Distributed consensus networks

Peter P. Rohde[1, 2, 3, 4, *]
[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] *Centre of Excellence in Engineered Quantum Systems, Brisbane, Australia*
[4] *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

* peter@peterrohde.org; https://www.peterrohde.org

I. OVERVIEW

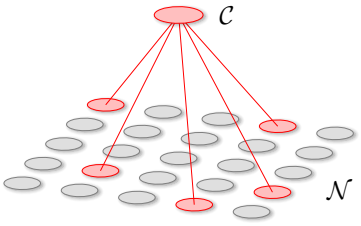
Introduction

Decentralised finance (DeFi) is the longstanding vision of a truly global and borderless financial system, the promise of universal, low-cost access to banking services and international financial markets, eliminating geographic and socio-political barriers of entry and reliance upon centralised financial institutions, subverting the power-base of monopolistic and hegemonic access to capital, facilitating a highly competitive global economic landscape.

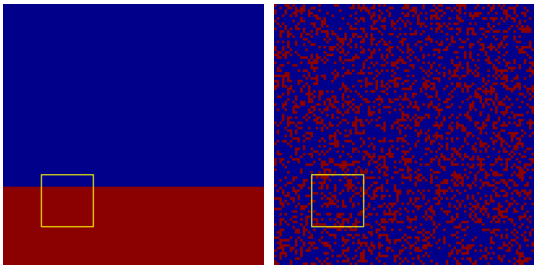
Smart-contracts enabling arbitrary, self-executing algorithmic exchange facilitate the synthesis of sophisticated financial instruments in the absence of conventional centralised exchanges. The algorithmic versatility of smart-contracts affords their use as a primitive in the construction of sophisticated distributed protocols such as Decentralised Autonomous Organisations (DAOs).

Consensus

In a decentralised environment transactions are approved via *consensus* (Sec. II) whereby a small number of randomly chosen network nodes form *consensus sets*, collectively acting as delegates for the network to authorise transactions by majority vote. Consensus outcomes should with high probability reflect the network majority, 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 assignment information, eliminating all avenues for strategic alignment.



Distributed algorithms for assigning consensus sets must be secure in the sense that their randomisation cannot be compromised by malicious parties.

Proof-of-work

The Bitcoin protocol (Nakamoto, 2008) 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, annualised energy consumption of the Bitcoin network is ∼155TWh (Cambridge Centre for Alternative Finance, 2023)[1], comparable to the total electricity consumption of medium sized countries[2].

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 ∼21TWh (PoW) to ∼2GWh (PoS) (Cambridge Centre for Alternative Finance, 2023)[3], a spontaneous reduction in energy consumption of 99.99%. 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).

Network structure

* Anonymous vs identifiable nodes
* Open vs closed
* Random subset problem
* Key establishment

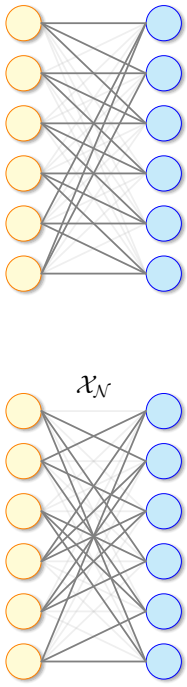
Consensus assignment problem

---

[1] Estimate as of January 19, 2024.
[2] Estimated annual electricity consumption of Sweden (2022): ∼164TWh.
[3] Estimates immediately prior and subsequent to *The Merge* on September 15, 2022.

Networks comprising known nodes afford more efficient distributed algorithms for solving the random subset problem. By utilising secure shared randomness (Sec. III.C) all network nodes may be simultaneously allocated to random subsets (Sec. III.D), ensuring full resource utilisation.

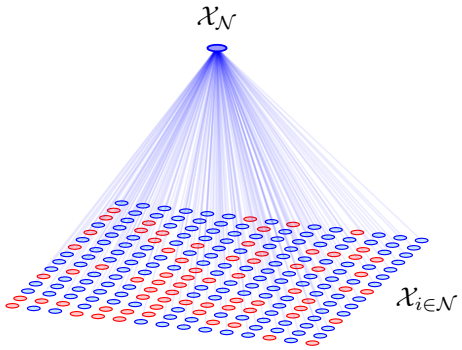
The generalisation of the random subset problem is the *consensus assignment problem* (Sec. IV), enabling the simultaneous allocation of network nodes to arbitrary sets of consensus sets under asymmetric load.



Distributed consensus networks

Distributed consensus networks (DCNs) combine the consensus assignment problem with an economic model that self-incentivises the honest participation of nodes (Sec. V). The product of DCNs is proofs-of-consensus (PoC) — timestamped, cryptographic proofs that a sets of nodes form secure random subsets of a network established via distributed execution of the consensus assignment problem (Sec. V.D).

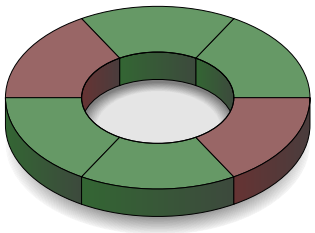
DCNs treat consensus as an abstract market commodity, an enabler of generic digital trade. The internal operation of DCNs is inherently non-monetary, a barter economy in which nodes contribute to consensus load in equal exchange for the consensus load they request. Nodes act as gateways to the DCNs to which they belong, and may independently utilise and monetise consensus load, collectively facilitating an externally-facing competitive bidding market for consensus (Sec. VII).



Consensus-time

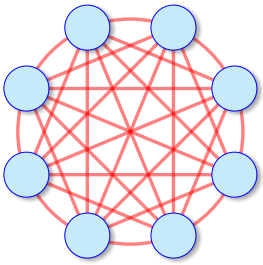
Synchronicity in the DCN protocol is enforced using *consensus time*.

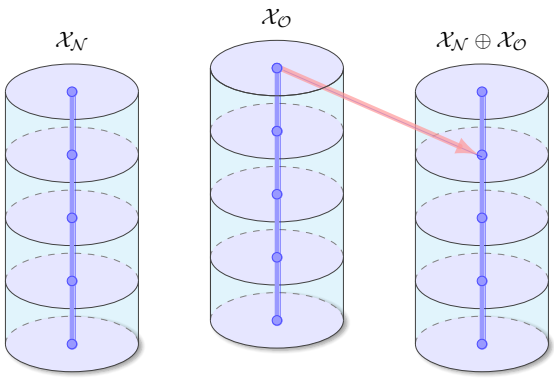
Proof-of-consensus



Quantum consensus networks

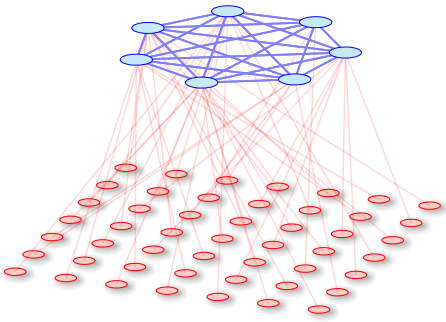
Quantum consensus networks (QCNs) (Sec. VI) have quantum-enabled nodes, possessing quantum computational or communications resources, acting as secure, certifiable quantum random number generators (QRNGs) upon forming consensus, a commodity with market value. Other (classical) DCNs may observe QCNs as oracles, utilising their random bitstreams to add entropy (Sec. VI.B) to their own global keys, enabling quantum-random consensus assignment.



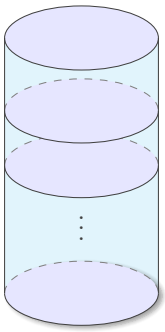


Economics

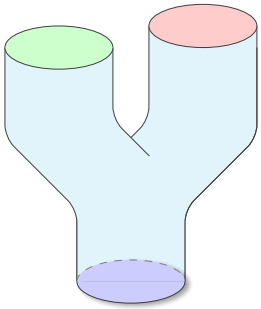
Existing blockchain design philosophy couples cryptocurrencies with the technological implementation of their mechanism for exchange, a vertically integrated paradigm which frustrates inter-ledger operations (Sec. X). The hard-wiring of currencies with specific consensus algorithms couples their cost of execution with monetary dynamics. The abstract separation between currencies and their mechanism for trade enables horizontal competition between currencies, likewise between competitors offering consensus as a commodity.



Blockchains



Strategic trust dynamics



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 set of network nodes ($\mathcal{N}$),

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

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,

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

where P_c is the probability of compromise, whereby $\mathcal{C}$ comprises a majority of dishonest parties attempting to subvert honest outcomes.

For $P_c = 0$ 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.

A. Compliance

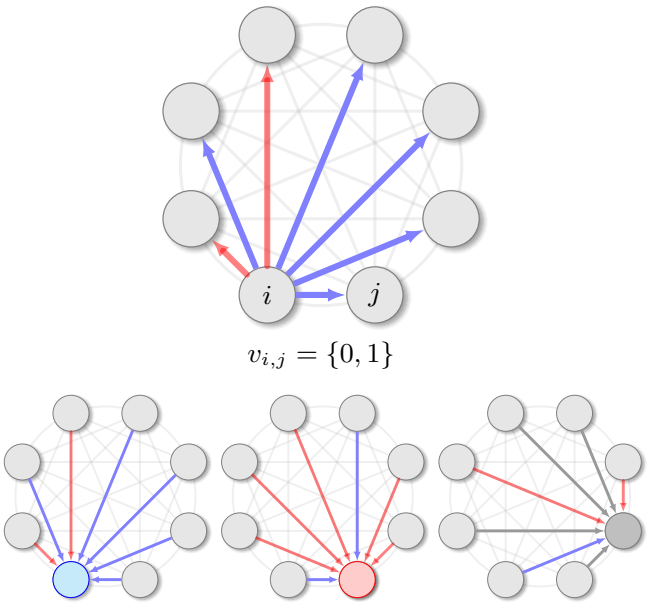


Figure 1: Compliance voting. ???

III. RANDOM SUBSET PROBLEM

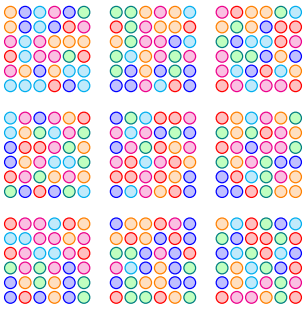


Figure 2: Random subsets. Vertices represent network nodes coloured by the consensus sets to which they are assigned. Each subfigure illustrates a distinct random assignment of 36 nodes to 6 consensus sets comprising 6 nodes.

The *random subset problem* is to choose a random subset A uniformly from B, where $A \subseteq B$, of which there are $\binom{|B|}{|A|}$ combinations. If the proportion of parties acting dishonestly and collusively is r, the probability a random subset of size N contains at least k dishonest parties is,

$$\begin{aligned} P(N, k, r) &= \sum_{n=k}^{N} \binom{N}{n} r^n (1-r)^{N-n} \\ &= I_r(k, N-k+1), \end{aligned} \tag{3.1}$$

where $I_x(a, b)$ is the regularised incomplete beta function. Compromising vote integrity via false-majority requires,

$$k_{\text{maj}} = \lfloor N/2 \rfloor + 1. \tag{3.2}$$

Hence, the likelihood of compromise is,

$$P_c(N, r) = I_r(\lfloor N/2 \rfloor + 1, N - \lfloor N/2 \rfloor). \tag{3.3}$$

While in principle all parties are associated with independent likelihood of compromise, 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,

$$r = \frac{1}{N} \sum_{i=1}^{N} r_i. \tag{3.4}$$

Hence, P_c is independent of how dishonest nodes are distributed and what strategies they might employ (Fig. 3).

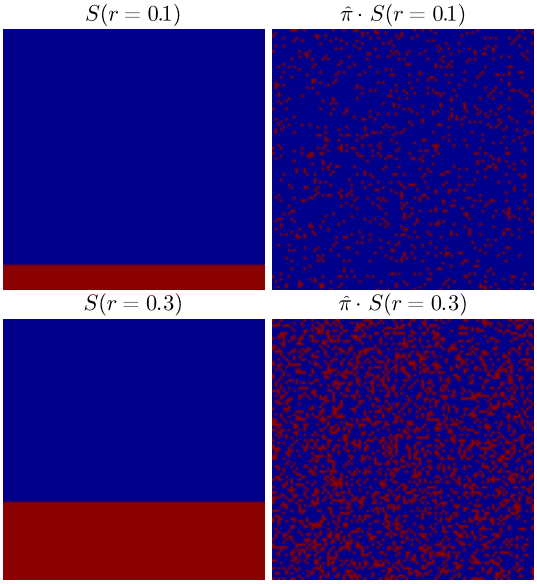


Figure 3: Random sampling as a defence against strategic alignment. (left) A minority of strategically aligned nodes may form false-majority if their alignment overlaps with a known allocation for consensus sets. (right) Random sampling erases strategic alignment and ensures node honesty is sampled with average-case r.

To uphold the integrity of majority votes, we require the likelihood of a minority gaining false-majority by

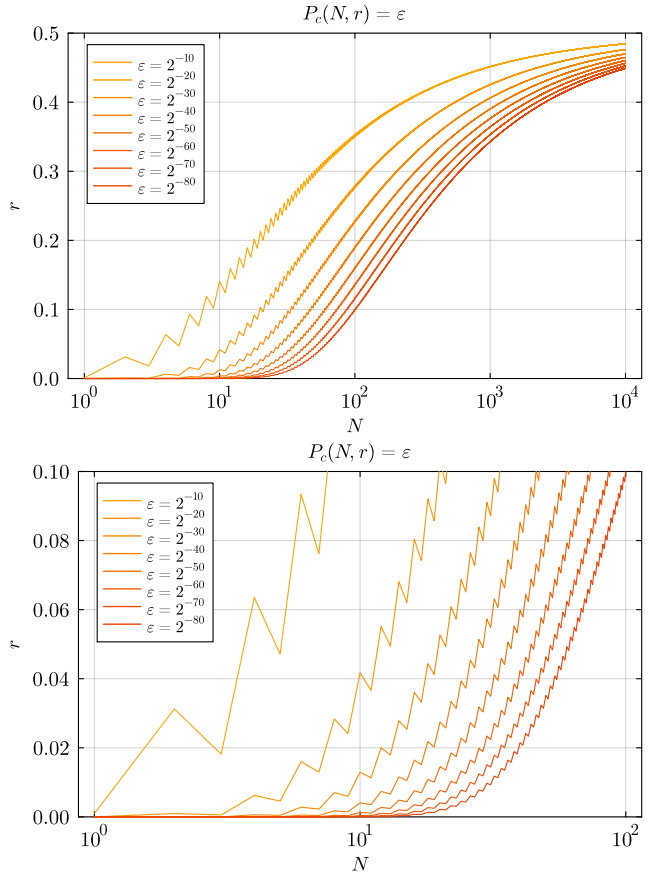


Figure 4: Security tradeoffs with consensus set size. $P_c(N,r) \leq \varepsilon$ is the probability that dishonest nodes within a random subset of size N form a false-majority when the proportion of dishonest nodes is r, where $\varepsilon = 2^{-\lambda}$ is the statistical security parameter. The oscillatory artefacts are associated with the non-linearity of $\lfloor N/2 \rfloor$ for even- and odd-valued N.

chance to be upper-bounded by an exponentially small threshold $\varepsilon = 2^{-\lambda}$,

$$P_c(N,r) \leq \varepsilon, \tag{3.5}$$

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

We define the required consensus set size, $N_{\min}$, as the smallest set size satisfying this inequality,

$$N_{\min}(r,\varepsilon) = \underset{N \in \mathbb{Z}^+}{\arg\min}(P_c(N,r) \leq \varepsilon). \tag{3.6}$$

For consensus to maintain ε-security, consensus sets should never be smaller than $N_{\min}$. Simultaneously, choosing consensus sets larger than $N_{\min}$ is unnecessary and wasteful. Tradeoff curves for $N_{\min}(r,\varepsilon)$ are shown in Fig. 4.

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 $N_{\min}$. These partitions form non-intersecting random subsets, each defining an independent consensus set.

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 $\{|\mathcal{C}|\}$.

function RANDOMSUBSETS($\mathcal{S}$, $\{|\mathcal{C}|\}$) $\rightarrow \{\mathcal{C}\}$
  ▷ *Assign a random number to each node* ◁
  for $i \in \mathcal{S}$ **do**
    $\texttt{random}_i \leftarrow$ RANDOM($\{0,1\}^n$)
  ▷ *Sort nodes by their random number* ◁
  $\texttt{ordered} =$ 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

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

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[4], 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.

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

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

```
function ProofOfWork(S, id, size) → C
    C = {}                                   ▷ Consensus set
    while |C| < size do
        for i ∈ N do                         ▷ In parallel
            random_i = Random({0,1}^n)
            output_i = Hash(id || random_i)
            if Valid(output_i) then
                C ← C ∪ S_i
    return C

function Valid(x ∈ {0,1}^n) → {0,1}
    if x/2^n ≤ p_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,

$$R_{\text{mine}} = p_{\text{mine}} \cdot R_{\text{hash}}, \tag{3.7}$$

where R_{hash} is the network's net hash-rate[5]. 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 *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,

$$p_{\text{waste}} = 1 - O\left(\frac{1}{n}\right), \tag{3.8}$$

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.

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

$$\mathcal{X}_i = \text{Hash}(\texttt{id}_i). \tag{3.9}$$

A global key is then defined relative to the set $\mathcal{N}$ as,

$$\mathcal{X}_{\mathcal{N}} = \text{Hash}\left[\,\Big|\Big|_{i\in\mathcal{N}}\, \mathcal{X}_i^{\downarrow}\right], \tag{3.10}$$

our shared random source, where $||_i$ denotes concatenation over elements i, and $\chi^{\downarrow} = \text{Sort}(\chi)$ denotes the sorted set.

Under the rules of entropy addition (Sec. VI.B), the randomness of the global key, $\mathcal{X}_{\mathcal{N}}$, cannot be compromised unless all $\mathcal{X}_i$ are compromised.

D. Secure random subsets

* Change 'global key' to 'secure shared random source'.

From a secure global key, $\mathcal{X}_{\mathcal{N}}$, as per Eq. (3.10) the random subset problem affords a simple solution. Individual keys are assigned to each node,

$$K_i = \text{Hash}(\mathcal{X}_{\mathcal{N}}||\mathcal{X}_i). \tag{3.11}$$

Ordering and partitioning nodes by their associated $\{K_i\}$ assigns them to consensus sets as per Alg. 1.

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

$$K_i^{(n)} = \text{Hash}^n(\mathcal{X}_{\mathcal{N}}||\mathcal{X}_i), \tag{3.12}$$

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

While nodes may choose their contributed `id`s, thereby influencing the established random source, $\mathcal{X}_{\mathcal{N}}$, this does not provide an avenue for compromising the integrity of random subset assignment. Adversarial nodes attempting to compromise ε-security by manipulating $\mathcal{X}_{\mathcal{N}}$ must find the necessary satisfying pre-images $\texttt{id}_i$. Searching over $\text{poly}(\lambda)$ pre-images, each associated with unique, independently random instances of $\mathcal{X}_{\mathcal{N}}$, preserves the exponentially decreasing likelihood of compromise,

$$\varepsilon' \leq \frac{\text{poly}(\lambda)}{2^{\lambda}}. \tag{3.13}$$

This approach to random subset assignment necessarily assumes a collectively agreed upon known set of nodes, as the global key $\mathcal{X}_{\mathcal{N}}$ cannot be established from an undefined set, nor can an undefined set be ordered.

[5] Bitcoin's cumulative network hash-rate was ∼500EH/s (500×10^{18}H/s) as of January, 2024 (YCharts, 2024).

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 consensus workload, 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 algorithm uniformly samples 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 the group action acting on an initial graph assignment (Sec. IV.C.6), where the pseudo-random algorithm is seeded (Sec. IV.C.5) by a secure random variable established by the network (Sec. III.C) to ensure consistent evaluation in a distributed context. The distributed algorithm for the consensus assignment problem is shown in Alg. 3.

Algorithm 3: Distributed algorithm for the consensus assignment problem. A secure shared random variable (Sec. III.C), $\mathcal{X}_\mathcal{N}$, collectively established by the network, $\mathcal{N}$, seeds pseudo-random sampling (Sec. IV.C) over consensus assignments, E, satisfying the consensus assignment graph constraints, (U, V, d) (Sec. IV.A).

function DistConsensusSampling($\mathcal{N}$, U, V, d) $\to E$
- $\mathcal{X}_\mathcal{N} \leftarrow$ SecureSharedRandomness($\mathcal{N}$)
- $E \leftarrow$ CanonicalGraph(U, V, d)
- $E \leftarrow$ ConsensusShuffle($\mathcal{X}_\mathcal{N}, E$)
- **return** E

A. Consensus assignment graphs

Consider a directed bipartite graph (Fig. 5),

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

where vertices $u \in U$ and $v \in V$ represent network nodes ($U = \mathcal{N}$) and consensus sets ($V = \{\mathcal{C}\}$) respectively, 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.

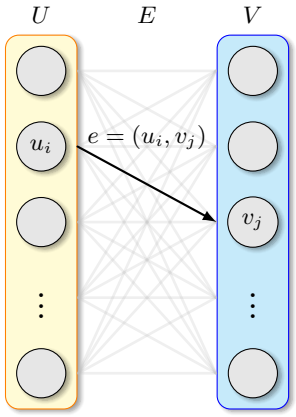


Figure 5: Generalised consensus assignment problem. Consensus assignment may be represented as a directed bipartite graph, $\mathcal{G} = (U, V, E)$, from a space of network nodes, $U \equiv \mathcal{N}$, to consensus sets, $V \equiv \{\mathcal{C}\}$, where vertex degree represents contributed/consumed consensus load. The set of all satisfying edge-sets $\{E\}$ represents the space of valid assignments. Under sampling, $p_{i,j}$ is the probability of the existence of edge $e = (u_i, v_j)$, which assigns node u_i to consensus set v_j.

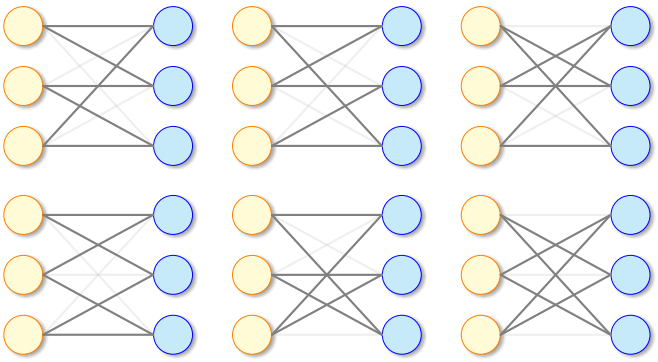


Figure 6: Satisfying solutions for the consensus assignment problem. The space of satisfying graphs assignments, $\{\mathcal{G}\} = C_d \times \mathcal{G}_{n,d}$, for $n = 3$ nodes of degree $d = 2$.

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

Vertex degrees represent workload, 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,

$$\sum_{u \in U} \deg(u) = \sum_{v \in V} \deg(v) = |E|, \tag{4.3}$$

where $|E|$ represents net consensus work.

Asymmetric load introduces sampling bias into the average-case likelihood of compromise, r, defined in

Eq. (3.4), which generalises to,

$$r = \frac{1}{|E|}\sum_{i=1}^{|\mathcal{N}|}\deg(u_i)\cdot r_i. \tag{4.4}$$

The assignment of network nodes to consensus sets is equivalent to assigning edge-sets E subject to the constraints imposed by the degree sequence,

$$\begin{aligned} d = (&\deg(u_1),\ldots,\deg(u_{|U|});\\ &\deg(v_1),\ldots,\deg(v_{|V|})), \end{aligned} \tag{4.5}$$

where,

$$|d| = |U| + |V| = |\mathcal{N}| + |\{\mathcal{C}\}|. \tag{4.6}$$

Expressing graphs by their edge-sets, E,

$$\mathcal{G} = \{(u_e \in U, v_e \in V)\}_{e\in E}, \tag{4.7}$$

where edges (u_e, v_e) denote the assignment of node u_e to consensus set v_e. In columnar representation,

$$\begin{aligned} E_U &= \{u_e\}_{e\in E},\\ E_V &= \{v_e\}_{e\in E}, \end{aligned} \tag{4.8}$$

elements have multiplicity given by their degree.

B. Consensus group

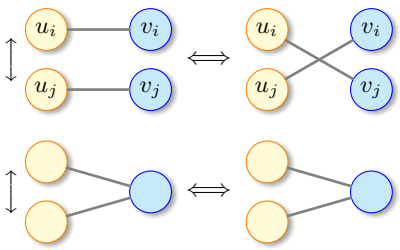


Figure 7: Swap symmetry.???.

* Space of satisfying graph assignments is given by the action of the consensus group on any valid graph assignment,

$$\{\mathcal{G}\} = C_d \times \mathcal{G}_{n,d}. \tag{4.9}$$

* Clarify edge transformations rather than U and V.
* Permutation group
* Graph automorphisms not the same as automorphisms over edges.
* Automorphism group in edge space describes invariances. Sym/Aut gives consensus group?
* Automorphism group over edge permutations is in general inequivalent to the conventional definition of a graph automorphism: for $\mathcal{G} = (V, E)$, $(u, v) \in E \Leftrightarrow (\sigma(u), \sigma(v)) \in E$
* Counting graph automorphisms is known to be
* While all permutations within E_U and E_V are degree preserving, not all are unique.

Transformations over the space of satisfying graph realisations of given degree sequence, d, defines the *consensus group*, $C(d)$, fully characterised by its degree sequence. Over the space of satisfying graphs the consensus group,

$$\phi : C(d) \times \mathcal{G}_d \to \mathcal{G}_d, \tag{4.10}$$

is identically the symmetric group,

$$\phi : C(d) = \mathrm{Sym}(\mathcal{G}_d). \tag{4.11}$$

The action of the consensus group may be equivalently defined via permutations within the columns of satisfying edge-sets,

$$\psi : C(d) \times (E_U \times E_V) \to (E_U \times E_V), \tag{4.12}$$

the group of non-degenerate permutations within the columns of E, which discounts permutations under which edge-sets remain invariant, a subgroup of the symmetric group acting on either column, both of length $|E|$,

$$(\psi, E_{U,V}) \subseteq S_{|E|}. \tag{4.13}$$

In the special case of 1-regular graphs, satisfying consensus assignments define bijective maps across graph partitions and the group action over edge-set columns is identically the symmetric group,

$$\psi : C(d) = S_{|E|},\ \delta(\mathcal{G}) = \Delta(\mathcal{G}) = 1, \tag{4.14}$$

as there are no degenerate permutations.

Hence, the order of the consensus group, equivalently the number of unique consensus assignments, is upper-bounded by the order of the symmetric group acting on edge-sets, $S_{|E|}$,

$$|C(d)| \leq |E|!, \tag{4.15}$$

with equality when $\Delta(\mathcal{G}) = 1$.

Employing element-wise inequality notation for degree sequences, of equal length, $|d| = |d'|$,

$$d' < d \coloneqq d'_i < d_i,\ \forall\, i, \tag{4.16}$$

affords the equivalent relations between degree sequences, satisfying assignment graphs and their respective subgroup structures,

$$d' < d \Rightarrow \mathcal{G}_{d'} \subset \mathcal{G}_d \Rightarrow C(d') \triangleleft C(d), \tag{4.17}$$

where $C(d')$ is a normal subgroup of $C(d)$ and $\mathcal{G}_{d'}$ is a subgraph of $\mathcal{G}_d$ under edge-removal.

The number of bits required to uniquely address the group's orbit scales as (Fig. 8),

$$n = \log_2(|\mathrm{Orb}_C(E)|) \leq \log_2(|E|!). \tag{4.18}$$

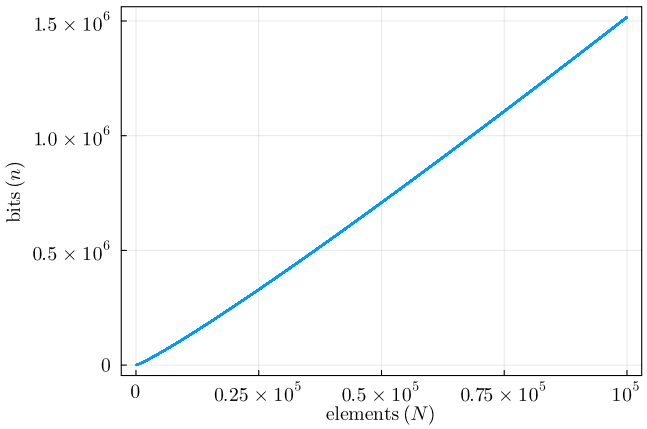


Figure 8: Number of bits (n) required to encode an arbitrary permutation (S_N) over N elements. This upper-bounds the required number of bits to uniquely address elements of the consensus group, $C(d)$.

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,

$$H(C(d)) = \log_2(|C(d)|), \tag{4.19}$$

where $|C(d)|$ 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. 4).

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,

$$\mathcal{L}_n \cong S_n, \tag{4.20}$$

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 4: (Fisher and Yates, 1953) 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.

function FISHERYATESSHUFFLE($\vec{v}$) $\to \vec{v}$ ▷ $O(n)$
  for $i \leftarrow |\vec{v}| - 1$ **to** 1 **do** ▷ $O(n)$
    $j \leftarrow$ RANDOM$(\{0, \ldots, i\})$ ▷ $O(1)$
    $v_i \leftrightarrow v_j$ ▷ *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 $v_i \leftrightarrow v_j$ 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,

$$\begin{aligned} \text{Fisher-Yates } (S_n) &: 0 \leq j \leq i, \\ \text{Sattolo } (C_n) &: 0 \leq j < i. \end{aligned} \tag{4.21}$$

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. 5).

Alg. 4 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,

$$\phi : G \times X \to X. \tag{4.22}$$

Elements of the set $x \in X$ individually transform under

the group action of G to their orbit,

$$\begin{aligned}\mathrm{Orb}_G(x) &= G \times x \\ &= \{y \in X \mid \exists\, g \in G \mid y = g \cdot x\} \end{aligned} \tag{4.23}$$

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

$$\mathrm{Orb}_{S_n}(x) = X, \tag{4.24}$$

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,

$$t = \mathrm{Orb}_G(x) \cap s. \tag{4.25}$$

Algorithm 5: Generalised Fisher-Yates shuffle for finite groups G.

```
function GroupShuffle(v⃗, G) → v⃗                          ▷ O(n²)
    for i ← |v⃗| − 1 to 1 do                               ▷ O(n)
        s ← {v⃗_j | 0 ≤ j ≤ i}                             ▷ Unassigned elements
        t ← TransitionSet(v⃗_i, s, G)                      ▷ O(n)
        j ← Random(t)                                     ▷ O(1)
        v_i ↔ v_j                                         ▷ Exchange
    return v⃗

function TransitionSet(x, s, G) → t                       ▷ O(n)
    return (G × x) ∩ s                                    ▷ Group action on reduced set
```

WRONG FOR THE CONSENSUS GROUP: 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,

$$P(\vec{d}) = \prod_{i=1}^{n} p(\vec{d}_i) = \prod_{i=1}^{n} \frac{1}{|t_i|} = \prod_{i=1}^{n} \frac{1}{|\mathrm{Orb}_{G^{(i)}}(x_i)|}, \tag{4.26}$$

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,

$$\begin{aligned} G^{(i)} \times X^{(i)} &\to X^{(i)}, \\ X^{(i)} &= X \backslash \{x_k\}_{k>i}, \\ G^{(i-1)} &\lhd G^{(i)}, \end{aligned} \tag{4.27}$$

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

$$1 = G^{(0)} \lhd \cdots \lhd G^{(n-1)} \lhd G^{(n)} = G, \tag{4.28}$$

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

As the consensus group is both free[6] and transitive[7] it has regular[8] group action and all group elements have the same orbit. Hence, $|t_i|$ is level-dependent on i but independent of group element $x \in X$ and execution pathway $l \in \mathcal{L}$. $P(\vec{d})$ is therefore 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. 6, whereby allowed transitions under edge exchanges $v_i \leftrightarrow v_j$ satisfy the Boolean constraint,

$$[(u_i, v_j) \notin E] \vee [(u_j, v_i) \notin E] = 1, \tag{4.29}$$

requiring that newly created edges not be previously existing ones. ??? CHECK

An $v_i \leftrightarrow v_j$ exchange within edge-set E is equivalent to eliminating existing edges,

$$e_i = (u_i, v_i),\ e_j = (u_j, v_j), \tag{4.30}$$

and replacing them with permuted edges,

$$e'_i = (u_i, v_j),\ e'_j = (u_j, v_i). \tag{4.31}$$

Thus, edge-set invariance under $v_i \leftrightarrow v_j$ requires,

$$E \backslash \{e_i, e_j\} \cup \{e'_i, e'_j\} = E, \tag{4.32}$$

which implies,

$$\begin{aligned} \{e'_i, e'_j\} &\in E, \\ E \backslash \{e_i, e_j\} &= E \backslash \{e'_i, e'_j\}, \\ \{e_i, e_j\} &= \{e'_i, e'_j\}. \end{aligned} \tag{4.33}$$

Hence, the requirement exchanges $v_i \leftrightarrow v_j$ not be edge-set invariant is,

$$\begin{aligned} \{e'_i, e'_j\} &\in E, \\ E \backslash \{e_i, e_j\} &= E \backslash \{e'_i, e'_j\}, \\ \{e_i, e_j\} &= \{e'_i, e'_j\}. \end{aligned} \tag{4.34}$$

---

[6] Free groups: all elements are invariant under the action of all group elements except the identity,

$$g \cdot x = x \Rightarrow g = e.$$

[7] Transitive groups: all elements $x \in X$ map to all elements $y \in X$ under the action of some group element $g \in G$,

$$x, y \in X \mid \exists\, g \in G \mid g \cdot x = y.$$

[8] Regular group action: $\forall\, x, y \in X \mid g \cdot x = y$, $g \in G$ is unique.

* Need to clarify notation here. i here refers to index in E, but usually we use it for a node number.

Algorithm 6: 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.

function CONSENSUSSHUFFLE($\mathcal{X}$, $\vec{u}$, $\vec{v}$) $\to \vec{v}$ ▷ $O(n^2)$
  for $i \leftarrow |\vec{v}| - 1$ **to** 1 **do** ▷ $O(n)$
    $t \leftarrow \{0 \le k < i \,|\, \text{TRANSITION}(i,k) = 1\}$ ▷ $O(n)$
    $j \leftarrow \text{RANDOM}(\mathcal{X}, t \cup i)$ ▷ $O(1)$
    $v_i \leftrightarrow v_j$ ▷ *Exchange*
  return $\vec{v}$

function ALLOWTRANSITION(i, j) $\to \{0,1\}$ ▷ $O(n)$
  return $[(u_i, v_j) \notin E] \wedge [(u_j, v_i) \notin E]$

The regular action of the consensus group implies that under symmetrisation via uniform sampling all satisfying edge assignments $\{E\}$ occur with equal probability, from which it follows,

$$\begin{aligned} p_{u,v} &= \frac{\deg(u)}{|V|}, \\ \sum_{v \in V} p_{u,v} &= \deg(u), \\ \sum_{u \in U} p_{u,v} &= 1, \\ \sum_{u \in U, v \in V} p_{u,v} &= |E|, \end{aligned} \tag{4.35}$$

where $p_{u,v}$ is the probability of node u being assigned to consensus set v. * Regular action.

* Use uniform bidding to preserve average case r. If non-uniform the effective r in consensus sets is biased towards the r of higher bidders.

* ??? * For a given edge $e = (u, v)$ in edge-set $e \in E$, the order of the orbit of vertex v under the group action ψ is,

$$\psi : |\text{Orb}_C(v)| = |V/v| = |V| - 1. \tag{4.36}$$

That is, vertex v may permute to any vertex other than itself.

* ??? TODO. Fig. 5

* Bipartite decomposition or bipartite dimension. Edge-disjoint union of complete bipartite graphs or bicliques.

* Degree-preserving biclique decomposition.

* Kmn defines invariant subgraphs under edge-permutation. Vertices within a Kmn subgraph define graph automorphisms under vertex permutations σ.

$$\varphi : (u, v) \in E \iff (\sigma_u, \sigma_v) \in E. \tag{4.37}$$

For complete graphs all vertex permutations within each bipartition define graph automorphisms,

$$\text{Aut}(K_{m,n}) = S_m \times S_n. \tag{4.38}$$

* Graph sum (disjoint union)

$$\mathcal{G}_d^{(K)} = \bigoplus_b K_{m_b, n_b}. \tag{4.39}$$

* Not all graphs can be decomposed this way. Determining this is NP-hard in general.

For $K_{m,n}$,

$$\begin{aligned} \deg(u) &= n, \, \forall\, u \in U, \\ \deg(v) &= m, \, \forall\, v \in V. \end{aligned} \tag{4.40}$$

* Admits graph automorphisms,

$$\bigtimes_b (S_{m_b} \times S_{n_b}) \subseteq \text{Aut}(\mathcal{G}_d^{(K)}). \tag{4.41}$$

* Is it equality?

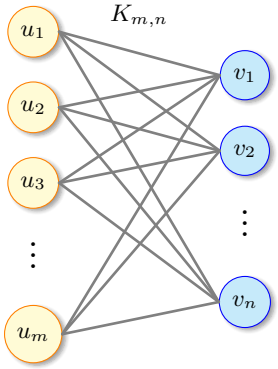


Figure 9: Complete bipartite graph, $K_{m,n}$. Vertices in each bipartition have uniform degree.

4. Counting bipartite graph realisations

The structure of the CONSENSUSSHUFFLE algorithm (Alg. 6) via Eq. (4.26) affords evaluation of the order of the consensus group equivalently counting bipartite graph realisations for a given degree sequence,

$$|C(d)| = |\mathcal{G}_d|. \tag{4.42}$$

* The regular action of $C(d)$ affords the freedom to As all execution paths $l \in \mathcal{L}$

u_1	v_1
u_1	v_2
u_1	$\vdots$
u_1	v_n
u_2	v_1
u_2	v_2
u_2	$\vdots$
u_2	v_n
$\vdots$	$\vdots$
u_m	v_1
u_m	v_2
u_m	$\vdots$
u_m	v_n

Figure 10: Canonically ordered edge-set for complete bipartite graph, $K_{m,n}$.

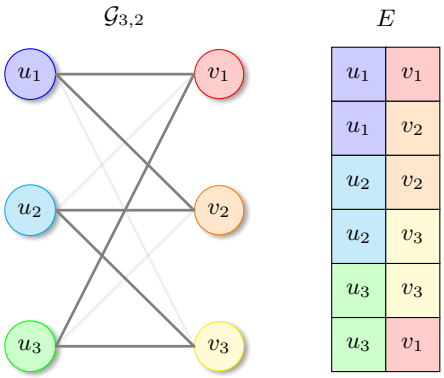


Figure 11: Consensus assignment graph ($\mathcal{G}_{3,2}$) and its respective edge-set (E).

Algorithm 7: Count satisfying bipartite graph realisations for degree sequence d. RedDeg(d) does not re-sort d, preserving its initial ordering. The elimination of zero-valued elements from d equates to relabelling, but does not change the respective graph strcuture.

function CountBipartiteReal(d) $\to N$
- $d \leftarrow$ DegSort(d)
- $N \leftarrow 1$ ▷ *Group orbit size*
- **for** $i \leftarrow 1$ **to** $|E|$ **do** ▷ $O(|U| \cdot |V|)$
  - $t \leftarrow \{0 \le k < i \,|\, \text{Transition}(i,k) = 1\}$ ▷ $O(n)$
  - $N \leftarrow N \cdot |t|$
- **return** N

5. Sampling via random bitstreams

As nodes must be in agreement on consensus assignment the Random functions relied upon during execution of Alg. 6 must evaluate consistently across network nodes, for which the established secure random bitstream $\mathcal{X}_\mathcal{N}$ (Sec. III.C) is employed as per Alg. 8.

Algorithm 8: Deterministically choose an element x from n choices where $\mathcal{X}$ is a secure shared random bitstream. At most $|\mathcal{X}| \le \lceil \log_2(n) \rceil \cdot \log_2(1/\delta)$ bits are required to ensure success with probability $p \ge 1 - \delta$.

function Random($\mathcal{X}$, n) $\to x$
- **repeat**
  - $b \leftarrow \lceil \log_2(n) \rceil$ ▷ *Minimum required number of bits*
  - $x \leftarrow \mathcal{X}$.pop(b) ▷ *Pop bits from bitstream*
- **until** $x < n$ ▷ *Until x is within bounds*
- **return** x

While this function is deterministic it is not guaranteed to halt if $n \neq 2^b$ where $b \in \mathbb{Z}^+$. The success probability for a single iteration of the repeat block in Alg. 8 is,

$$p_1(n) = \frac{n}{2^{\lceil \log_2(n) \rceil}} \ge \frac{1}{2}, \tag{4.43}$$

where the worst-case lower-bound of $p = 1/2$ arises when $n = 2^b + 1$ for $b \gg 1$,

$$\begin{aligned} p_1(2^b + 1) &= \frac{1}{2} + \frac{1}{2^{b+1}}, \\ \lim_{b \to \infty} p_1(2^b + 1) &= \frac{1}{2}. \end{aligned} \tag{4.44}$$

With m rounds the success probability is,

$$p_m(n) = 1 - (1 - p_1)^m > 1 - \frac{1}{2^m}. \tag{4.45}$$

Limiting the probability of failure to $\delta = 1/2^m$ requires at most,

$$m \ge \log_2(1/\delta), \tag{4.46}$$

rounds. Hence, the worst-case consumption of random bits is,

$$|\mathcal{X}^{(n)}| \le \lceil \log_2(n) \rceil \cdot \log_2(1/\delta). \tag{4.47}$$

The respective number of bits required to address all elements of the symmetric group, S_n, with δ-likelihood of failure for all calls to Random($\mathcal{X}, \cdot$) is,

$$\begin{aligned} |\mathcal{X}|^{(S_n)} &= \sum_{i=1}^{n} |\mathcal{X}^{(i)}| \\ &= \log_2(1/\delta) \cdot \sum_{i=1}^{n} \lceil \log_2(i) \rceil \\ &\approx \log_2(1/\delta) \cdot \log_2(n!), \end{aligned} \tag{4.48}$$

an effective $\log_2(1/\delta)$ multiplicative overhead on the scaling relationship shown in Fig. 8.

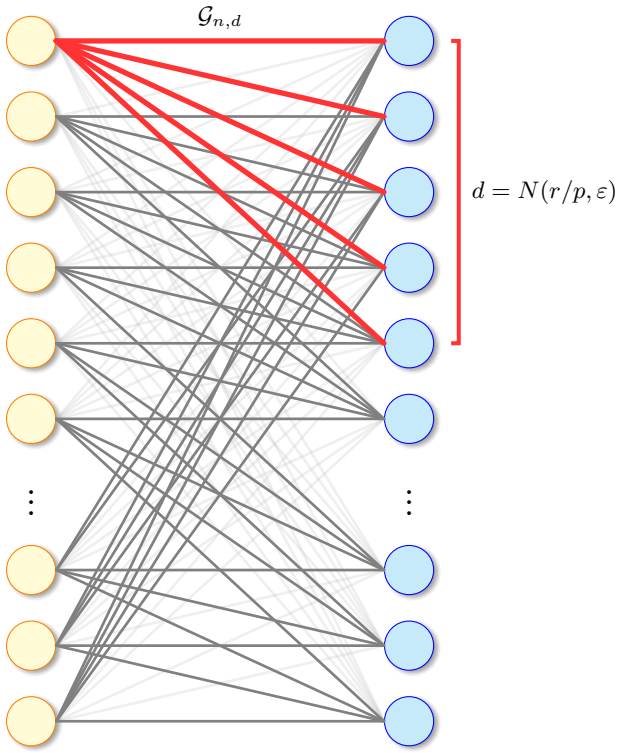


Figure 12: Uniform bidding initial assignment graph. $\mathcal{G}_{n,d}$ has n nodes and n consensus sets, all with degree d.

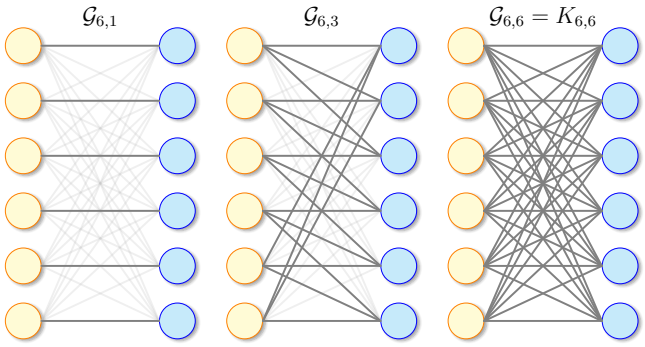


Figure 13: Uniform bidding initial assignment graph. $\mathcal{G}_{n,d}$ has n nodes and n consensus sets, all with degree d, where $d \leq n$. For $n = d$ we obtain the complete bipartite graph $\mathcal{G}_{n,n} = K_{n,n}$.

Uniformly sampling consensus allocation requires applying sampled consensus group transformations (Alg. 6) to an initial satisfying graph assignment (Fig. 5) — a solution to the *bipartite realisation problem*. Via the Gale-Ryser theorem (Gale, 1957; Ryser, 1957), realisable bipartite graphs (those with *bigraphic* degree sequences) demands,

$$\sum_{i=1}^{|U|} \deg(u_i) = \sum_{j=1}^{|V|} \deg(v_j), \tag{4.49}$$
$$\sum_{i=1}^{k} \deg(u_i) \leq \sum_{j=1}^{|V|} \min(\deg(v_j), k),\ \forall k \in \{1, \ldots, |V|\},$$

where the second constraint assumes U and V are ordered decreasingly by degree. Realisable bipartite graphs may be efficiently realised using the Havel-Hakimi algorithm (Hakimi, 1962; Havel, 1955), shown in Alg. 9, which we employ for initial canonical graph assignment.

The simplest bidding format for ensuring realisable consensus assignment is *uniform bidding* where all nodes request a single consensus set of size $|\mathcal{C}|$, and all vertices have constant degree,

$$\deg(u) = \deg(v) = |\mathcal{C}|,\ \forall u \in U, v \in V, \tag{4.50}$$

which is realisable as per Eq. (4.49) if and only if,

$$|\mathcal{C}| \leq |\mathcal{N}'|, \tag{4.51}$$

where,

$$|U| = |V| = |\mathcal{N}'| = |\mathcal{C}|, \tag{4.52}$$

is the number of participating network nodes, equivalently the number of consensus sets. The same condition holds if all nodes request a constant number of consensus sets of uniform size.

* 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: ...

Algorithm 9: (Hakimi, 1962; Havel, 1955) Deterministic consensus assignment via bipartite graph realisation of a given degree sequence, with $O(|U| \cdot |V|)$ time-complexity.

function CANONICALGRAPH(U, V, d) $\to E$
- $E \leftarrow \{\}$ ▷ *Edge set*
- **for** $u \in U$ **do** ▷ $O(|U| \cdot |V|)$
  - **if** $\deg(u) > 0$ **then**
    - **for** $v \in V$ **do**
      - **if** $\deg(v) > 0$ **then**
        - $E \leftarrow E \cup (u, v)$ ▷ *Assign edge*
        - $\deg(u) \leftarrow \deg(u) - 1$ ▷ *Decrement valence*
        - $\deg(v) \leftarrow \deg(v) - 1$
        - **if** $\deg(u) = 0$ **then**
          - **break**
- $x \leftarrow \sum_{u \in U} \deg(u) + \sum_{v \in V} \deg(v)$ ▷ *Unassigned edges*
- **if** $x \neq 0$ **then**
  - $E \leftarrow \emptyset$ ▷ *Failure: unrealisable graph*
- **return** E

function DEGSORT(U, V) $\to (U, V)$
- $U \leftarrow \text{SORT}(U \,|\, \deg(u_i) \geq \deg(u_{i+1}))$ ▷ $O(|U| \log |U|)$
- $V \leftarrow \text{SORT}(V \,|\, \deg(v_i) \geq \deg(v_{i+1}))$ ▷ $O(|V| \log |V|)$
- **return** (U, V)

Fig. 12

$$\mathcal{G}_{n,d} : e_{i,j} = \begin{cases} 1, & (j-i) \bmod n < d \\ 0, & \text{otherwise} \end{cases}, \tag{4.53}$$

V. DISTRIBUTED CONSENSUS NETWORKS

A. Model

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. 10.

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

Algorithm 10: Communications primitives, where $\mathcal{B}$ denotes the shared broadcast channel and numbers denote distinct synchronous steps.

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

function 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

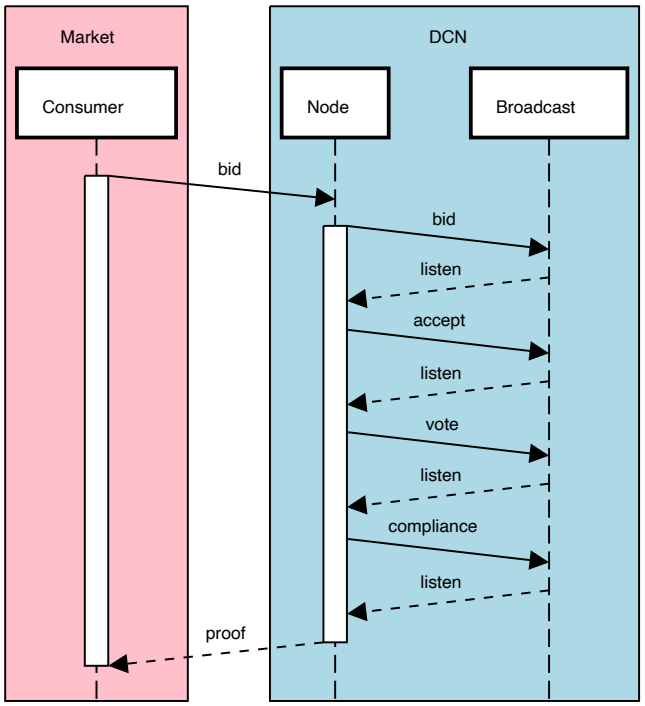


Figure 14: DCN protocol sequence diagram. (blue) DCN internal network. (red) External consensus market.

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 ($|\mathcal{C}_{i,j}|$),

$$\mathcal{R}_{i,j} = (\texttt{id}_{i,j}, |\mathcal{C}_{i,j}|),$$
$$\mathcal{R}_i = \bigcup_j \mathcal{R}_{i,j}, \tag{5.1}$$

and stating recognised network nodes via the set of

their public keys ($\texttt{PubKey}_j$),

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

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 ($\mathcal{N}'$),

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

where $\textsc{Accept}(\cdot)$ is a network-agreed deterministic function, and $\tilde{}$ 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 $\mathcal{X}_\mathcal{N}$ 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).

$$\mathcal{B} \to \mathcal{X}_\mathcal{N} \to \{\mathcal{C}\}. \tag{5.6}$$

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

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

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.

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

1. Voting

* Assume the network composition is,

$$N = N_H + N_D, \tag{5.9}$$

where $N_H > N_D$.

* A network majority requires,

$$N_{\text{maj}} = \lfloor N/2 \rfloor + 1, \tag{5.10}$$

votes.

* Requiring at least,

$$N_V = N_{\text{maj}} + N_D, \tag{5.11}$$

votes guarantees the inclusion of N_{maj} honest votes, thereby upholding network integrity (correctness?).

* Requiring at least,

$$\begin{aligned} N_P &= N_V + N_D \\ &= N_{\text{maj}} + 2N_D, \end{aligned} \tag{5.12}$$

participants (i.e bidders) guarantees at least N_V honest voters amongst them, ranging between N_V (all of whom are honest) to N_P (N_V honest + N_D dishonest).

* Since there are at least N_V honest participants, reaching quorum does not rely on dishonest parties who therefore do not have the ability inhibit quorum.

* Treating all dishonest votes as ambiguous ($\perp$) and honest votes as known with $N_H = N_+ + N_-$ votes for ($+$) and against ($-$) ACCEPT.

* Here $0 \leq N_+ \leq N_D$ is a subjective value.

* A simple majority vote will be ambiguous when,

$$N_{\text{maj}} - N_D < N_+ < N_{\text{maj}} \tag{5.13}$$

Within this window dishonest voters may swing the simple-majority outcome in either direction. If dishonest votes are in favour we have,

$$N_+ + N_D \geq N_{\text{maj}}, \tag{5.14}$$

and the outcome is unambiguous ACCEPT. If dishonest votes go against we have,

$$N_+ < N_{\text{maj}}, \tag{5.15}$$

and the outcome is unambiguous REJECT.

* When,

$$N_+ + N_D < N_{\text{maj}}, \tag{5.16}$$

it is not possible for withheld dishonest votes to swing the outcome to ACCEPT and the result is unambiguous REJECT.

* Using three-valued logic the subjective simple majority vote outcome is,

$$\textsc{MajorityVote}^{(i)}(\cdot) = \begin{cases} 0, & N_+^{(i)} < N_{\text{maj}} - N_D \\ 1, & N_+^{(i)} \geq N_{\text{maj}} + N_D \\ \perp, & N_{\text{maj}} - N_D < N_+^{(i)} < N_{\text{maj}} + N_D \end{cases},$$

where the subjective outcome is relative to a party's understanding of $N_+^{(i)}$.

* As different nodes may have different subjective interpretations of N_+, when the majority vote is ambiguous for some it may be unambiguous for others. As honest nodes are compliant all parties have the same subjective understanding of honest vote outcomes. Dishonest nodes may be non-compliant, resulting in fragmentation from inconsistent subjective understanding of their outcomes.

* If a node subjectively believes there are $N_+ \geq N_{\text{maj}}$ votes to ACCEPT, hence an ACCEPT majority vote outcome, other nodes may subjectively believe there are a minimum of $N_+ - N_D$ votes to ACCEPT. Hence, a voter (A) who subjectively observes $N_+^{(A)} \geq N_{\text{maj}}$ can only be certain other nodes share that understanding when $N_+^{(A)} - N_D \geq N_{\text{maj}}$.

* Subsequent to a simple majority vote, let us hold a supermajority vote requiring N_V votes to pass. As there are at least N_V honest voters amongst the N_P participants quorum exists.

* If the simple majority vote is unambiguous the supermajority vote is required to reflect that outcome (confirmation). If the simple majority vote is ambiguous this implies $N_{\text{maj}} - N_D < N_+ < N_{\text{maj}}$. A supermajority of N_V requires $N_+ + N'_+ \geq N_V$, where $N'_+ \leq N_D$ are any unknown dishonest votes that may yet additionally contribute.

In the ambiguous regime, a supermajority of N_V requires,

$$\begin{aligned} N_+ + N'_+ &\geq N_V, \\ N_+ + N'_+ &\geq N_{\text{maj}} + N_D, \\ N_+ + N'_+ - N_D &\geq N_{\text{maj}}, \end{aligned} \tag{5.17}$$

Since $N_+ < N_{\text{maj}} + N_D$

See Fig. 15

* Since $N_P = N_{\text{maj}} + 2N_D$ comprises (at most) N_D dishonest parties,

$$\dot{r} = \frac{N_D}{N_P} = \frac{N_D}{p \cdot N} = \frac{r}{p} \tag{5.18}$$

With N_V received votes, it is guaranteed a network majority of N_{maj} have cast honest votes.

2. Network acceptance

??? TO DO

Decomposing a network into honest and dishonest nodes,

$$\begin{aligned} \mathcal{N} &= \mathcal{N}_H \cup \mathcal{N}_D, \\ N &= N_H + N_D, \end{aligned} \tag{5.19}$$

the ratios of malicious (r) and participating (p) nodes in the network are,

$$r = \frac{N_D}{N}, \quad p = \frac{N_P}{N}, \tag{5.20}$$

where $\mathcal{N}'$ are the participating nodes,

In a worst-case adversarial model we assume all dishonest nodes are present in any participating set. The ratio of malicious nodes in the participating set is,

$$r' = \frac{N_D}{p \cdot N} = \frac{r}{p}, \tag{5.21}$$

modulating the effective ratio of dishonest nodes by the ratio of participating nodes, where $p > 1/2$ and $r < 1/2$. Maintaining $r' < 1/2$ to ensure an honest majority imposes a lower bound on the participation ratio,

$$p_{\text{min}} > \max(2r, 1/2). \tag{5.22}$$

As the effective proportion of dishonest nodes, $r' = r/p$, is a function of the variable network participation rate, p, the required consensus set size to maintain ε-security may be algorithmically specified upon acceptance. Now the dynamic minimum consensus set size scales as,

$$N_{\text{min}}(r/p, \varepsilon), \tag{5.23}$$

shown in Fig. 16.

C. Consensus time

DEFINITIONS

* Define consensus time of a message broadcast by node i relative to a set of nodes $\mathcal{S}$,

$$\textsc{ConsensusTime}(i, \mathcal{S}) = \text{median}(\{t_{i,j}\}_{j \in \mathcal{S}}), \tag{5.24}$$

where $t_{i,j}$ is the reported time-of-receipt of message i by node j, and $\{t_{i,j}\}_{j \in \mathcal{S}}$ is the set of reported times-of-receipt of message i by elements of $\mathcal{S}$.

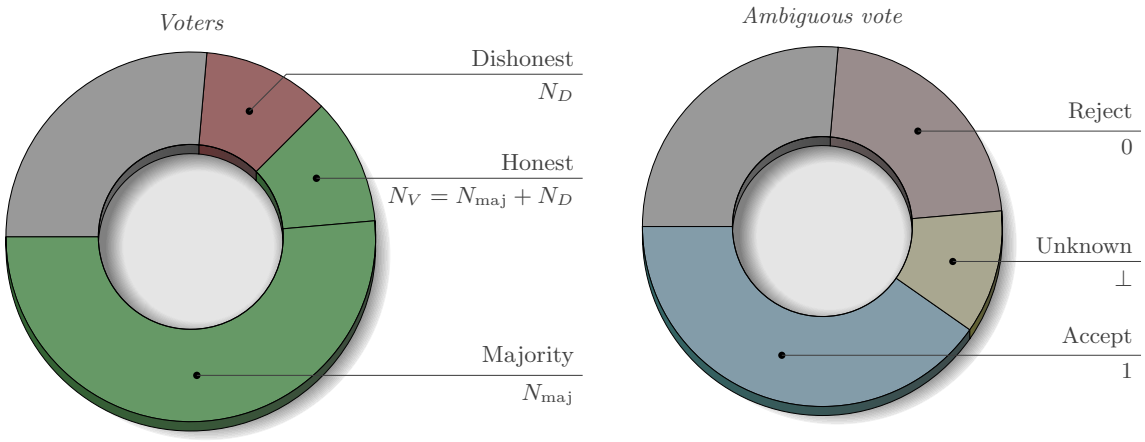


Figure 15: ???

* Use notation,

$$\vec{t}_{i,\mathcal{S}} = \{t_{i,j}\}_{j\in\mathcal{S}},$$
$$\tilde{t}_{i,\mathcal{S}} = \text{median}(\vec{t}_{i,\mathcal{S}}). \tag{5.25}$$

* For any majority subset,

$$\mathcal{S}_+ \subseteq \mathcal{S},\ |\mathcal{S}_+| > \frac{|\mathcal{S}|}{2}, \tag{5.26}$$

we have,

$$\min(\vec{t}_{i,\mathcal{S}_+}) \leq \textsc{ConsensusTime}(i,\mathcal{S}) \leq \min(\vec{t}_{i,\mathcal{S}_+}). \tag{5.27}$$

Hence, the consensus-time of a set is bounded by any constituent majority.

* A majority of nodes reporting identical times-of-receipt,

$$t_{i,j} = t_{i,k}\ \forall\ j,k \in \mathcal{S}_+, \tag{5.28}$$

forces convergence of consensus-time,

$$\textsc{ConsensusTime}(i,\mathcal{S}) = t_{i,\mathcal{S}_+}, \tag{5.29}$$

invariant under times reported by the minority, $\vec{t}_{i,\mathcal{S}_-}$.

PROTOCOL:

1. All nodes $i \in \mathcal{S}$ report their times-of-receipt of a given message,

$$\mathcal{B} \leftarrow t_i, \tag{5.30}$$

and initialise their set of recognised compliant nodes as those whose timestamps subjectively arrived on time,

$$\mathcal{S}_i^{(0)} = \{j \in \mathcal{S} \,|\, \textsc{ReceivedOnTime}_i(t_{j\to i})\}. \tag{5.31}$$

2. Nodes i record the arrival times of all timestamps announced by j they recognise as compliant ($j \in \mathcal{S}_i$),

$$\mathcal{B} \to \{t_{j\to i}\}_{j\in\mathcal{S}_i}. \tag{5.32}$$

and update their set of recognised compliant nodes accordingly,

$$\mathcal{S}_i^{(n)} = \{j \in \mathcal{S}_i^{(n-1)} \,|\, \textsc{ReceivedOnTime}_i(t_{j\to i})\}. \tag{5.33}$$

With increasing n, subjective subsets of compliant nodes are non-expanding,

$$\mathcal{S}_j^{(n)} \subseteq \mathcal{S}_j^{(n-1)}. \tag{5.34}$$

3. Nodes announce the set of received timestamps they deem compliant (for the nth round),

$$\mathcal{T}_i^{(n)}(S_i^{(n)}) = \{t_{j\to i}^{(n)}\}_{j\in S_i^{(n)}},$$
$$\mathcal{B} \leftarrow \mathcal{T}_i^{(n)}(S_i^{(n)}). \tag{5.35}$$

The median of $\mathcal{T}_i^{(n)}(S_i^{(n)})$ is node i's subjective consensus-time.

4. Nodes i and j mutually recognise the compliance of the set $\mathcal{S}_i \cap \mathcal{S}_j$. Node i interprets the relative consensus-time implied by j via their mutually recognised nodes,

$$\mathcal{T}_{j\to i}^{(n)} = \mathcal{T}_j^{(n)}(\mathcal{S}_i^{(n)} \cap \mathcal{S}_j^{(n)}). \tag{5.36}$$

5. On a pairwise basis relative consensus-times are unambiguous.

6. Nodes update their sets of timestamps to their new pairwise mutually recognised relative consensus-times,

$$t_i^{(n)} = \text{median}_j(\mathcal{T}_{j\to i}^{(n)}(\mathcal{S}_i \cap \mathcal{S}_j)) \tag{5.37}$$

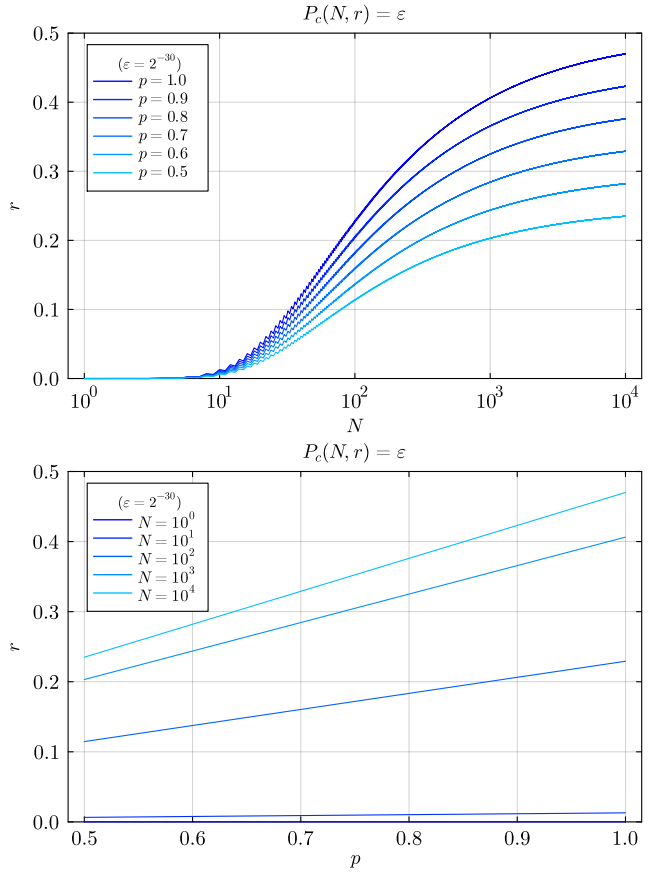


Figure 16: Security tradeoffs with consensus set size and variable network participation. $P_c(N, r/p) \leq \varepsilon$ is the probability that dishonest nodes within a random subset of size N form a false-majority when the proportion of dishonest nodes is $r' = r/p$, with network participation p. Shown for constant $\varepsilon = 2^{-30}$, where the $p = 1$ curve corresponds to the respective curve in Fig. 4.

7. (Repeat to 1): All nodes announce their updated timestampe,

$$\mathcal{B} \leftarrow t_i^{(n)}. \tag{5.38}$$

8. When an majority of nodes j report the same subjective understandings of consensus-time of i, all honest nodes necessarily converge (Eq. (5.29)).

9. Failure to converge arises when different nodes have different subjective interpretations of consensus-time, which can only occur via minority timing manipulation. This is only possible if when,

$$\mathcal{S}_j \neq \mathcal{S}_k,\ j, k \in \mathcal{S}_H. \tag{5.39}$$

10. If a majority of reported updated times (i.e subjective consensus-times) are identical convergence has been reached: TERMINATE.

* ??? TODO

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. 11).

Algorithm 11: Consensus time of a broadcast message is given by the median of the times of receipt reported by nodes.

function 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,

$$|t_i - \text{ConsensusTime}(\mathcal{C}, \texttt{message})| \leq \delta, \tag{5.40}$$

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,

$$\text{MajorityVote}(\mathcal{C}, \text{Compliant}(\texttt{message})). \tag{5.41}$$

Let the worst-case network latencies be,

$$\tau_{\max} = \max_{i,j \in \mathcal{N}} (\tau_{i,j}), \tag{5.42}$$

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,

$$t_B \leq \text{ConsensusTime}(\mathcal{C}, \texttt{message}) - \tau_{\max}. \tag{5.43}$$

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

$$\delta \geq \frac{\tau_{\max}}{2}. \tag{5.44}$$

The possible values for the median of a set of numbers $\vec{t} = \{t_i\}_i$ is discretised, limited to being one of the values t_i or the mean of two nearest ones, $(t_i + t_{i+1})/2$, where $t_{i+1} \geq t_i$ are ordered. If $\vec{t}$ are the times reported by a majority, consensus-time is similarly constrained.

Considering a set of parties with maximal dishonest minority, with $|\mathcal{N}_H|$ honest nodes and $|\mathcal{N}_D| = |\mathcal{N}_H| - 1$ dishonest nodes, such that $|\mathcal{N}| = 2 \cdot |\mathcal{N}_H| - 1$. Then we have,

* Let $\mathcal{S}_{j,k}$

* Consider an honest majority of nodes with different subjective

* Under an honest-majority assumption, if honest nodes

-

* For a network with honest majority

$$\min(\mathcal{N}_H) \leq \textsc{ConsensusTime}(\mathcal{N}_H) \leq \max(\mathcal{N}_H). \tag{5.45}$$

Including dishonest timestamps the inequality remains unchanged,

$$\min(\mathcal{N}_H) \leq \textsc{ConsensusTime}(\mathcal{N}) \leq \max(\mathcal{N}_H). \tag{5.46}$$

Given,

$$\max(\mathcal{N}_H) - \min(\mathcal{N}_H) \leq \tau_{\max}. \tag{5.47}$$

If all honest nodes $\mathcal{N}_H$ report the same time their subjective consensus times converge ($\delta \to 0$) and cannot be manipulated by any minority tactic.

Consensus time may be subjective when nodes include timestamps from different sets of nodes, which may arise when dishonest nodes manipulate message timing such that their arrival times yield ambiguous inclusion.

$$\textsc{ConsensusTime}(\{\mathcal{C}\}_i) \neq \textsc{ConsensusTime}(\{\mathcal{C}\}_j), \tag{5.48}$$

for different subjective sets of timestamps to include, $\{\mathcal{C}\}_i$ and $\{\mathcal{C}\}_j$.

1. Consensus-time convergence

All nodes i broadcast time-of-receipt of a message, $t_i^{(0)}$. Messages must be received within cutoff time T_0. Assume all honest nodes time-of-receipt are within cutoff. Dishonest nodes may manipulate their transmission timing to create subjective ambiguity in which timestamps are acknowledged by different nodes.

The consensus-time based only on honest nodes is bounded by,

$$\begin{aligned} t_H &= \textsc{ConsensusTime}(\mathcal{N}_H), \\ \min(\mathcal{N}_H) &\leq t_H \leq \max(\mathcal{N}_H). \end{aligned} \tag{5.49}$$

For all nodes it is similarly bounded under honest majority, $|\mathcal{N}_H| > |\mathcal{N}_D|$,

$$\begin{aligned} t_N &= \textsc{ConsensusTime}(\mathcal{N}_H \cup \mathcal{N}_D), \\ \min(\mathcal{N}_H) &\leq t_N \leq \max(\mathcal{N}_H) \end{aligned} \tag{5.50}$$

Following the initial announcements of recorded arrival times, $t_i^{(0)}$, all nodes update their times to their subjectively observed consensus-times,

$$t_i^{(1)} = \textsc{ConsensusTime}(\mathcal{N}(i)), \tag{5.51}$$

where $\mathcal{N}(i)$ is the set of times observed by node i, which necessarily includes the reported times of all honest nodes, dictating common upper and lower bounds across all honest subjective $t_i^{(k)}$.

Updated times are announced to the network and employed for the subsequent round. The local update rule is recursively defined,

$$t_i^{(k)} = \textsc{ConsensusTime}(\mathcal{N}^{(k-1)}(i)). \tag{5.52}$$

In the absence of any dishonest nodes, all subjective $t_i^{(1)}$ will be equivalent and consensus-time convergence is achieved.

With only honest nodes, all $t_i^{(k)}$ converge after one round. Dishonest participants inhibit convergence by creating discrepancy between subjective consensus-times, but nonetheless remain bounded between the maximum and minimum of honest nodes. This creates a deadlock scenario where convergence is prevented.

We achieve convergence in consensus-time when subjective consensus-times are stable.

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,

$$\mathcal{P}_i(\mathcal{C}). \tag{5.53}$$

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

$$\mathcal{P}(\mathcal{C}) = \bigcup_{i \in \textsc{Majority}(\mathcal{C})} \mathcal{P}_i(\mathcal{C}). \tag{5.54}$$

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.

* Variation in consensus time preserves compliance outcomes.

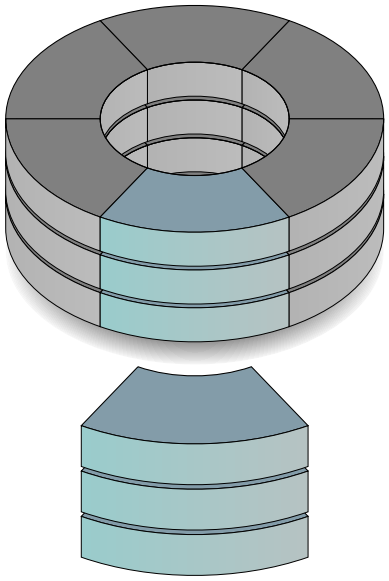


Figure 17: Proof-of-consensus.

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,

$$\texttt{tally} \in \mathbb{Z}^{|\mathcal{N}|}. \tag{5.55}$$

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 ACCEPT stage of the protocol, the network must also agree on the network's updated `tally` state.

* Policy: propose update.
* Accept current constitution.

F. Resource consumption

* Latency limits speed.
* Multiplier via independent parallel sub nets or more virtual nodes.
* Inter subnet balancing of load into single.
* Digital signature consumption.

VI. QUANTUM CONSENSUS NETWORKS

Quantum consensus networks (QCNs) 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.C) to inject entropy into their own global keys using *entropy addition* (Sec. VI.B), 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.D).
- Interactive proofs of quantumness: require quantum computation but no quantum communication (Sec. VI.E).

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 contributing QRN is genuine, combining them under bit-wise XOR yields a collective QRN source.

Independent of the underlying two-party QRNG protocol, QCNs operate as follows:

1. All nodes execute two-party QRNG with all other nodes, a total of $n(n-1)$ QRNG rounds.
2. All n nodes commit their versions of their QRNG outcomes with all other $n-1$ nodes.
3. A corresponding *compliance graph* is implied by the committed data. This may be realised by any entity observing the published data.
4. A graph reduction algorithm eliminates graph edges associated with inconsistent QRNG outcomes, followed by eliminating nodes not connected by a majority of edges.
5. The resultant graph is a unique fully-connected subgraph representing the unanimous-majority outcome (Sec. VI.D.1).

A. Entropy sources: quantum vs. classical

* Entropy rate

$$H(\mathcal{X}) = \lim_{n\to\infty} H(\mathcal{X}_n|\mathcal{X}_{n-1},\ldots,\mathcal{X}_1), \tag{6.1}$$

where $H(X|Y)$ is the conditional entropy of X given knowledge of Y.

TO DO:

* What is quantum random vs classical random.

* Correlations over repeats?

For iid (quantum)

$$H(\mathcal{X}) = H(\mathcal{X}_n) \tag{6.2}$$

B. Entropy addition

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

$$\mathcal{X}_i \to \{0, 1\}, \tag{6.3}$$

with probability distributions,

$$\mathcal{X}_i \sim \text{Ber}(p_i) : \; p_i = p_i(0) = 1 - p_i(1), \tag{6.4}$$

the individual binary Shannon entropies are given by,

$$H_2(\mathcal{X}_i) = -p_i \log_2 p_i - (1 - p_i) \log_2(1 - p_i), \tag{6.5}$$

where,

$$0 \leq H_2(\cdot) \leq 1, \tag{6.6}$$

the entropy of the random variable given by their bit-wise XOR,

$$\mathcal{X} = \bigoplus_i \mathcal{X}_i, \tag{6.7}$$

is lower-bounded by the maximum entropy of the contributing random variables,

$$\max_i(\{H_2(\mathcal{X}_i)\}) \leq H_2(\mathcal{X}) \leq 1. \tag{6.8}$$

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

Note that hash functions do not exhibit the entropy addition property of the bitwise XOR operation and cannot be employed in this context.

C. Quantum random oracles

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

$$\mathcal{X}_\mathcal{O}(t) = \bigoplus_{i \in \mathcal{O}(t)} \mathcal{X}_i(t), \tag{6.9}$$

whose combined entropy is bounded by,

$$\max_{i \in \mathcal{O}}(\{H_2(\mathcal{X}_i)\}) \leq H_2(\mathcal{X}_\mathcal{O}) \leq 1. \tag{6.10}$$

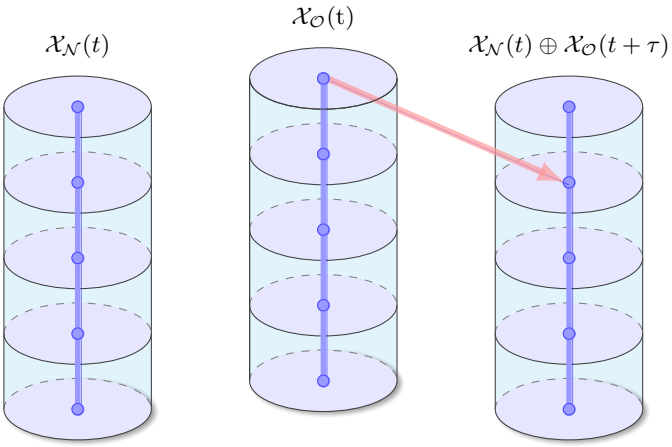


Figure 18: Quantum random oracles. A classical DCN establishing hash-based secure random source $\mathcal{X}_\mathcal{N}(t)$, where t denotes time, may add entropy from a QCN acting as oracle for quantum random source $\mathcal{X}_\mathcal{O}(t)$. To maintain security $\mathcal{X}_\mathcal{N}$ must be established in advance of $\mathcal{X}_\mathcal{O}$, achieved by adding entropy from a QRN outcome at pre-agreed future time $t+\tau$, $\tilde{\mathcal{X}}_\mathcal{N}(t) = \mathcal{X}_\mathcal{N}(t) \oplus \mathcal{X}_\mathcal{O}(t+\tau)$.

A classical DCN may observe a QRNG oracle and add its entropy $\mathcal{X}_\mathcal{O}$ to its own global key. For this to be secure it is required that the DCN's own global key, $\mathcal{X}_\mathcal{N}$, be committed prior to the availability of the external entropy source,

$$\tilde{\mathcal{X}}_\mathcal{N}(t+\tau) = \mathcal{X}_\mathcal{N}(t) \oplus \mathcal{X}_\mathcal{O}(t+\tau), \tag{6.11}$$

where $\tilde{\mathcal{X}}_\mathcal{N}$ is the network's oracle-modulated global key, and $\tau > 0$ is a pre-agreed future point in time, subsequent to commitment of the networks initially established global key, $\mathcal{X}_\mathcal{N}$.

D. 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,

$$|\Psi\rangle_{i,j} = \frac{1}{\sqrt{2}}(|0\rangle_i |0\rangle_j + |1\rangle_i |1\rangle_j), \tag{6.12}$$

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,

$$b_i(n), b_j(n) \in \{0, 1\}, \tag{6.13}$$

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

$$m_i(n), m_j(n) \in \{0, 1\}. \tag{6.14}$$

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

$$s_{i,j} = \{m(n) \,|\, b_i(n) = b_j(n)\}_n. \tag{6.15}$$

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.

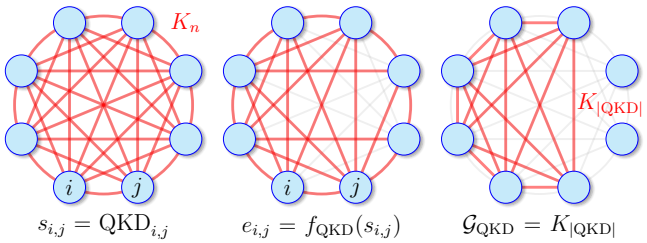


Figure 19: QKD proof-chain for shared quantum randomness. Amongst n nodes with full $O(n^2)$ quantum communications connectivity given by the complete graph K_n, every node executes a QKD protocol with every other node, associating a shared random bit-string $s_{i,j}$ (received by i from j) with every edge. Bit-strings passing a QRN certification function $f_{\mathrm{QKD}}(s_{i,j})$ define the edges of a subgraph, where certification acts as an implied vote of honesty. Maintaining only vertices connected to a majority of other nodes followed by eliminating those not connected to every other, we obtain a complete subgraph $\mathcal{G}_{\mathrm{QKD}}$ reflecting the unanimous majority.

1. Consensus protocol

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

$$f_{\mathrm{QKD}}(s_{i,j}) \to \{0, 1\}, \tag{6.16}$$

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,

$$\mathcal{G}^{(\mathrm{comp})}_{\mathrm{QKD}} : e_{i,j} = f_{\mathrm{QKD}}(s_{i,j}). \tag{6.17}$$

Letting the QKD compliance of nodes be,

$$\mathcal{G}^{(\mathrm{comp})}_{\mathrm{QKD}} : v_i = \textsc{Majority}\left(\bigcup_{j \neq i} e_{i,j}\right), \tag{6.18}$$

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[9], achieved by eliminating all vertices in $\mathcal{G}^{(\mathrm{comp})}_{\mathrm{QKD}}$ not connected by an edge to every other node,

$$\mathcal{G}_{\mathrm{QKD}} : v_i = \begin{cases} 1 & \text{if } |u_i \in \mathcal{G}^{(\mathrm{comp})}_{\mathrm{QKD}}| = |\mathcal{G}^{(\mathrm{comp})}_{\mathrm{QKD}}| - 1 \\ 0 & \text{otherwise} \end{cases}. \tag{6.19}$$

The fully connected $\mathcal{G}_{\mathrm{QKD}} = K_{|\mathcal{G}_{\mathrm{QKD}}|}$ subgraph now represents the accepted subset of QKD-compliant nodes under consensus. The associated collectively established shared random bit-string is defined as,

$$s(\mathcal{G}_{\mathrm{QKD}}) = \bigoplus_{v_i, v_j \in \mathcal{G}_{\mathrm{QKD}}} s_{i,j}. \tag{6.20}$$

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 $f_{\mathrm{PP}}(\vec{b}, \vec{m}) \to \vec{s}$.

E. 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.

[9] 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.

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,

$$f_{\mathcal{I}}(x) \to w, \tag{6.21}$$

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,

$$f(x_0) = f(x_1) = w, \tag{6.22}$$

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[10].

For matrix,

$$A \in \mathbb{Z}_q^{m \times n}, \tag{6.23}$$

and vectors,

$$x, y, s, e \in \{0,1\}^n, \tag{6.24}$$

related by,

$$y = A \cdot s + e, \tag{6.25}$$

under modulo q arithmetic where q is prime, a TCF may be constructed as,

$$f_{\mathcal{I}}(b, x_b) = \lfloor A \cdot x + b \cdot y \rceil, \tag{6.26}$$

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

$$x_0 = x_1 + s. \tag{6.27}$$

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. 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 via a Hadamard transform,

$$|\psi_H\rangle = \hat{H}^{\otimes n} |0\rangle^{\otimes n} = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle . \tag{6.28}$$

[10] 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. Evaluating $f_{\mathcal{I}}(x)$ into an output register yields[11],

$$|\psi_{\mathcal{I}}\rangle = \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle\,|f_{\mathcal{I}}(x)\rangle\,,$$

which may be efficiently prepared using a quantum circuit with,

$$O(n^2 \log^2 n), \tag{6.29}$$

gate count (Kahanamoku-Meyer *et al.*, 2022).

4. The prover measures the output register, obtaining measurement outcome w which is communicated to the verifier. Measuring w collapses the x-register onto the equal superposition of associated satisfying pre-images,

$$|\psi_w\rangle = \frac{1}{\sqrt{2}}(|x_0\rangle + |x_1\rangle)\,|w\rangle\,. \tag{6.30}$$

As the x-register was initialised into a uniform superposition over all length-n bit-strings and the TCF is a 2-to-1 function, w is sampled uniformly at random.

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

6. When measuring in the $b = 0$ ($\hat{Z}$) basis, the prover randomly measures either $m = x_0$ or $m = x_1$, easily verified by direct evaluation of $f(x) \to w$ and comparison with the prover's previously reported w. When measuring in the $b = 1$ ($\hat{X}$) basis verification succeeds if $m \cdot x_0 = m \cdot x_1$. The verification rules are,

$$\begin{aligned} \hat{Z}\,(b=0): &\quad m = \{x_0, x_1\}, \\ \hat{X}\,(b=1): &\quad m \cdot x_0 = m \cdot x_1. \end{aligned} \tag{6.31}$$

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

The key observation is that since $\hat{X}$ and $\hat{Z}$ 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 $p = 1/2^n$ 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. (6.18) are now made from the verifier perspective.

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.

---

[11] The unitarity of quantum circuits prohibits direct evaluation of classical functions on quantum registers in general,

$$\hat{U}_f\,|x\rangle \not\to |f(x)\rangle\,,$$

where $\hat{U}_f$ denotes a quantum circuit evaluating classical function $f(\cdot)$. Introducing ancillary quantum register $|y\rangle$ affords the reversible classical transformation $(x, y) \leftrightarrow (x, y \oplus f(x))$, which may be implemented unitarily in general,

$$\hat{U}_f\,|x\rangle\,|y\rangle \to |x\rangle\,|y \oplus f(x)\rangle\,.$$

Considering a single output bit of $f(\cdot)$, $\hat{U}_f$ admits the decomposition,

$$\hat{U}_f = \hat{\Pi}_0 \otimes \hat{I} + \hat{\Pi}_1 \otimes \hat{X},$$

where,

$$\hat{\Pi}_i = \sum_{x\,|\,f(x)=i} |x\rangle\,\langle x|\,,$$

are projectors onto the subspaces of x satisfying $f(x) = i$ (Nb: $\hat{\Pi}_0 + \hat{\Pi}_1 = \hat{I}$, $\hat{\Pi}_0 \cdot \hat{\Pi}_1 = 0$). The unitarity of $\hat{U}_f$ follows, independent of $f(\cdot)$,

$$\hat{U}_f^\dagger \cdot \hat{U}_f = (\hat{\Pi}_0 \otimes \hat{I} + \hat{\Pi}_1 \otimes \hat{X})^2 = \hat{I}.$$

Repeating for all output bits and letting $y = 0$ yields,

$$\hat{U}_f\,|x\rangle\,|0\rangle \to |x\rangle\,|f(x)\rangle\,.$$

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. (6.16),

$$f_{\text{IPQ}}(\mathcal{I}_i, \mathcal{T}_i, w_{i,j}, b_i, m_{i,j}) \to \{0, 1\}, \tag{6.32}$$

similarly defines a directed IPQ compliance graph,

$$G^{\text{comp}}_{\text{IPQ}} : e_{i,j} = f_{\text{IPQ}}(\mathcal{I}_i, \mathcal{T}_i, w_{i,j}, b_i, m_{i,j}) \to \{0, 1\}. \tag{6.33}$$

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 equilibrating 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,

$$N_{\min}(r, \varepsilon) = N^{(n)}_{\min}(r, \varepsilon^n), \tag{8.6}$$

which implies,

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

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. 20). 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.

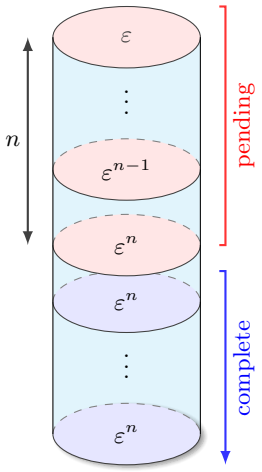


Figure 20: Blockchain with n-level reverse integrity checking. Upon addition of the top-level block consensus requires verifying integrity going back n blocks. If consensus has ε-security, for $i \leq n$ the ith past block will have been verified i times, exhibiting cumulative ε^i-security. Blocks $i \geq n$ 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.

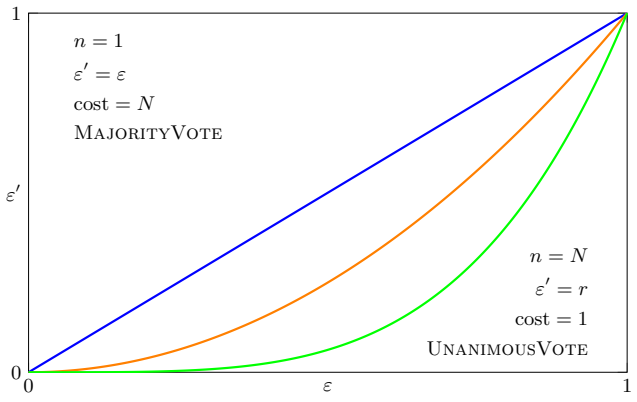


Figure 21: Tradeoffs in blockchain integrity with retrospective consensus verification.

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.

* Consider asymmetric bidding if preserving r is not a consideration.

E. Distributed signature authorities

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

IX. STRATEGIC CONSIDERATIONS

??? TO DO

Fig. 23

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

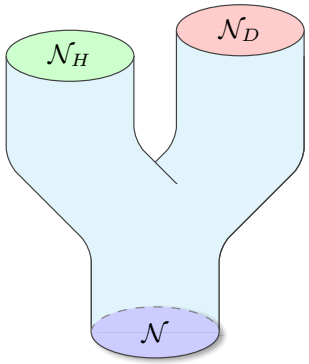


Figure 22: Bifurcation in network trust. When the network $\mathcal{N} = \mathcal{N}_H \cup \mathcal{N}_D$ supporting a blockchain segregates into two non-interacting networks, $\mathcal{N}_H$ and $\mathcal{N}_D$, a fork is created, forming two unique, legitimate blockchains, one associated with each network.

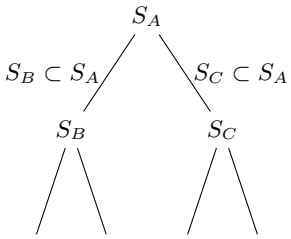


Figure 23: Trust hierarchies represented as subset-trees define retreat strategies.

X. CONCLUSION

ACKNOWLEDGMENTS

REFERENCES

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

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.

Hakimi, S L (1962), "On realizability of a set of integers as degrees of the vertices of a linear graph. I," Journal of the Society for Industrial and Applied Mathematics **10**, 496.

Havel, Václav (1955), "A remark on the existence of finite graphs," Časopis Pro Pěstování Matematiky **080**, 477.

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," .

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), *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.

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.

YCharts, (2024), "Bitcoin network hash rate," .

Zhu, Daiwei, Gregory D. Kahanamoku-Meyer, Laura Lewis, Crystal Noel, Or Katz, Bahaa Harraz, Qingfeng Wang, Andrew Risinger, Lei Feng, Debopriyo Biswas, Laird Egan, Alexandru Gheorghiu, Yunseong Nam, Thomas Vidick, Umesh Vazirani, Norman Y. Yao, Marko Cetina, and Christopher Monroe (2023), "Interactive cryptographic proofs of quantumness using mid-circuit measurements," Nature Physics **19**, 1725.