vocdoni

Complaint-Free Distributed Key Generation (DKG)

Link to draft paper

Blockchain-assisted protocol with proactive zk-SNARK validation

Thesis. Replace interactive on-chain complaint/dispute phases in smart-contract DKGs with **proactive zk-SNARK validation**: each participant submits one proof that their entire contribution (Feldman VSS consistency + correctly encrypted shares) is well-formed. The contract **rejects invalid contributions at submission time**, so there is nothing to dispute.

Motivation and context

Blockchain-assisted DKGs often handle malformed shares via on-chain complaints and timeouts. This adds extra rounds and creates practical attack surfaces (griefing, deadline games, liveness failures). The goal of our scheme is to make DKG contributions **self-validating** at submission time.

What the protocol delivers

- **Complaint-free DKG:** invalid contributions are rejected by SNARK verification; no dispute phase.
- **Public verifiability:** observers can audit correct execution from on-chain data + proofs.
- **Block-delimited phases:** no interactive deadlines; non-submitters are excluded by policy.
- **Complete lifecycle:** threshold decryption + optional secret-key disclosure with proofs/checks.

DKG Protocol at a glance

1. **Initiation:** organizer posts (n, t) + optional policy parameters.
2. **Show of hands:** eligible nodes signal readiness (optionally with stake).
3. **Node selection:** select n nodes using chain randomness; abort if policy not met.
4. **Main DKG process:** every node posts polynomial commitments and encrypted shares + zk proof of correctness.
5. **Finalization:** the organizer computes the public key PK and commitments to private shares, and posts these to the blockchain together with a zk proof of correctness.

Cryptographic building blocks

- **Shamir + Feldman VSS:** degree-(t−1) polynomials with public coefficient commitments.
- **Hashed ElGamal:** publish shares as ciphertexts (ECIES/DHIES-style masking).
- **Chaum-Pedersen DLEQ:** DLOG equality proof, used for partial decryptions.
- **zk-SNARKs (Groth16 , FFLONK, ...):** on-chain verification and (optionally) offloading of group operations.

ZK circuits (what is proven)

Circuits are specified for **DKG phase 4** (shares, commitments, Feldman checks, encryption), **DKG phase 5** (aggregate commitments, derive PK and Di), **threshold decryption** (submission of partial decryption by the nodes; submission of the decrypted ciphertext), and **optional key disclosure** (submission of secret key shares by the nodes; submission of the secret key).

Practicality: public inputs and on-chain cost

A central engineering constraint is public input size. Assuming $t \approx n/2$, a naïve approach would result in ~6n scalar inputs per participant in phase 4 and ~$n^2/2$ for the phase-5 aggregation proof. FFLONK is input-limited (~38), while Groth16 supports ~700 inputs but is expensive per input. This motivates input-commitment techniques.

Input reduction strategies in the draft

- **Hash-to-single-signal:** commit to long vectors via Poseidon hashes, with the circuit proving preimage knowledge; relatively expensive but necessary when no suitable randomness is available.
- **BRLC:** commit via a random linear combination; cheap on-chain but requires careful, unpredictable challenge derivation.

Limitations and assumptions

- **Partial synchrony:** phases are delimited by block numbers, the protocol is therefore not fully asynchronous.
- **Static corruption:** participant set fixed pre-run (no adaptive security).
- **Scale:** large committees likely need L2 economics + input reduction to be practical.

Where researchers can contribute next

- **Formal security:** model and prove robustness/correctness in the bulletin-board + smart-contract setting under ZK enforcement.
- **Concrete ZK engineering:** implement circuits, measure constraints/proving times, and explore better vector commitments / batching.
- **Deployment economics:** benchmark calldata vs storage and Groth16 vs FFLONK vs other provers tradeoffs on specific EVM L2s; evaluate extensions (refresh, threshold signatures).