Zero-Knowledge Verification of Distributed Key Generation

1. Introduction

Purpose and Scope

This specification provides a framework for zero-knowledge (ZK) verification circuits within a Distributed Key Generation (DKG) protocol. It emphasizes cryptographic formulations to ensure the correctness and security of each verification step.

Overview of Distributed Key Generation (DKG)

DKG enables a set of n participants to collaboratively generate a shared public key without any single party knowing the corresponding private key. This is achieved through:

- **Shamir's Secret Sharing [1]**: Distributes a secret among participants such that any subset of $t+1$ can reconstruct it, but no subset of t or fewer can.
- **Verifiable Secret Sharing (VSS)**: Enhances Shamir's scheme by allowing participants to verify the correctness of their received shares.
- **Zero-Knowledge Proofs (ZKPs) [2]**: Allow participants to prove the validity of their actions without revealing any secret information, ensuring that deviations can be detected without compromising the underlying secrets.

The protocol outputs:

- Each participant will reconstruct a partial secret S_i such that the shared secret SS can be derived by evaluating the Lagrange interpolation[5] of these partial secrets at $x = 0$, where the partial secrets correspond to points $(1, S_1), (2, S_2), \ldots, (n, S_n)$ on a polynomial $F(x)$ of degree at most t.
- SS is the shared secret between the participants.

The protocol ensures that:

- The secret is generated according to Shamir's Secret Sharing scheme.

- Any deviation — whether intentional or accidental — can be detected, and the malicious participant can be identified. Zero-knowledge proofs play a crucial role in enabling this detection without revealing any sensitive information exchanged during the process.

2.0 High-Level Overview of Provable Distributed Key Generation (PDKG)

1. **Initialization (Public Setup Phase):**
   One participant initializes the session by publishing the setup on a shared, publicly accessible platform (e.g., a blockchain smart contract, shared database, or bulletin board).
   The setup includes:
   - `n` : Total number of participants
   - `t` : Threshold number of participants required to reconstruct the secret
   - `generationId` : A unique identifier for this specific key generation session
2. **Polynomial Generation (Secret Sharing Phase):**
   Each participant independently generates a random polynomial of degree t, as per Shamir's Secret Sharing.
3. **Commitment Broadcast:**
   Each participant computes cryptographic commitments to their polynomial coefficients and the setup.
   These commitments are published to the public board to enable verifiable consistency checks.
4. **Share Distribution:**
   Participants privately send encrypted shares (i.e., evaluations of their polynomial), along with any additional data required to prove correctness — such as the verification vector — to each of the other participants. The verification vector helps recipients verify the validity of the share without needing to know the sender's secret polynomial. A dispute mechanism is included to handle missing or invalid shares via public challenges (shares can be posted publicly in encrypted).
5. **Verification and Acknowledgment:**
   Upon receiving shares from others, each participant:
   - Verifies the correctness of each received share against the sender's public commitment, the session setup, and the corresponding verification vector.
   - If a share is invalid or inconsistent:
     - Constructs and publishes a proof of misbehavior (e.g., using cryptographic evidence) on the public board.
   - If misbehavior cannot be proven or if a participant refuses to participate:
     - Posts a challenge on the public board to trigger a dispute-resolution mechanism that ensures the protocol's progress.

6. **Partial Key Generation:**
   Each participant computes their partial secret key by summing the valid shares they received from all other participants.
7. **Finalization and Proof Construction:**
   Once enough valid shares and acknowledgments have been collected, any participant can construct a publicly verifiable proof that:
   - All distributed shares are consistent with the published commitments.
   - The collective secret can be reconstructed from the valid shares.

   If a participant misbehaves — for example, by submitting an invalid proof (e.g., a signature that doesn't correspond to their expected public key):
   - A proof of misbehavior can be constructed and published.

   If a participant refuses to cooperate (e.g., by failing to submit their signature):
   - A challenge can be posted on the public board to trigger the dispute resolution mechanism.

   This ensures the key was generated honestly and can be used securely in threshold cryptographic schemes.

2.1 Initialization

Participants agree on:

- Threshold t, total number of participants n, message M.
- A unique $\mathrm{generation\_id}$.
- Authentication key $\mathrm{AuthKey}_i$ and corresponding public key AuthPK_i for each participant P_i. We assume these are ECDSA or similar public/private key pairs.
- A homomorphic function $\mathrm{PK}(x)$, satisfying $\mathrm{PK}(x+y) = \mathrm{PK}(x) + \mathrm{PK}(y)$. For example, $\mathrm{PK}(x) = g \cdot x$ in BLS12-381. Being homomorphic is crucial is crucial for verifying the correctness of combined shares without revealing the individual shares.

2.2 Commitment Phase

Each participant P_i creates a random polynomial:

$$f_i(x) = a_{i,0} + a_{i,1}x + \cdots + a_{i,t}x^t$$

Where:

- $a_{i,j} \in \mathbb{F}_q$: Random coefficients.
- $f_i(x)$: Secret polynomial of P_i.

The secret partial share of participant P_j is $s_{i,j} = f_i(j)$.

We define a verification vector V_i as:

$$(\mathrm{PK}(a_{i,0}), \ldots, \mathrm{PK}(a_{i,t}))$$

Commitment:

$$C_i = \mathrm{HASH}(n, t, \mathrm{generation\_id}, \mathrm{V}_i)$$

Published to a public board and signed with $\mathrm{AuthKey}_i$.

2.3 Share Distribution

Each P_i computes:

$$s_{i,j} = f_i(j)$$

Sends to P_j along with V_i, all signed with $\mathrm{AuthKey}_i$. The verification vector V_i allows P_j to verify that the received share $s_{i,j}$ is consistent with the polynomial committed to by P_i (P_j can take the C_i from the public board).

2.4 Share Verification

Upon receiving $s_{i,j}$ and V_i from P_i, participant P_j performs the following verifications:

- **Hash Consistency**:

  $$C_i \stackrel{?}{=} \mathrm{HASH}(n, t, \mathrm{generation\_id}, \mathrm{V}_i)$$

  If the hash does not match and the discrepancy cannot be cryptographically proven (e.g., a ZKP showing a collision in the hash function, which is highly unlikely), initiate the fallback challenge mechanism.
- **Signature Authentication**:
  Verify the authenticity of C_i using the public authentication key AuthPK_i. If the signature is invalid and no cryptographic proof of misbehavior is provided (e.g., a ZKP showing a flaw in the signature scheme), fall back to the challenge protocol (use Circuit 3 to prove misbehavior).
- **Polynomial Evaluation**:
  Define the verification polynomial:

  $$p_i(x) = \sum_{k=0}^{t} \mathrm{PK}(a_{i,k}) \cdot x^k, \quad \text{where } \mathrm{PK}(a_{i,k}) \in \mathrm{V}_i$$

Then verify:

$$\mathrm{PK}(s_{i,j}) \overset{?}{=} p_i(j)$$

A mismatch here provides verifiable evidence of an invalid share because the homomorphic property of PK allows checking the evaluation without knowing the secret coefficients $a_{i,k}$ (Circuit 1).

If a participant submits a share that cannot be verified or if there is insufficient evidence to validate its correctness, and malicious intent is suspected, then participant P_j should initiate a **challenge** against P_i on the public board.

The challenge must include an expiration timestamp. The response to the challenge must be **encrypted using a deterministic ECDH scheme[5]** based on the sender's published verification vector.

Specifically:

- The verification vector $\mathbf{V}_i$ (published by P_i) is **sorted deterministically**, for example, lexicographically by public key bytes.
- The **last element** of this sorted vector is used for key agreement:
  - P_i uses the **private key** from their own last element.
  - P_j uses the **public key** from the corresponding last element of P_i's vector.
- They derive a shared secret using ECDH:

$$K_{i,j} = \mathrm{ECDH}(\mathrm{PrivKey}_i^{\mathrm{last}}, \mathrm{PubKey}_j^{\mathrm{last}})$$

- This shared secret is passed through a key derivation function (e.g., HKDF) to obtain a symmetric key K_{enc}.

This encryption scheme ensures that only the intended recipient (with access to the corresponding private key) can decrypt the response, while preserving deterministic behavior and cryptographic soundness necessary for verifiability and zero-knowledge applications.

If the challenge expires without a valid response, or if the response is invalid, the protocol can demonstrate misbehavior by P_i, which may result in penalties (e.g., slashing). If the protocol fails, all participants can verify the failure based on the data recorded on the public board.

The use of encryption and zero-knowledge proofs guarantees that sensitive information is never exposed on the public board at any stage of the protocol.

2.5 Partial Key Generation

Each P_i computes their partial secret key S_i by summing the valid shares received from all other participants:

$$S_i = \sum_{j=1}^{n} s_{j,i}$$

where $s_{j,i}$ is the share sent from P_j to P_i.

2.6 Finalization

During the final round, each participant P_i broadcasts a signature over a message SM_i (predefined constant agreed upon by the participants) to all other participants.

Given that each participant possesses the verification vectors of all others, they can independently verify that each SM_i is signed using the correct partial secret key associated with P_i. Specifically, they can reconstruct the partial public key PK_i corresponding to S_i by evaluating the sum of the verification vectors at $x = i$:

$$PK_i = \sum_{k=1}^{n} p_k(i) = \sum_{k=1}^{n} \sum_{j=0}^{t} \mathrm{PK}(a_{k,j}) \cdot i^j$$

and then verify the signature using PK_i.

If an invalid signature is detected, it constitutes cryptographic proof of misbehavior (Circuit 2). However, if a participant withholds participation or provides malformed data that cannot be conclusively proven malicious, a fallback challenge mechanism is triggered. At this stage, since the partial public keys are derived from public commitments, the challenge and response might occur without encryption, focusing on proving the validity of the derived public key or the signature.

Once all valid signatures are collected, any participant (or a subset thereof) can construct a final proof (Circuit 4) that the secret sharing protocol has completed successfully, and that the reconstructed shared secret is SS. This proof is published to the public board, finalizing the protocol execution.

3. Verification Circuit Analysis

Circuit 1: Incorrect Share Detection

- **Objective**: Determine whether the share $s_{i,j}$ sent by P_i to P_j is invalid, either due to incorrect polynomial evaluation or inconsistent public commitments.
- **Verification Process**:
  i. **Commitment Hash Check**:
     Confirm that the provided commitment hash C_i matches the expected hash derived from $(n, t, \text{generation\_id}, V_i)$. If the hash does not match and no cryptographic proof of a hash collision is provided, initiate the fallback challenge mechanism.
  ii. **Signature Authentication**:
     Verify the authenticity of C_i using the public authentication key AuthPK_i. If the signature is invalid and no cryptographic proof of a flaw in the signature scheme is available, fall back to the challenge protocol.
  iii. **Share Evaluation**:
     Evaluate whether:

$$\text{PK}(s_{i,j}) \stackrel{?}{=} \sum_{k=0}^{t} \text{PK}(a_{i,k}) \cdot j^k$$

     A mismatch here provides verifiable cryptographic evidence of an invalid share, as the homomorphic property of PK ensures the equality should hold for a valid share.
- **Result**:
  The circuit succeeds if it can produce a verifiable contradiction (a mismatch in hash, signature, or polynomial evaluation). If no contradiction is detected and no cryptographic proof of misbehavior is available, the circuit defers to the challenge mechanism.
- **Public outputs**
  - All commitement set C = { C_1, ..., C_t}.
  - Public key of the sender AuthKey_i.

Circuit 2: Incorrect Partial Public Key Detection

1. **Signature Verification**:
   Verify the signature over the input data (likely related to their partial key contribution) using the authentication key AuthPK_i. If the signature is invalid but not provably malicious, fallback to the challenge mechanism.

2. **Construct the Aggregated Polynomial**:

$$P(x) = \sum_{k=1}^{n} f_k(x) = \sum_{k=1}^{n} \left(\sum_{j=0}^{t} a_{k,j} x^j \right) = \sum_{j=0}^{t} \left(\sum_{k=1}^{n} a_{k,j} \right) x^j$$

   Applying the homomorphic function PK to this polynomial gives:

$$\mathrm{PK}(P(x)) = \sum_{j=0}^{t} \left(\sum_{k=1}^{n} \mathrm{PK}(a_{k,j}) \right) x^j, \quad \text{where } \mathrm{PK}(a_{k,j}) \in V_k$$

   Note:
   The partial public key of participant i, denoted as PK_i, is the homomorphic encryption of their partial secret key S_i:

$$PK_i = \mathrm{PK}(S_i) = \mathrm{PK}\left(\sum_{j=1}^{n} s_{j,i} \right) = \sum_{j=1}^{n} \mathrm{PK}(s_{j,i})$$

3. **Proof of Correct Reconstruction**:
   Prove that the partial public key PK_i is consistent with the aggregated polynomial evaluated at $x = i$:

$$PK_i \stackrel{?}{=} \mathrm{PK}(P(i)) = \sum_{j=0}^{t} \left(\sum_{k=1}^{n} \mathrm{PK}(a_{k,j}) \right) i^j$$

4. **Signature Validation**:
   Verify that the message signature SM_i was generated using the private key corresponding to the partial public key PK_i.

- **Expected Output**:
  Successfully generate a proof if either step (3) or (4) fails, indicating incorrect share reconstruction leading to a wrong partial public key or an invalid signature using the claimed partial private key.
- **Public outputs**
  - All commitement set C = { C_1, ... C_t}.
  - Public key of the sender $\mathrm{AuthKey}_i$.

Circuit 3: Malicious Encryption Detection

- **Objective**:
  Verify that an encrypted share sent from P_i to P_j can be correctly decrypted using a shared key derived via ECDH and that the resulting plaintext corresponds to a valid share consistent with P_i's committed polynomial.
- **Verification Steps**:
  i. **Deterministic Key Derivation via ECDH**:
    - The vector $\mathbf{V}_i$ is **sorted deterministically** (e.g., lexicographically by public key bytes).
    - The **last element** of the sorted verification vector is used for key agreement:
      - Participant P_i uses the **private key** from their own last element.
      - Participant P_j uses the **public key** from the corresponding last element of P_i's vector.
    - They perform ECDH:

$$K_{i,j} = \mathrm{ECDH}(\mathrm{PrivKey}_i^{\mathrm{last}}, \mathrm{PubKey}_j^{\mathrm{last}})$$

  - The result is passed through a key derivation function (e.g., HKDF) to produce a symmetric key K_{enc} for encryption with a cipher like ChaCha20-Poly1305[6].
  - This shared secret is used to derive a symmetric encryption key (e.g., via HKDF), which is then used with a cipher like ChaCha20[6] to encrypt:
    - The generation id
    - The share $s_{i,j}$
    - The $\mathrm{HASH}(n, t, \mathrm{generation\_id}, \mathbf{V}_i)$
    - AuthPK_i
    - $\mathrm{SIGN}(\mathrm{AuthPK}, \mathrm{HASH})$

  ii. **Decryption and Share Extraction**:
    - Participant P_j derives the shared key:

$$K_{j,i} = \mathrm{ECDH}(\mathrm{AuthKey}_j, \mathrm{AuthPK}_i)$$

    - Since ECDH is symmetric, $K_{i,j} = K_{j,i}$.
    - P_j uses the derived key to decrypt the ciphertext and extract the share $s_{i,j}$ and accompanying data.

  iii. **Share Validation**:
    - The circuit first proves that the correct decryption key was derived from the initial secrets using the specified ECDH protocol and verification vector.
    - Then, two possible outcomes are evaluated:
      a. If decryption fails (e.g., due to incorrect key derivation, incomplete or malformed ciphertext, or tampering), this constitutes a verifiable failure of proper encryption.

b. If decryption succeeds but the decrypted share is inconsistent with the committed polynomial (e.g., does not satisfy the homomorphic evaluation check), the logic from **Circuit 1** is applied to prove inconsistency or misbehavior in the share itself.

- **Expected Output**:
  The circuit succeeds if it can produce a verifiable contradiction:
  - Enability to decrypt the message.
  - Parsing of the mssages fail.
  - A mismatch in hash.
  - Invalid signature.
  - Invalid polynomial evaluation.
- **Public outputs**
  - All commitement set C = { C_1, ... C_t}.
  - $\mathrm{PubkKey}_i$ and $\mathrm{PubkKey}_j$ (they are used to prove that the prover use correct keys in ECDH).
  - Whole encrypted message.

Circuit 4: Successful Finalization

- **Objective**: Verify the correctness of the final reconstructed key and confirm that all participants have properly completed the protocol.
- **Verification Steps**:
  i. **Commitment Validation**:
  Prove that each commitment C_i is consistent with the corresponding verification vector V_i. This likely involves demonstrating that hashing V_i (along with $n, t, \mathrm{generation\_id}$) results in C_i.
  ii. **Partial Key Consistency**:
  Verify that each participant's public key satisfies:

$$PK_i = \mathrm{P}(i), \quad \text{where } i \in [1, n]$$

  This ensures that each participant has correctly reconstructed their partial key from the shared polynomial.
  iii. **Message Signature Validation**:
  Prove that each signature over message M was generated using the corresponding partial secret key.
  iv. **Final Key Reconstruction via Lagrange Interpolation[4]**:
  Use Lagrange interpolation L to reconstruct the final public key:

$$L(PK_0, \dots, PK_n) = \mathrm{PK}(SS) = \mathrm{P}(0)$$

Note:

The interpolation can be performed over any subset S satisfying:

$$|S| = m, \quad \text{where } m \in [k, n], \quad \text{and} \quad S \subseteq \{PK_0, \dots, PK_n\}$$

This demonstrates that any subset of at least k participants can reconstruct the shared secret.
However, to confirm correctness for all participants, it is recommended to use the full set of partial public keys.

- **Expected Output**:
  The circuit succeeds only if all commitments, partial keys, signatures, and the reconstructed final key are valid.
- **Public outputs**
  - All commitement set C = { C_1, ... C_t}.
  - Public key of the sender $\mathrm{AuthKey}_i$.

References for Zero-Knowledge Verification of Distributed Key Generation

1. **Shamir's Secret Sharing (SSS)**
   A method to divide a secret into multiple parts, requiring a minimum number of parts to reconstruct the original secret.
   Shamir's Secret Sharing - Wikipedia
2. **Zero-Knowledge Proofs (ZKPs)**
   Cryptographic methods enabling proof of validity without revealing underlying information.
   Zero-Knowledge Proof (ZKP) — Chainlink
3. **Distributed Key Generation Protocols**
   Protocols allowing multiple participants to jointly generate a shared public/private key pair without exposing private keys.
   Distributed Key Generation - Wikipedia
4. **Lagrange Interpolation**
   Mathematical method used in Shamir's Secret Sharing to reconstruct secrets from polynomial interpolation.
   Lagrange Polynomial - Wikipedia
5. **Elliptic Curve Diffie–Hellman (ECDH)**
   Key agreement protocol for securely deriving shared keys over an insecure channel.

Elliptic-Curve Diffie–Hellman - Wikipedia

6. **ChaCha20-Poly1305**

   Authenticated encryption algorithm combining confidentiality and integrity protection.

   ChaCha20-Poly1305 - Wikipedia