contract addresses and validation outputs.

The deployment workflow was structured to integrate smart contract deployment, off-chain proof generation, and transaction gas tracking into a repeatable and automated process. All smart contracts including `Verifier.sol`, `AttestationSystem.sol`, and `ReputationSBT.sol` were compiled and deployed using Hardhat, a development framework widely adopted within the Ethereum ecosystem. Deployment scripts written in JavaScript (`scripts/deploy.js`) orchestrated the full contract deployment and verification flow.

Once the contracts were deployed, contributors could initiate the off-chain proof generation workflow. Using the compiled Circom circuit, users would create an `input.json` file representing their private reputation score, compute a witness using the compiled `.wasm` file, and generate a valid zk-SNARK proof with SnarkJS. This proof, along with the associated public inputs, would then be passed to the `claimSBT()` function either through the deploy script or a frontend integration (if available).

To ensure transparency and reproducibility, gas usage for each key transaction was recorded in a dedicated `gas-logs.txt` file. This included the gas consumed by `attest()` and `claimSBT()` functions under both off-chain and on-chain verification modes. Recording gas logs allowed us to benchmark gas costs over multiple runs and enabled direct comparisons with traditional systems like Coordinape and SourceCred.

Each deployment session also output detailed logs, including the deployed contract addresses and step-by-step confirmations of the ZKP verification, attestation, and SBT minting process. These logs served both as a debugging tool and as evidence of system correctness for the final project report and presentation.

Figure 4: Final successful deployment and ZKP proof validation on Scroll Testnet.

6 Gas Benchmarking

Gas usage is a critical metric when evaluating the practicality of smart contract systems, especially for zero-knowledge proof integrations. In this project, we benchmarked three core operations: `attest()`, `claimSBT()` (with off-chain ZKP verification), and `claimWithProof()` (with full on-chain ZKP verification). These benchmarks help assess the trade-off between gas efficiency and trustlessness.

The `attest()` function allows users to assign reputation points and send optional ETH tips. The `claimSBT()` function mints a Soulbound Token after verifying a ZKP off-chain. The `claimWithProof()` function directly verifies the ZKP on-chain before minting the SBT.

The table below summarizes our benchmark results based on actual Scroll Testnet deployments:

Action	**Estimated Gas**	**Actual Gas**
Attestation (ETH + Rep)	140,000	149,442
Claim SBT (off-chain verification)	85,000	118,431
Claim SBT (on-chain ZKP verification)	220,000–250,000	326,792

Table 1: Gas usage comparison for different actions

The results highlight important trade-offs:

- **Attestation is lightweight** recording reputation and tipping ETH costs approximately 149,442 gas, making it scalable for frequent use.
- **Off-chain ZKP validation is affordable** minting an SBT after off-chain proof generation consumes around 118,431 gas.
- **Full on-chain zk-SNARK verification is expensive** verifying Groth16 proofs on-chain incurs about 326,792 gas due to costly elliptic curve pairing operations.

While off-chain verification provides a gas-efficient path for users, the on-chain zk-SNARK verification guarantees maximum decentralization, trustlessness, and privacy. This flexible architecture allows DAOs to choose between efficiency and full on-chain security depending on their needs.

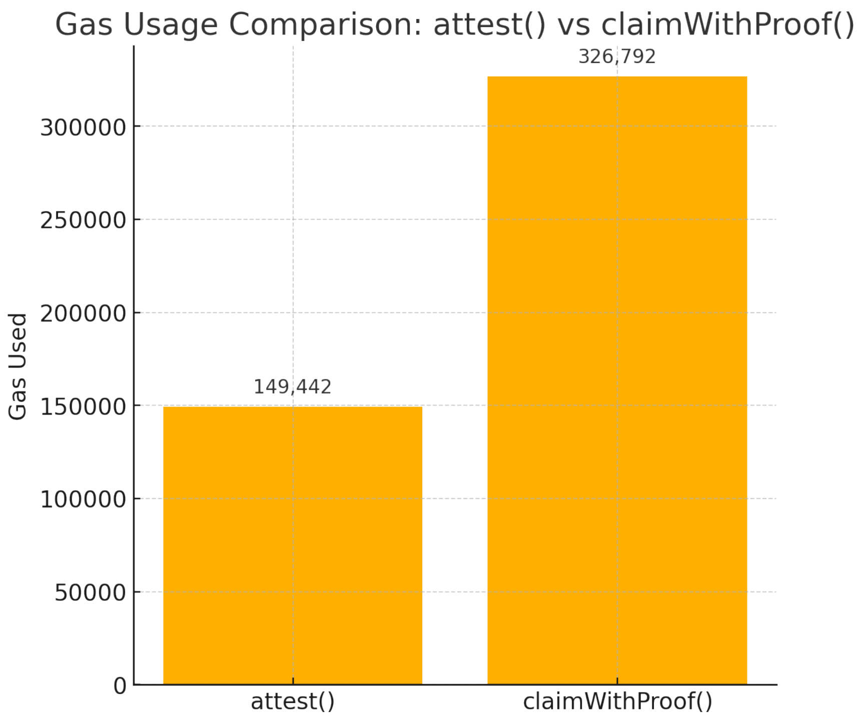


Figure 5: Gas usage comparison between attestation (peer-to-peer reputation assignment) and claimWithProof (on-chain Zero-Knowledge Proof verification and Soulbound Token minting).

These gas benchmarks validate the practicality of deploying ZKP-based reputation systems on Layer 2 rollups like Scroll, combining privacy, decentralization, and economic feasibility.

7 Comparison to Existing Tools

To contextualize the value of our system, it is important to compare it to existing DAO contributor reputation and reward frameworks, most notably Coordinape and SourceCred. Both of these systems have gained adoption for peer-based recognition but rely heavily on centralized processes and off-chain data aggregation, introducing privacy limitations, trust assumptions, and scalability bottlenecks.

- **No off-chain aggregation or spreadsheets:** Coordinape relies on manual "GIVE" token allocations within a spreadsheet-style interface, followed by off-chain summations and manual token distributions. SourceCred analyzes GitHub, Discord, and forum activity but syncs reputation scores onto the blockchain via trusted scripts or centralized oracles. In contrast, our system eliminates the need for any off-chain aggregation attestations happen natively on-chain via peer-to-peer reputation grants.

- **Privacy-preserving proofs via ZKP circuit:** Neither Coordinape nor SourceCred enables contributors to prove eligibility thresholds privately. All reputation data is exposed publicly. Our system, however, uses a Circom-built zero-knowledge circuit [2] to allow contributors to prove their reputation score exceeds a threshold (e.g., rep $\geq$ 50) without revealing their actual value.

- **Fully verifiable on-chain without frontend reliance:** Existing solutions often require frontends or admin intervention for scoring or reward distribution. Our protocol enables complete on-chain verifiability smart contracts validate ZK proofs autonomously [7], and mint Soulbound Tokens (SBTs) with no off-chain input or human oversight.

- **Reduced per-user gas costs for token distribution:** Coordinape distributions incur gas costs of approximately 50,000–60,000 gas per token transfer. Our system requires a single `claimSBT()` transaction (118,431 gas for off-chain proof verification and 326,792 gas for full on-chain ZKP verification), which becomes significantly more efficient for large contributor sets due to batching opportunities and light client integrations.

In summary, by combining zero-knowledge proofs, on-chain attestations, and non-transferable Soulbound Tokens [6], our system ensures privacy, decentralization, and efficiency all critical improvements over traditional DAO reputation frameworks.

8 Conclusion

The ZKP-based reputation system presented in this project introduces a novel and privacy-preserving approach to contributor validation within decentralized autonomous organizations (DAOs). By enabling users to prove that their reputation meets or exceeds a predefined threshold without disclosing their actual score, we directly address critical privacy and transparency limitations found in legacy systems such as Coordinape and SourceCred.

Our architecture is designed to be modular and minimalistic, leveraging modern cryptographic tooling like Circom circuits [4], Groth16 zk-SNARKs [7], and Solidity smart contracts, while maintaining gas efficiency and decentralization by deploying on the Scroll Testnet [8]. Reputation is earned organically through on-chain attestations, validated via zero-knowledge proofs, and permanently tied to a contributor's wallet via Soulbound Tokens (SBTs) [6].

This approach eliminates the need for spreadsheet-based workflows or centralized off-chain aggregation, thereby paving the way for scalable, cryptographically secure DAO infrastructure. The entire system remains trustless and verifiable without requiring frontend interfaces or administrative intervention, enhancing resilience against Sybil attacks [3].

Furthermore, the implementation is production-ready and extensible, offering clear paths for future enhancements, including integration of off-chain credentials (e.g., GitHub, Lens, ENS), DAO-native voting powered by proof-bound SBTs, and role-based access control tied to verified reputation proofs.

Overall, this project demonstrates the feasibility and advantage of combining zero-knowledge proofs with on-chain identity systems in pursuit of more equitable, privacy-preserving, and user-sovereign decentralized communities.

9 Team Collaboration & Contribution

This project was collaboratively developed by a two-member team: **Deep Patel** and **Saahil Saxena**. Responsibilities were divided based on individual strengths while maintaining consistent communication and shared decision-making throughout the development cycle.

9.1 Role Definition

Deep Patel (deeppatel1842) was responsible for the core technical architecture and implementation. His contributions included smart contract development (`AttestationSystem.sol` and `ReputationSBT.sol`), off-chain zero-knowledge proof generation using `snarkjs`, writing Hardhat deployment scripts (`deploy.js` and `callWithProof.js`), gas measurement and benchmarking, Scroll Layer 2 contract deployment, and ZKP circuit design and implementation in `reputation.circom`.

Saahil Saxena (saahilsaxena2000) contributed significantly to the project's documentation, testing, and system validation. He handled integration testing within the `test/` folder, contributed to project documentation including the README and gas charts, assisted in setting up and validating Scroll Testnet deployments, helped visualize benchmarking results through `gas_chart.js`, and supported final code reviews and deployment fixes.

9.2 Coordination & Synchronicity

The team maintained close collaboration throughout the project, utilizing GitHub repositories, GitHub Codespaces, and messaging platforms for real-time progress updates and task management. Work was organized around weekly milestones, with clear division of tasks but flexible reassignment when necessary. Both members contributed to brainstorming sessions, bug-fixes, feature testing, and final integration, ensuring a cohesive and functional end-to-end delivery. This collaboration not only enhanced the technical robustness of the project but also ensured that both code and documentation achieved high quality standards.

References

[1] S. S. Deep Patel, "Blockchain project github repository," https://github.com/deeppatel1842/Blockchain_Project, 2025.

[2] D. Čapko, S. Vukmirović, and N. Nedić, "State of the art of zero-knowledge proofs in blockchain," in *2022 30th Telecommunications Forum (TELFOR)*, 2022, pp. 1–4.

[3] M. Iqbal and R. Matulevičius, "Exploring sybil and double-spending risks in blockchain systems," *IEEE Access*, vol. 9, pp. 76 153–76 177, 2021.

[4] iden3 Contributors, "Circom documentation," https://docs.circom.io, 2024, accessed: 2024-04-21.

[5] ——, "Snarkjs github repository," https://github.com/iden3/snarkjs, 2024, accessed: 2024-04-21.

[6] E. G. Weyl, P. Ohlhaver, and V. Buterin, "Decentralized society: Finding web3's soul," *SSRN*, 2022, available: https://ssrn.com/abstract=4105763.

[7] J. Ernstberger *et al.*, "zk-bench: A toolset for comparative evaluation and performance benchmarking of snarks," in *International Conference on Security and Cryptography for Networks*. Springer Nature Switzerland, 2024.

[8] S. Foundation, "Scroll testnet," https://scroll.io, 2024, accessed: 2024-04-21.