**PrivateVote: ZK-Nullifier Anonymous Governance**

**Anonymity and Integrity Guaranteed by Zero-Knowledge Proofs on the Midnight Network**

**The Problem and Our ZK Solution**

Traditional digital voting systems force users to trust a central authority to enforce fairness (voter eligibility, one-vote-per-person). This inherently compromises privacy.

**PrivateVote** solves this by using the **Compact** language and **Zero-Knowledge Proofs (ZKPs)** on the Midnight Network. We separate privacy from integrity, allowing the system to guarantee election fairness without recording any identifying metadata about the voter. Our core innovation is the **ZK Nullifier Logic** within the contract's IsVoteValid circuit.

**Technical Deep Dive: The ZK Nullifier Pattern**

The Nullifier pattern is the critical mechanism that mathematically guarantees a user can only vote once, while still remaining fully anonymous.

* **The Secret Identity (Witness):** The voter provides a voter\_id\_secret (a u64 in the VoteDetails witness). This private token is used to generate the proof but is **never stored on the public ledger.**
* **The Anonymous Receipt (Nullifier):** A nullifier\_hash (Field) is generated from the secret. This hash is publicly written to the ledger and acts as an anonymous "already spent" flag, fulfilling the **Nullifier** function.
* **Fraud Prevention (Circuit Check):** The core IsVoteValid circuit uses an assertion (assert(!prev\_state.nullifiers.contains(...))) to check the public ledger's set of spent nullifiers. If the hash exists, the transaction is rejected, preventing the double-spend.
* **Admin Control (Architectural Security):** The transition InitializeElection requires an IsAdmin(witness) proof, ensuring only a privileged party, verified via ZK, can deploy and set up the contract.

The final public **Ledger** (state) holds only the final vote tally and the set of anonymous spent Nullifiers, guaranteeing transparency and privacy simultaneously.

**Project Files and Status**

This project demonstrates a fully specified architecture for a winning ZK DApp.

* **PrivateVote.compact**: The complete smart contract containing all ZK logic (IsAdmin, IsVoteValid), the Nullifier set (nullifiers), and state transitions. **Status: Code Complete**
* **index.html**: A single-page application demonstrating the full voting workflow (Initialize, Input Secret, Vote, See Tally) using mock logic for a clear demo. **Status: Demo Ready**

**How to Run the Demo**

1. Download the **index.html** file.
2. Open the file in any web browser to see the live, simulated ZK voting experience.

**Future Roadmap (Next Steps for the Fellowship)**

1. **Compiler Integration:** Complete integration with the working compactc compiler once toolchain support is stabilized.
2. **ZK Commitment Logic:** Implement a more robust Merkle Tree or cryptographic commitment scheme for voter registration lists.
3. **Encrypted Tally:** Explore homomorphic encryption to allow complex vote tallies (e.g., ranked choice) directly on encrypted data.

Built by: Nikita Sharma

Hacked on: The Midnight Network