Distributed consensus networks

Peter P. Rohde[1, 2, 3]
[1] *BTQ Technologies, 16-104 555 Burrard Street, Vancouver BC, V7X 1M8 Canada*
[2] *Center for Engineered Quantum Systems, School of Mathematical & Physical Sciences, Macquarie University, NSW 2109, Australia*
[3] *Hearne Institute for Theoretical Physics, Department of Physics & Astronomy, Louisiana State University, Baton Rouge LA, United States*[*]

Blockchains rely on distributed consensus algorithms to decide whether a proposed transaction is valid and should be added to the blockchain. The purpose of consensus is to act as an independent arbiter for transactions, robust against adversarial manipulation. This can be achieved by choosing random subsets of nodes to form consensus sets. In an economy where consensus is the commodity, consensus must be secure, computationally efficient, fast and cheap. Most current blockchains operate in the context of open networks, where algorithms such as proof-of-work are highly inefficient and resource-intensive, presenting long-term scalability issues. Inefficient solutions to allocating consensus sets equates to high transaction costs and slow transaction times. Closed networks of known nodes afford more efficient and robust solutions. We describe a secure distributed algorithm for solving the random subset problem in networks of known nodes, bidding to participate in consensus for which they are rewarded, where the randomness of set allocation cannot be compromised unless all nodes collude. Unlike proof-of-work, the algorithm allocates all nodes to consensus sets, ensuring full resource utilisation, and is highly efficient. While the protocol is synchronous, a staking mechanism ensures self-synchronisation with no dependence on an external reference. The protocol follows self-enforcing where adversarial behaviour only results in self-exclusion. Signature-sets produced by the protocol act as timestamped, cryptographic proofs-of-consensus, a commodity with market-determined value. The protocol is highly strategically robust against collusive adversaries, affording subnetworks defence against denial-of-service and majority takeover.

CONTENTS

I. Introduction 1

II. Consensus 2

III. Random subset problem 3
A. Centralised algorithm 3
B. Proof-of-work 4
C. Secure shared randomness 5
D. Secure random subsets 5

IV. Consensus assignment problem 5
A. Consensus assignment graphs 5
B. Consensus group 6
C. Uniform consensus sampling 7
1. Fisher-Yates shuffle 7
2. Generalised Fisher-Yates shuffle for finite groups 7
3. Uniformly sampling the consensus group 8
4. Sampling via random bitstreams 8
5. Initial assignment 9

V. Distributed consensus networks 9
A. Model 9
B. Protocol 9
C. Consensus time 10
D. Proof-of-consensus 10
E. Network policy 11

VI. Quantum consensus networks 11
A. Entropy addition 11
B. Quantum random oracles 12
C. Quantum key distribution (QKD) 12
1. Consensus protocol 12
D. Interactive proofs of quantumness 13
1. Trapdoor claw-free (TCF) functions 13
2. The LWE problem 13
3. Interactive proof protocol 14
4. Consensus protocol 14

VII. Economics 14

VIII. Applications 15
A. Protocols 15
B. Ledgers 15
C. Blockchains 15
D. Distributed computing 16
E. Distributed signature authorities 16

IX. Strategic considerations 16

X. Conclusion 17

References 17

[*] dr.rohde@gmail.com

I. INTRODUCTION

* DeFi
* In a decentralised environment transactions are approved via *consensus* (Sec. II) whereby a small number of randomly chosen network nodes (*consensus sets*) collectively act as delegates for the network to authorise transactions. Consensus outcomes should with high probability reflect honest outcomes, a function of consensus set size and the ratio of dishonest nodes.

The randomisation of nodes forming consensus is vital to ensure the integrity of consensus outcomes (Sec. III). If the choice of consensus nodes were known in advance this would provide avenues for malicious parties to compromise security by targeting the known allocation. Randomisation effectively erases all assignment information of consensus node assignment, eliminating all avenues for strategic alignment.

The Bitcoin protocol first introduced proof-of-work (PoW) as mechanism for securely randomly selecting a small number of network nodes at random to form consensus (Sec. III.B). This protocol operates in the context of a network of unknown and unidentifiable nodes in which anyone may freely participate. The algorithm requires nodes to compete to solve inverse-hashing problems, a randomised algorithm that effectively chooses winners by lottery.

While the proof-of-work mining algorithm provides an ingenious solution to the random subset problem it is highly inefficient, resulting in enormous net energy consumption to maintain the network. Currently, the Bitcoin protocol consumes on the order of $\sim$150TWh per year (Cambridge Centre for Alternative Finance, 2023), comparable to the energy consumption of a medium sized country.

The inefficiency of proof-of-work presents a significant obstacle for scalability, motivating the development of more efficient alternate consensus algorithms. Proof-of-stake (PoS) (Bentov *et al.*, 2017; King and Nadal, 2012) is a leading alternative, recently adopted by the Ethereum network to improve scalability and reduce transaction costs. Ethereum's transition to proof-of-stake, famously known as *the merge*, reduced its annualised energy consumption from $\sim$21TWh (PoW) to $\sim$2GWh (PoS) (Cambridge Centre for Alternative Finance, 2023). However, despite its radically improved energy efficiency, proof-of-stake has been criticised for affording reduced security owing to its increased vulnerability to manipulation and reduced level of randomisation. Proof-of-work based on quantum sampling problems has also been investigated as a means for enhanced energy efficiency (Singh *et al.*, 2023).

* Sec. III.C

* Sec. III.D

The generalisation of the random subset problem is the *consensus assignment problem* (Sec. IV), enabling the simultaneous allocation of network nodes to an arbitrary set of consensus sets under asymmetric load.

Distributed consensus networks (DCNs) (Sec. V) combine the consensus assignment problem with an economic model that self-incentivises honest participation of nodes.

* Setting

* DCN's treat consensus as an abstract market commodity, an enabler of digital trade.

* Proof-of-consensus (Sec. V.D)

* Major existing blockchains tie consensus to their respective cryptocurrencies (or *coins*), resulting in the execution of trade via consensus having monetary effects.

* Detached from specific

Quantum consensus networks (QCNs) (Sec. VI) are quantum-enabled, possessing quantum computational or communications resources, acting as secure, certifiable quantum random number generators (QRNGs) upon forming consensus, a valuable commodity with many applications. Other (classical) DCNs may observe QCNs as oracles, utilising their random bitstreams to add entropy (Sec. VI.A) to their own networks, enabling quantum-random consensus assignment.

Sec. VII

Sec. VIII

Sec. IX

Hashcash: (Back, 2002; Dwork and Naor, 1993)

(Schneier, 1996)

II. CONSENSUS

The purpose of consensus is to provide a decentralised mechanism for sets of parties to collectively act as an independent arbiter in judging and signing off on the validity of statements. In the context of blockchains, this is employed to determine whether a newly submitted transaction block is legitimate and should be added to the blockchain.

In an environment where a minority subset of parties are dishonest and conspiring to form false consensus, the purpose of consensus protocols is to uphold the will of the majority, independent of the behaviour of collusive, dishonest parties. As dishonest parties are assumed to be in the minority, this can always be achieved by ensuring that all parties are involved in consensus. However, this is highly resource-intensive and consensus needn't involve all parties. A subset of parties suffices to form consensus if their decision reflects the will of the majority, enabling them to act as delegates.

Choosing a subset of parties to form consensus there is some probability of dishonest parties forming a false majority by chance. Consensus sets should therefore be chosen to upper-bound this probability, independent of the strategy employed by dishonest parties.

Consensus is formed by majority vote on the validity of some statement ($\texttt{id}$) by members of a consensus set ($\mathcal{C}$) chosen from a pool of consensus nodes in a network ($\mathcal{N}$),

$$\mathcal{C} \subseteq \mathcal{N}. \tag{2.1}$$

The majority vote of the entire network is a deterministic decision function defining the *source of truth*,

$$\textsc{MajorityVote}(\mathcal{N}, \texttt{id}) \to \{0,1\}, \tag{2.2}$$

which consensus sets must uphold. Based on their statistical composition the majority votes of consensus sets are in general probabilistic,

$$\textsc{MajorityVote}(\mathcal{C},\texttt{id}) = \begin{cases} \Pr(1-P_M), & \textsc{MajorityVote}(\mathcal{N},\texttt{id}) \\ \Pr(P_M), & \neg\textsc{MajorityVote}(\mathcal{N},\texttt{id}) \end{cases},$$

where P_M is the probability that $\mathcal{C}$ comprises a majority of dishonest parties attempting to subvert majority rule.

For $P_M = 0$ where an honest majority is guaranteed, this reduces to the deterministic case as per Eq. (2.2). The purpose of distributed consensus algorithms is to choose consensus sets $\mathcal{C}$ operating in this regime to a close approximation.

III. RANDOM SUBSET PROBLEM

The *random subset problem* is to choose a random subset A uniformly from B, where $A \subseteq B$, of which there are,

$$\binom{|B|}{|A|}, \tag{3.1}$$

combinations. If the proportion of parties acting dishonestly and collusively is r, the probability a random subset of size N contains a majority of dishonest players is,

$$\begin{aligned} P_M(N,r) &= \sum_{n=\lfloor N/2\rfloor+1}^{N} \binom{N}{n} r^n (1-r)^{N-n} \\ &= 1 - I_{1-r}(\lfloor N/2\rfloor+1, \lfloor N/2\rfloor+2) \end{aligned} \tag{3.2}$$

where $I_p(a,b)$ is the regularised incomplete beta function. While in principle all elements are associated with their own r_i, choosing subsets uniformly at random ensures the probabilities of set members being dishonest are independently and identically sampled with the average-case probability r. Hence, P_M is independent of how dishonest nodes are distributed and what strategies they might employ (Fig. 1).

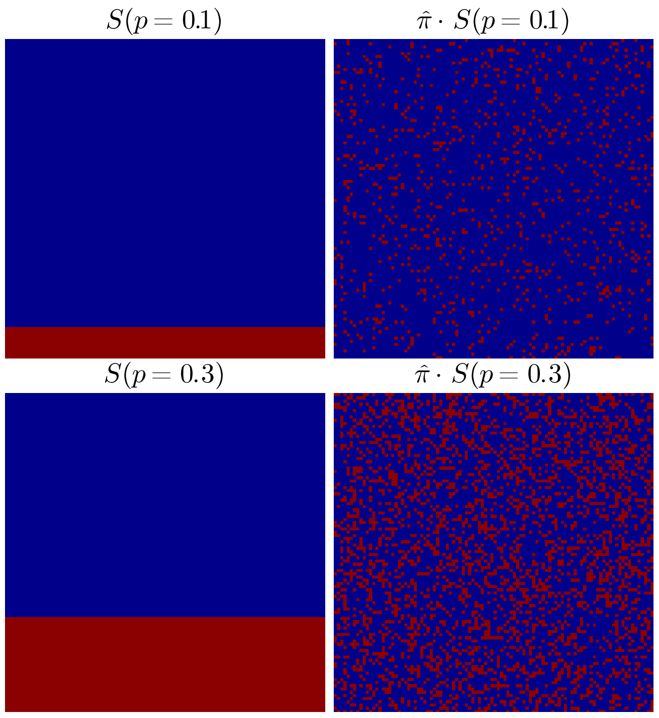

Figure 1: Random sampling as a defence against strategic alignment. (left) A minority of strategically aligned nodes may form false-majority if their alignment overlaps with a known allocation for consensus sets. (right) Random sampling erases strategic alignment and ensures node honesty is sampled with average-case r.

To uphold the integrity of majority votes, we require the likelihood of a minority gaining false-majority by chance to be upper-bounded by an exponentially small threshold $\varepsilon = 2^{-\lambda}$,

$$P_M(N,r) \leq \varepsilon, \tag{3.3}$$

for statistical security parameter $\lambda > 1$. Here, ε is a subjective security parameter which may be application- or user-dependent.

We define the required consensus set size, N_C, as the smallest set size satisfying this inequality,

$$N_C(r,\varepsilon) = \underset{N\in\mathbb{Z}^+}{\arg\min}(P_M(N,r) \leq \varepsilon). \tag{3.4}$$

For consensus to maintain ε-security, consensus sets should never be smaller than N_C. Simultaneously, choosing consensus sets larger than N_C is unnecessary and wasteful. Tradeoff curves between N_C, r and ε are shown in Fig. 2.

A. Centralised algorithm

The random subset problem has a trivial centralised solution (Alg. 1). We assign unique random bit-strings to each node, order them by their random number, and partition the ordered list piecewise into units of length N_C. These partitions form non-intersecting random subsets, each defining an independent consensus set.

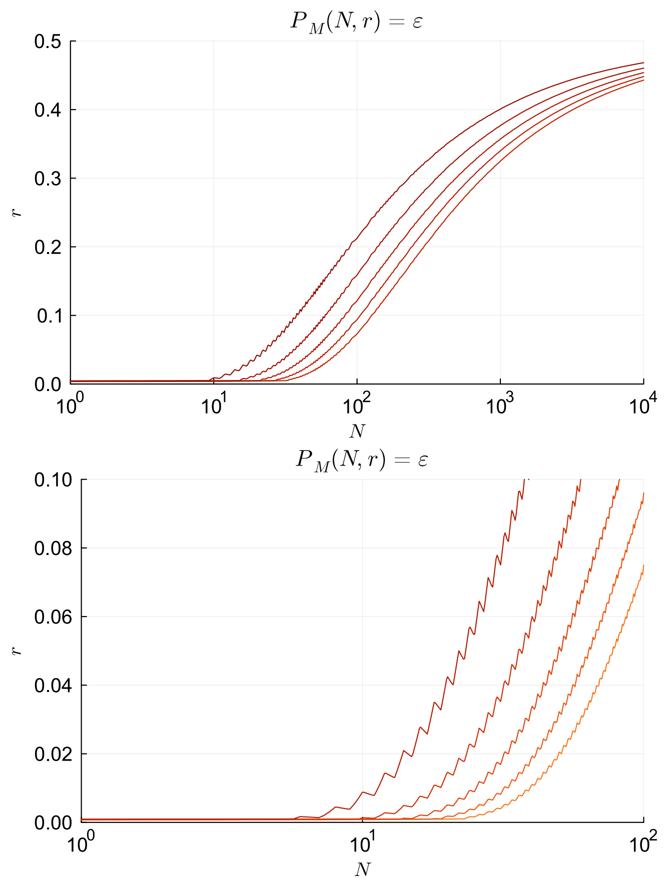

Figure 2: Security tradeoffs with consensus set size. $P_M(N,r) \leq \varepsilon$ is the probability that dishonest nodes within a random consensus set of size N form a false majority when the proportion of dishonest nodes is r, where ε is the security parameter.

Algorithm 1: Centralised algorithm for the random subset problem, assigning a set of nodes $\mathcal{S}$ to a set of independent random subsets $\{\mathcal{C}\}$ of given sizes $\{|\mathcal{C}|\}$.

```
procedure RandomSubsets($\mathcal{S}$, $\{|\mathcal{C}|\}$) $\to \{\mathcal{C}\}$
    ▷ Assign a random number to each node ◁
    for $i \in \mathcal{S}$ do
        random$_i$ $\leftarrow$ Random($\{0,1\}^n$)
    ▷ Sort nodes by their random number ◁
    ordered = Sort($\mathcal{S}$, random)
    ▷ Partition the list into consensus sets ◁
    $\{\mathcal{C}\}$ = Partition(ordered, $\{|\mathcal{C}|\}$)
    return $\{\mathcal{C}\}$
```

The challenge is to securely implement this algorithm in a decentralised environment, such that nodes are unable to compromise the randomness of the assignments of consensus sets.

B. Proof-of-work

Proof-of-work has been widely employed in blockchains such as Bitcoin (Nakamoto, 2008) as a distributed protocol for choosing random subsets of nodes. Here, nodes compete to find satisfying inputs to hash functions whose outputs satisfy a constraint dictating the likelihood of success. This distributed algorithm effectively asks nodes to find solutions to randomised problems with very low probability of success, $p_{\text{mine}} \ll 1$, such that winners are randomly allocated across nodes in the network.

Since hash functions are pseudo-random and exhibit pre-image resistance[1], the only viable approach to finding such solutions is via brute-force, repeatedly hashing random bit-strings until a satisfying input is found. Winning nodes are hence allocated at random and cannot be spoofed under a computational hardness assumption. The distributed algorithm for proof-of-work is shown in Alg. 2.

Algorithm 2: Random subsets via proof-of-work. Nodes $\mathcal{S}$ hash random bitstrings, salted by a problem instance specified by the previous block `id`, where the per-hash success rate is p_{mine}.

```
procedure ProofOfWork($\mathcal{S}$, id, size) $\to \mathcal{C}$
    $\mathcal{C} = \{\}$                              ▷ Consensus set
    while $|\mathcal{C}| <$ size do
        for $i \in \mathcal{N}$ do                    ▷ In parallel
            random$_i$ = Random($\{0,1\}^n$)
            output$_i$ = Hash(id|random$_i$)
            if Valid(output$_i$) then
                $\mathcal{C} \leftarrow \mathcal{C} \cup \mathcal{S}_i$
    return $\mathcal{C}$

procedure Valid($x \in \{0,1\}^n$) $\to \{0,1\}$
    if $x/2^n \leq p_{\text{mine}}$ then
        return true
    else
        return false
```

Proof-of-work has no requirement that nodes be known or trusted, providing a very general protocol suited to open networks in which anyone can participate. However, while affording a distributed solution to the random subset problem, this approach is inherently wasteful. The expected mining rate is,

$$R_{\text{mine}} = p_{\text{mine}} \cdot R_{\text{hash}}, \tag{3.5}$$

where R_{hash} is the network's net hash-rate[2]. To inhibit the formation of multiple, simultaneous consensus sets (double-mining), potentially manifesting itself as forks in the blockchain, proof-of-work systems may introduce

[1] For Hash$(x) \to y$ it is computationally hard to find a satisfying input $x \in \{0,1\}^*$ for a given output $y \in \{0,1\}^n$.

[2] The Bitcoin network's cumulative network hash-rate is $\sim$500EH/s (500×10^{18}H/s).

friction by algorithmically adjusting the *difficulty parameter* to ensure consistent mining times. To achieve constant mining time, difficulty must scale with cumulative network hash-rate, making the distributed algorithm less efficient as network size grows. For example, maintaining constant mining rate implies efficiency scales inversely with network size,

$$p_{\text{waste}} = 1 - O\left(\frac{1}{n}\right), \tag{3.6}$$

asymptotically perfectly inefficient.

C. Secure shared randomness

An environment comprising a network of known nodes affords a more efficient solution to the random subset problem. The key observation is to utilise *secure shared randomness* to randomly permute and partition sets of nodes as per Alg. 1. The source of shared randomness should be secure, in the sense that its randomness be robust against manipulation by dishonest nodes.

Can only change to a different random

Secure shared randomness can be achieved by first requiring network nodes ($\mathcal{N}$) to present transaction `ids` whose hashes act as unique random numbers,

$$X_i = \textsc{Hash}(\texttt{id}_i). \tag{3.7}$$

A global key is then defined as,

$$X_{\mathcal{N}} = \bigoplus_{i \in \mathcal{N}} X_i, \tag{3.8}$$

our shared random source, where $\oplus$ denotes the bitwise XOR operation (addition modulo 2).

* Comment on poly(n) attempts at epsilon leaves it exponentially small.

$$\varepsilon' \leq \frac{\text{poly}(\lambda)}{2^\lambda}. \tag{3.9}$$

D. Secure random subsets

From a secure global key, $X_{\mathcal{N}}$, as per Eq. (3.8) the random subset problem affords a simple solution. Individual keys are assigned to each node,

$$K_i = \textsc{Hash}(X_i | X_{\mathcal{N}}). \tag{3.10}$$

Ordering and partitioning nodes by their associated $\{K_i\}$ assigns them to consensus sets as per Alg. 1. Importantly, the randomness of the global key, $X_{\mathcal{N}}$, cannot be compromised unless all X_i are controlled.

Multiple random subset allocations may be derived from a single global key by rehashing individual keys,

$$K_i^{(n)} = \textsc{Hash}^n(X_i | X_{\mathcal{N}}), \tag{3.11}$$

where $K_i^{(n)}$ denote keys for the nth round.

This approach necessarily assumes the network comprises a known set of nodes, as the global key $X_{\mathcal{N}}$ cannot be established from an undefined set, nor can an undefined set be ordered.

IV. CONSENSUS ASSIGNMENT PROBLEM

The random subset problem randomly partitions network space into consensus sets, imposing a subset-sum constraint on consensus set sizes.

In the context of symmetric load where all nodes request $|\mathcal{C}|$ consensus load the net consensus load is,

$$L = |\mathcal{C}| \cdot |\mathcal{N}|. \tag{4.1}$$

This requires performing $|\mathcal{C}|$ independent re-partitionings within each of which every node will be assigned exactly once. Under this allocation every node both contributes and requests $|\mathcal{C}|$ units of work.

The *consensus assignment problem* generalises the random subset problem to accommodate arbitrary consensus set sizes and asymmetric load, allowing nodes to contribute and request arbitrary load, a prerequisite for enabling a free consensus market.

Representing consensus assignment via *consensus assignment graphs* (Sec. IV.A reflecting the contributed and requested load by nodes and consensus sets, the goal of the algorithm is to uniformly sample from the space of satisfying assignments (Sec. IV.C), achieved by defining consensus assignment via its algebraic group structure (Sec. IV.B) and uniformly sampling group elements.

A. Consensus assignment graphs

Consider a directed bipartite graph (Fig. 3),

$$G = (U, V, E), \tag{4.2}$$

where vertices $u \in U$ and $v \in V$ represent network nodes ($\mathcal{N}$) and consensus sets respectively ($\{\mathcal{C}\}$), and the directed edge set E defines the assignment of nodes to consensus sets where each edge is associated with a unit of consensus work. The constraint that nodes may be assigned at most once to a given consensus set imposes that G is a simple graph.

Consensus load may be arbitrarily partitioned and nodes may make multiple requests for consensus sets, hence in general $|\mathcal{S}| \neq |\{\mathcal{C}\}|$. Expressing vertex sets as degree sequences,

$$\begin{aligned} U &= (\deg^+(u_1), \dots, \deg^+(u_{|U|})), \\ V &= (\deg^-(v_1), \dots, \deg^- v_{|V|})), \end{aligned} \tag{4.3}$$

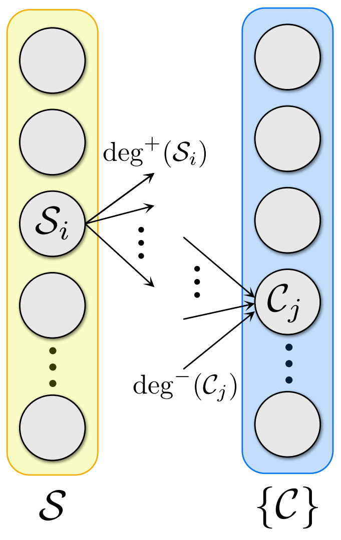

Figure 3: Generalised consensus assignment problem. Consensus assignment may be represented as a directed bipartite graph from a space of network nodes ($\mathcal{N}$) to consensus sets ($\{\mathcal{C}\}$), where vertex degree represents contributed/consumed consensus load. The set of all satisfying edge-sets E represents the space of valid assignments.

where $\deg^+(u)$ denotes work contributed by node u and $\deg^-(v)$ the load requested by consensus set v. The degree sum formula for bipartite graphs equates to a conservation of work constraint in the system,

$$\sum_{i\in U}\deg^+(u_i) = \sum_{j\in V}\deg^-(v_j) = |E|, \tag{4.4}$$

where $|E|$ represents net consensus work.

The assignment of network nodes to consensus sets is equivalent to assigning edge sets E subject to the constraints imposed by the degree sequences. Expressing E via its edge-set,

$$\{(u \in U, v \in V)\}_E, \tag{4.5}$$

elements in the two columns have multiplicity given by their degree.

B. Consensus group

The space of satisfying graph realisations for given degree sequences U and V defines the *consensus group*, $C(U,V)$. Over the space of satisfying edge-sets ($\{E\}$) the consensus group,

$$\phi : C(U,V) \times \{E\} \to \{E\}, \tag{4.6}$$

is identically the symmetric group,

$$C(U,V) = \mathrm{Sym}(\{E\}). \tag{4.7}$$

Equivalently, over vertex space (U or V) the consensus group,

$$\phi : C(U,V) \times U \to U, \tag{4.8}$$

is the group of non-degenerate vertex permutations, permutations under which edge sets E are invariant.

The order of the consensus group, equivalently the number of unique consensus assignments, is,

$$|C(U,V)| = \frac{|E|!}{\prod_{u\in U}\deg^+(u)! \cdot \prod_{v\in V}\deg^-(v)!} \leq |E|!, \tag{4.9}$$

reflecting the set of node permutations discounted by their degeneracies, upper-bounded by the order of the symmetric group $|S_n| = n!$ with no degeneracies.

Consensus groups exhibit a recursive normal subgroup structure,

$$C(U',V') \triangleleft C(U,V), \tag{4.10}$$

reflecting the associated subgraph structure,

$$G(U',V') \subset G(U,V). \tag{4.11}$$

As edge exchange operations are symmetric across U and V the consensus group may equivalently be associated with the space of edge permutations on either. Despite having support over vertex sets with distinct degree sequences they afford the same space of edge permutations.

The number of bits required to uniquely address the group's orbit scales as,

$$n = \log_2(|\mathrm{orb}_C(E)|) \leq \log_2(|E|!), \tag{4.12}$$

shown in Fig. 4.

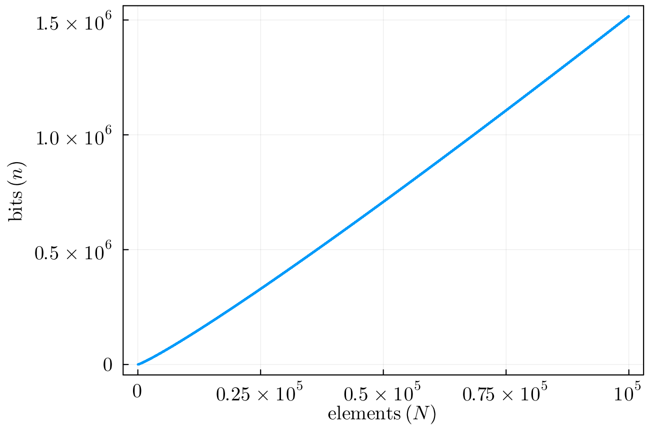

Figure 4: Number of bits (n) required to encode an arbitrary permutation (S_N) over N elements. This upper-bounds the required number of bits to uniquely address elements of the consensus group.

C. Uniform consensus sampling

Secure consensus assignment requires assigning edge-sets uniformly at random over the space of all satisfying edge-sets. Uniformity implies maximisation of entropy, given by,

$$H(C(U,V)) = \log_2(|C(U,V)|), \tag{4.13}$$

where $|C(U,V)|$ is the order of the respective consensus group. Degeneracies in permutations result in sampling bias thereby reducing entropy, requiring that algorithmic implementation does not double-count degenerate permutations from the symmetric group.

1. Fisher-Yates shuffle

The Fisher-Yates shuffle (Fisher and Yates, 1953) is an efficient algorithm for applying in-place permutations $\pi \in S_n$ to an array of n elements uniformly at random (Alg. 3).

The algorithm iteratively allows every array index to randomly exchange itself with any element not already assigned. Assuming $O(1)$ random number generation the Fisher-Yates shuffle exhibits $O(n)$ time-complexity. As the operational primitive is the exchange operation the algorithm permutes arrays in-place.

The sequence of $O(n)$ random numbers chosen during execution defines a decision tree whose ith level assigns the ith array index. Individual execution paths correspond to individual permutations with a one-to-one correspondence. Hence, the algorithm's execution space $\mathcal{L}_n$ is isomorphic to the symmetric group,

$$\mathcal{L}_n \cong S_n, \tag{4.14}$$

under mapping between execution paths $l \in \mathcal{L}_n$ and permutations $\pi \in S_n$. As all execution paths occur with uniform probability $1/n!$ the algorithm uniformly samples from the symmetric group.

Algorithm 3: The Fisher-Yates shuffle algorithm for applying a random permutation $\pi \in S_{|\vec{v}|}$ to the elements a vector $\vec{v}$. The algorithm permutes vectors in-place with $O(n)$ runtime assuming an $O(1)$ Random$(\cdot)$ function.

```
procedure FisherYatesShuffle($\vec{v}$) → $\vec{v}$                ▷ $O(n)$
    for $i \leftarrow |\vec{v}| - 1$ to 1 do                        ▷ $O(n)$
        $j \leftarrow$ Random($0 \le j \le i$)                       ▷ $O(1)$
        $v_i \leftrightarrow v_j$                                    ▷ Exchange
    return $\vec{v}$
```

The uniqueness of execution paths may be seen by noting that an element at index $j \leq i$ has exactly one path to be reassigned to index i, via the direct exchange $i \leftrightarrow j$ when the loop reaches index i.

The execution path $l \in \mathcal{L}_n$ of the algorithm directly relates to the cycle decomposition of the respective group element. As the algorithm iterates through i the series of exchange operations defines a path. When a trivial $v_i \leftrightarrow v_i$ exchange occurs it terminates that path, closing it as a cycle, after which the next i opens a new one.

Closely related to the Fisher-Yates shuffle is Sattolo's algorithm (Sattolo, 1986) for uniformly sampling the cyclic group (C_n) differing only from the Fisher-Yates shuffle in the set of allowed exchanges,

$$\begin{aligned}\text{Fisher-Yates } (S_n) &: 0 \leq j \leq i, \\ \text{Sattolo } (C_n) &: 0 \leq j < i.\end{aligned} \tag{4.15}$$

Note that this distinction serves to prohibit trivial exchanges ($v_i \leftrightarrow v_i$) thereby forcing all execution paths to define single cycles.

2. Generalised Fisher-Yates shuffle for finite groups

The Fisher-Yates shuffle may be generalised to uniformly sample over other finite groups (Alg. 4).

Alg. 3 may be interpreted as follows: for every index i the associated value v_i is chosen uniformly from the set of unassigned values v_j (where $0 \leq j \leq i$). More generally, we would choose v_i uniformly from the set of elements it is allowed to transition to under the action of the respective group. For the symmetric group, S_n, this includes all elements, whereas other groups in general constrain the set of allowed transitions.

Consider a group G defined over set X,

$$\phi : G \times X \to X. \tag{4.16}$$

Elements of the set $x \in X$ individually transform under the group action of G to their orbit,

$$\text{orb}_G(x) = G \times x, \tag{4.17}$$

defining sets for the allowed transitions of x under the action of G. For the symmetric group we have,

$$\text{orb}_{S_n}(x) = X, \tag{4.18}$$

as all elements may map to all others, while for other groups the orbit of x may be a subset of elements.

We define the *transition set*, t, to be the set of allowed transitions of x under the group action of G, given by the intersection unassigned elements (s) and the orbit of x,

$$t = \text{orb}_G(x) \cap s. \tag{4.19}$$

The uniqueness of execution paths in the generalised algorithm follows the same argument as for the the original scheme. The probability associated with an execution path is,

$$P(\vec{d}) = \prod_{i=1}^{n} p(\vec{d}_i) = \prod_{i=1}^{n} \frac{1}{|t_i|} = \prod_{i=1}^{n} \frac{1}{|\text{orb}_{G^{(i)}}(x_i)|}, \tag{4.20}$$

Algorithm 4: The generalised Fisher-Yates shuffle for finite groups G.

```
procedure GroupShuffle($\vec{v}$, $G$) $\to \vec{v}$                ▷ $O(n^2)$
    for $i \leftarrow |\vec{v}| - 1$ to 1 do                        ▷ $O(n)$
        $s \leftarrow \{\vec{v}_j \,|\, 0 \leq j \leq i\}$          ▷ Unassigned elements
        $t \leftarrow$ TransitionSet($\vec{v}_i, s, G$)             ▷ $O(n)$
        $j \leftarrow$ Random($t$)                                  ▷ $O(1)$
        $v_i \leftrightarrow v_j$                                   ▷ Exchange
    return $\vec{v}$

procedure TransitionSet($x$, $s$, $G$) $\to t$                      ▷ $O(n)$
    return $(G \times x) \cap s$                                    ▷ Group action on reduced set
```

where $\vec{d}_i = 1/|t_i|$ is the probability of making choice $j = \vec{d}_i$ at level i under uniform sampling, and $G^{(i)}$ denotes the ith level subgroup of $G = G^{(n)}$ acting on the reduced set predicated on the removal of already assigned elements of X,

$$G^{(i)} \times X^{(i)} \to X^{(i)},$$
$$X^{(i)} = X \backslash \{x_k\}_{k>i},$$
$$G^{(i-1)} \lhd G^{(i)}, \tag{4.21}$$

defining a *composition series* followed by the algorithm to iteratively assign set elements,

$$1 = G^{(0)} \lhd \cdots \lhd G^{(n-1)} \lhd G^{(n)} = G, \tag{4.22}$$

where each subgroup acts on the set predicated on the removal of an element from the set upon which the supergroup acts.

The consensus group is *transitive* by definition (all element $x \in X$ map to all elements $y \in X$ under the action of some group element $g \in G$),

$$x, y \in X \; \exists \; g \in G \mid g \cdot x = y, \tag{4.23}$$

and its non-degeneracy implies it is also a *free group* (all elements are invariant under the action of all group elements except the identity),

$$g \cdot x = x \;\Rightarrow\; g = e. \tag{4.24}$$

Hence, the action of the consensus group is *regular* which implies the orbits of all elements in the set are equal. This implies $|t_i|$ is level-dependent on i but independent of group element $x \in X$ and execution pathway $l \in \mathcal{L}$. Hence, $P(\vec{d})$ is a function of the group but uniform across all group elements.

3. Uniformly sampling the consensus group

Uniformly sampling the consensus group is achieved by defining transition sets as per Alg. 5. Here the allowed transitions under edge exchanges are defined by the non-degeneracy constraint,

$$(u_i \neq u_j) \wedge (v_i \neq v_j) = 1, \tag{4.25}$$

which implies that exchange operations between vertices u_i and u_j (equivalently v_i and v_j) are only allowed when distinct (Fig. 5).

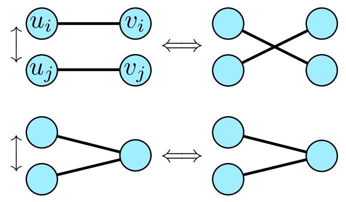

Figure 5: Degeneracy rule for vertex exchanges in the consensus group. Allowed vertex exchanges $u_i \leftrightarrow u_j$ (equivalently $v_i \leftrightarrow v_j$) require vertices in both the U and V partitions of the bipartite graph to be distinct, as per Eq. (4.25).

Algorithm 5: The transition set for the consensus group may be characterised by the constraints imposed by the group relations characterising non-degenerate transitions. The Transition function is a binary operator specifying whether index j is an allowed transition for index i, defining the respective transition set.

```
procedure ConsensusShuffle($\vec{v}$) $\to \vec{v}$                  ▷ $O(n^2)$
    for $i \leftarrow |\vec{v}| - 1$ to 1 do                         ▷ $O(n)$
        $t \leftarrow \{0 \leq j \leq i \,|\,$ Transition$(i, j) = 1\}$  ▷ $O(n)$
        $j \leftarrow$ Random($t$)                                   ▷ $O(1)$
        $v_i \leftrightarrow v_j$                                    ▷ Exchange
    return $\vec{v}$

procedure Transition($i$, $j$) $\to \{0, 1\}$                         ▷ $O(1)$
    return $(u_i \neq u_j) \wedge (v_i \neq v_j)$                    ▷ Non-degeneracy rule
```

Deriving the Random source from the network's secure global key $X_{\mathcal{N}}$ ensures all nodes follow the same execution pathway upon evaluation.

Symmetrisation over the consensus group ensures each node is assigned with uniform probability to all consensus sets, and every consensus set has uniform probability of comprising any subset of nodes,

$$p_{i,j} = const., \tag{4.26}$$

where $p_{i,j}$ is the probability of node i being assigned to consensus set j.

4. Sampling via random bitstreams

Sampling the consensus group relies on the Random function to uniformly sample an element from a set. Given a secure random bitstream X (Sec. III.C) the Random function may be implemented as per Alg. 6.

While this function is deterministic it is not guaranteed to halt if $n \neq 2^b$ for $b \in \mathbb{Z}^+$. The success probability for a single iteration of the repeat block in Alg. 6 is,

$$p_1(n) = \frac{n}{2^{\lceil \log_2(n) \rceil}} \geq 1/2. \tag{4.27}$$

Algorithm 6: Deterministically choose an index x from n choices where X is an infinite random bitstream. At most $|X| \leq \lceil \log_2(n) \rceil \cdot \log_2(1/\delta)$ random bits are required to ensure success with probability $p \geq 1 - \delta$.

```
procedure RANDOM($X$, $n$) → $x$
    repeat
        $b \leftarrow \lceil \log_2(n) \rceil$        ▷ Minimum required number of bits
        $x \leftarrow X.\mathrm{pop}(b)$        ▷ Pop bits from bitstream
    until $x < n$        ▷ Until x is within bounds
    return $x$
```

With m rounds the success probability is,

$$p_m(n) = 1 - (1 - p_1)^m > 1 - \frac{1}{2^m} > 1 - \delta. \tag{4.28}$$

Thus to achieve success with at most δ probability of failure requires at most,

$$m \geq \log_2(1/\delta), \tag{4.29}$$

rounds, and the worst-case required number of random bits is,

$$|X| \leq \lceil \log_2(n) \rceil \cdot \log_2(1/\delta). \tag{4.30}$$

5. Initial assignment

Uniformly sampling the consensus group requires applying Alg. 5 to an initial assignment, the *bipartite realisation problem*. Via the Gale-Ryser theorem (Gale, 1957; Ryser, 1957), realisable bipartite graphs (*bigraphic* degree sequences) demands,

$$\begin{aligned}\sum_{i=1}^{|U|} \deg^+(u_i) &= \sum_{j=1}^{|V|} \deg^-(v_j),\\ \sum_{i=1}^{k} \deg^+(u_i) &\leq \sum_{j=1}^{|V|} \min(\deg^-(v_j), k),\end{aligned} \tag{4.31}$$

where the second constraint assumes U and V are ordered decreasingly by degree and holds for all k.

* Policy approaches to ensure bipartite realisation:

- Uniform bidding: All nodes request consensus sets of equal size as per the random subset problem.
- Multiple uniform bidding: Nodes request multiple consensus sets of varying size, where the arrays of consensus set sizes are uniform across users.
- Hierarchical bidding: ...

V. DISTRIBUTED CONSENSUS NETWORKS

A. Model

* Sign time with accept.

We consider a network of known nodes ($\mathcal{N}$) with access to a shared, public broadcast channel ($\mathcal{B}$), which nodes may `broadcast()` into and `listen()` to. The communications primitives our model relies upon are shown in Alg. 7.

Network nodes BID to participate in consensus by announcing transaction `ids` with their required consensus set size. The network forms consensus on the set of nodes and bids to accept, thereby establishing the global key $X_{\mathcal{N}}$ which defines the assignment of consensus sets via distributed implementation of the random subset problem.

Algorithm 7: Communications primitives, where $\mathcal{B}$ denotes the shared broadcast channel and numbers denote distinct synchronous steps.

```
procedure ANNOUNCE(node, statement)
    message = SIGN(node, statement)
    $\mathcal{B} \leftarrow \mathcal{B} \cup$ message        ▷ Broadcast

procedure COMMITREVEAL(node, statement)
    salt = RANDOM($\{0,1\}^n$)

    ▷ 1. Commit hash ◁
    ANNOUNCE(node, HASH(statement|salt))

    ▷ 2. Reveal pre-image ◁
    ANNOUNCE(node, message|salt)
```

B. Protocol

The DCN protocol is synchronous with the following steps for each network node $i \in \mathcal{N}$:

1. BID: Nodes (i) bid to participate by presenting a set of requests (j) for consensus sets of given sizes ($|\mathcal{C}_{i,j}|$),

$$\begin{aligned}\mathcal{R}_{i,j} &= (\mathtt{id}_{i,j}, |\mathcal{C}_{i,j}|),\\ \mathcal{R}_i &= \bigcup_j \mathcal{R}_{i,j},\end{aligned} \tag{5.1}$$

   and stating recognised network nodes via the set of their public keys ($\mathtt{PubKey}_j$),

$$\mathcal{N}^{(i)} = \{\mathtt{PubKey}_j\}. \tag{5.2}$$

   All nodes broadcast (commit-reveal):

$$\mathcal{B} \leftarrow (\mathcal{R}_i, \mathcal{N}^{(i)}). \tag{5.3}$$

2. ACCEPT: Observing the broadcast channel $\mathcal{B}$, all nodes infer the accepted network state, bids and participating nodes ($\mathcal{N}'$),

$$\begin{aligned}\mathcal{B} &\to (\tilde{\mathcal{R}}, \tilde{\mathcal{N}}', \tilde{\mathcal{N}}),\\ \tilde{\mathcal{N}} &= \textsc{MajorityVote}(\{\mathcal{N}^{(j)}\}),\\ (\tilde{\mathcal{R}}, \tilde{\mathcal{N}}') &= \textsc{Accept}(\{\mathcal{R}_j\}),\end{aligned} \tag{5.4}$$

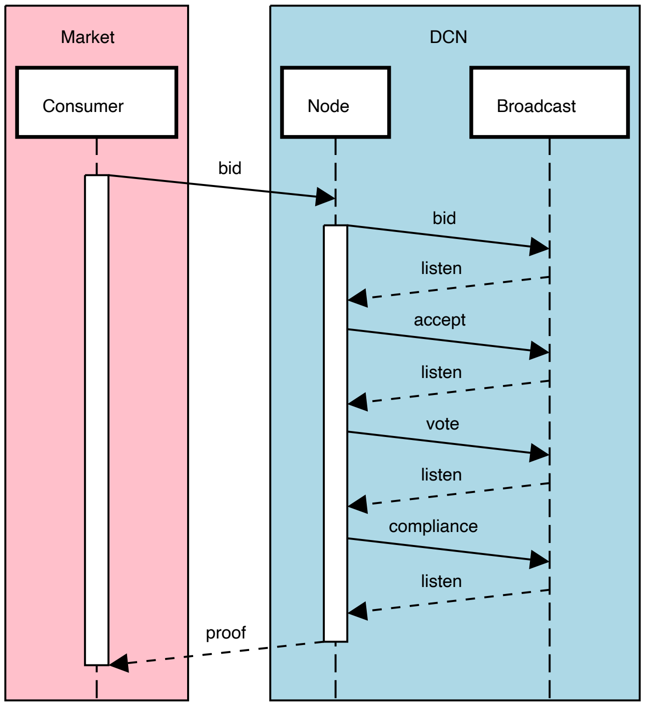

Figure 6: DCN protocol sequence diagram. (blue) DCN internal network. (red) External consensus market.

where $\text{Accept}(\cdot)$ is a network-agreed deterministic function, and $\sim$ denotes values inferred from the broadcast channel.

In this step consensus is performed at the network level where all network nodes form a single consensus set, requiring that participants form a network majority,

$$|\mathcal{N}'| > \frac{|\mathcal{N}|}{2}. \tag{5.5}$$

All other votes in the protocol are conducted at the level of assigned consensus sets.

3. Consensus: The global key $X_{\mathcal{N}}$ is implied by the accepted parameters (Sec. III.C) as is the network's allocation to consensus sets via the random subset algorithm (Sec. 1).

$$\mathcal{B} \rightarrow X_{\mathcal{N}} \rightarrow \{\mathcal{C}\}. \tag{5.6}$$

Participating nodes vote (commit-reveal) on their assigned consensus requests,

$$\mathcal{B} \leftarrow \text{Vote}_i(\tilde{\mathcal{R}}). \tag{5.7}$$

4. Compliance: Nodes vote (announce) on the compliance of participating nodes and reveal the time-of-receipt of all broadcast messages associated with the respective consensus'. Compliant nodes follow all required protocol steps, are in agreement with the majority on all votes, where timestamps (time-of-receipt) of all announcements are within the network's δ threshold.

$$\mathcal{B} \leftarrow \text{CompliantSet}_i(\mathcal{N}). \tag{5.8}$$

C. Consensus time

To enforce synchronisation of the protocol compliance requires nodes' broadcast messages to satisfy timing constraints under majority vote. To establish majority vote outcomes on the timing of messages we introduce *consensus time*, given by the median of the reported times of receipt of messages (Alg. 8).

Algorithm 8: Consensus time of a broadcast message is given by the median of the times of receipt reported by nodes.

```
procedure ConsensusTime($\mathcal{C}$, message) $\rightarrow \mathbb{R}$
    times $\leftarrow$ {TimeOfReceipt($i$, message)}$_{i \in \mathcal{C}}$
    return Median(times)
```

Consensus time exhibits the property that if for any majority of consensus nodes,

$$|t_i - \text{ConsensusTime}(\mathcal{C}, \texttt{message})| \leq \delta, \tag{5.9}$$

no reported t_i for the remaining minority can shift consensus time by more than δ, making consensus time robust against minority manipulation. Consensus time may therefore be utilised as an implied majority vote on nodes' timing compliance,

$$\text{MajorityVote}(\mathcal{C}, \text{Compliant}(\texttt{message})). \tag{5.10}$$

Let the worst-case network latencies be,

$$\tau_{\max} = \max_{i,j \in \mathcal{N}}(\tau_{i,j}), \tag{5.11}$$

where $\tau_{i,j}$ is the matrix of point-to-point latencies between nodes i and j. A message broadcast at time t_B will be received by all network nodes by at latest $t_B + \tau_{\max}$. By majority vote, the latest time at which a message could have been broadcast is,

$$t_B \leq \text{ConsensusTime}(\mathcal{C}, \texttt{message}) - \tau_{\max}. \tag{5.12}$$

Ensuring majority votes on consensus time are well-defined requires,

$$\delta \geq \frac{\tau_{\max}}{2}. \tag{5.13}$$

D. Proof-of-consensus

The sequence of signed, broadcast announcements made by compliant network nodes (i) is:

- $\mathcal{R}_i$: Request for consensus sets.
- $\mathcal{N}^{(i)}$: Public keys of all network nodes.
- $\textsc{Vote}_i(\tilde{\mathcal{R}}_j)$: Votes on assigned consensus'.
- $\textsc{Timestamp}_i(\mathcal{B})$: Time-of-receipt of broadcast messages from other nodes in the consensus set.
- $\textsc{CompliantSet}_i(\mathcal{C})$: Recognised compliant nodes in the consensus set.

We denote a complete set of such announcements from a given node ($i \in \mathcal{N}$) participating in consensus set $\mathcal{C}$ as,

$$\mathcal{P}_i(\mathcal{C}). \tag{5.14}$$

A proof-of-consensus for consensus set $\mathcal{C}$ comprises any self-consistent majority set of partial proofs,

$$\mathcal{P}(\mathcal{C}) = \bigcup_{i \in \textsc{Majority}(\mathcal{C})} \mathcal{P}_i(\mathcal{C}). \tag{5.15}$$

Note that while the proofs for a given consensus are not unique they are equivalent proofs of the same statement.

A proof-of-consensus in a self-contained proof system, comprising all necessary information to verify its validity.

E. Network policy

Let the network maintain its own ledger for tallying the contributed consensus workload of all network nodes, comprising an array of zero-initialised integer registers, one for each node,

$$\texttt{tally} \in \mathbb{Z}^{|\mathcal{N}|}. \tag{5.16}$$

When node i faithfully participates in consensus its tally register is incremented. Conversely, when requesting consensus load its tally must have sufficient balance to make the request. Under steady-state operation where nodes request what they contribute registers do not change.

When the network accepts new nodes they are unable to immediately make consensus requests and instead make null-bids, offering to contribute load without return. This increases their tally, enabling them to make subsequent requests for consensus load. This effectively forces new nodes to buy into the network by pre-contributing load they will subsequently request.

During the $\textsc{Accept}$ stage of the protocol, the network must also agree on the network's updated `tally` state.

VI. QUANTUM CONSENSUS NETWORKS

Quantum consensus networks (QCN) have quantum-enabled nodes, whose goal it is to form consensus on the generation of certifiable quantum randomness, an important resource in cryptography and numerous other applications.

As quantum hardware is costly compared to classical hardware it is expected that few networks will be quantum-enabled. However, they may exploit the quantum randomness provided by dedicated QCNs acting as *quantum random oracles* (Sec. VI.B) to inject entropy into their own global keys using *entropy addition* (Sec. VI.A), effectively enabling classical DCNs to achieve quantum random consensus assignment.

We consider two approaches for quantum random number generation (QRNG):

- Quantum key distribution (QKD): requires quantum communication but no quantum computation (Sec. VI.C).
- Interactive proofs of quantumness: require quantum computation but no quantum communication (Sec. VI.D).

As these are two-party protocols, every instance may be associated with a graph edge between the respective nodes. Random numbers associated with edges may be spoofed if both nodes collude, bypassing the need for expensive quantum resources. However, so long as at least one QRN is genuine combining them under bit-wise XOR yields a collective QRN source.

A. Entropy addition

The bit-wise XOR operator is a strictly non-entropy-decreasing function. For binary random variables,

$$X_i \to \{0,1\}, \tag{6.1}$$

with probability distributions,

$$p_i = p_i(0) = 1 - p_i(1), \tag{6.2}$$

the individual binary Shannon entropies are given by,

$$H_2(X_i) = -p_i \log_2 p_i - (1-p_i)\log_2(1-p_i), \tag{6.3}$$

where,

$$0 \leq H_2(\cdot) \leq 1. \tag{6.4}$$

Their combined entropy under the bit-wise XOR operation is lower-bounded by their maximum,

$$H_2\left(\bigoplus_i X_i\right) \geq \max_i(\{H_2(X_i)\}). \tag{6.5}$$

Hence, a random source derived from multiple sources via entropy addition is at least as random as any of them. Consequently, if any one contributing source was a QRNG the combined source will be too.

B. Quantum random oracles

Let $\mathcal{O}(t)$ denote a dynamic set of contributing random oracles at time t. We define a collective random bit-stream,

$$X_{\mathcal{O}}(t) = \bigoplus_{i\in\mathcal{O}(t)} X_i(t), \tag{6.6}$$

whose combined entropy is bounded by,

$$\max_{i\in\mathcal{O}}(\{H_2(X_i)\}) \leq H_2(X_{\mathcal{O}}) \leq 1. \tag{6.7}$$

A classical DCN may observe a QRNG oracle an add its entropy $x_{\mathcal{O}}$ to its own global key. For this to be secure it is important that the hash-based global key $X_{\mathcal{N}}$ be committed prior to the availability of the external entropy source,

$$\tilde{X}(t+\Delta) = X(t) \oplus X_{\mathcal{O}}(t+\Delta), \tag{6.8}$$

where $\Delta > 0$ is a pre-agreed future point in time subsequent to commitment of the initially established hash-based key.

C. Quantum key distribution (QKD)

Quantum key distribution (QKD) (Bennett and Brassard, 1984; Ekert, 1991) enables the secure establishment of shared randomness between two parties with information theoretic security. While ordinarily utilised for secret key exchange, here we exploit not the secrecy of shared randomness but its inability to be spoofed under honest execution of the protocol.

Assuming the existence of a quantum internet (Rohde, 2021) capable of arbitrary point-to-point entanglement routing, all node-pairs (i,j) have access to an indefinite supply of maximally-entangled Bell pairs,

$$|\Psi\rangle_{i,j} = \frac{1}{\sqrt{2}}(|0\rangle_i |0\rangle_j + |1\rangle_i |1\rangle_j), \tag{6.9}$$

requiring full $O(n^2)$ quantum communications connectivity.

For the nth copy of $|\Psi\rangle_{i,j}$ both nodes independently and privately choose measurement bases,

$$b_i(n), b_j(n) \in \{0,1\}, \tag{6.10}$$

where $b=0$ denotes the Pauli-Z basis and $b=1$ the Pauli-X basis, and record their associated measurement outcomes,

$$m_i(n), m_j(n) \in \{0,1\}. \tag{6.11}$$

The subset of measurement outcomes where both parties operate in the same basis defines a shared random bit-string,

$$s_{i,j} = \{m(n)\,|\,b_i(n) = b_j(n)\}_n. \tag{6.12}$$

These post-selected bit-strings correspond identically to those provided by the E91 (Ekert, 1991) QKD protocol. The BB84 protocol (Bennett and Brassard, 1984) can be similarly employed with the interpretational difference that for one party b and m denote encoding, for the other measurement.

Physical and implementation errors reduce the otherwise perfect measurement correlations between nodes measuring in the same basis resulting in inconsistent shared strings. However, privacy amplification (Impagliazzo *et al.*, 1989) may be used to reduce an imperfect random bit-string to a shorter one with higher entropy using classical post-processing.

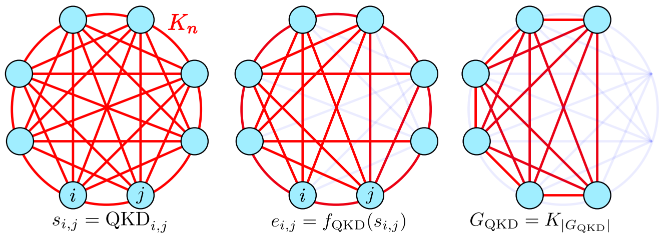
K_n
i
j
$s_{i,j} = \mathrm{QKD}_{i,j}$
i
j
$e_{i,j} = f_{\mathrm{QKD}}(s_{i,j})$
$G_{\mathrm{QKD}} = K_{|G_{\mathrm{QKD}}|}$

Figure 7: QKD proof-chain for shared quantum randomness. Amongst n nodes with full $O(n^2)$ quantum communications connectivity given by the complete graph K_n, every node executes a QKD protocol with every other node, associating a shared random bit-string $s_{i,j}$ with every edge. Bit-strings passing a QRN certification function $f_{\mathrm{QKD}}(s_{i,j})$ define the edges of a subgraph. Maintaining only vertices connected to a majority of other nodes followed by eliminating those not connected to every other, we obtain a complete subgraph G_{QKD} reflecting the unanimous majority.

1. Consensus protocol

Associating QKD bit-strings $s_{i,j}$ with graph edges we assume a certification function,

$$f_{\mathrm{QKD}}(s_{i,j}) \to \{0,1\}, \tag{6.13}$$

which evaluates `true` if $s_{i,j}$ passes a certification test for randomness. We define a QKD compliance graph with edge-inclusion based on the validity of the respective QKD bit-strings,

$$G_{\mathrm{QKD}}^{(\mathrm{comp})} : e_{i,j} = f_{\mathrm{QKD}}(s_{i,j}). \tag{6.14}$$

Letting the QKD compliance of nodes be,

$$G_{\mathrm{QKD}}^{(\mathrm{comp})} : v_i = \textsc{Majority}\left(\bigcup_{j\neq i} e_{i,j}\right), \tag{6.15}$$

the reduced graph now only contains nodes whose associated QKD bit-strings $s_{i,j}$ are majority valid.

Additionally requiring unanimity demands finding a fully-connected subgraph, or clique[3], achieved by eliminating all vertices in $G_{\text{QKD}}^{(\text{comp})}$ not connected by an edge to every other node,

$$G_{\text{QKD}} : v_i = \begin{cases} 1 & \text{if } |u_i \in G_{\text{QKD}}^{(\text{comp})}| = |G_{\text{QKD}}^{(\text{comp})}| - 1 \\ 0 & \text{otherwise} \end{cases}. \tag{6.16}$$

The fully connected $G_{\text{QKD}} = K_{|G_{\text{QKD}}|}$ subgraph now represents the accepted subset of QKD-compliant nodes under consensus.

The associated collectively established shared random bit-string is defined as,

$$s(G_{\text{QKD}}) = \bigoplus_{v_i, v_j \in G_{\text{QKD}}} s_{i,j}. \tag{6.17}$$

Although nodes could commit post-processed QKD strings obtained following post-selection and privacy amplification, this requires interaction between respective nodes. In the interests of maintaining a broadcast-only communications interface nodes may simply commit their raw unprocessed strings (b and m) from which the associated QRNs s are implied under a network-agreed post-processing function $f_{\text{PP}}(\vec{b}, \vec{m}) \to \vec{s}$.

D. Interactive proofs of quantumness

An interactive proof of quantumness (Liu and Gheorghiu, 2022; Zhu *et al.*, 2023) comprises two parties, a *prover* and a *verifier*, where the goal is for the prover to prove to the verifier that they have honestly executed a quantum implementation of some function $f(\cdot)$ that cannot be spoofed by classical simulation. The verifier has only classical resources and both parties may classically communicate.

While such protocols are not known in general for arbitrary $f(\cdot)$, they have been described in the context of a restricted class of functions known as trapdoor claw-free functions.

1. Trapdoor claw-free (TCF) functions

Trapdoor claw-free functions (TCF) are a class of cryptographic, 2-to-1, one-way functions,

$$f_{\mathcal{I}}(x) \to w, \tag{6.18}$$

which are classically efficient to evaluate in the forward direction, but for which it is hard to find simultaneously satisfying inputs $\{x_0, x_1\}$ (the 'claw') mapping to the same output,

$$w = f(x_0) = f(x_1), \tag{6.19}$$

where $x \in \{0,1\}^n$ and $w \in \{0,1\}^{n-1}$ are bit-strings.

Here, $\mathcal{I}$ denotes a problem instance derived from a secret (the trapdoor). If the secret is known, finding claws $\{x_0, x_1\}$ is classically efficient for any w. Since $f(\cdot)$ is easy to evaluate in the forward direction, verifying solutions is classically efficient, and the problem of claw-finding by definition resides in the complexity class **NP**.

2. The LWE problem

A candidate TCF is the lattice-based learning with errors (LWE) problem (Goldwasser *et al.*, 1985; Regev, 2009, 2010). This problem is believed to be post-quantum, where the associated claw-finding problem lies outside of **BQP**, the class of problems efficiently solvable by quantum computers[4].

For matrix,

$$A \in \mathbb{Z}_q^{m \times n}, \tag{6.20}$$

and vectors,

$$x, y, s, e \in \{0,1\}^n, \tag{6.21}$$

related by,

$$y = A \cdot s + e, \tag{6.22}$$

under modulo q arithmetic, a TCF may be constructed as,

$$f_{\mathcal{I}}(b, x_b) = \lfloor A \cdot x + b \cdot y \rceil, \tag{6.23}$$

where $b = \{0,1\}$ is a single bit and claws are related by,

$$x_0 = x_1 + s. \tag{6.24}$$

Here, $\mathcal{I} = \{A, y\}$ specifies the problem instance derived from the secret trapdoor $\mathcal{T} = \{s, e\}$ secretly held by the verifier, enabling efficient classical claw-finding and verification if known.

Since $f(x) \to w$ is classically efficient to evaluate in the forward direction, it is easy to find a w for which a single satisfying input x is known. The challenge lies in finding simultaneously satisfying pairs of inputs, believed to be hard for both classical and quantum computers.

[3] While the MaxClique problem of finding the largest cliques in a graph is known to be **NP**-complete in general, here we are not finding maximal cliques, affording an efficient solution.

[4] An alternate number-theoretic TCF based on Rabin's function has been described (Goldwasser *et al.*, 1988; Rabin, 1979). Since here the complexity of inverting the trapdoor reduces to integer factorisation this candidate TCF is vulnerable to quantum attack via Shor's algorithm (Shor, 1997), making it less applicable in the assumed context of universal quantum computation.

3. Interactive proof protocol

Taking a cryptographic TCF function, $f_{\mathcal{I}}(x) \to w$, an interactive proof of quantumness may be implemented as follows:

1. The verifier specifies a problem instance $\mathcal{I}$, without revealing the associated secret $\mathcal{T}$ from which it was derived.

2. The prover prepares a uniform superposition of all length-n bit-strings x using Hadamard gates,
$$|\psi\rangle = H^{\otimes n} |0\rangle^{\otimes n} = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle\,. \tag{6.25}$$

3. Evaluating $f_{\mathcal{I}}(\cdot)$ on the input x register into an output register yields,
$$|\psi\rangle = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle\,|f(x)\rangle\,. \tag{6.26}$$
This state may be efficiently prepared using a quantum circuit with,
$$O(n^2 \log^2 n), \tag{6.27}$$
gate count (Kahanamoku-Meyer *et al.*, 2022).

4. The prover measures the output register, collapsing the state onto,
$$|\psi\rangle = \frac{1}{\sqrt{2}}(|x_0\rangle + |x_1\rangle)\,|w\rangle\,, \tag{6.28}$$
for random w, which is communicated to the verifier. Since the input space x is a uniform superposition and $f(\cdot)$ is a 2-to-1 function, w is also uniform.

5. The verifier specifies a random measurement basis in which the prover should measure qubits in the x register, where $b = \{0, 1\}$ correspond to the Pauli-Z and X bases respectively.

6. When measuring in the Z basis, the prover randomly measures either $m = x_0$ or $m = x_1$, easily verified by directly evaluating $f(\cdot)$ and comparing with the prover's previously reported w. When measuring in the X basis verification succeeds if,
$$m \cdot x_0 = m \cdot x_1. \tag{6.29}$$

7. The above is repeated for some constant number of rounds, randomising the measurement basis b at every round.

The key observation is that since X and Z measurements do not commute, it is not possible for the prover to know both measurement outcomes simultaneously and therefore must measure in accordance with the verifier's stated measurement basis to pass verification of a single round. While a single round can be classically spoofed if the measurement basis b is known in advance of announcing w, if unknown, b can only be guessed with a probability of $1/2$. Upon repetition, the probability of correctly guessing all measurement bases scales as $1/2^n$ for n rounds, ensuring asymptotic confidence in the honesty of the prover.

4. Consensus protocol

To incorporate IPQs into the QCN framework we require all nodes to act as both prover and verifier for all other nodes.

In the verifier capacity every node prepares a single random TCF instance for all other nodes to prove. Despite solving the same problem instance their proofs will be distinct.

Following the same approach as with QKD we represent the proofs-of-quantumness via a complete graph with the distinction that as this is an asymmetric protocol the graph is now directed (from prover to verifier) with edges in both directions for every node-pair.

Majority votes as per Eq. (6.15) are now made from the verifier persepctive.

The additional synchronous steps required to accommodate IPQs are:

1. Nodes commit a single random problem instance $\mathcal{I}_i$.

2. Nodes execute the quantum problem instance specified by every other node j and commit the obtained $w_{i,j}$.

3. Nodes commit the random measurement bases b_i other nodes will be required to measure in.

4. Nodes complete their quantum computations and commit the obtained measurements $m_{i,j}$.

5. Nodes reveal their secrets $\mathcal{T}_i$.

Assuming a verification function analogous to Eq. (6.13),
$$f_{\mathrm{IPQ}}(\mathcal{I}_i, \mathcal{T}_i, w_{i,j}, b_i, m_{i,j}) \to \{0,1\}, \tag{6.30}$$
similarly defines a directed IPQ compliance graph,
$$G_{\mathrm{IPQ}}^{\mathrm{comp}} : e_{i,j} = f_{\mathrm{IPQ}}(\mathcal{I}_i, \mathcal{T}_i, w_{i,j}, b_i, m_{i,j}) \to \{0,1\}. \tag{6.31}$$

VII. ECONOMICS

Internally, the DCN operates as a cooperative, profit-neutral, barter economy, whose nodes facilitate an externally-facing competitive bidding market for consensus. Nodes' subjective cost of consensus is identically their own computational execution cost, individually incentivising computational and communications implementation efficiency.

Proof-of-work associates the creation of money with exchange, creating a monetary dependence on algorithmic inefficiency. DCNs are monetarily neutral, providing consensus as a generic and abstract commodity in a competitive market environment.

The DCN is a floating market instrument with instantaneous value. Consensus is necessarily consumed upon production and cannot be saved.

Consensus nodes individually utilise and monetise their gateway to the network facilitating highly competitive and high volume consensus markets.

VIII. APPLICATIONS

A. Protocols

Protocols are user-level applications for consensus, defined as arbitrary time-dependent functions acting on the current state of the broadcast channel,

$$\textsc{Protocol}(\mathcal{B}(t=0),\star) \to \star, \tag{8.1}$$

where time is relative to the present. The state of the broadcast channel a time t contains all previous messages broadcasts, where,

$$\mathcal{B}(t=0) = \bigcup_{x\in\mathcal{B}(t\leq 0)} x. \tag{8.2}$$

The state of the broadcast channel is in general subjective as individual users may have imperfect knowledge of $\mathcal{B}$ as a result of information loss. The subjective state of the broadcast channel for user i is,

$$\mathcal{B}_i \subseteq \mathcal{B}. \tag{8.3}$$

Consequently, Protocol outputs are also subjective and may differ in general,

$$\textsc{Protocol}_i(\mathcal{B}_i,\star) \neq \textsc{Protocol}(\mathcal{B},\star) \tag{8.4}$$

B. Ledgers

Consensus is formed on state register transformations.

Ledgers are defined by policies stipulating the consensus networks they recognise. Inter-ledger operations require only mutual recognition of consensus networks.

Ledgers as oracles.

Finite state machine oracles.

The combined public broadcasts across all networks acts as a global oracle for ledger states, facilitating a high arbitrage environment in an algorithmic context, an equilibriating force across the ledgers of parallel markets or interconnected markets.

Inter-ledger transactions require only mutual recognition of consensus, defined by the networks from which they are drawn.

Sec: ledger transaction queues.

* Hierarchical bidding to maximise resource utilisation.

C. Blockchains

Blockchains are protocol-level applications for consensus, following their own rules on what consensus is formed on. A blockchain's transaction history is immutable and may be retrospectively evaluated. Blockchain implementations typically consider asynchronous operating environments. In this setting simultaneous block additions manifest themselves as forks, requiring error correction mechanisms to maintain the integrity of the chain. Formally, a pool of valid block additions defines a directed tree graph which the blockchain implementation must correct to a directed linear graph.

In a synchronous setting where a ledger is associated with consensus derived from a given network these considerations change. Rather than performing consensus assignment on the basis of unique transaction identifiers we assign on the basis of transaction queue identifiers associated with individual ledgers. From the pool of accepted bids nodes assigned to a given transaction queue consensus set process all bids associated with that queue, batch processed in accordance with the ledger's transaction amalgamation rules. Employing queue assignment rather than transaction assignment mitigates the possibility of fork formation. In a proof-of-work setting this issue may be addressed by introducing friction. However, while this hinders double-mining it also undermines transaction processing rates, an undesirable tradeoff.

If n consensus sets of size N independently form honest majority their union of size nN necessarily forms honest majority. Hence,

$$P(nN,r) \leq \varepsilon \;\Rightarrow\; P(N,r)^n \leq \varepsilon. \tag{8.5}$$

For $n=1$ this reduces to consensus by majority vote amongst N parties, while the opposing limiting case of $n=N$ reduces to consensus by unanimity amongst N parties. A blockchain with unanimous n-level retrospective verification affords ε^n-security, and the associated tradeoff in required consensus set size scales as,

$$N_C(r,\varepsilon) = N_C^{(n)}(r,\varepsilon^n), \tag{8.6}$$

which implies,

$$P_M(N,r) = P_M(N^{(n)},r)^n. \tag{8.7}$$

Now ε represents the effective error rate in new block additions while $\varepsilon' = \varepsilon^n$ is the effective security parameter which applies only to blocks at least n steps back, which have been subject to n independent verifications. These blocks are considered *complete* whereas more recent blocks are considered *pending*, potentially still subject to being invalidated (Fig. 8). Only the most recent *complete* block must persist to maintain the blockchain, the point to which the blockchain is reverted if *pending* blocks are invalidated.

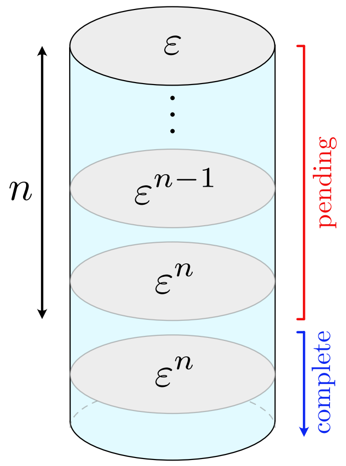

Figure 8: Blockchain with n-level reverse integrity checking. Upon addition of the top-level block consensus requires verifying integrity going back n blocks. If consensus has ε-security, for $i \leq n$ the ith past block will have been verified i times, exhibiting cumulative ε^i-security. Blocks $i \geq n$ all exhibit $\varepsilon' = \varepsilon^n$ integrity, the security parameter of the blockchain. The ε-security of the top block may be interpreted as the effective error rate in block addition, where error correction is implemented at the blockchain's protocol level.

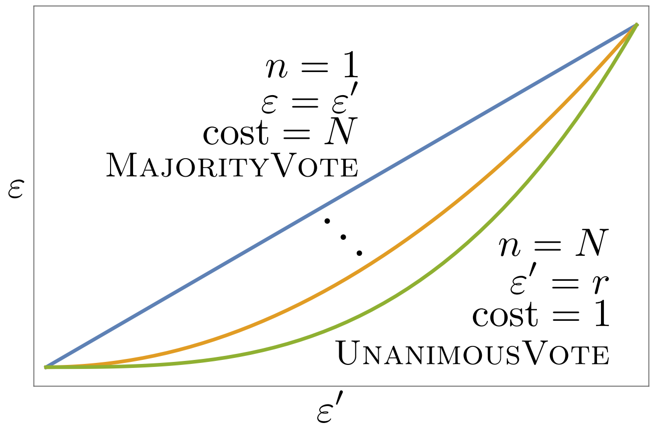

Figure 9: Tradeoffs in blockchain integrity with retrospective consensus verification.

D. Distributed computing

The consensus assignment problem may be interpreted more generically as randomised dynamic load allocation.

Consider the case of $|\mathcal{C}| = 1$ consensus sets, the delegation of computational workloads to single randomly allocated nodes. In this context the consensus assignment algorithm facilitates dynamic load balancing across the network. Similarly, $|\mathcal{C}| > 1$ equates to dynamic allocation with $|\mathcal{C}|$-fold redundancy where consensus is formed on the outcome.

More generally, MapReduce-type (Dean and Ghemawat, 2008) computations may be delegated to consensus sets of arbitrary size, where the Map routine corresponds to the assignment of consensus nodes and the Reduce routine is evaluated by consensus.

* Distributed queries.

E. Distributed signature authorities

* Consensus on state of knowledge. Trust in knowledge. Oracle.

IX. STRATEGIC CONSIDERATIONS

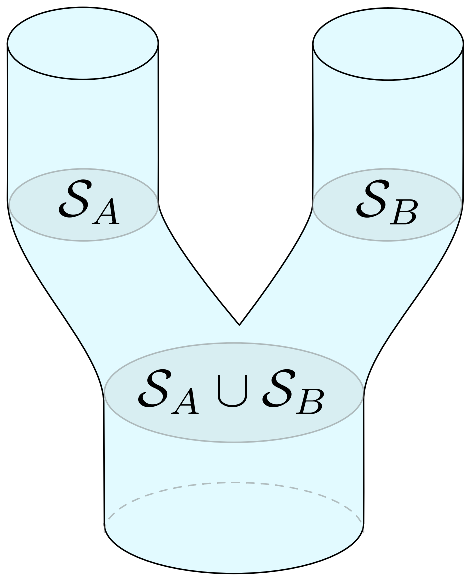

Figure 10: Bifurcation in network trust. When the network $\mathcal{S}_A \cup \mathcal{S}_B$ supporting a blockchain segregates into two non-interacting networks, $\mathcal{S}_A$ and $\mathcal{S}_B$, a fork is created, forming two unique, legitimate blockchains, one associated with each network.

Fig. 11

'Rebasing' network upon strategic retreat.

Retreat strategy in trust hierarchy where root is level $i = 0$. Levels represent set containment, $s_{i+1} \subset s_i$. Retreat to smallest i (i.e largest trusted subset). Assume

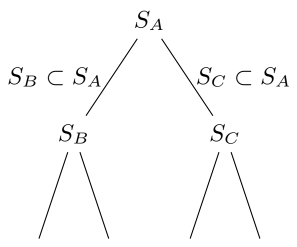

Figure 11: Trust hierarchies represented as subset-trees define retreat strategies.

that $r_{i+1} < r_i$. Inequality could work in either direction??

X. CONCLUSION

REFERENCES

Back, Adam (2002), "Hashcash - a denial of service counter-measure," .

Bennett, C H, and G. Brassard (1984), "Quantum cryptography: Public key distribution and coin tossing," Proceedings of the IEEE , 175.

Bentov, Iddo, Ariel Gabizon, and Alex Mizrahi (2017), "Cryptocurrencies without proof of work," arXiv:1406.5694.

Cambridge Centre for Alternative Finance, (2023), "Cambridge bitcoin electricity consumption index (CBECI)," .

Dean, Jeffrey, and Sanjay Ghemawat (2008), "Mapreduce: Simplified data processing on large clusters," Commun. ACM **51**, 107.

Dwork, Cynthia, and Moni Naor (1993), "Pricing via processing or combatting junk mail," in *Advances in Cryptology — CRYPTO' 92*, edited by Ernest F. Brickell (Springer Berlin Heidelberg) p. 139.

Ekert, Artur K (1991), "Quantum cryptography based on bell's theorem," Physical Review Letters **67**, 661.

Fisher, Ronald Aylmer, and Frank Yates (1953), *Statistical tables for biological, agricultural, and medical research* (Hafner Publishing Company).

Gale, David (1957), "A theorem on flows in networks," Pacific J. Math. **7**, 1073.

Goldwasser, Shafi, Silvio Micali, and Ronald L. Rivest (1985), "A "paradoxical" solution to the signature problem," in *Advances in Cryptology* (Springer Berlin Heidelberg) p. 467.

Goldwasser, Shafi, Silvio Micali, and Ronald L. Rivest (1988), "A digital signature scheme secure against adaptive chosen-message attacks," SIAM Journal on Computing **17**, 281.

Impagliazzo, R, L. A. Levin, and M. Luby (1989), "Pseudo-random generation from one-way functions," in *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*, STOC '89 (Association for Computing Machinery) p. 12.

Kahanamoku-Meyer, Gregory D, Soonwon Choi, Umesh V. Vazirani, and Norman Y. Yao (2022), "Classically verifiable quantum advantage from a computational bell test," Nature Physics **18**, 918.

King, Sunny, and Scott Nadal (2012), "Ppcoin: Peer-to-peer crypto-currency with proof-of-stake," .

Liu, Zhenning, and Alexandru Gheorghiu (2022), "Depth-efficient proofs of quantumness," Quantum **6**, 807.

Nakamoto, Satoshi (2008), "Bitcoin: A peer-to-peer electronic cash system," .

Rabin, M O (1979), *Digitalized Signatures and Public-key Functions as Intractable as Factorization*, Tech. Rep.

Regev, Oded (2009), "On lattices, learning with errors, random linear codes, and cryptography," Journal of ACM **56**, 1.

Regev, Oded (2010), "The learning with errors problem (invited survey)," in *2010 IEEE 25th Annual Conference on Computational Complexity*, p. 191.

Rohde, Peter P (2021), in *The Quantum Internet: The Second Quantum Revolution* (Cambridge University Press).

Ryser, H J (1957), "Combinatorial properties of matrices of zeros and ones," Canadian Journal of Mathematics **9**, 371.

Sattolo, Sandra (1986), "An algorithm to generate a random cyclic permutation," Information Processing Letters **22**, 315.

Schneier, Bruce (1996), *Applied Cryptography* (Wiley).

Shor, Peter W (1997), "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," SIAM Journal on Computing **26**, 1484.

Singh, Deepesh, Boxiang Fu, Gopikrishnan Muraleedharan, Chen-Mou Cheng, Nicolas Roussy Newton, Peter P. Rohde, and Gavin K. Brennen (2023), "Proof-of-work consensus by quantum sampling," arXiv:2305.19865.

Zhu, D, G.D. Kahanamoku-Meyer, and L. et al. Lewis (2023), "Interactive cryptographic proofs of quantumness using mid-circuit measurements," Nature Physics 10.1038/s41567-023-02162-9.