Consensus on SNARK pre-processed circuit polynomials

Jehyuk Jang *

Tokamak Network

October 2024

Abstract

This paper addresses verifiable consensus of pre-processed circuit polynomials for succinct non-interactive argument of knowledge (SNARK). More specifically, we focus on parts of circuits, referred to as wire maps, which may change based on program inputs or statements being argued. Preparing commitments to wire maps in advance is essential for certain SNARK protocols to maintain their succinctness, but it can be costly. SNARK verifiers can alternatively consider receiving wire maps from an untrusted parties.

We propose a consensus protocol that reaches consensus on wire maps using a majority rule. The protocol can operate on a distributed, irreversible, and transparent server, such as a blockchain. Our analysis shows that while the protocol requires over 50% honest participants to remain robust against collusive attacks, it enables consensus on wire maps with a low and fixed verification complexity per communication, even in adversarial settings. The protocol guarantees consensus completion within a time frame ranging from a few hours to several days, depending on the wire map degree and the honest participant proportion.

Technically, our protocol leverages a directed acyclic graph (DAG) structure to represent conflicting wire maps among the untrusted deliverers. Wire maps are decomposed into low-degree polynomials, forming vertices and edges of this DAG. The consensus participants, or deliverers, collaboratively manage this DAG by submitting edges to branches they support. The protocol then returns a commitment to the wire map that is written in the first fully grown branch. The protocol's computational efficiency is derived from an interactive commit-prove-verify scheme that enables efficient validation of submitted edges.

Our analysis implies that the practical provides a practical solution for achieving secure consensus on SNARK wire maps in environments with dynamic proportion of honest participants. Additionally, we introduce a tunable parameter N that allows the protocol to minimize cost and time to consensus while maintaining a desired level of security.

Contents

*jake@tokamak.network

1 Introduction

Succinct non-interactive argument of knowledge (SNARK) protocols allow provers to generate a proof or argument of a statement. Verifiers, the counterpart on the protocol, are convinced of correctness of the statement, if and only if the proof passes a verification algorithm. The proof is succinct, meaning that its size and the complexity of verification remain constant independent of the computation effort required to deduce the statement.

SNARK protocols can deal with any NP statements; however, they require a pre-compiler to construct a circuit, which attests validity of a specific statement in polynomial time in a deterministic way. Considering a program that takes an input and returns an output, a statement can be a tuple of (the program, an input, an output), where the input and output, combined, are referred to as an instance. For non-deterministic programs, a circuit can be defined only when specific input(s) are provided.

As the complexity of the pre-compiler is asymptotically similar to that of running the program, SNARK verification achieves succinctness only when the circuit is *pre-processed* and readily accessible to verifiers. One practical approach to maintain this succinctness is to involve a third party to deliver the circuit. However, this requires careful consideration; to avoid relying on trust in the third party, circuit delivery should be designed to incorporate significant time delays, which mitigate potential security risks.

In this paper, we propose a protocol that enables network participants to deliver a circuit to verifiers through consensus. Although our protocol relies on the assumption that over 50% of participants are honest, it remains both secure and efficient. Consensus can typically be reached within a few hours to a few days, depending on the circuit's degree of complexity.

1.1 SNARK definition

We define a SNARK protocol that consists of four algorithms: $\text{Compile}_{\text{P}}$, $\text{Compile}_{\text{NP}}$, Prove, and Verify, as follows:

- $\text{Compile}_{\text{P}}$: It takes as input a pre-defined set of unit operations and returns a library $\mathcal{K}$ of subcircuits, each corresponding to the unit operations, respectively, where
  - the unit operations can vary from elementary operations such as multiplication or addition gates to complex but deterministic modules such as MUX-based ALU or cryptographic hash functions.
- $\text{Compile}_{\text{NP}}$: It takes as input a program P with input in and the subcircuit library $\mathcal{K}$ and generates a *wire map* S along with an instance ϕ and a witness ν, where
  - a *wire map* S contains connectivity between interface wires of the library subcircuits as a permutation, encoded in a polynomial.
  - There exist a deterministic linear map $\text{Derive}_S(\mathcal{K}) \mapsto C$ for each wire map S. Here C is a linear combination of the elements in $\mathcal{K}$ such that
  $$(\phi = (\text{in}, \text{out})) \wedge (P(\text{in}) = \text{out}) \iff C(\phi, \nu) = 1.$$
  - Given $\mathcal{K}$, $\text{Compile}_{\text{NP}}$ is a subroutine of a relation generator that generates a binary relation R such that, for a finite field $\mathbb{F}$ and a statement length l,
  $$R = \{(\phi, \pi) \in \mathbb{F}^l : C(\phi, \pi) = 1\}.$$
- $\text{Prove}_{\text{SNARK}}$: It takes as input srs, $\mathcal{K}$, S, ϕ, and ν and generates a proof π, where
  - a structured reference string srs can be a string of random encoding of monomial evaluations provided by an Oracle.
- $\text{Verify}_{\text{SNARK}}$: It takes as input srs, $\mathcal{L}$, s, ϕ, and π and either accepts or rejects π, where
  - $\mathcal{L}$ is a set of commitments to the library subcircuits in $\mathcal{K}$, and
  - s is a commitment to the wire map S.

1.1.1 SNARK examples

- If $\mathcal{K}$ contains only a single element and S is identity, Groth16[1] matches our SNARK definition.
- If $\mathcal{K}$ only consists of addition and multiplication gates and their linear combinations (custom gates), PlonK[2] fits our SNARK definition.
- If $\mathcal{K}$ varies from elementary gates to complex but deterministic modules, Tokamak zk-SNARK[3] fits our SNARK definition.

1.2 Security and efficiency of SNARK

According to SNARK theory, informally, if π is accepted by $\text{Verify}_{\text{SNARK}}$, then the prover who generated π, regardless of the algorithm used, has knowledge of a pair $(\phi, \nu) \in R$ with overwhelming probability. Furthermore, SNARK can provide an efficient verifiable computation due to succinct verification properties:

- The proof length is $O(1)$,
- proof generation requires $O(|P| \log_2 |P|)$ time, and
- proof verification takes $O(|\phi|)$ time.

However, this security and efficiency are only achievable under an assumption that the verifier running $\text{Verify}_{\text{SNARK}}$ has access to honestly generated commitments $\mathcal{L}$ and s, corresponding to the library subcircuits and a wire map, respectively.

1.3 Paper outline

1.3.1 Problem statement

As mentioned above, the SNARK verification algorithm $\text{Verify}_{\text{SNARK}}$ relies on commitments to the library subcircuits $\mathcal{L}$ and (a commitment to) a wire map s as auxiliary inputs, in addition to the primary inputs an instance ϕ and a proof π. The security and efficiency of the SNARK are guaranteed only if $\mathcal{L}$ and s are delivered honestly. However, ensuring reliable delivery of these commitments is not in the scope of SNARK protocols.

Perfect reliability for both $\mathcal{L}$ and s can be achieved only if they are reproduced inside $\text{Verify}_{\text{SNARK}}$, which compromises the succinctness of SNARKs. Fortunately, for more reliable delivery of $\mathcal{L}$, researchers have developed multi-party computation (MPC) [4–6]. While an MPC ceremony typically takes several weeks to a month and involves considerable verification complexity, these costs are practically acceptable, as $\mathcal{L}$ is independent of specific programs and can be reused for multiple SNARK verifications following a single ceremony.

The primary challenge, however, lies in reliably delivering the wire map s, which depends on the specific program and its input. Repeating such high costs for every program input is impractical and not a feasible solution in most cases.

1.3.2 Contribution

In this paper, we propose a protocol that reaches a consensus on s among network participants based on a majority rule. Although our protocol requires more than 50% of participants to be honest, we show that our protocol is efficient in both computational and communication complexities, with a guarantee of consensus completion within a few hours to several days, depending on the degree of the wire map and the honest portion.

Specifically, letting N be a parameter for the minimum number of votes required from participants to reach a consensus, the time required for consensus and the overall communication cost for the entire process are $O(N)$. We also show that increasing N reduces the reliance on the honest portion for security. For example, when all malicious participants collaborate and collude, by making N sufficiently large, we can make it unlikely that a malicious group will win the consensus even with a low honest portion slightly greater than 50%. In other words, a larger N provides greater security at the cost of a longer consensus duration and increased communication complexity.

Furthermore, our protocol incurs only a small verification complexity for each participant. Letting D be the degree of the wire map being agreed upon, each participant needs to compute $O(D)$ multiplications in a field while performing only $O(1)$ exponentiations in a multiplicative group (Table 1). This reduced complexity is a strength of our protocol compared to traditional MPC protocols, allowing it to be deployed on cost-sensitive networks such as a blockchain.

1.3.3 Technical overview

In Section 2, we will define an algebraic decomposition of polynomials, called *accumulative decomposition*. Given a set $\mathcal{X}$ of any D distinct evaluation points, this decomposes a polynomial $p(X)$ of degree D into N contributions $\{c_i(X)\}_{i=0}^{N-1}$, each being a polynomial of degree $\lceil D/N \rceil$. Reconstruction can be done by dividing $\mathcal{X}$ into N accumulative subsets $\{\mathcal{X}_i\}_{i=0}^{N-1}$, where $\mathcal{X}_i \subset \mathcal{X}_{i+1}$ for $i = 0, \cdots, N-2$ and $\mathcal{X}_{N-1} = \mathcal{X}$. By defining vanishing polynomials $t_{\mathcal{X}_i}$, each of which vanishes on $\mathcal{X}_i$, the polynomial p can be reconstructed by

$$p(X) = \sum_{i=1}^{N-1} c_i(X) t_{\mathcal{X}_{i-1}}(X) + c_0(X). \tag{1}$$

Considering the cases where multiple polynomials of degree D are decomposed by the accumulative decomposition, we observe that the decomposition results can be placed on a directed acyclic graph (DAG). Figure 1 illustrates a DAG. Simply put, a tuple $\mathbf{v}_i = (\mathcal{X}_i, c_i, t_{\mathcal{X}_i})$ can form a vertex. A pair of vertices $(\mathbf{v}_i, \mathbf{v}_j)$ can form an edge only if $\mathcal{X}_i \subset \mathcal{X}_j$. We can reconstruct a polynomial from a branch of N consecutive edges, e.g., $(\mathbf{v}_0, \mathbf{v}_1, \cdots, \mathbf{v}_{N-1})$, by computing (1). As a result, each branch of length N represents a polynomial of degree D.

Recall our main problem in which a number of network participants attept to deliver a commitment to a wire map to the SNARK verifier. Under an untrusted environment, there can be conflicts among wire maps during the delivery. The protocol we will propose in Section 3 is to solve this conflict by allowing the participants to collaboratively manage a DAG. When a new consensus session begins, the DAG is initialized as an empty graph. As the session proceeds, the participants can vote on a wire map by submitting an edge to extend an existing branch or a vertex to propose a new branch so that branches on the DAG grow. The protocol ends when the first branch that reaches a target height is found, at which point the commitment to the wire map represented by that branch is returned. Consequently, the wire map that that first receives N votes is chosen and delivered to the SNARK verifier.

To reduce communication complexity, the protocol utilizes a representative selection algorithm that periodically selects a contributor from among the participants. Each contributor is permitted to submit a single vote. Assuming that the selection process is uniform, the resulting wire map can be expected to receive the support of the largest number of participants.

To reduce computational complexity of the protocol's online algorithms, we carefully separate the whole procedure of voting into two parts, computing a vote and validating a vote. Vote computation is run offline by a contributor that includes running the selection algorithm, choosing a branch to vote for, computing a new edge, and generating witnesses and proofs for the validity of the contributor selection and the edge. A vote is then validated on a server through a commit-prove-verify interactive scheme under the random oracle model, based on the KZG commitment scheme[7]. We have also optimized the scheme with techniques introduced in [2].

1.4 Notation and assumption

- Given a security parameter λ, we use a bilinear group $(\mathbb{F}, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$, where

- $\mathbb{F}$ is a finite field,
- $\mathbb{G}_1$ is an additive group defined over $\mathbb{F}$,
- $\mathbb{G}_2$ and $\mathbb{G}_T$ are additive and multiplicative groups defined over an extension of $\mathbb{F}$, respectively, and
- e is a bilinear map defined as $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$.
- Letting g and h be the generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively, for $x \in \mathbb{F}$, we write $[x]_1 = xg$ and $[x]_2 = xh$.

- $\mathbb{F}_d[X]$ denotes a ring of polynomials of degree d defined over $\mathbb{F}$.
  - similarly, $\mathbb{F}_{\leq d}[X]$ and $\mathbb{F}_{<d}[X]$ denote rings of polynomials of degree at most d and of degree less than d, respectively.
- Given $f \in \mathbb{F}_{d-1}[X]$ and a set $\mathcal{X} = (x_0, \cdots, x_{n-1})$ of $n \leq d$ evaluation points, we write $\tilde{f}_{\mathcal{X}} \in \mathbb{F}_{n-1}[X]$ as the Lagrange interpolation based on a data set $\{(x, f(x))\}_{x \in \mathcal{X}}$, which is calculated by

$$\tilde{f}_{\mathcal{X}}(X) = \sum_{i=0}^{l-1} f(x_i) \prod_{k=0, k\neq i}^{l-1} \frac{(X - x_k)}{(x_i - x_k)}.$$

  - It is straightforward to see that if $n = d$, $\tilde{f}_{\mathcal{X}}(X) = f(X)$.
  - Analogously, for $n = 0$ or $\mathcal{X} = \emptyset$, we write $\tilde{f}_{\emptyset}(X) := 0$.
- We assume that for some $D_p \in \mathbb{F}$, all protocol entities are given srs defined as, for $\tau \in \mathbb{F}$,

$$\text{srs} := ([\tau]_1, [\tau^2]_1, \cdots, [\tau^{D_p-1}]_1, [\tau]_2, [\tau^{D_p}]_2).$$

2 Polynomial analysis

Motivated by polynomial multipoint evaluation in [8, 9], we introduce the accumulative decomposition of polynomials and subsequently define a directed acyclic graph (DAG) constructed based on this decomposition.

2.1 Accumulative decomposition of a polynomial

Definition 1. Let $D_p, D_c \in \mathbb{F}$ be parameters such that $D_c \mid D_p$. Consider a polynomial $f \in \mathbb{F}_{D_p-1}[X]$. Define $N := D_p/D_c$. Let $(x_0, x_1, \cdots, x_{D_p-1}) \in \mathbb{F}^{D_p}$ be a sequence of evaluation points in any order. *Accumulative decomposition* of a polynomial $f(X)$ is expressed by

$$f(X) = \sum_{i=0}^{N-1} c_i(X) t_{\mathcal{X}_{i-1}}(X),$$

where

- We let $\mathcal{X}_i$ denote the *incremental sets* of evaluation points such that, for $i = 0, \cdots, N-2$,

$$\mathcal{X}_{i+1} := \mathcal{X}_i \cup \{x_{(i+1)D_c}, \cdots, x_{(i+2)D_c-1}\},$$

  and $\mathcal{X}_0 := \{x_0, \cdots, x_{D_c-1}\}$,

- We let $t_{\mathcal{X}_i}(X)$ denote the *accumulation polynomials*, defined for $i = 0, \cdots, N-1$,

$$t_{\mathcal{X}_i}(X) := \prod_{i \in \mathcal{X}_i} (X - x_i),$$

and $t_{\mathcal{X}_{-1}} := 1$,

- We let $c_i \in \mathbb{F}_{D_c - 1}[X]$ denote the *contributions* to $f(X)$ such that, for $i = 1, \cdots, N-1$

$$\tilde{f}_{\mathcal{X}_i}(X) - \tilde{f}_{\mathcal{X}_{i-1}}(X) = c_i(X) \prod_{x \in \mathcal{X}_{i-1}} (X - x),$$

and $c_0(X) = \tilde{f}_{\mathcal{X}_0}(X)$.

The accumulative decomposition uses the following straightforward identity:

$$\begin{aligned} f(X) - \tilde{f}_{\mathcal{X}_0}(X) =& (\tilde{f}_{\mathcal{X}_{N-1}}(X) - \tilde{f}_{\mathcal{X}_{N-2}}(X)) + \\ & (\tilde{f}_{\mathcal{X}_{N-2}}(X) - \tilde{f}_{\mathcal{X}_{N-3}}(X)) + \cdots + \\ & (\tilde{f}_{\mathcal{X}_1}(X) - \tilde{f}_{\mathcal{X}_0}(X)). \end{aligned}$$

2.2 DAG of contributions

Definition 2. Given parameters (D_p, D_c) with $D_c \mid D_p$, we define a DAG $\mathcal{G} = (V, E)$ as follows,

- **Vertex**: Each vertex $v \in V$ is represented as $v = (\mathcal{X}, c(X), t(X), p(X))$, where
  - $\mathcal{X} \subset \mathbb{F}$ is the set of D_c evaluation points,
  - $c \in \mathbb{F}_{D_c - 1}[X]$ is the contribution,
  - $t \in \mathbb{F}_{\leq D_p}[X]$ is the accumulation polynomial, and
  - $p \in \mathbb{F}_{< D_p}[X]$ is the branch polynomial.
- **Edge**: The set of edges is defined as $E \subseteq \{(v_1, v_2) : v_1, v_2 \in V \wedge v_1 \neq v_2\}$.
- **Branch**: A sequence $(v_0, v_1, \cdots, v_{k-1}) \in V^k$ forms a branch if each pair pair $e_i = (v_{i-1}, v_i)$, for $i = 1, \cdots, k-1$, belongs to E.
- **Branch polynomial**: For a branch $(v_0, v_1, \cdots, v_{k-1})$ where $v_i = (\mathcal{X}^{(i)}, c_i(X), t_{\mathcal{X}_i}(X), p_i(X))$, each branch polynomial $p_i(X)$ is defined as

$$p_i(X) := \sum_{i=0}^{k-1} c_i(X) t_{\mathcal{X}_{i-1}}(X),$$

where $\mathcal{X}_i = \bigcup_{j=0}^{i} \mathcal{X}^{(j)}$.

- **Maximum height**: The length of a branch does not exceed $N := D_p / D_c$.
- **Complete branch**: We say a branch of length k is *N-incomplete* if $k < N$, and *N-complete* otherwise.

- **Finishing edge**: An edge $e = (u, v) \in E$ is called a finishing edge, if v is the last vertex in an N-complete branch.

Figure 1 illustrates a DAG of contributions. Each complete branch in this DAG corresponds to a polynomial of degree $D_p - 1$, which can be retrieved from the branch polynomial of the last vertex. The representative branch polynomials may be either identical or different. All the intermediate contributions on each complete branch must be reproducible by their representative branch polynomial $f(X)$, as follow,

$$c_i(X) = \frac{\tilde{f}_{\mathcal{X}_i}(X) - \tilde{f}_{\mathcal{X}_{i-1}}(X)}{\prod_{x \in \mathcal{X}_{i-1}}(X - x)}.$$

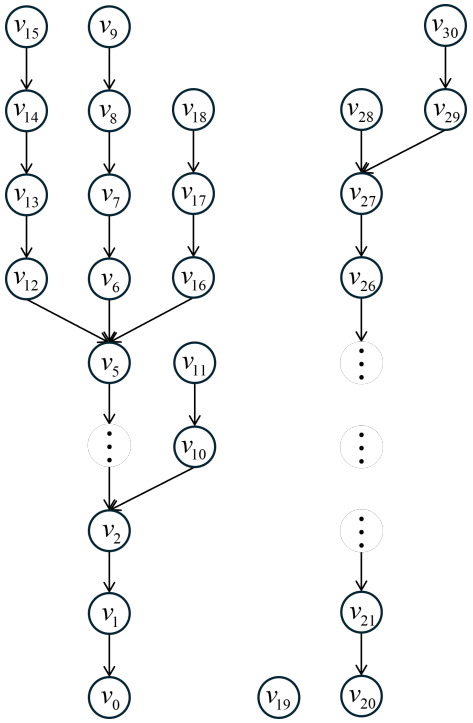

Figure 1: Example DAG of contributions

3 Proposed consensus protocol

Recall that the verification algorithm of Tokamak zk-SNARK, $\text{Verify}_{\text{SNARK}}$, requires a wire map as an auxiliary input. In this section, we propose a consensus protocol to resolve conflicts among wire maps that could occur during their elivery from network participants to the SNARK verifiers, by reaching a consensus among network participants.

In our protocol, each participant is assumed to propose a wire map of degree $D_p - 1$. Some of the proposed wire maps may be identical, especially when they are honest. As discussed previously in Section 1, we assume there is a contributor selection protocol that uniformly selects contributors from the participants.

Given the uniform selection algorithm, we can model the consensus procedure upon our protocol as the clustering of wire map conflicts: participants possessing identical wire maps can be clustered (Figure 2), and the distribution of wire maps in votes submitted by the selected contributors follows the proportions that the clusters occupy relative to the total participant population. In Section 4, we will show that our protocol can achieve a negligible chance of attack success by collusive adversaries, provided that honest participants form the majority cluster.

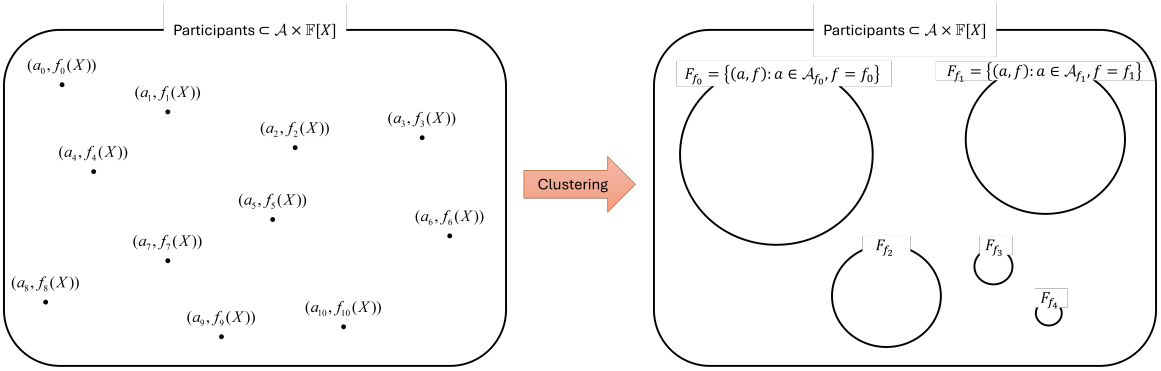

Figure 2: Clustering of wire maps. In the left box, a_i indicates each participant purporting a wire map $f_i(X)$. In the right box, each subset F_{f_k} represents a cluster of the participants purporting the same wire map.

The clusters are represented as DAG branches. Thus, each contributor can support a cluster by submitting a vote, which contains a new edge on the corresponding branch that they are supporting. The main functionality of our protocol is to provide the algorithm UpdateDAG, which validates the votes and accepts their new edges as elements of the DAG.

Definition 3. Our consensus protocol consists of three algorithms (InitDAG, VoteGen, UpdateDAG). The protocol uses the following terminologies:

- **Server**: The protocol runs on a transparent and irreversible server, where data is gathered and published in **blocks**, every T seconds. All important outputs generated by the protocol users are recorded on this server upon requests.
- **Consensus manager**: There is a consensus manager, who raises a consensus agenda (ϕ, π) for (the commitment to) a wire map f , such that $\text{Verify}_{\text{SNARK}}(\phi, \pi, [f(\tau)]_1) = 1$. The consensus manager configures consensus parameters (D_p, D_c), and initiates a consensus session (See Appendix A for the initiating algorithm InitDAG).
- **Participant**: Participants of the protocol are possessing their wire maps. We assume a list of participants is available on the server before each consensus is initiated. The participants in $\mathcal{A}$ are represented by their public keys in $\mathbb{G}_2$. The set $\mathcal{A}$ may be structured for fast data access and retrieval.
- **Contributor**: Contributors are selected from the set $\mathcal{A}$ under a predefined selection rule (An example of the selection rule is illustrated in Appendix B). Each contributor may compute a contribution based on their wire map[1]. A contribution is then presented as a *vote*, packaged

[1]Although an example of the contribution algorithm VoteGen is provided in Appendix C, we do not assume that contributors behave honestly.

with any witnesses and proofs for its validity. The format of votes will be specified in the next subsection.

- **Consensus session**: A consensus session is initiated by the consensus manager to initialize a DAG $\mathcal{G}$. During the session, at most one participant in the list is selected as the contributor for each block. Their vote is validated by an algorithm UpdateDAG before being recorded onto the server. As the server blocks are generated, the DAG grows with contributions from the contributors. The session lasts for $O(N)$ blocks, concluding when the first branch of length $N := D_p/D_c$ is found, such that the branch polynomial f satisfies $\text{Verify}_{\text{SNARK}}(\phi, \pi, f) = 1$.

3.1 Data Format

We define the format of data exchanged between the participants and the server. Suppose a DAG $\mathcal{G} = (V, \mathcal{E})$ has been initialized, along with a preallocation of server storage for $\mathcal{E}$. The set $\mathcal{E}$ is assumed to be structured to allow fast access by indexing, meaning that $\mathcal{E}(a)$ for $a \in \mathbb{F}$ returns the element of $\mathcal{E}$ indexed by a.

3.1.1 Vertex Format

For every $v \in V$, $v = (x, c, t, p)$, where

- $x \in \mathbb{F}$ is a base evaluation point to span $\mathcal{X}$,
- $c \in \mathbb{G}_1$ is the commitment to a contribution of degree $D_c - 1$,
- $t \in \mathbb{G}_1$ is the commitment to an accumulation polynomial,
- $p \in \mathbb{G}_1$ is the commitment to a branch polynomial.

3.1.2 Edge Format

For every $\mathbf{e} \in \mathcal{E}$, $\mathbf{e} = (w_0, v_1)$, where

- $w_0 \in \mathbb{F} \cup \{\perp\}$ is the pointer to the previous vertex, specifically, $\exists v_0 \in V$ such that $\mathcal{E}(w_0) = (w_{-1}, v_0)$, if and only if $w_0 \in \mathbb{F}$,
- $v_1 \in V$ is the next vertex.

3.1.3 Vote Format

A vote is defined as a tuple $\mathbf{o} = (\mathbf{e}, b, \tilde{t}, \pi_\lambda, t_\zeta, a, w_a, C)$, where

- $b \in \{0, 1\}$ is a flag such that $b = 1$ if and only if $\mathbf{e}$ is the last edge on an N-complete branch,
- $(\tilde{t}, \pi_\lambda, t_\zeta) \in \mathbb{G}_1^2 \times \mathbb{F}$ are an auxiliary commitment, an opening proof, and an opening, respectively, for efficient vertex and edge validation based on a KZG commitment scheme,
- $a \in \mathbb{G}_2$ is the public key of the contributor of this vote such that $\text{Hash}(a) = x$,
- w_a is a witness that a is the selected contributor,
- $C \in \mathbb{G}_1$ is a signature for this vote.

3.2 Vote Validation

Let a vote $\mathbf{o}$ be in the correct format. We define conditions for this vote to be validated.

3.2.1 Contributor Selection

By the definition, a contributor a is selected from $\mathcal{A}$ for each block under a predefined selection rule. We model this contributor selection process as an NP problem. Let Select denote an NP-time algorithm to decide a contributor, which is run on participants' local. Let IsSelected denote a P-time algorithm to verify the result of contributor selection, which is run on the server. We define the two algorithms as follows:

- $\text{Select}(\mathcal{A}, h, a) \mapsto w_a$: It takes as input the participant list $\mathcal{A}$, a public coin toss h, and the public key a of the participant and returns any witness w_a to the correct selection of a as the contributor.
- $\text{IsSelected}(\mathcal{A}, h, a, w_a) \mapsto 0/1$: It takes as input $\mathcal{A}$, h, a, and a witness w_a and returns either acceptance or rejection.

We exemplify the contributor selection and verification algorithms in Appendix 2, but our protocol does not mandate its use. Instead, in the next section for the analysis of our protocol, we assume the distribution of Select follows that of h.

3.2.2 Signature Verification

To prevent spoofing, a signature is enveloped in every vote submission. We define a signature as

$$C := sp_1,$$

where $s \in \mathbb{F}$ is the *private key* of the contributor such that $a = [s]_2$, and p_1 is the branch polynomial commitment in the second vertex v_1 of the edge $\mathbf{e} = (v_0, v_1)$. This signature can be verified by the following pairing equation:

$$e(C, [1]_2) = e\left(p_1, a\right). \tag{2}$$

3.2.3 Summary of Vote Validation

Definition 4 (Valid votes)**.** We say a vote $\mathbf{o}$ is valid if and only if:

- $\text{IsSelected}(\mathcal{A}, h, a, w_a) = 1$,
- $e(C, [1]_2) = e\left(p_1, a\right)$,
- $\mathbf{e}$ is a valid edge.

3.3 Edge Validation

Let an edge $\mathbf{e}$ be in the correct format. We define conditions for $\mathbf{e}$ to be validated.

3.3.1 Fast Vertex Retrieval

Since the server is transparent and irreversible, the server can quickly access the previous block pointed by the index w_0 to retrieve a vertex v_0 such that $\mathcal{E}(w_0) = (w_{-1}, v_0)$. This fast access is feasible only when the index w_0 is provided by the contributor. We assume there is an algorithm for the contributor to retrieve the index w_0 as follows:

- GetIndex$(\mathcal{E}, v_0) \mapsto w_0$: It takes as input a (sorted) edge set $\mathcal{E}$ and a vertex v_0 and returns the index w_0 such that $\mathcal{E}(w_0) = (w_{-1}, v_0)$, if $v_0 \in V$, and returns $\perp$ otherwise.

3.3.2 Verifying the Edge Direction

Consider an (retrieved) edge (v_0, v_1) where $v_i = (x^{(i)}, c_i, t_i, p_i)$ for $i \in \{0, 1\}$. The base evaluation points $x^{(i)}$ can generate the sets $\mathcal{X}^{(i)}$ of evaluation points defined for the DAG as follows:

$$\mathcal{X}^{(i)} = \bigcup_{k=0}^{D_c-1} \{x^{(i)} + k\}.$$

We write $t_i = [t_{\mathcal{X}_i}(\tau)]_2$, where $t_{\mathcal{X}_i}(X)$ are the accumulation polynomials, and by the definition of the incremental sets of evaluation points, they must satisfy

$$t_{\mathcal{X}_1}(X) = t_{\mathcal{X}_0}(X)\tilde{t}_a(X), \tag{3}$$

where $\tilde{t}_a(X) := \prod_{x \in \mathcal{X}^{(1)}}(X - x)$, and we can write $t_{\mathcal{X}_0}(X) = 1$, if $v_0 = \perp$. If the following auxiliary inputs $\tilde{t} := [\tilde{t}_a(\tau)]_1$, $\pi_{\tilde{t}} := [\pi_{\tilde{t}}(\tau)]_1$, and $\pi_{\text{dir}} := [\pi_{\text{dir}}(\tau)]_1$ for some polynomials $\pi_{\tilde{t}}, \pi_{\text{dir}} \in \mathbb{F}[X]$ are provided by the contributor, equation (3) can be equivalently checked by the two following equations:

$$\tilde{t}_a(X) - \tilde{t}_\zeta = \pi_{\tilde{t}}(X)(X - \zeta), \tag{4}$$

$$t_{\mathcal{X}_1}(X) - t_{\mathcal{X}_0}(X)\tilde{t}_\zeta = \pi_{\text{dir}}(X)(X - \zeta), \tag{5}$$

where ζ is a coin tossed by a random oracle given $(t_0, t_1, \tilde{t})$, and $\tilde{t}_\zeta := \tilde{t}_a(\zeta)$.

Checking (4)-(5) can be replaced with the following procedure:

1. Compute $\zeta = \text{Hash}(t_0, t_1, \tilde{t})$.
2. Compute $\tilde{t}_\zeta = \tilde{t}_a(\zeta)$.
3. Check

$$e(\tilde{t} - \tilde{t}_\zeta[1]_1, [1]_2) = e(\pi_{\tilde{t}}, [\tau]_2 - \zeta[1]_2), \text{ and} \tag{6}$$

$$e(t_1 - \tilde{t}_\zeta t_0, [1]_2) = e(\pi_{\text{dir}}, [\tau]_2 - \zeta[1]_2). \tag{7}$$

3.3.3 Exception for Finishing Edges

For a finishing edge, the commitment $t_1 = [t_{\mathcal{X}_1}(\tau)]_1$ cannot be derived from the universal reference string srs, since $t_{\mathcal{X}_1}(X)$ is of degree D_p, whereas the $\mathbb{G}_1$ elements of srs only support monomials of degree up to $D_p - 1$. Thus, for a finishing edge, the last pairing equation of the above procedure is changed to

$$e(t_1 - \tilde{t}_\zeta t_0, [1]_2) e\left([1]_1, [\tau^{D_p}]_2\right) = e(\pi_{\text{dir}}, [\tau]_2 - \zeta[1]_2), \tag{8}$$

which allows an exception for t_1 to be computed as

$$t_1 = [t_{\mathcal{X}_1}(X) - X^{D_p}|_{X=\tau}]_1.$$

To sum up, letting $b \in \{0,1\}$ denote a flag for notifying a finishing edge ($b = 1$, if and only if $\mathbf{e}$ is a finishing edge), the pairing checks (7) and (8) can be combined as

$$e(t_1 - \tilde{t}_\zeta t_0, [1]_2)e\left(b[1]_1, [\tau^{D_p}]_2\right) = e(\pi_{\mathrm{dir}}, [\tau]_2 - \zeta[1]_2). \tag{9}$$

3.3.4 Commitment to the Branch Polynomial

Consider an (retrieved) edge (v_0, v_1) where $v_i = (x^{(i)}, c_i, t_i, p_i)$ for $i \in \{0,1\}$. We write $c_i = [c_i(\tau)]_1$, $t_i = [t_{\mathcal{X}_i}(\tau)]_1$, and $p_i = [p_i(\tau)]_1$. By Definition 2, the branch polynomials are expected to satisfy the following relationship:

$$p_1(X) - p_0(X) = c_1(X)t_{\mathcal{X}_0}(X),$$

where if $v_0 = \perp$, $p_0(X) = 0$ and $t_{\mathcal{X}_0}(X) = 1$. Let $\pi_{t_0} := [\pi_{t_0}(\tau)]_1$ and $\pi_{\mathrm{brc}} := [\pi_{\mathrm{brc}}(\tau)]_1$ for some polynomials $\pi_{t_0}, \pi_{\mathrm{brc}} \in \mathbb{F}[X]$, ζ denote a coin tossed by a random oracle given (c_0, t_0, p_0, p_1), and $t_\zeta := t_{\mathcal{X}_0}(\zeta)$. If $(\pi_{t_0}, \pi_{\mathrm{brc}}, t_\zeta)$ is provided by the contributor as auxiliary input, the above equation can be equivalently checked by the two following equations:

$$t_0(X) - t_\zeta = \pi_{t_0}(X)(X - \zeta), \tag{10}$$
$$p_1(X) - p_0(X) - c_1(X)t_\zeta = \pi_{\mathrm{brc}}(X)(X - \zeta). \tag{11}$$

Checking (10)-(11) can be replaced with the following procedure: If $v_0 \neq \perp$,

1. Compute $\zeta = \mathrm{Hash}(c_0, t_0, p_0, p_1)$,
2. Check
$$\begin{aligned} e(t_0 - t_\zeta[1]_1, [1]_2) &= e(\pi_{t_0}, [\tau]_2 - \zeta[1]_2), \text{ and} \\ e(p_1 - p_0 - t_\zeta c_1, [1]_2) &= e(\pi_{\mathrm{brc}}, [\tau]_2 - \zeta[1]_2), \end{aligned} \tag{12}$$

and if $v_0 = \perp$, it is sufficient to simply check $p_1 = c_1$.

3.3.5 Summary of Edge Validation

Remark (Aggregation of the proofs)**.** Let $\lambda := \mathrm{Hash}(\tilde{t}_\zeta, t_\zeta)$. The four proofs $\pi_{\tilde{t}}, \pi_{t_0}, \pi_{\mathrm{dir}}, \pi_{\mathrm{brc}}$ and the four pairing equations in (6), (9), and (12) can be aggregated into π_λ and

$$e\left(\mathrm{LHS}_\lambda, [1]_2\right) = e(\pi_\lambda, [\tau]_2 - \zeta[1]_2)e\left(-\lambda^2 b[1]_1, [\tau^{D_p}]_2\right), \tag{13}$$

respectively, where

$$\begin{aligned} \mathrm{LHS}_\lambda &:= \tilde{t} + \lambda t_0 + \lambda^2 t_1 + \lambda^3(p_1 - p_0 - t_\zeta c_1) - (\tilde{t}_\zeta + \lambda t_\zeta + \lambda^2 \tilde{t}_\zeta t_\zeta)[1]_1, \\ \pi_\lambda &:= \pi_{\tilde{t}} + \lambda \pi_{t_0} + \lambda^2 \pi_{\mathrm{dir}} + \lambda^3 \pi_{\mathrm{brc}}. \end{aligned}$$

Definition 5 (Valid edges)**.** We say an edge $\mathbf{e}$ is valid, if and only if there exists $\pi_\lambda \in \mathbb{G}_1$ such that equation (13) holds, where $\zeta = \mathrm{Hash}(c_1, t_0, t_1, p_0, p_1, \tilde{t})$ and $\lambda = \mathrm{Hash}(\tilde{t}_\zeta, t_\zeta)$.

3.4 Main algorithm

As the main algorithm of our clustering protocol, we define UpdateDAG. The algorithm takes as input the vote of a contributor, validates it, and reflects the edge in it on the DAG $\mathcal{G}$. It also runs the SNARK verify algorithm $\text{Verify}_{\text{SNARK}}$, if there is an N-complete branch in $\mathcal{G}$.

Remark (All the vote validation in one)**.** Consider a vote $\mathbf{o}$ in the correct format and denote $\mathbf{e} = (v_0, v_1)$ and $v_i = (x^{(i)}, c_i, t_i, p_i)$ for $i \in \{0, 1\}$. Let $\gamma = \text{Hash}(\text{LHS}_\lambda, \pi_\lambda, C, p_1)$ be a random oracle instantiation. A vote is valid if and only if the following equation holds:

$$e\left(\text{LHS}_\lambda + \gamma C, [1]_2\right) = e(\pi_\lambda, [\tau]_2 - \zeta[1]_2) e\left(-\lambda^2 b[1]_1, [\tau^{D_p}]_2\right) e\left(\gamma p_1, a\right),$$

which is an aggregation of (2) and (13).

Based on this vote check, we provide a pseudo-code of UpdateDAG in Algorithm 1.

4 Analysis

In the clustering protocol, we allow using a parameter D_c to balance cost and security. We define a target height $N := D_p/D_c$. In this section, we show that larger N increases the time and cost to reach consensus, but enhances security. We also provide specific numerical values for time, cost, and security, which will help the consensus manager choose a suitable D_c.

4.1 Algorithm Complexity

During a clustering session, our main algorithm UpdateDAG is called at most once per block, leading to at least N calls in total to reach consensus. Table 1 summarizes the communication and computation complexities for a single call to UpdateDAG. The cost cost associated with the sub-functions IsSelected, Retrieve, and $\text{Verify}_{\text{SNARK}}$, as well as retrieving the structured reference string (srs) are not included.

Complexity type			**Any contributors**	**The last contributor**
Communication	**Input**	Read-only	$6\mathbb{G}_1 + 1\mathbb{G}_2 + 2\mathbb{F} + \lvert w_a \rvert$	-
		R&W	$5\mathbb{G}_1 + 4\mathbb{F} + \lvert\mathcal{A}\rvert$	$\lvert\phi\rvert + \lvert\pi\rvert$
	Output	R&W	$3\mathbb{G}_1 + 2\mathbb{F}$	-
Computation	$\mathbb{G}_1$	Multipl.	8	$\lvert\text{Verify}_{\text{SNARK}}\rvert$
	$\mathbb{G}_1$	Pairing	4	
	$\mathbb{F}$	Multipl.	$O(D_c)$	
	$\mathbb{F}$	Hash	4	

Table 1: The complexity of UpdateDAG. The last contributor who feeds a finishing edge in the algorithm performs additional computation. Regarding the complexity types, the read-only input refers to a part of input that is used locally by the algorithm but does not affect the original data stored on the server. In contrast, the R&W input and output, referring to read and write input and output, indicate the data that can be modified by the algorithm. MSM refers to multi-scalar multiplication.

In terms of communication complexity, UpdateDAG takes inputs of constant sizes in addition to the variable-size inputs W_a and $\mathcal{A}$, used by the IsSelected function. Read and write inputs, such as edges, are provided by the server, while read-only inputs are provided by the contributor and used temporarily within the algorithm.

The variable input sizes $|W_a|, |\mathcal{A}|$ are designed with the algorithm's efficiency in mind, particularly for the (Select, IsSelected) pair. For example, given $N_{\mathcal{A}}$ participants, if IsSelected is based on a Merkle tree, $|W_a|$ and $|\mathcal{A}|$ can be $O(\log N_{\mathcal{A}})$ and $O(N_{\mathcal{A}})$, respectively, and if it is based on a KZG commitment scheme, $|W_a|$ and $|\mathcal{A}|$ can each consist of single $\mathbb{G}_1$ elements.

Regarding computation complexity, UpdateDAG performs a fixed number of operations in $\mathbb{G}_1$, which remain constant regardless of the parameters D_p and D_c.

4.2 Consensus Time and Security

4.2.1 Stochastic Model for Clustering Sessions

Suppose participants are forming two groups, one group of honest participants and another of malicious attackers, each supporting an honest wire map and a forged wire map, respectively. In other words, each group advocates for one branch. We assume the participants are rational and avoid forking their branch, as such behavior reduces the chance for their wire map to be selected as the final consensus. We also assume that the distribution of Select strictly follows that of h, which is uniform.

We model the growth of DAG branches as random walks $H_n, A_n \in \mathbb{Z}$ for a block index n. Let the portion of honest participants be denoted by $p_H \in [0, 1]$. Analogously, the portion of malicious participants is $p_A = 1 - p_H$. When a consensus instance is initiated, the random walks are initialized to $H_0 = A_0 = 0$. As the consensus proceeds, either H_n or A_n increases by one for every block formation. More specifically, the state transition probability follows:

$$\begin{aligned}
\Pr(H_{n+1} = x + 1, A_{n+1} = y \mid H_n = x, A_n = y) &= p_H, \\
\Pr(H_{n+1} = x, A_{n+1} = y + 1 \mid H_n = x, A_n = y) &= p_A, \\
\Pr(H_{n+1} = x + 1, A_{n+1} = y + 1 \mid H_n = x, A_n = y) &= 0, \\
\Pr(H_{n+1} = x, A_{n+1} = y \mid H_n = x, A_n = y) &= 0.
\end{aligned}$$

4.2.2 Consensus Time

The clustering process is completed and reaches a consensus as soon as the random walks first encounter $H_n = N$ or $A_n = N$. To estimate the consensus time, we observe the first block number R at which the consensus is reached, defined as:

$$R := \min\{k : (H_k \geq N \wedge A_k < N) \vee (H_k < N \wedge A_k \geq N)\}.$$

Clearly, $R \in [N..2N - 1]$. The distribution of R is given by:

$$\Pr[R = r] = \begin{cases} 0, & \text{for } r < N \text{ or } r \geq 2N, \\ \binom{r-1}{N-1} \left(p_A^N (1 - p_A)^{r-N} + p_A^{r-N} (1 - p_A)^N \right), & \text{for } N \leq r < 2N. \end{cases}$$

Given the block formation period T_{blk} of the server, the time for consensus completion, T_{con} is calculated as follow:

$$T_{\text{con}} = T_{\text{blk}} R.$$

Table 2 shows the numerical results of the expected T_{con} in hours for various N and $p_H \in \{0.51, 0.75, 0.9\}$, where $T_{\text{blk}} = 13$ seconds. As illustrated, given a fixed N, $\mathbb{E}[T_{\text{con}}]$ is affected only by p_H. Since T_{con} is observable from every consensus session, while p_H is not, one can estimate p_H within a network by comparing the actual T_{con} with $\mathbb{E}[T_{\text{con}}]$.

$\log_2 N$	$\min(T_{\text{con}})$	$\mathbb{E}[T_{\text{con}}]$ in hours when			$\max(T_{\text{con}})$
		$p_H = 0.51$	$p_H = 0.75$	$p_H = 0.9$	
0	0.00	0.00	0.00	0.00	0.01
1	0.01	0.01	0.00	0.00	0.02
2	0.02	0.02	0.01	0.01	0.03
3	0.03	0.05	0.04	0.02	0.06
4	0.06	0.10	0.08	0.05	0.12
5	0.12	0.21	0.15	0.12	0.23
6	0.23	0.43	0.31	0.26	0.46
7	0.46	0.88	0.62	0.51	0.92
8	0.92	1.78	1.23	1.03	1.85
9	1.85	3.59	2.47	2.05	3.70
10	3.70	7.22	4.93	4.11	7.40
11	7.40	14.48	9.86	8.22	14.79
12	14.79	28.99	19.72	16.43	29.58
13	29.58	58.00	39.44	32.87	59.16
14	59.16	116.01	78.89	65.74	118.33
15	118.33	232.02	157.77	131.48	236.66

Table 2: Expected consensus time T_{con} for different N and p_H values.

4.2.3 Attack Success Probability

The forged wire map supported by the attackers can be selected as the final consensus if the random walks first reach $H_r < N$ and $A_r \geq N$ in r movements. Since r can vary in $[N..2N-1]$, the probability $\Pr[A_{\text{win}}]$ can be obtained by:

$$\Pr[A_{\text{win}}] = \sum_{r=N}^{2N-1} \binom{r-1}{N-1} p_A^N (1-p_A)^{r-N}.$$

Table 3 presents numerical results of $\Pr[A_{\text{win}}]$ for various N and $p_H \in \{0.51, 0.75, 0.9\}$. In this table, as N increases, the protocol becomes more robust against attackers. If we can estimate p_H in a network in advance, this table can help the consensus manager determine the minimum N required for a desired security level.

4.3 Choosing a Good D_c

To summarize our analysis, a lower D_c increases the robustness of the protocol's against collusive attackers but also makes reaching a consensus more time-consuming and costly. Thus, the goal of the consensus manager is to find a suitable D_c that minimizes cost and time while ensuring a desired level of security. However, since p_H is not observable, the theoretically optimal D_c may not be practical.

$\log_2 N$	$\Pr[A_{\text{win}}]$ when			$\min(p_H)$ for	
	$p_H = 0.51$	$p_H = 0.75$	$p_H = 0.9$	$\Pr[A_{\text{win}}] < 10^{-2}$	$\Pr[A_{\text{win}}] < 10^{-3}$
0	0.4900	0.2500	0.1000	0.990	0.999
1	0.4850	0.1562	0.0280	0.942	0.982
2	0.4781	0.0706	0.0027	0.858	0.924
3	0.4686	0.0173	0.0000	0.772	0.839
4	0.4553	0.0013	0.0000	0.669	0.756
5	0.4367	0.0000	0.0000	0.643	0.687
6	0.4107	0.0000	0.0000	0.602	0.635
7	0.3746	0.0000	0.0000	0.573	0.596
8	0.3255	0.0000	0.0000	0.552	0.569
9	0.2611	0.0000	0.0000	0.537	0.549
10	0.1827	0.0000	0.0000	0.526	0.535
11	0.1003	0.0000	0.0000	0.519	0.525
12	0.0351	0.0000	0.0000	0.513	0.518
13	0.0052	0.0000	0.0000	0.510	0.513
14	0.0001	0.0000	0.0000	0.507	0.509
15	0.0000	0.0000	0.0000	0.505	0.507

Table 3: Attack success probabilities and minimum p_H values for different thresholds.

We can estimate p_H based on $\mathbb{E}[T_{\text{con}}]$ and its Table 2. Although the real-world p_H may be heterogeneous (i.e., p_H can change over time), we assume that p_H changes slowly. A traditional estimate of p_H can be developed using the distribution of T_{con}. For example, letting $\mathbb{E}[T_{\text{con}}(p_H)]$ denote the expected consensus time as a function of p_H, and t_{con} denote an observation of T_{con}, a simple (though imperfect) estimate $\hat{p_H}$ of p_H can be made by:

$$\hat{p_H} := \arg\min_{p_H} |t_{\text{con}} - \mathbb{E}[T_{\text{con}}(p_H)]|.$$

Better estimates, such as the Bayes estimator, can be developed based on higher moments of T_{con}, but they are beyond the scope of this paper.

Given an estimate $\hat{p_H}$, based on $\Pr[A_{\text{win}}]$ and its Table 3, the consensus manager can decide on the maximum D_c that guarantees the attack success probability does not exceed a desired threshold. For example, the table shows that the attack success probability is guaranteed to be below 10^{-3} for N not less than 2^4, given that $p_H \geq 0.76$. The consensus manager can repeat this observation-estimation-decision process for every consensus session.

5 Conclusion

In this paper, we have proposed a consensus protocol for SNARK pre-processed circuit polynomials, referred to as wire maps. Pre-processing the wire maps is essential for some SNARKs, but it is costly. Our protocol is particularly useful for securely delivering wire maps from untrusted network participants to SNARK verifiers. We developed a framework to resolve conflicts over wire maps among the participants by representing the wire maps within a DAG structure and allowing the participants to collaboratively manage this DAG. Our design aimed to enhance efficiency in distributed networks by enabling verifiable consensus with constant verification complexity.

Through our analysis, we highlighted the critical role of parameters D_c and p_H in determining the protocol's security and efficiency. Our findings suggest that while lower values of D_c improve security by mitigating attack success probabilities, they also increase the time and resource expenditure needed to reach consensus. Consequently, we provided insights into selecting suitable values for D_c that satisfy both performance and security requirements, given dynamic nature of p_H.

The results of this study contribute to the broader discourse on secure consensus mechanisms in cryptographic systems, with practical implications for blockchain applications and other decentralized networks. Future research could further refine the model by integrating real-time parameter adjustments and exploring advanced statistical estimators for p_H to enhance protocol responsiveness and adaptability to changing network dynamics.

References

[1] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016*, pages 305–326, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[2] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Paper 2019/953, 2019.

[3] Jehyuk Jang and Jamie Judd. An efficient SNARK for field-programmable and RAM circuits. Cryptology ePrint Archive, Paper 2024/507, 2024.

[4] Sean Bowe, Ariel Gabizon, and Ian Miers. Scalable multi-party computation for zk-SNARK parameters in the random beacon model. Cryptology ePrint Archive, Paper 2017/1050, 2017.

[5] Markulf Kohlweiss, Mary Maller, Janno Siim, and Mikhail Volkhov. Snarky ceremonies. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2021*, pages 98–127, Cham, 2021. Springer International Publishing.

[6] Episode 133: Trusted setup ceremonies explored. `https://zeroknowledge.fm/133-2`. Accessed: 2024-10-11.

[7] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *Advances in Cryptology - ASIACRYPT 2010*, pages 177–194, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[8] Joachim von zur Gathen and Jürgen Gerhard. *Fast polynomial evaluation and interpolation*, page 295–312. Cambridge University Press, 2013.

[9] Alin Tomescu, Robert Chen, Yiming Zheng, Ittai Abraham, Benny Pinkas, Guy Golan Gueta, and Srinivas Devadas. Towards scalable threshold cryptosystems. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 877–893, 2020.

Appendix

A Protocol initiation

The consensus manager initiates each consensus session by running the algorithm InitMem, which takes as input a pair (ϕ, π) representing SNARK instance and proof and consensus parameters D_p

and D_c. As shown in Algorithm 2, there is no data processing performed by this algorithm. Instead, its role is to publish the inputs to the server and pre-allocate slots for DAG edges that will be submitted by contributors during the session.

B Example: Contributor selection rule based on PoS

We provide an example algorithm for selecting contributors from the public key pool $\mathcal{A}$ of participants. Assume that public keys in the pool $\mathcal{A}$ have been sorted and mapped to equally spaced indices during the protocol initiation. In other words, the pool can be written as $\mathcal{A} = ((x_0, a_0), (x_1, a_1), \cdots, (x_{N-1}, a_{N-1}))$, where $x_i = i\Delta$ for all i. Then, given a hash of the previous block $h \in \mathbb{F}$, the participant holding the public key a_k that satisfies the following condition is selected as the contributor:

$$(x_k, a_k) \in \mathcal{A} \wedge |h - x_k| \leq 0.5\Delta.$$

With this rule, we present two algorithms Select and IsSelected. Define a polynomial $t_{\mathcal{A}} \in \mathbb{F}[X]$ such that $t_{\mathcal{A}}(x_i) = \text{Hash}(a_i)$ for all i. Assume that $\mathcal{A}$ is public, and $t_{\mathcal{A}} = [t_{\mathcal{A}}(\tau)]_1$ and Δ are given. The two algorithms are provided in Algorithm 3 and Algorithm 4, respectively.

C Vote generation

We provide guidance on how a contributor computes a DAG contribution and forms a vote. However, our guidance cannot compel the actions of participants, and we are also open to the possibilities that malicious participants or those prioritizing computational efficiency may not adhere to our guides.

Consider a participant possessing a polynomial $f(X)$ of degree D_p and a private key s, with the corresponding public key $a = [s]_2$. Suppose the participant has access to $\mathcal{A}$ and the latest $\mathcal{E}$. If the participant is a contributor (i.e., the participant can generate a witness w_a through Select that, along with the public key a, can pass IsSelected), they may use a traditional efficient algorithm named DepthFirstSearch to collect all DAG branches from $\mathcal{E}$.

Conjecture (Branch selection criteria)**.** When choosing a DAG branch to contribute to, a rational contributor possessing $f(X)$ may choose the longest D_p-incomplete branch that can be reproduced by $f(X)$.

A pseudocode for vote generation is illustrated in Algorithm 5.

Algorithm 1 UpdateDAG

procedure UPDATEDAG($\mathbf{o}; h, \mathcal{M}, \mathcal{E}(w_0)$)
  parse $\mathcal{M} = (\mathcal{A}, \phi, \pi, D_p, D_c)$
  assert $\mathcal{A} \neq \emptyset$
  assert $\mathcal{E} \neq \perp$
  assert IsSelected$(\mathcal{A}, h, a, w_a) = 1$ ▷ Validate the contributor eligibility
  parse $\mathbf{o} = (\mathbf{e}, b, \tilde{t}, \pi_\lambda, t_\zeta, a, w_a, C)$ ▷ Check the format of the vote
  assert $b \in \{0, 1\}$
  parse $\mathbf{e} = (w_0, v_1)$
  if $w_0 = \perp$ **then**
    $t_0 \leftarrow [1]_1$
    $p_0 \leftarrow [0]_1$
  else
    $(v_{-1}, v_0) \leftarrow \mathcal{E}(w_0)$ ▷ Retrieve the previous vertex from the server
    parse $v_0 = (x^{(0)}, c_0, t_0, p_0)$
  end if
  parse $v_1 = (x^{(1)}, c_1, t_1, p_1)$
  assert $x^{(1)} = \text{Hash}(a)$
  $\zeta \leftarrow \text{Hash}(c_1, t_0, t_1, p_0, p_1, \tilde{t})$ ▷ Toss the first coin
  $\tilde{t}_\zeta \leftarrow 1$
  for $i = 0 \rightarrow D_c - 1$ **do** ▷ Reproducing $\tilde{t}_\zeta$
    $x_i \leftarrow x^{(1)} + i$
    $\tilde{t}_\zeta \leftarrow \tilde{t}_\zeta(\zeta - x_i)$
  end for
  $\lambda \leftarrow \text{Hash}(\tilde{t}_\zeta, t_\zeta)$ ▷ Toss the second and last coins
  $\gamma \leftarrow \text{Hash}(\text{LHS}_\lambda, \pi_\lambda, C, p_1)$
  $\text{LHS}_\lambda \leftarrow \tilde{t} + \lambda t_0 + \lambda^2 t_1 + \lambda^3(p_1 - p_0 - t_\zeta c_1) - (\tilde{t}_\zeta + \lambda t_\zeta + \lambda^2 \tilde{t}_\zeta t_\zeta)[1]_1$
  $E_0 \leftarrow e(\text{LHS}_\lambda + \gamma C, [1]_2)$
  $E_1 \leftarrow e(\pi_\lambda, [\tau]_2 - \zeta[1]_2)$
  $E_2 \leftarrow e\left(-\lambda^2 b[1]_1, [\tau^{D_p}]_2\right)$
  $E_3 \leftarrow e\left(\gamma p_1, a\right)$
  assert $E_0 = E_1 E_2 E_3$ ▷ Verify the pairing equations
  if $b = 1$ **then** ▷ Return the validated edge to the server
    assert $\text{Verify}_{\text{SNARK}}(\phi, \pi, p_1) = 1$
  end if
  $\mathcal{E} \leftarrow \mathcal{E} \cup \{\mathbf{e}\}$
  return $\mathcal{E}$
end procedure

Algorithm 2 InitDAG

procedure INITDAG($\mathcal{A}, \phi, \pi, D_p, D_c$)
  assert $\mathcal{V} = \perp$
  $\mathcal{M} \leftarrow (\mathcal{A}, \phi, \pi, D_p, D_c)$
  $\mathcal{E} \leftarrow \emptyset$
  return $(\mathcal{M}, \mathcal{E})$
end procedure

Algorithm 3 Select

procedure SELECT($\mathcal{A}, h, \Delta, x, a$)
  assert $(x, a) \in \mathcal{A}$
  assert $|h - x| \leq \lfloor 0.5\Delta \rfloor$
  $t_{\mathcal{A}}(X) \leftarrow 0$
  for $i = 0$ to $|\mathcal{A}| - 1$ **do**
    Define $L_i(X)$ such that $L_i(i\Delta) = 1$ and $L_i(k\Delta) = 0$ for all $k \not\equiv i \mod |\mathcal{A}|$
  end for
  for $m \in \mathcal{A}$ **do**
    parse $m = (\tilde{x}, \tilde{a})$
    $i \leftarrow \tilde{x}/\Delta$
    $t_{\mathcal{A}}(X) \leftarrow t_{\mathcal{A}}(X) + \text{Hash}(\tilde{a})L_i(X)$
  end for
  $\pi_a(X) \leftarrow (t_{\mathcal{A}}(X) - \text{Hash}(a))/(X - x)$
  $\pi_a \leftarrow [\pi_a(\tau)]_1$
  return π_a
end procedure

Algorithm 4 IsSelected

procedure ISSELECTED($t_{\mathcal{A}}, h, \Delta, a, x, \pi_a$)
  assert $x \in \mathbb{F}, \pi \in \mathbb{G}_1$
  assert $|h - x| \leq \lfloor 0.5\Delta \rfloor$
  assert $e(t_{\mathcal{A}} - \text{Hash}(a)[1]_1, [1]_2) = e(\pi, [\tau]_2 - x[1]_2)$
  return True
end procedure

Algorithm 5 VoteGen

procedure VoteGen($f(X), s, \mathcal{M}, \mathcal{E}$)
  parse $\mathcal{M} = (\mathcal{A}, \phi, \pi, D_p, D_c)$
  assert $\deg(f(X)) = D_p$
  assert $\text{Verify}_{SNARK}(\phi, \pi, [f(\tau)]_1) = 1$
  $(\mathcal{X}, t_{\text{pre}}(X), p_{\text{pre}}(X), v_{\text{pre}}) \leftarrow \text{Choose}(f(X), \mathcal{E}, D_p, D_c, \phi, \pi)$
  $x \leftarrow \text{Hash}(a)$
  $\mathcal{X}_a \leftarrow \emptyset$
  $\tilde{t}_a(X) \leftarrow 1$
  for $k = 0$ to $D_c - 1$ **do**
    $\mathcal{X}_a \leftarrow \mathcal{X}_a \cup \{x + k\}$
    $\tilde{t}_a(X) \leftarrow \tilde{t}_a(X) \cdot (X - (x + k))$
  end for
  $p(X) \leftarrow \text{Interpolate}(f(X), \mathcal{X} \cup \mathcal{X}_a)$
  $c(X) \leftarrow (p(X) - p_{\text{pre}}(X)) / t_{\text{pre}}(X)$
  $t(X) \leftarrow t_{\text{pre}}(X) \cdot \tilde{t}_a(X)$
  if $\deg(t) < D_p$ **then**
    $b \leftarrow 0$
    $t \leftarrow [t(\tau)]_1$
  else
    $b \leftarrow 1$
    $t \leftarrow [t(X) - X^{D_p}|_{X=\tau}]_1$
  end if
  Package vote: $\mathbf{o} = (\mathbf{e}, b, \tilde{t}, \pi_\lambda, t_\zeta, a, w_a, C)$
  return o
end procedure

Algorithm 6 Choose

procedure Choose($f(X), \mathcal{E}, D_p, D_c$)
  $\mathcal{X}_{\text{long}} \leftarrow \emptyset$
  $t_{\text{long}}(X) \leftarrow 1$
  $p_{\text{long}}(X) \leftarrow 0$
  if $\mathcal{E} \neq \emptyset$ **then**
    $N \leftarrow D_p / D_c$
    $\mathcal{B} \leftarrow \text{DepthFirstSearch}(\mathcal{E})$
    Find the longest reproducible branch
  end if
  return $(\mathcal{X}_{\text{long}}, t_{\text{long}}(X), p_{\text{long}}(X), v_{\text{long}})$
end procedure

Algorithm 7 Reproduce

procedure Reproduce($\mathbf{b}, f(X), D_p, D_c$)
  Parse branch $\mathbf{b} = (v_0, v_1, \cdots, v_{l-1})$
  Cut off non-reproducible vertices and return $(l, v_{l-1}, \mathcal{X}, t(X), p(X))$
end procedure

Algorithm 8 Interpolate

1: **procedure** INTERPOLATE($f(X), \mathcal{X}$)
2: $l \leftarrow |\mathcal{X}|$
3: **for** $x \in \mathcal{X}$ **do**
4: Compute interpolation using Lagrange formula
5: **end for**
6: **return** $\tilde{f}_{\mathcal{X}}(X)$
7: **end procedure**