Proof of Work Without All the Work

Diksha Gupta[1], Jared Saia[2], and Maxwell Young*[3]

[1]Dept. of Computer Science, University of New Mexico, NM, USA
dgupta@unm.edu
[2]Dept. of Computer Science, University of New Mexico, NM, USA
saia@cs.unm.edu
[3]Computer Science and Engineering Dept., Mississippi State University, MS, USA
myoung@cse.msstate.edu

Abstract

Proof-of-work (PoW) is a popular algorithmic tool used to enhance network security by imposing a computational cost on participating devices. Unfortunately, traditional PoW schemes require that correct devices work perpetually in order to remain vigilant against *possible* malicious behavior. In other words, computational costs are unbounded regardless of whether the network is under attack. This shortcoming is highlighted by recent studies showing that PoW is highly inefficient with respect to operating cost and ecological footprint.

We address this issue by designing a general PoW scheme for securing open distributed systems, whereby the cost to devices grows slowly as a function of the cost incurred by an attacker. Consequently, if the network is attacked, our scheme guarantees security, with algorithmic costs that are commensurate with the cost of the attacker. Conversely, in the absence of attack, algorithmic costs are small.

Our results hold in a dynamic system where participants join and depart over time. We demonstrate how our PoW scheme can be leveraged to address important security problems in distributed computing including: Sybil attacks, Byzantine consensus, and Committee election.

*This work is supported by the National Science Foundation grant CCF 1613772 and by a research gift from C Spire.

1 Introduction

Twenty-five years after its introduction by Dwork and Naor [33], ***proof-of-work (PoW)*** is enjoying a research renaissance. Originally, PoW was conceived of as a technique for preventing malicious users from acquiring more than their "fair share" of a system resource such as bandwidth[1] or a server's computational power. In recent years, PoW provides a critical primitive for cryptocurrencies such as Bitcoin [74], along with other blockchain technologies such as Ethereum [36], BlockStack [12], and Chain Incorporated [51, 81].

Yet, despite success with BitCoin and its analogs, PoW has not fulfilled its promise of mitigating a wider range of malicious behaviors such as application-layer distributed denial-of-service (DDoS)[2] and Sybil attacks [31]. These are well-known and enduring security problems for which PoW seems well-suited, and yet proposals [15, 20, 46, 54, 63, 67, 82, 93, 94] built around PoW have yet seen only limited deployment.

A Barrier to Widespread Use. A major impediment to the widespread use of PoW is "the work". Current PoW schemes require a significant expenditure of resources to secure a system, *even when the system is not under attack*.

Cryptocurrency systems have cleverly provided monetary incentives for performing the computational work necessary to ensure PoW-based security. They have been extremely successful, both in terms of practical impact [25,32,35,49,66], and research impact [14, 37, 45, 74, 91].

However, the perpetual resource burning inherent to systems like Bitcoin is undesirable for several reasons. First, *energy consumption:* in 2015, the Economist calculated that Bitcoin consumes at least 1.46 terawatt-hours of electricity per year, or enough to power $135,000$ American homes [34]. This has significant environmental and economic impact that will increase as technologies like Bitcoin become more widely used. Second, *scalability:* high energy consumption prevents using current PoW approaches on the many other large open systems that also require security. Third, *feasibility:* in networks of battery-powered devices – for example, in many ad-hoc wireless settings – energy must be used sparingly, so current PoW systems are simply infeasible. Finally, *security:* when the mechanism that provides security is expensive, agents will likely selfishly seek to reduce costs and thereby compromise security.

[1]Dwork and Naor were motivated to reduce spam email.

[2]For example, see [20, 46, 54, 67, 82, 93, 94].

In light of these problems, we seek to reduce the cost of PoW systems and focus on the following question: **Can we design PoW systems where the resource costs are low in the absence of attack, and grow commensurately with the effort expended by an attacker?**

In this paper, we design and analyze algorithms that answers this question in the affirmative. Initially, our result is presented in the context of the Sybil attack; however, it applies more generally to safeguarding a distributed system from any attack where an adversary seeks to obtain significantly more than its fair share of network resources. We formalize this guarantee in Theorem 1 of Section 1.3, and we later elaborate on the application of our result in more general scenarios in Section 5.

1.1 Related Work on Sybil Attacks

There is a large body of literature on mitigating Sybil attacks (for example, see surveys [53, 71, 78]). Due to space constraints, we summarize only closely related results. *Critically, none of these prior results have the desired property that the resource costs to the good IDs grow commensurately with the cost incurred by the adversary.*[3]

Most relevant to our work are prior results that employ testing:

1. PoW via *computational testing* [15] is a popular technique and has been used to safeguard open systems in [58, 63, 92]. Unfortunately, in these prior works, nodes must continually issue puzzles – for example, every 5 seconds in the experiments of [63] – in order to protect against potential attacks. That is, even in the absence of an attack, the system is always in a "hyperalert" state leading to significant computational waste. This leads to the good IDs incurring a cost that is linear in the lifetime of the system, not linear in the amount spent by the adversary.

2. In wireless settings, *radio-resource testing* is a well-known defense [73, 78]. Radio-resource testing has been employed in a setting with a single resource provider [43] and later extended to decentralized systems [42]. However, these approaches suffer the same drawback; the system must constantly perform tests to guarantee security which leads to wasted bandwidth in the absence of attack. Again, the good IDs incur a cost that is linear in the lifetime of the system, not linear in the adversary's cost.

[3]We note that this can be characterized as a ***resource-competitive*** approach [8,41]. However, due to space constraints, we omit a discussion of resource-competitive algorithms since it is not critical to understanding our results.

Finally, as an alternative to testing approaches, there has been significant work on leveraging social networks to mitigate Sybil attacks [62, 70, 95, 98, 99]. However, information on social-network connections may not be accessible, or may not even exist (also, interestingly, more recent work argues that topological assumptions necessary to some of these defenses may not hold in practice [97]). A similar idea is to establish unique IDs using geographical constraints [5, 87] or the physics of wireless signals [30,40,64]. Containment strategies – where good IDs are insulated to an extent from bad IDs – are examined in overlays [28, 86].

1.2 Our Model

Attack Model. We consider a system composed of physical ***nodes*** and virtual ***identifiers (IDs)***. A ***good node*** is defined by having at most one ID; this is referred to as a ***good ID***. Any node with more than a single ID is assumed to be under the control of an adversary who controls these corresponding ***bad IDs***.

Speaking of a single adversary – rather than a collection of adversarial nodes – allows us to address the worst-case scenario where such nodes can communicate and collude perfectly in order to launch attacks on the system. We emphasize that the adversary may create as many IDs as it wishes.[4]

The adversary is assumed to be computationally bounded so that we can make use of public key cryptography. We emphasize however that **our solutions do not require any public key (PKI)**. That is, nodes generate their own key pair, but we do not require a certificate authority to bind a node to its public key, nor do we require any type of key management system.

At any time, we assume that the adversary has computational power at most a $1/\alpha$ fraction of the computational power of the good nodes, where $\alpha > 2$ is some known constant. That is, the adversary has less computational power than the cumulative computational power held by the good nodes by a factor of α; this is a standard assumption for PoW schemes [65, 68, 75]. Note that we are not assuming we know the adversary's computational power exactly, only a (possibly loose) upper bound.

System Dynamism. The system is dynamic with IDs joining and departing over time, and time is discretized into ***rounds***. We pessimistically assume that all join/departure events are scheduled in a worst-case fashion by an adversary. Good IDs are assumed to tell the server when they depart, while bad IDs may decide to leave without any notification.

The minimum number of good IDs in the system at any point is assumed to be at least $\boldsymbol{n_0}$. Our goal is to provide security and performance guarantees for

[4]In practice, for example, an attacker can spoof IP and MAC addresses.

$\boldsymbol{N = O(n_0^{\gamma})}$ joins and departures of IDs, for any desired constant $\gamma \geq 1$. In other words, the guarantees on our system hold with high probability (w.h.p.)[5] over this polynomial number of dynamic events.

1.3 Main Results

Our main results are summarized below. We measure ***computational cost*** as effort required to solve computational puzzles (see Section 1.4 for details), and we measure ***bandwidth cost*** as the number of messages sent and received. The ***lifetime*** of the system is defined as the time until at least N join or leave events.

Theorem 1. *Let T_C and T_B denote the total computational and bandwidth costs, respectively, incurred by the adversary. Let g_{new} denote the number of good IDs that have joined the system. Then, w.h.p.* COM^2 *has the following properties:*

- *The fraction of good IDs always exceeds at least $1/2$.*
- *The cumulative computational cost to the good IDs is $O(T_C + g_{new})$.*
- *The cumulative bandwidth cost to the good IDs is $O(T_B + g_{new})$.*

There are two important ramifications of Theorem 1. First, the good IDs obtain a large, constant fraction of the system resources. In particular, there is nothing special about a $1/2$-fraction and this can be increased as we increase the value of α in our attack model.

Second, the computational and bandwidth costs incurred by the good IDs grow slowly with the cost incurred by the adversary. Recalling our discussion in Section 1, this is precisely the type of result we sought. When there is no attack on the system, the costs are low and solely a function of the number of good IDs; there is no exorbitant overhead. But as the adversary spends more to attack the system, the costs required to keep the system secure grow commensurately with T_C and T_B.

1.4 Computational Puzzles

All nodes have access to a hash function, $\boldsymbol{h}$, about which we make the standard *random oracle assumption* [7,17,59]. Succinctly, this assumption is that when first computed on an input, x, $h(x)$ is selected independently and uniformly at random from the output domain, and that on subsequent computations of $h(x)$ the same output value is always returned. We assume that both the input and output domains are the real numbers between 0 and 1. In practice, h may be a cryptographic hash function, such as SHA-2 [80], with inputs and outputs of sufficiently large bit lengths.

[5]With probability at least $1 - n_0^{-c}$ for any desired $c > 0$.

In general, a node must find an input x such that $h(x)$ is less than some threshold. Decreasing this threshold value will increase the difficulty, since one must compute the hash function on more inputs to find an output that is sufficiently small.[6]

We assume that each node can perform μ hash-function evaluations per round for $\mu > 0$. Additionally, we assume that μ is of some size polynomial in n_0 so that $\log \mu = \Theta(\log n_0)$.[7]

For any integer $\rho \geq 1$, we define a ***ρ-round puzzle*** to consist of finding $\ell = C \log \mu$ solutions, each of difficulty $\tau = \rho(1-\delta)\mu/(C \log \mu)$, where $\delta > 0$ is a small constant and C is a sufficiently large constant depending on δ and μ.

Let X be a random variable giving the expected number of hash evaluations needed to compute ℓ solutions. Then X is a negative binomial random variable and we have the following concentration bound (see, for example, Lemma 2.2 in [4]).

For every $0 < \epsilon \leq 1$, it holds that

$$Pr(|X - E(X)| \geq \epsilon E(X)) \leq 2e^{-\epsilon^2 \ell/(2(1+\epsilon))}$$

Given the above, we can show[8] that every good ID will solve a ρ-round puzzle with at most $\rho\mu$ hash function evaluations, and that the adversary must compute at least $(1-2\delta)d\rho\mu$ hash evaluations to solve every ρ-round puzzle. Note that for small δ, the difference in computational cost is negligible, and that μ is also unnecessary in comparing costs. Thus, for ease of exposition, **we assume that each ρ-round puzzle requires computational cost ρ to solve**.Finally, a node v uses a ***public key K_v***, generated via public key encryption, as its ID. The input to a puzzle always incorporates K_v and s, where the ***solution string s*** is selected by v (for good IDs, s will be a random string).

Uses of Puzzles. In our algorithms, puzzles are used in two distinct ways (although, they are constructed in the same manner). First, when a new ID wishes to join the system, it must provide a solution for a β-round puzzle; this is Step (1) in the pseudocode of Figure 1. Here, the input to the puzzle is $K_v||s$. We note that, in the case of a bad ID, this solution may have been precomputed by the adversary. This is not a problem since the purpose of this puzzle is only to force the adversary to incur a computational cost (it is not used to preserve guarantees on the fraction of good IDs in the system).

[6]Many other types of puzzles exist (for examples, see [52, 55, 84]) and our results are likely compatible with other designs.

[7]It is reasonable to assume large μ since, in practice, the number of evaluations that can be performed per second is on the order of millions to low billions [9, 10, 50].

[8]By using a union bound over the N joins and departure events, for C sufficiently large as a function of δ and γ.

Algorithm 1: COM^2

```
Input. The following sets are defined:
$\mathcal{S}_{\text{old}} \leftarrow$ set of IDs present at beginning of current epoch
$\mathcal{S}_{\text{t}} \leftarrow$ set of IDs present at time $t$

Execute the following steps for the lifetime of the system:
 1. Upon joining, each ID $v$ solves a $\beta$-round puzzle and sends
    the solution $s_v$ to the server which adds $v$ to $\mathcal{S}_{\text{t}}$ upon confirming the validity
    of $s_v$
 2. For any round $t$, if $|(\mathcal{S}_{\text{t}} \cup \mathcal{S}_{\text{old}}) - (\mathcal{S}_{\text{t}} \cap \mathcal{S}_{\text{old}})| \geq |\mathcal{S}_{\text{old}}|/3$, then the server :
        • Issues a 1-round puzzle to each ID via broadcasting random string $r$
        • $\mathcal{S}_{\text{old}} \leftarrow$ set of IDs that returned valid solution
        • $\mathcal{S}_{\text{t}} \leftarrow \mathcal{S}_{\text{old}}$
```

The second way in which puzzles are used is to limit the fraction of bad IDs in the system; see Step (2) in Figure 1. An announcement is periodically made that *all* IDs *already in the system* should solve a puzzle. When this occurs, a random string, $\boldsymbol{r}$, of $\Theta(\log N)$ bits is generated and included as part of the announcement. The value r must also be appended to the inputs for all requested solutions in this round; that is, the input is $\boldsymbol{K_v}||\boldsymbol{s}||\boldsymbol{r}$. These random bits ensure that the adversary cannot engage in a pre-computation attack — where it solves puzzles and stores the solutions far in advance of launching an attack — by keeping the puzzles unpredictable. For ease of exposition, we omit further discussion of this issue and consider the use of these random bits as implicit whenever this second type of puzzle is issued.

While the same r is used in the puzzle construction for all IDs, we emphasize that a *different* puzzle is assigned to each ID since the public key used in the construction is unique. Again, this is only of importance to the second way in which puzzles are used. Using the public key in the puzzle construction also prevents puzzle solutions from being stolen. That is, ID K_v cannot lay claim to a solution found by ID K_w since the solution is tied to the public key K_w.

2 A Centralized Defense

To simplify the presentation of our result, we begin by demonstrating a centralized solution. To this end, we consider a ***server*** which can communicate directly with each ID in the system. We emphasize that our assumption of a server is removed later in Section 3.

Our first algorithm *Commensurate Computation* ($\textsc{Com}^2$) is described in Algorithm 1. In this algorithm, each ID that wishes to join the system and receive service is required to solve a β-round puzzle in order to register with the server. We refer to this as an ***entrance fee***. These puzzles are of the first type described in Section 1.4 (which do not require a "time stamp" value r); we reiterate that the only purpose of this puzzle Step (1) is to force a computational cost.

The server tracks the membership in the system using the set $\boldsymbol{\mathcal{S}}_{\text{t}}$. Whenever an ID registers with the server, $\mathcal{S}_{\text{t}}$ is updated. Similarly, when a good ID informs the server that it is departing, $\mathcal{S}_{\text{t}}$ is also updated. However, bad IDs may not provide such a notification and, therefore, $\mathcal{S}_{\text{t}}$ is not necessarily accurate at all times.

At the beginning of the system, the server knows the existing membership denoted by $\mathcal{S}_{\text{old}}$; assume $|\boldsymbol{\mathcal{S}}_{\text{old}}| = \boldsymbol{n_0}$ initially. At some point, $|(\mathcal{S}_{\text{t}} \cup \mathcal{S}_{\text{old}}) - (\mathcal{S}_{\text{t}} \cap \mathcal{S}_{\text{old}})| \geq |\mathcal{S}_{\text{old}}|/3$. When this happens, it triggers the execution of Step 2 whereby all IDs are issued a 1-round puzzle and must respond with a valid solution within 1 round. The issuing of these puzzles is performed by the server by announcing r to all IDs. A round is of sufficient length that the computation time to solve a 1-round puzzle dominates the round trip communication time between the client and server.

The server verifies a puzzle solution by checking that (1) all $C \log \mu$ inputs to h submitted generate an output that is at most $\frac{C \log \mu}{(1-\delta)\mu}$; and (2) each of these inputs contains the string r and also the individual public key of the ID. Those IDs that fail to submit valid puzzle solutions during step 2 are de-registered and permanently blacklisted — that is, they are effectively removed from the system. The actions in Step 2 are referred to as a ***test***. Over the lifetime of the system, the execution of $\textsc{Com}^2$ is conceptually broken into sets of consecutive rounds called ***epochs*** which are delineated by tests. The process outlined above is repeated where now the server knows the existing membership *exactly*, $\mathcal{S}_{\text{old}}$, at the beginning of each epoch (thus, $\mathcal{S}_{\text{t}}$ is set to equal $\mathcal{S}_{\text{old}}$ at this point in Figure 1).

Initialization. Prior to the first epoch, the server issues puzzles to all IDs, and sets variables by running the bullets under step 2 of Algorithm 1. Thus, initially $\mathcal{S}_{\text{old}}$ contains less than a $1/3$ fraction of bad nodes.

2.1 Analysis

For any epoch i, we let B_i (respectively G_i) denote the number of bad (respectively good) IDs in the system at the end of epoch i, and we let $N_i = B_i + G_i$. Recall that $B_0 < N_0/3$.

Lemma 1. *For all* $i \geq 0$, $B_i < N_i/3$.

Proof. This is true by assumption for $i = 0$. For $i > 0$, note that the adversary has computational power less than $G_i/2$. Thus, after Step 2 that ends epoch i, we have $B_i < G_i/2$. Adding $B_i/2$ to both sides of this inequality yields $3/2B_i < N_i/2$, from which, $B_i < N_i/3$. $\square$

Let n_i^a, g_i^a, b_i^a denote the total, good, and bad IDs that arrive over epoch i. Similarly, let n_i^d, g_i^d, b_i^d denote the total, good, and bad IDs that depart over epoch i.

Note that the server will always have an accurate value for all of these variables except for possibly b_i^d — recall from Section 1.2 that bad IDs do not need to notify the server when they depart — and, consequently, n_i^d; in these two cases, the server may hold values which are underestimates of the true values.

Lemma 2. *The fraction of bad IDs is always at most* $1/2$.

Proof. Fix some epoch $i > 0$. Recall that an epoch ends when $|(\mathcal{S}_{\text{t}} \cup \mathcal{S}_{\text{old}}) - (\mathcal{S}_{\text{t}} \cap \mathcal{S}_{\text{old}})| \geq |\mathcal{S}_{\text{old}}|/3$ where $|\mathcal{S}_{\text{old}}| = N_{i-1}$. Therefore, we have $b_i^a + g_i^d \leq N_{i-1}/3$. We are interested in the maximum value of the ratio of bad IDs to total IDs at any point during the epoch. Thus, we pessimistically assume all additions of bad IDs and removals of good IDs come first in the epoch. We are then interested in the maximum value of the ratio:

$$\frac{B_{i-1} + b_i^a}{N_{i-1} + b_i^a - g_i^d}.$$

By Lemma 1, $B_{i-1} \leq N_{i-1}/3$. Thus, the we want to find the maximum of $\frac{N_{i-1}/3 + b_i^a}{N_{i-1} + b_i^a - g_i^d}$, subject to the constraint that $b_i^a + g_i^d \leq N_{i-1}/3$. This ratio is maximized when the constraint achieves equality, that is when $g_i^d = N_{i-1}/3 - b_i^a$. Plugging this back into the ratio, we get

$$\begin{aligned} \frac{N_{i-1}/3 + b_i^a}{N_{i-1} + b_i^a - g_i^d} &\leq \frac{N_{i-1}/3 + b_i^a}{2N_{i-1}/3 + 2b_i^a} \\ &= 1/2 \end{aligned}$$

Therefore, the maximum fraction of bad IDs is $1/2$ at any point during any arbitrary epoch $i > 0$.

Finally, we note that this argument is valid even though $\mathcal{S}_{\text{t}}$ may not account for bad IDs that have departed *without* telling the server (recall this is possible as stated in Section 1.2). Intuitively, this is not a problem since such departures can only lower the fraction of bad IDs in the system; formally, the critical equation in the above argument is $b_i^a + g_i^d \leq N_{i-1}/3$, and this does not depend on b_i^d. $\square$

Lemma 3. *For any $i > 0$, the total cost to the good IDs in epoch i is at most $\beta\, g_i^a + 3\left(n_i^a + n_i^d\right)$.*

Proof. The cost to each good ID in epoch i arises from (1) the entrance fee paid by newly arrived IDs, and (2) the test that marks the end of epoch i. This aggregate cost is thus at most:

$$\begin{aligned}\beta\, g_i^a + G_i &\leq \beta\, g_i^a + N_i\\ &\leq \beta\, g_i^a + N_{i-1} + (n_i^a - n_i^d)\\ &\leq \beta\, g_i^a + 4n_i^a + 2n_i^d\end{aligned}$$

The last inequality follows from noting that $\left(n_i^a + n_i^d\right) \geq N_{i-1}/3$ for a test to occur; again, note that this holds even if bad IDs do not alert the server that they are departing over epoch i (as n_i^d is an upper bound on the true number of IDs that have departed). $\square$

Lemma 4. *For all $j \geq 0$, the total cost to the good IDs over j epochs is at most $(\beta + 6)\left(\sum_{i=0}^{j} g_i^a\right) + 6T_j$*

Proof. The total cost to the good IDs is the sum of the cost paid by the good IDs in each epoch. In other words, for any $j \geq 0$, the total cost is:

$$\begin{aligned}\sum_{i=0}^{j}(\beta\, g_i^a + G_i) &\leq \sum_{i=0}^{j}\left(\beta\, g_i^a + 4n_i^a + 2n_i^d\right)\\ &\leq \sum_{i=0}^{j}\beta\, g_i^a + 4\sum_{i=0}^{j} n_i^a + 2\sum_{i=0}^{j} n_i^d\\ &\leq \sum_{i=0}^{j}\beta\, g_i^a + 6\sum_{i=0}^{j} n_i^a\\ &\leq (\beta+6)\sum_{i=0}^{j} g_i^a + 6T_j\end{aligned}$$

Where the third line follows since $\sum_{i=0}^{j} n_i^a \geq \sum_{i=0}^{j} n_i^d$. In the last inequality, we denote the cost paid by the adversary by T_j. Again, this holds even if bad IDs do not alert the server that they are departing over epoch i, since n_i^d is an upper bound on the true number of IDs that have departed. $\square$

We now give the proof for Theorem 1:

Proof. By Lemma 2, w.h.p. the fraction of bad IDs is less than $1/2$, and so the first claim follows. The computational cost follows directly from Lemma 4. The bandwidth cost follows by noting that each good ID must send a message (to the server) upon joining the system and also each time it solves a puzzle. $\square$

3 A Distributed Version of COM^2

We now consider defending against attacks in the absence of a server. We spread the server's responsibilities over a subset of the IDs, called a ***committee***, and this gives rise to new challenges:

- First, there is no longer a central authority that can be trusted to conduct a computational test. Therefore, this test must be done in a distributed fashion.
- Second, while a server can unilaterally decide whether an ID is good or bad, this determination now needs to be made and agreed upon by the committee members.
- Third, while the correctness of the server is not impacted by joins and departures, the correctness of the committee may be compromised if the fraction of good IDs becomes too small.

Our distributed version of COM^2 overcomes these challenges through a combination of techniques embodied by the two protocols: COMMITTEE MAINTENANCE, which is analogous to COM^2, and MEMBERSHIP ELECTION, which deals with the rebuilding the committee. The details of our approach are described below; however, we first discuss the system model.

System Model. As before, the bad IDs can collude to maximize their gains and are under the control of an adversary with computational power at most a $1/\alpha$ fraction of the computational power of the good nodes. We assume that $\alpha \geq 5$ (see proof of Lemma 5).

Again, the minimum number of good IDs in the system at any point is assumed to be at least $\boldsymbol{n_0}$ and we aim to provide guarantees for $\boldsymbol{N = O(n_0^{\gamma})}$ joins and departures by IDs, for any desired constant $\boldsymbol{\gamma} \geq 1$.

Over time, committees are disbanded and rebuilt; the details are presented later. An ***epoch*** begins when a new committee is completed and begins to be used (the old committee being disbanded). We maintain the same notation: for any epoch i, B_i and G_i denote the number of bad and good IDs in the system at the end of epoch i, respectively, and we let $N_i = B_i + G_i$. We assume that at the time of system creation, the fraction of bad IDs is less than $3/10$; in particular, $B_0 < (3/10)N_0$ (this aligns with the result proved in Lemma 5).

Our models for communication and joins/departures of IDs are described below.

Communication Model. We now change our communication model somewhat to align more closely with the related literature. All IDs may communicate using a diffuse primitive, denoted by **BCAST**, which allows a good ID to send a value to all other good IDs within a known bounded constant amount of time, despite the presence of an adversary. The assumption of such a "blackbox" protocol is common in the literature on PoW schemes such as Bitcoin [11, 39, 65, 68]. Moreover, empirical studies suggest that the Bitcoin overlay, for example, allows for such a communication protocol [69]. For ease of exposition, we assume that the time to diffuse a message is negligible when compared to the time to solve computational puzzles.

Can a message m_v from a committee member ID K_v be spoofed? This is prevented in the following manner. ID K_v signs m_v with its private key to get $\texttt{sign}_v$, and then sends $(m_v||\texttt{sign}_v||K_v)$ via BCAST. Any other ID can use K_v to check that the message was signed by the ID K_v and thus be assured that ID K_v is the sender. This technique is implicitly used in MEMBERSHIP ELECTION to send a system-wide random string r; however, for succinctness, we omit the details from the pseudocode.

We again mention that while our solution employs public/private key pairs, *no public key infrastructure is required* for the same reasons as given in Section 1.2.

Joins and Departures. As before, the system size may vary wildly over time, increasing and decreasing polynomially in n_0 above a minimum number of n_0 good IDs.

We pessimistically assume that the adversary schedules the joins and departures by IDs in the system. The adversary clearly knows whether an ID is good or bad, and can inform its scheduling decisions based on this information. However, the adversary is *oblivious* to which IDs belong to a committee.[9]

Unlike our model in Section 2, we do *not* assume that a good ID can make its departure known explicitly. This corresponds to a more challenging scenario where the adversary can abruptly eject good IDs from the system by causing them to crash or fail via a network attack (for example, a DDoS attack).

We do impose a loose constraint on the rate of departures: at most an ϵ_0-fraction of good IDs may depart in any single round, where $\boldsymbol{\epsilon_0 > 0}$ is a small, known constant. This assumption is necessary since otherwise an extremely rapid

[9]One can imagine a variety of attacks aimed at causing good nodes/IDs to fail, and the array of techniques available for mitigating such attacks in the literature. Such security issues are important against an adversary that might target committee members, but only complementary to the focus of our work here; thus, we omit further discussion of them.

number of departures can leave the system bereft of a functioning committee. Note that this constraint still allows for an amount of dynamism that is linear in the current system size.

Committee Construction. In constructing a new committee, there is a sequence of ***membership intervals*** $\left(\frac{1}{2^{k/d}}, \frac{1}{2^{(k+1)/d}}\right]$ for $\boldsymbol{k \geq 1}$ and where $\boldsymbol{d}$ is a constant we can set subject to $d \geq 20\gamma$ (this is required for the proof of Lemma 5). For each interval, the ID that generates the smallest hash function output in the k^{th} interval becomes the k^{th} committee member. This determination is handled by the current committee; the details are given in COMMITTEE MAINTENANCE and MEMBERSHIP ELECTION (see Algorithms 2 and 3).

Given that the adversary has a $1/\alpha$-fraction of the computational power held by the good IDs, we expect the newly-formed committee to have a majority of good IDs. This is argued formally in Section 3.1.

Algorithm Overview and Intuition. In our distributed version of COM^2, the role of the server is now taken over by a committee. Over the lifetime of the system, this committee should always satisfy two invariants: (1) a majority of the IDS are good and (2) it has size $\Theta(\log n_0)$.

As joins and departures occur, these two invariants may become endangered. In response, a new committee is built from scratch. The details of this construction are described below and in the pseudocode of Algorithms 2 and 3.

Note that we are only describing/analyzing the construction and maintenance of the committee in this section. This committee can replace the server in the algorithm of Section 2 (pseudocode in Algorithm 1), and all prior cost analysis remains valid.

Throughout, let $\boldsymbol{\mathcal{C}_{\text{old}}}$ denote the set of IDs who are members of the committee at the beginning of the current epoch (which the formation of this new committee delineates). Let $\boldsymbol{\mathcal{C}_t}$ denote the set of IDs forming the committee at any time t in the current epoch. A new committee needs to be constructed whenever the number of departures from the committee is at least $|\mathcal{C}_{\text{old}}|/3$.

When this condition is met, the members of $\mathcal{C}_t$ agree on and diffuse (via BCAST) a random string r as in Section 2. Agreement can be accomplished via any secure multi-party computation technique for generating random bits; for example, the result in [89] suffices.

How is r sent such that each good ID trusts it came from the committee? Recall that a node v that joins the system generates a public key, K_v, and the corresponding private key, k_v; the public key is used as its ID. If ID K_v is a committee member, it will sign the random string r using its private key k_v to obtain $\texttt{sign}_v$. Then, $(r||\texttt{sign}_v||K_v)$ is sent using BCAST. Any good ID can verify $\texttt{sign}_v$ via

K_v to ensure it returns r. This implicitly occurs in Step 2 of COMMITTEE MAINTENANCE, but we omit the details in the pseudocode for simplicity.

A good ID K_v that is a member of the committee is assumed to be issuing "heartbeat" messages; these are periodic announcements that the ID is still present in the system. As with r, these messages are signed. We assume that the absence of a heartbeat message for some sufficiently large time indicates to all IDs that the corresponding committee member has departed. To simplify our presentation, we omit further mention of this aspect.

MEMBERSHIP ELECTION (Algorithm 3) In MEMBERSHIP ELECTION, each good ID, K_v must evaluate $h(K_v||s||r)$ throughout a *single* round using randomly chosen strings s. Of these evaluations, the string s_v corresponding to the *smallest output found by* K_v is BCAST as the message $(K_v||s_v)$ by ID K_v. Note that there is no need to sign these messages; only messages originating from the committee need to be signed.[10]

A good ID K_w that receives such a message checks whether $h(K_v||s||r)$ yields the smallest output it has seen so far during this round and, if so, K_w tentatively sets K_v as the k^{th} committee member and propagates $(K_v||s_v)$ as dictated by BCAST; otherwise, the message is discarded. By this process, if s_v is the smallest value in some membership interval, then the entire system (including K_v) learns this via BCAST, and each good ID will consider K_v to belong to the committee being constructed. Note that this means that all good IDs eventually agree on the membership of the committee.

Over the round during which this construction is taking place, the existing committee still satisfies the invariants (1) and (2) above (this is proved in Lemma 5). In this way, the system is always *live*; that is, a functioning committee is always available. After the round completes, the new committee members are known to all good IDs in the system. Those "old" committee members that were present prior to this construction are no longer recognized by the good IDs; we say the old committee is disbanded. Finally, any messages that are received after this single round are considered late and discarded.

Initialization. Prior to the first epoch, our algorithm is initialized as follows. First, a global random string r is generated; this can be done using a "heavy-weight" algorithm such as that of [2]. Next, we run MEMBERSHIP ELECTION using this random string r as the seed for the required puzzles.

[10]Recall that a solution cannot be stolen given that it is tied to the ID that obtained it via the puzzle construction (recall our discussion in Section 1.2). The adversary might (bizarrely) obtain a solution for a puzzle that incorporates a good ID K_v and then diffuse $(K_v||s_v)$, but that does not help the adversary.

Algorithm 2: COMMITTEE MAINTENANCE

```
Input. The following sets are defined:
$\mathcal{C}_{\text{old}} \leftarrow$ IDs in committee at beginning of the current epoch
$\mathcal{C}_t \leftarrow$ IDs in committee at time $t$ of the current epoch

Execute the following steps for the lifetime of the system:
1. Each new ID generates a public key, $K_v$, and private key, $k_v$
2. At any time $t$, if $|\mathcal{C}_t| \leq \frac{2}{3}|\mathcal{C}_{\text{old}}|$, then the following is executed:
    • $\mathcal{C}_t$ uses BCAST to diffuse a random string $r$
    • $\mathcal{C}_{old} \leftarrow$ MEMBERSHIP ELECTION
    • $\mathcal{C}_t \leftarrow \mathcal{C}_{old}$
```

Algorithm 3: MEMBERSHIP ELECTION

```
Each good ID $K_v$ executes the following for a single round:
1. Perform hash-function evaluations with $K_v||s||r$ as input; let $s_v$ be the input
   for the smallest solution found in this round.
2. Use BCAST to diffuse $(K_v||s_v)$
3. Upon receiving any $(K_w||s_w)$ from some ID $K_w$:
    • If $h(K_w||s_w||r)$ is the smallest value received in the interval
      $\left(\frac{1}{2^{k/d}}, \frac{1}{2^{(k+1)/d}}\right]$ for some $k \geq 1$, then tentatively set $K_w$ to be the $k^{\text{th}}$
      member of the committee; and propagate $(K_w||s_w)$.
    • At the end of the round, set the current tentative members of the committee
      to the final members.
```

3.1 Analysis of Distributed COM^2

In this section, we prove that COMMITTEE MAINTENANCE and MEMBERSHIP ELECTION maintain the invariants that the committee has (1) a majority of good IDs and (2) size $\Theta(\log N_i)$ for any epoch i over the lifetime of the system. Furthermore, we derive bounds on the bandwidth and computational costs of these algorithms.

3.2 Good Majority and Committee Size

To simplify our presentation, the claims made throughout this section are proved to hold with probability at least $1 - O(1/n_0^{\gamma+2})$. Of course, we wish the claims of Theorem 1 to hold with probability at least $1 - 1/n_0^{\gamma+1}$ such that a union bound over n_0^{γ} joins and departures yields a w.h.p. guarantee. By providing this "slack" of an $\Omega(1/n_0)$-factor in each of the guarantees of this section, we demonstrate this is feasible while avoiding an analysis cluttered with specific settings for the constants used in our arguments.

Lemma 5. *The following properties hold with probability at least* $1 - O(1/n^{\gamma+2})$, *for all* $t > 0$*:*

- $\mathcal{C}_{old}$ *has more than a* $7/10$*-fraction of good IDs,*
- $\mathcal{C}_t$ *has more than a* $1/2$*-fraction of good IDs,*
- $|\mathcal{C}_t| = \Theta(\log n_0)$,

for $\alpha \geq 5$ *and* $d \geq 35\gamma$.

Proof. Recall that for an ID K_v to become a committee member, it must obtain the *smallest* value in $\left(\frac{1}{2^{k/d}}, \frac{1}{2^{(k+1)/d}}\right]$ for the integer $k \geq 1$ during the single round of MEMBERSHIP ELECTION.

Let the indicator random variable $X_{v,k} = 1$ if ID K_v finds a solution in the k^{th} membership interval, although *not* necessarily the smallest value in this interval; otherwise, $X_{v,k} = 0$.

Let the set of good IDs present at the beginning of epoch i be denoted by $\mathcal{G}$. These are the good IDs that execute MEMBERSHIP ELECTION. Let the random variable $X_{\mathcal{G},k} = \sum_{K_{v,k} \in \mathcal{G}} X_{v,k}$ which counts the number of hash-function evaluations by good IDs that land in the k^{th} membership interval.

Let the set of all bad IDs present at any point in epoch i be denoted by $\mathcal{B}$. Define $X_{\mathcal{B},k} = \sum_{K_{v,k} \in \mathcal{B}} X_{v,k}$ which counts the number of hash-function evaluations by bad IDs that land in the k^{th} membership interval.

Expected Value Calculation. We have:

$$E[X_{\mathcal{G},k}] = E\left[\sum_{K_{v,k} \in \mathcal{G}} X_{v,k}\right] \geq |\mathcal{G}| \frac{\mu}{2^{k/d}} = G_{i-1} \frac{n_0^{\ell}}{2^{k/d}}$$

for some constant $\ell \geq 1$ since μ is a polynomial in n_0 (recall Section 1.2). Similarly:

$$E[X_{\mathcal{B},k}] = E\left[\sum_{K_v \in \mathcal{B}} X_{v,k}\right] \leq |\mathcal{G}| \frac{\mu}{\alpha 2^{(k-1)/d}} = G_{i-1} \frac{n_0^{\ell}}{\alpha 2^{(k-1)/d}}$$

Concentration Bounds on $X_{\mathcal{G},k}$. Let $\eta = G_{i-1}\mu/(\kappa(\gamma+2)\ln n_0)$ for a sufficiently large constant $\kappa > 0$. Since the $X_{v,k}$s are independent, a Chernoff bound tightly bounds $X_{\mathcal{G},k}$ such that for a small constant $\delta > 0$, we have:

$$\begin{aligned} Pr\left(X_{\mathcal{G},k} < (1-\delta)\frac{G_{i-1}\mu}{2^{k/d}}\right) &\leq Pr\left(X_{\mathcal{G},k} < (1-\delta)\frac{G_{i-1}n_0^{\ell}}{2^{k/d}}\right) \\ &\leq \exp\left(-\frac{\delta^2 G_{i-1}\, n_0^{\ell}}{2^{(k/d)+1}}\right) \end{aligned}$$

noting that $\mu = n_0^{\ell}$. Consider k over the range from 1 to $d \lg \eta$:

$$\begin{aligned} \exp\left(-\frac{\delta^2 |\mathcal{G}|\, n_0^{\ell}}{2^{(k/d)+1}}\right) &\leq \exp\left(-(\delta^2/2)\kappa(\gamma+2)\ln n_0\right) \\ &\leq O\left(1/n_0^{\gamma+2}\right) \end{aligned}$$

for a sufficiently large constant $\kappa \geq 2/\delta^2$.

Therefore, the number of hashes by good IDs that fall in the k^{th} interval is at least $(1-\delta)G_{i-1}\mu/2^{k/d}$ with probability at least $1 - O(1/n_0^{\gamma+2})$. A similar calculation yields upper bounds $X_{\mathcal{G},k}$:

$$Pr\left(X_{\mathcal{G},k} > (1+\delta')|\mathcal{G}|\mu/2^{k/d}\right) \leq O\left(1/n_0^{\gamma+2}\right)$$

Concentration Bounds on $X_{\mathcal{B},k}$. Over the range of $1 \leq k \leq d \lg \eta$, a similar argument proves that for a small constant $\delta'' > 0$, $X_{\mathcal{B},k} = (1 \pm \delta'')G_{i-1}\mu/\alpha 2^{(k-1)/d}$ with probability at least $1 - O(1/n_0^{\gamma+2})$.

However, can the adversary obtain outputs that belong to membership intervals for $k > d\lg(\eta)$? In such intervals, we expect only a small number of outputs and we pessimistically assume that, if one exists, it belongs to the adversary (not to a good ID).

Each such membership interval has size less than $1/(2^j\eta)$ for $j \geq 1$. With probability at least $1 - O(1/n_0^{\gamma+2})$, the adversary will not obtain an output in any interval for $j \geq (\gamma+2)\lg\eta$. Thus, the adversary gains at most $(\gamma+2)\lg\eta$ additional committee members.

Size of Committee. We note that the same argument used above to bound the number of bad IDs for membership intervals with $k > d\lg(\eta)$ also applies to good IDs. By this fact, and the above bounds, the total size of the committee is:

$$d\lg(\eta) \leq |\mathcal{C}_{\text{old}}| \leq d\lg(\eta) + 2(\gamma+2)\lg\eta$$

since G_{i-1} and μ are each a polynomial in n_0, $|\mathcal{C}_{\text{old}}| = \Theta(\log n_0)$ with probability at least $1 - O(1/n^{\gamma+2})$. Given that the committee size decreases by at most $|\mathcal{C}_{\text{old}}|/3$ in an epoch, it follows that $|\mathcal{C}_{\text{t}}| = \Theta(\log n_0)$.

Fraction of Good IDs. Each output that falls in the k^{th} membership interval has the same probability as any other of being the smallest. Therefore, pulling the above pieces together: with probability at least $1 - O(1/n^{\gamma+2})$, the probability that a good ID obtains the smallest output in the k^{th} interval for $1 \leq k \leq d\lg(\eta)$ is at least:

$$\frac{(1 \pm \epsilon)G_{i-1}\mu/2^{k/d}}{(1 \pm \epsilon)G_{i-1}\mu/2^{k/d} + (1 \pm \epsilon')G_{i-1}\mu/\alpha 2^{(k-1)/d}} \geq \frac{(1-\epsilon'')\alpha}{\alpha + 2^{1/d}}$$

for a small constant ϵ'' depending on small constants $\epsilon, \epsilon' > 0$.

To obtain a lower bound on the number of good IDs in $\mathcal{C}_{\text{old}}$, define the indicator random variable $Y_k = 1$ if a good ID has the smallest output in k^{th} interval; otherwise, $Y_k = 0$. Let $Y = \sum_{k=1}^{d\lg(\eta)} Y_k$, noting our sum is up to the *lower bound* on $|\mathcal{C}_{\text{old}}|$,.

$$\begin{aligned} E[Y] &= E\left[\sum_{k=1}^{d\lg(\eta)} Y_k\right] = \sum_{k=1}^{d\lg(\eta)} E[Y_k] \\ &\geq \sum_{k=1}^{d\lg(\eta)} \frac{(1-\epsilon'')\alpha}{\alpha+2^{1/d}} = \left(\frac{(1-\epsilon'')\alpha}{\alpha+2^{1/d}}\right) d\lg\eta \end{aligned}$$

We now set $d \geq 20\gamma$ and make two observations. First, by a Chernoff bound, with probability at least $1 - O(1/n_0^{\gamma+2})$, the number of good IDs in $\mathcal{C}_{\text{old}}$ is at least $\frac{(1-\epsilon''')\alpha}{\alpha+2^{1/20\gamma}}(20\gamma)\lg\eta$ for a small constant $\epsilon''' > 0$.

Second, we derived above that $|\mathcal{C}_{\text{old}}| \leq (1 \pm \delta'')d\lg(\eta) + 2(\gamma+2)\lg\eta$. Therefore, the fraction of good IDs in $\mathcal{C}_{\text{old}}$ is:

$$\begin{aligned} &\geq \frac{(1-\epsilon''')\frac{\alpha}{\alpha+2^{1/d}}(d\lg\eta)}{(1+\delta'')(d)\lg(\eta) + 2(\gamma+2)\lg\eta} \\ &\geq (1-\varepsilon)\left(\frac{\alpha}{\alpha+2^{1/d}}\right)\left(\frac{d}{d+2\gamma+4}\right) \\ &> 7/10 \end{aligned}$$

for $\alpha \geq 5$, $d \geq 35\gamma$, and a sufficiently small constant $\varepsilon > 0$ depending on ϵ''' and δ''. This holds with probability at least $1 - O(1/n_0^{\gamma+2})$.

Over the epoch, at most $|\mathcal{C}_{\text{old}}|/3$ IDs can depart prior to MEMBERSHIP ELECTION being executed. In the worst case, these are all good IDs. Since for any positive quantities X and Y, $\frac{X-(Y/3)}{(2/3)Y} = (3/2)(X/Y) - (1/2)$, the fraction of good IDs is at least $11/20$, and this holds with probability at least $1 - O(1/n_0^{\gamma+2})$.

Finally, recall that at most a constant ϵ_0-fraction of good IDs may depart over the single round in which MEMBERSHIP ELECTION is executing (see our model for joins and departures described in Section 3). By a Chernoff bound, for a suitably small $\epsilon_0 < 1/20$, the fraction of good IDs that leave the committee is tightly bounded such that the fraction of good IDs remaining in the committee exceeds $1/2$ with probability at least $1 - O(1/n_0^{\gamma+2})$ during this final round. Therefore, for the current epoch, a majority of good IDs exists in the committee until MEMBERSHIP ELECTION completes and a new committee is then available. $\square$

3.3 Cost Analysis

When computing the bandwidth cost for the decentralized network, we assume that the good nodes are connected in a sparse overlay network. In particular, for epoch i, we assume that the number of edges in this network is $\tilde{O}(G_i)$. Then we have the following lemma.

Lemma 6. *Let T_C and T_B denote the total computational and bandwidth costs, respectively, incurred by the adversary. Let g_{new} denote the number of good IDs that have joined the system. Then* COMMITTEE MAINTENANCE *(Algorithm 2) has costs as follows*

- *The total computational cost to the good nodes is $O\left(T_C + g_{new}\right)$.*
- *The total bandwidth cost to the good nodes is $\tilde{O}\left(T_B + g_{new}\right)$.*

Proof. (Sketch) We begin with computational cost. In epoch i, the computational cost to the good nodes occurs only during the call to MEMBERSHIP ELECTIONat the end of the epoch, and equals G_i, which is no more than $G_{i-1} + g_i^a$.

By the end of epoch i, we know that a $1/3$ fraction of the IDs in $\mathcal{C}_{old}$ leave. Let $Cost_i$ be the computational cost to the algorithm in epoch i, and let T_i be the computation cost spent by the adversary in epoch i. There are two cases.

Case 1: At least a $1/6$ fraction of the IDs that leave the committee are good. Since the good IDs in the committee are a random sample of the good IDs in the system, $g_i^d = \theta(G_{i-1})$, with high probability. Thus, $Cost_i \leq c_1 g_i^d + g_i^a$ for some constant c_1.

Case 2: At least a $1/6$ fraction of the IDs that leave the committee are bad. Then, at the beginning of the epoch, the adversary must have incurred $\theta(G_{i-1})$ computational cost in order to obtain positions in the committee for this constant fraction of bad IDs. Thus, $T_{i-1} = \theta(G_{i-1})$, and so $Cost_i \leq c_2 T_{i-1} + g_i^a$ for some constant c_2.

Combining these two cases, we have that for every epoch i, it is the case that $Cost_i \leq c_1 g_i^d + c_2 T_{i-1} + g_i^a$. By summing over all epochs, we get that the total computational cost to the algorithm is no more than the following.

$$\sum_i c_1 g_i^d + c_2 T_{i-1} + g_i^a \leq c_2 T_C + \sum_i (c_1+1) g_i^a$$
$$= O\left(T_C + g_{\text{new}}\right)$$

The communication costs are dominated by the cost of the call to MEMBERSHIP ELECTIONduring epoch i for each positive i. Recall that we assume the nodes are connected in a network with $\tilde{O}(G_i)$ edges.[11] A record breaking analysis shows that each edge in this network will send an expected $\theta(\log G_i)$ messages during the call to MEMBERSHIP ELECTION.[12] Thus the total bandwidth cost during the call to MEMBERSHIP ELECTIONis $\tilde{O}(G_i)$.

Since this is within logarithmic factors of the computational cost, we can apply the same analysis as for the computational cost to achieve the bound given in the lemma. □

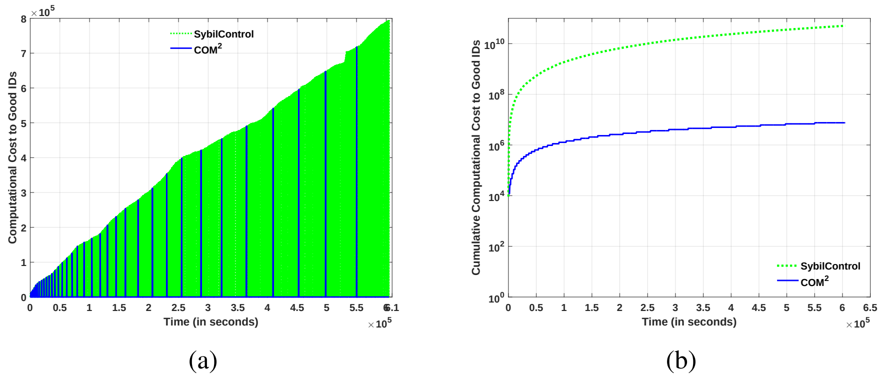


Figure 1: Computational cost and cumulative computational for good IDs in absence of an attack using the Bitcoin Dataset.

4 Empirical Validation

We simulate COM^2 to evaluate its performance against a well-known PoW scheme named SYBILCONTROL [63]. The experiments allow us to (1) compare the computational cost incurred by the good IDs and (2) investigate the extent to which a Sybil Attack can impact the fraction of good IDs under these two algorithms.

[11]The notation $\tilde{O}$ indicates that polylog(n_0) factors are omitted.

[12]The source of each edge perceives a random sequence of at most $\theta(G_i)$ values, and in expectation $\theta(\log G_i)$ of these will be the smallest seen so far, and so will merit a message along the edge. See [47] for details.

We simulate the centralized version of COM^2. Our goal is to offer a proof of concept, not a full-fledged comparison. In this vein, we simulate the centralized algorithm as it greatly reduces the number of parameters that we need to correctly estimate. Because the distributed version has asymptotically equal resource costs, we expect similar performance for the distributed algorithm.

Finally, note that we performed Monte-Carlo Simulations for generating each set of plots, where each observed value was averaged over 20 separate simulations.

Overview of SYBILCONTROL. Under this algorithm, the number of bad IDs are limited through the use of computational puzzles. Each ID must solve a puzzle to join the system. Additionally, each ID periodically (every 5 seconds) tests its neighbors with a puzzle, removing those IDs from its list of neighbors that failed to provide a solution within a time limit. It may be the case that an ID is a neighbor to more than one ID and thus, receives multiple puzzles to solve simultaneously; in this case, they are combined into a single puzzle whose solution satisfies all the received puzzles. We implement our own simulation of the SYBILCONTROL algorithm.

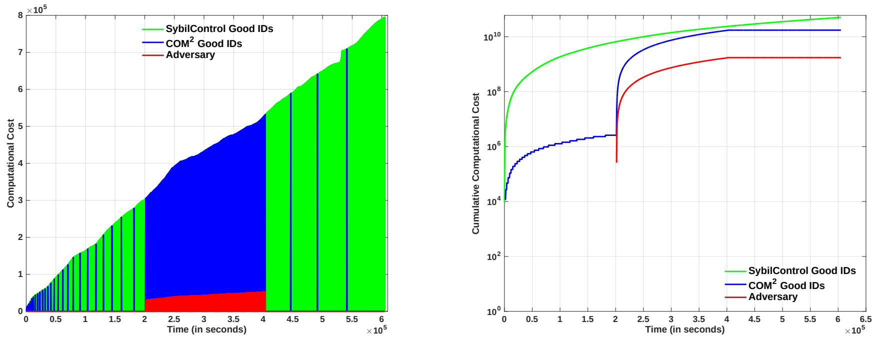
Points of Comparison. To compare both algorithms fairly, we assume that the computational cost for solving a single PoW is 1 for both algorithms. Since in both algorithms require that a new ID solves a puzzle to join the system, we refrain from measuring this computational cost. We let the fraction of computational power of the adversary, $\frac{g}{\alpha}$ be the same for both algorithms.

We first examine performance on real-world Bitcoin data [76] [77] in Section (4.1). In Section (4.2), we present a comparative study on three real-world peer-to-peer networks, namely - BitTorrent, Skype and Bitcoin [90] [48] [76] [77] . Next, we study the effect of joins and departures on the two algorithms presented in Section (4.3), where we simulate session lengths governed by Weibull Distribution. Finally, we execute a large-scale Sybil attack on a static system, initially consisting of all good IDs and present our results in Section (4.4).

4.1 The Bitcoin Network

We simulate SYBILCONTROL and COM^2 on a real-world dataset for the Bitcoin Network [76] [77].This Bitcoin Dataset spans a period of one week and consists of timestamps for join/departure events. In our experiments, we assume $\alpha = 10$ and investigate the computational cost under two scenarios: (1) in absence of bad IDs and (2) when bad IDs are present (i.e., the system is under attack).

We simulate scenario (1) by assuming all the events in the dataset are result of good IDs joining/departing the system. Figure 1a shows the computational cost to the good IDs for SYBILCONTROL and COM^2 in the absence of an attack using the Bitcoin Dataset. In Figure 1a, the dense green area signifies the high frequency of computational cost paid by good IDs in SYBILCONTROL in comparison to the increasingly-spaced blue plot for COM^2. Also, from Figure 1b, it can be seen that the cumulative computational cost to the good IDs differs by greater than 4 orders of magnitude just after 13 hours of simulation and this gap keeps widening with time. This empirical observation strengthens our belief in the cost efficiency of COM^2 in comparison to SYBILCONTROL.



(a) (b)

Figure 2: Computational cost and cumulative computational cost versus time under a large-scale attack using the Bitcoin Dataset.

Figure 2 shows the empirical results for the second scenario, where we simulate a large-scale Sybil attack using the Bitcoin Dataset. The attack is orchestrated using the following adversarial strategy. Every 5 seconds, the adversary adds $\frac{n}{3}$ bad IDs to the network from time $\frac{t}{3}$ to $\frac{2t}{3}$, where n is the total number of IDs at the time and $t = 60, 4970$ seconds (7 days) is the total execution time. Given that the adversary adds $\frac{n}{3}$ IDs every 5 seconds, a computational test is triggered in both algorithms every 5 seconds.

In Figure 2a, the dense green region represents the computational cost paid by the good IDs in SYBILCONTROL, the dense blue region represents that paid by good IDs in COM^{2}and the red region by the adversary. Due to the setup of the stress test, the computational cost paid by good IDs in both the algorithms from $\frac{t}{3}$ to $\frac{2t}{3}$, is equivalent, thus the blue region overlaps the green. The cumulative cost to the good IDs in SYBILCONTROL, COM^2 and the adversary can be compared through Figure 2b, which shows that the cost incurred in COM^2 is less than that to SYBILCONTROL and also, COM^2's cost is a function of the cost paid by the adversary, unlike SYBILCONTROL.

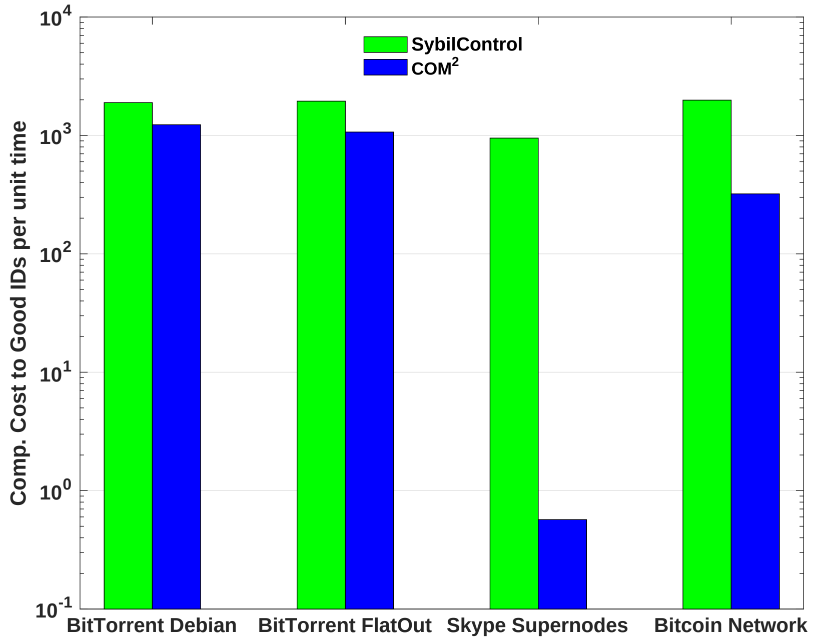


Figure 3: Computational Cost per unit Time for Peer-to-Peer Networks.

4.2 Peer-to-Peer Networks

We next compare the performance of the two algorithms on three different peer-to-peer (P2P) networks, namely BitTorrent, Skype and Bitcoin [90] [48] [76] [77]. For the BitTorrent network, we performed experiments on two BitTorrent servers, namely - BitTorrent Debian and BitTorrent Flatout. The shape (k) and scale (λ) parameter values of Weibull Distribution for session time of $k = 0.38$ and $\lambda = 42.2$ for Debian, and $k = 0.59$ and $\lambda = 41.9$ for FlatOut. The Skype supernodes had also have a Weibull Distribution for session time with median session time of 5.5 hours, and shape parameter of 0.64. We generate the session time for 10,000 good IDs from these parameter values and sample for the bitcoin network from the real-world data obtained from [76] [77]. For all empirically generated data, 1 unit of time corresponds to 1 second.

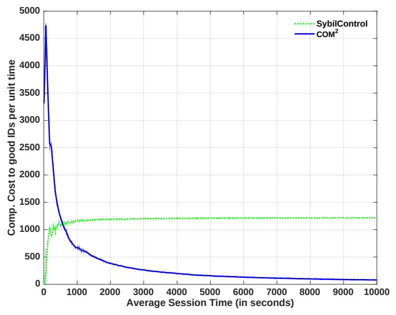
We simulate SYBILCONTROL and COM^2 on each of these networks, and compare the computational cost to the good IDs per unit time. Figure 3 illustrates the results of the simulations. We observe that COM^2 outperforms SYBILCONTROL in all three peer-to-peer systems in terms of computational costs of the network. COM^2 outperforms SYBILCONTROL by 34.51% in BitTorrent Debian, by 45.56% in BitTorrent FlatOut, by 99.94% in Skype Supernodes and 83.88% in

Bitcoin.[13]

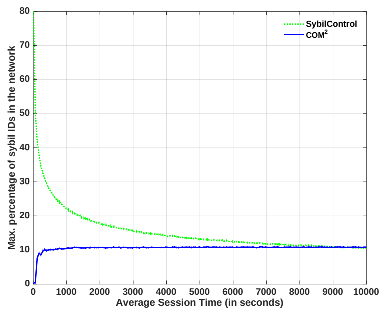
4.3 Effect of Joins and Departures

Empirical studies of session lengths in peer-to-peer network show Weibull Distribution [96] as a better fit on real time data in comparison to the popularly-used Poisson Distribution [90]. Hence, we used the Weibull distribution to generate the session lengths of IDs.

The average session length ranged from as low as 0.1 seconds to 10000 seconds. The system was initialized with 10000 good IDs and the simulations were carried out until 15000 new IDs joined the network. Each new ID could be good or bad with probability 0.5. For each value of average session length, we took the mean of our observations over 30 runs.



(a)



(b)

Figure 4: Computational Cost per unit time and Maximum Percentage of bad IDs in system versus Average Session length (Churn)

As can be seen from the plots, with increase in the average session length (order of hours), COM^2 is able to give similar guarantees as SYBILCONTROL in terms of the maximum percentage of bad IDs in the system at much lower computation cost to the good IDs (See Figure 4). On the other hand, when session length are small (order of minutes), COM^2 is able to maintain a constant percentage of bad IDs in the system whereas SYBILCONTROL is not able to give any such guarantees (the percentage of bad IDs can become unbounded).

[13]We use the following definition of performance: $(\frac{\text{COM}^2}{\text{SYBILCONTROL}} - 1) * 100$

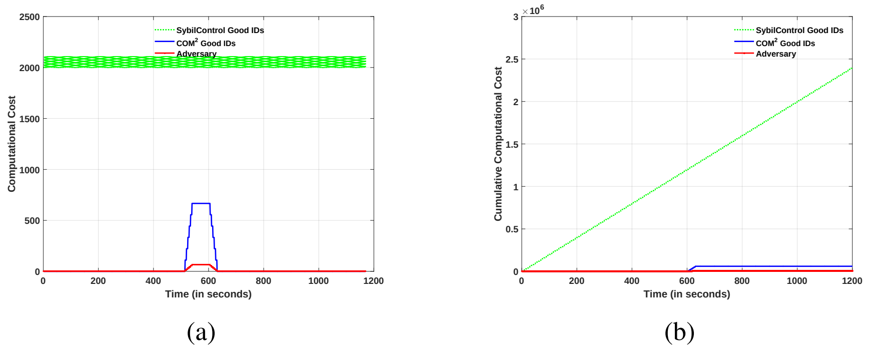


Figure 5: Growth of Computational Cost and Cumulative Computational Cost versus Time for system under threat

4.4 Performance Under Threat

Figure 5 shows the results of a Stress Test on the system: Initially, the system consisted of 10000 good IDs, after 10 minutes of this static system, every 5 seconds $\frac{n}{3}$ bad IDs attempted to joined the system over a span of next 30 seconds, following which $\frac{n}{3}$ bad IDs left the system every 5 seconds over the next 30 seconds. The system was allowed to remain static for next 10 minutes. For Figure 5b, the computational cost at each time step in the plot is the average of the computational cost for last 100 time steps.

As can be seen in Figure 5a, the good IDs pay a computational cost even in the absence of an attack for SYBILCONTROL, which is zero for COM^2 until time step 500. The plot for Cumulative Computation Cost to good IDs i.e., Figure 5b grows linearly with time, no matter whether the system is under attack or not for SYBILCONTROL, whereas in COM^2 this cost is incurred only when system is under attack from time step 600 to 660.

5 Applications

5.1 Byzantine Consensus

Background. The problem of ***Byzantine Consensus (BC)*** was introduced by Lamport, Shostak and Pease [61] (this problem is also referred to as Byzantine Agreement). There are n nodes, of which some hidden subset are *bad*. These bad nodes may deviate from a prescribed algorithm in an arbitrary way. The remaining nodes are *good*, and will follow our algorithm faithfully. Each good node has an initial input bit. The goal is for (1) all good nodes to decide on the same bit; and (2) this bit to equal the input bit of at least one good node.

Byzantine consensus enables the creation of a reliable system from unreliable components. Therefore, it is not surprising that BC is used in areas as diverse as: maintaining blockchains [14, 38, 45, 68]; trustworthy computing [16, 18, 19, 22, 23, 60, 88]; P2P networks [1, 79]; and databases [72, 83, 100]

A major shortcoming of current algorithms for BC is that they do not scale. For example, Rhea *et al*. write: *"Unfortunately, Byzantine agreement requires a number of messages quadratic in the number of participants, so it is infeasible for use in synchronizing a large number of replicas"* [85] (see also [24], [21]). This quadratic message cost hinders deployment in modern systems where the number of participants can be large.

Recent results reduce the total number of messages to $\tilde{O}(n^{3/2})$ [56, 57]. However, these algorithms (1) have high cost in practice; (2) are complicated to implement; and (3) require non-constructive combinatorial objects such as expander graphs.

How Our Result Applies. Our result allows us to reduce the communication cost for BC. The bad nodes are again incarnated as a single adversary with equivalent computational power. Each good node has a single ID, while the adversary is not constrained in its creation of IDs.

We employ the result in Section 3. The members of the committee execute any BC algorithm that requires a quadratic number of messages; this implies a message cost of $O(\log^2 n)$ generated by the committee. The committee then diffuses the signed consensus value to all other IDs. On receiving these agreed upon values from their committee members, the non-committee members take the majority of these verified consensus values. Since committee has a majority of good members, this results in all IDs holding the correct consensus value.

In this way, we are able to solve traditional BC with $\tilde{O}(n)$ number of messages in total. Additionally, via our guarantees in Section 3, we can solve a dynamic version of BC, where IDs are joining and leaving, and can do so with computational cost to the good IDs that is commensurate with the cost incurred by the adversary.

5.2 ELASTICO: Committee Election

Background. ELASTICO is a secure protocol [65], which aims to achieve agreement on a set of transactions in a blockchain i.e., the state of the blockchain. Informally, the core idea is to partition the system into smaller fragments called committees, where each committee executes a BC protocol to agree upon its set of transactions. Then, the committee that was formed last — referred to as the ***final committee*** — computes the final digest of all transactions in the system; these transactions having been received from other committees. This final digest is broadcast to all other participants in the system.

How Our Result Applies. In order to reduce message cost, ELASTICO makes use of a special committee referred to as the ***directory committee (DC)***, which coordinates the formation of all other committees. We propose the election of the DC using our algorithms COMMITTEE MAINTENANCE and MEMBERSHIP ELECTION in Section 3. This would ensure (1) the committee contains a majority of good IDs for a polynomial number of join and leave events; and (2) bandwidth and computational costs grow commensurately with the costs of the adversary.

6 Conclusion

We have described algorithms to efficiently use PoW computational puzzles to reduce the fraction of bad IDs in open systems. Unlike previous work, our algorithms require the good nodes to expend computational resources that grow only linearly with the computational resources expended by the adversary. In particular, assume the adversary incurs computational cost T_C and sends T_B messages, and that g_{new} good IDs enter the system. Then our algorithm requires $O(T_C + g_{new})$ computational cost and sends $O(T_B + g_{new})$ messages.

Many open problems remain. One of particular interest is: Can we adapt our technique so secure multi-party computation? The problem of ***secure multi-party computation (MPC)*** involves designing an algorithm for the purpose of computing the value of an n-ary function f over private inputs from n nodes $x_1, x_2, ..., x_n$, such that the nodes learn the value of $f(x_1, x_2, ..., x_n)$, but learn nothing more about the inputs than what can be inferred from this output of f. The problem is generally complicated by the assumption that an adversary controls a hidden subset of the nodes. In recent years, a number of attempts have been made to solve this problem for very large n manner [3] [6] [13] [26] [27] [29] [44]. We believe that our technique for forming committees in a dynamic network could be helpful to solve a dynamic version of this problem, while ensuring that the resource costs to the good nodes is commensurate with the resource costs to an adversary.

References

[1] Atul Adya, William J. Bolosky, Miguel Castro, Gerald Cermak, Ronnie Chaiken, John R. Douceur, Jon Howell, Jacob R. Lorch, Marvin Theimer, and Roger P. Wattenhofer. FARSITE: Federated, available, and reliable storage for incompletely trusted environment. In 5^{th} USENIX *Symposium on Operating Systems Design and Implementation (OSDI)*, pages 1–14, 2002.

[2] Marcin Andrychowicz and Stefan Dziembowski. Pow-based distributed cryptography with no trusted setup. In *Internationl Cryptology Conference (CRYPTO)*, pages 379–399. Springer, 2015.

[3] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. *Automata, languages and Programming*, pages 152–163, 2010.

[4] Baruch Awerbuch and Christian Scheideler. Towards a scalable and robust DHT. In *18th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 318–327, 2006.

[5] Rida A Bazzi and Goran Konjevod. On the Establishment of Distinct Identities in Overlay Networks. In *Proceedings* 24^{th} *Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 312–320, 2005.

[6] Zuzana Beerliová-Trubıniová and Martin Hirt. Efficient multi-party computation with dispute control. In *TCC*, volume 3876, pages 305–328. Springer, 2006.

[7] Mihir Bellare and Phillip Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pages 62–73. ACM, 1993.

[8] Michael A. Bender, Jeremy T. Fineman, Mahnush Movahedi, Jared Saia, Varsha Dani, Seth Gilbert, Seth Pettie, and Maxwell Young. Resource-Competitive Algorithms. *SIGACT News*, 46(3):57–71, September 2015.

[9] BitcoinWiki. Mining Hardware Comparison, 2016. `https://en.bitcoin.it/wiki/Mining_hardware_comparison`.

[10] BitcoinWiki. Non-Specialized Hardware Comparison, 2016. `https://en.bitcoin.it/wiki/Non-specialized_hardware_comparison`.

[11] BitcoinWiki. BitcoinWiki: Network, 2017. `https://en.bitcoin.it/wiki/Main_Page`.

[12] Blockstack. Blockstack: The New Decentralized Internet, 2016. `https://blockstack.org/`.

[13] Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas P Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus

Nielsen, Kurt Nielsen, Jakob Pagter, et al. Secure multiparty computation goes live. In *Financial Cryptography*, volume 5628, pages 325–343. Springer, 2009.

[14] Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Narayanan, Joshua A. Kroll, and Edward W. Felten. SoK: Research Perspectives and Challenges for Bitcoin and Cryptocurrencies. In *Proceedings of the IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, pages 104–121, 2015.

[15] Nikita Borisov. Computational Puzzles As Sybil Defenses. In *Proceedings of the Sixth IEEE International Conference on Peer-to-Peer Computing*, P2P '06, pages 171–176, 2006.

[16] Christian Cachin and Jonathan A. Poritz. Secure Intrusion-Tolerant Replication on the Internet. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*, pages 167–176, 2002.

[17] Ran Canetti. Towards Realizing Random Oracles: Hash Functions That Hide All Partial Information. In *Proceedings of the Annual International Cryptology Conference*, pages 455–469. Springer, 1997.

[18] Miguel Castro and Barbara Liskov. Practical Byzantine Fault Tolerance. *Operating Systems Review*, 33:173–186, 1998.

[19] Miguel Castro and Barbara Liskov. Practical Byzantine Fault Tolerance and Proactive Recovery. *ACM Transactions on Computer Systems (TOCS)*, 20(4):398–461, 2002.

[20] Wu chang Feng, Ed Kaiser, Wu chi Feng, and Antoine Luu. The Design and Implementation of Network Puzzles. In *Proceedings of the Annual Joint Conference of IEEE Computer and Communications Societies (INFOCOM)*, pages 2372–2382, 2005.

[21] Chien-Fu Cheng, Shu-Ching Wang, and Tyne Liang. The Anatomy Study of Server-Initial Agreement for General Hierarchy Wired/Wireless Networks. *Computer Standards & Interfaces*, 31(1):219 – 226, 2009.

[22] Allen Clement, Mirco Marchetti, Edmund Wong, Lorenzo Alvisi, and Mike Dahlin. Byzantine Fault Tolerance: The Time is Now. In *Proceedings of the Second Workshop on Large-Scale Distributed Systems and Middleware (LADIS)*, pages 1–4, 2008.

[23] Allen Clement, Edmund Wong, Lorenzo Alvisi, Mike Dahlin, and Mirco Marchetti. Making Byzantine Fault Tolerant Systems Tolerate Byzantine Faults. In *Proceedings of the Sixth USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 153–168, 2009.

[24] James Cowling, Daniel Myers, Barbara Liskov, Rodrigo Rodrigues, and Liuba Shrira. HQ Replication: A Hybrid Quorum Protocol for Byzantine Fault Tolerance. In *Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 177–190, 1999.

[25] Anthony Cuthbertson. Bitcoin Now Accepted by 100,000 Merchants Worldwide. February 2015. www.ibtimes.co.uk/bitcoin- now-accepted-by-100000-merchants-worldwide-1486613.

[26] Ivan Damgard and Yuval Ishai. Scalable secure multiparty computation. In *Crypto*, volume 4117, pages 501–520. Springer, 2006.

[27] Ivan Damgård, Yuval Ishai, Mikkel Krøigaard, Jesper Nielsen, and Adam Smith. Scalable multiparty computation with nearly optimal work and resilience. *Advances in Cryptology–CRYPTO 2008*, pages 241–261, 2008.

[28] George Danezis, Chris Lesniewski-laas, M. Frans Kaashoek, and Ross Anderson. Sybil-resistant DHT Routing. In *Proceedings of the* 10^{th} *European Symposium On Research In Computer Security (ESORICS)*, pages 305–318, 2005.

[29] Varsha Dani, Valerie King, Mahnush Movahedi, and Jared Saia. Quorums quicken queries: Efficient asynchronous secure multiparty computation. In *International Conference on Distributed Computing and Networking*, pages 242–256. Springer, 2014.

[30] Murat Demirbas and Youngwhan Song. An RSSI-based Scheme for Sybil Attack Detection in Wireless Sensor Networks. In *Proceedings of the 2006 International Symposium on on World of Wireless, Mobile and Multimedia Networks*, WOWMOM '06, pages 564–570, 2006.

[31] John Douceur. The Sybil Attack. In *Proceedings of the Second International Peer-to-Peer Symposium (IPTPS)*, pages 251–260, 2002.

[32] Jamie Doward. City Banks Plan to Hoard Bitcoins to Help Them Pay Cyber Ransoms. October 2016. https://www.theguardian.com/technology/2016/oct/22/city-banks-plan-to-hoard-bitcoins-to-help-them-pay-cyber-ransoms.

[33] Cynthia Dwork and Moni Naor. Pricing via Processing or Combatting Junk Mail. In *Proceedings of the* 12th *Annual International Cryptology Conference on Advances in Cryptology*, pages 139–147, 1993.

[34] The Economist. The Magic of Mining. 2015. www.economist.com/news/business/21638124-minting-digital-currency-has-become-big-ruthlessly-competitive-business-magic.

[35] The Economist. The Trust Machine. 2015. www.economist.com/news/leaders/21677198-technology-behind-bitcoin-could-transform-how-economy-works-trust-machine.

[36] Ethereum. Ethereum: Blockchain App. Program, 2016. `https://www.ethereum.org/`.

[37] Andy Extance. The Future of Cryptocurrencies: Bitcoin and Beyond. *Nature*, 526:21–23, 2015.

[38] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert Van Renesse. Bitcoin-NG: A Scalable Blockchain Protocol. In *Proceedings of the* 13th *USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 45–59, 2016.

[39] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. *The Bitcoin Backbone Protocol: Analysis and Applications*, pages 281–310. 2015.

[40] Stephanie Gil, Swarun Kumar, Mark Mazumder, Dina Katabi, and Daniela Rus. Guaranteeing Spoof-Resilient Multi-Robot Networks. In *Proceedings of Robotics: Science and Systems*, Rome, Italy, July 2015.

[41] Seth Gilbert, Valerie King, Jared Saia, and Maxwell Young. Resource-Competitive Analysis: A New Perspective on Attack-Resistant Distributed Computing. In *Proceedings of the* 8th *ACM International Workshop on Foundations of Mobile Computing*, 2012.

[42] Seth Gilbert, Calvin Newport, and Chaodong Zheng. Who Are You? Secure Identities in Ad Hoc Networks. In *Proceedings of the* 28th *International Symposium on Distributed Computing (DISC)*, pages 227–242, 2014.

[43] Seth Gilbert and Chaodong Zheng. SybilCast: Broadcast on the Open Airwaves. In *Proceedings of the* 25th *Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 130–139, 2013.

[44] Oded Goldreich. Secure multi-party computation. *Manuscript. Preliminary version*, pages 86–97, 1998.

[45] Sergey Gorbunov and Silvio Micali. Democoin: A Publicly Verifiable and Jointly Serviced Cryptocurrency. Cryptology ePrint Archive, Report 2015/521, 2015. `http://eprint.iacr.org/2015/521`.

[46] Jeff Green, Joshua Juen, Omid Fatemieh, Ravinder Shankesi, Dong Jin, and Carl A. Gunter. Reconstructing Hash Reversal Based Proof of Work Schemes. In *Proceedings of the 4th USENIX Conference on Large-scale Exploits and Emergent Threats*, LEET'11, pages 10–10, 2011.

[47] Charles Miller Grinstead and James Laurie Snell. *Introduction to probability*. American Mathematical Soc., 2012.

[48] Saikat Guha and Neil Daswani. An experimental study of the skype peer-to-peer voip system. Technical report, Cornell University, 2005.

[49] Robert Hackett. Can This 22-year-old Coder Out-Bitcoin Bitcoin? 2016. `http://fortune.com/ethereum-blockchain-vitalik-buterin/`.

[50] Hashcat. Benchmark Results, 2016. `http://thepasswordproject.com/oclhashcat_benchmarking`.

[51] Chain Incorporated. Chain: Enabling a New Medium for Assets, 2016. `https://chain.com/`.

[52] Yves Igor Jerschow and Martin Mauve. Modular Square Root Puzzles: Design of Non-Parallelizable and Non-Interactive Client Puzzles. *Computers and Security*, 35:25–36, June 2013.

[53] R. John, J. P. Cherian, and J. J. Kizhakkethottam. A Survey of Techniques to Prevent Sybil Attacks. In *Proceedings of the International Conference on Soft-Computing and Networks Security (ICSNS)*, pages 1–6, 2015.

[54] Ed Kaiser and Wu chang Feng. Mod_kaPoW: Mitigating DoS with Transparent Proof-of-Work. In *Proceedings of the 2007 ACM CoNEXT Conference*, CoNEXT '07, pages 74:1–74:2, 2007.

[55] Ghassan O. Karame and Srdjan Čapkun. Low-Cost Client Puzzles Based on Modular Exponentiation. In *Proceedings of the 15th European Conference on Research in Computer Security (ESORICS)*, pages 679–697, 2010.

[56] Valerie King and Jared Saia. Breaking the $O(n^2)$ Bit Barrier: Scalable Byzantine Agreement with an Adaptive Adversary. In *Proceedings of the* 29^{th} *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 420–429, 2010.

[57] Valerie King and Jared Saia. Breaking the $O(n^2)$ Bit Barrier: Scalable Byzantine Agreement with an Adaptive Adversary. *Journal of the ACM (JACM)*, 58(4), 2011.

[58] Marek Klonowski and MichaB Koza. Countermeasures Against Sybil Attacks in WSN Based on Proofs-of-Work. In *Proceedings of the Sixth ACM Conference on Security and Privacy in Wireless and Mobile Networks*, WiSec '13, pages 179–184, 2013.

[59] Neal Koblitz and Alfred J Menezes. The Random Oracle Model: A Twenty-Year Retrospective. *Designs, Codes and Cryptography*, 77(2-3):587–610, 2015.

[60] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. Zyzzyva: Speculative Byzantine Fault Tolerance. In *Proceedings of* 21^{st} *ACM SIGOPS Symposium on Operating Systems Principles*, pages 45–58, 2007.

[61] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

[62] Chris Lesniewski-Laas and M. Frans Kaashoek. Whanau: A Sybil-Proof Distributed Hash Table. In *Proceedings of the* 7^{th} *USENIX Conference on Networked Systems Design and Implementation*, NSDI'10, pages 8–8, 2010.

[63] Frank Li, Prateek Mittal, Matthew Caesar, and Nikita Borisov. SybilControl: Practical Sybil Defense with Computational Puzzles. In *Proceedings of the Seventh ACM Workshop on Scalable Trusted Computing*, STC '12, pages 67–78, 2012.

[64] Y. Liu, D. R. Bild, R. P. Dick, Z. M. Mao, and D. S. Wallach. The Mason Test: A Defense Against Sybil Attacks in Wireless Networks Without Trusted Authorities. *IEEE Transactions on Mobile Computing*, 14(11):2376–2391, 2015.

[65] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. A Secure Sharding Protocol For Open

Blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 17–30, 2016.

[66] Pablo Magro. What Greece can learn from bitcoin adoption in Latin America . July 2015. www.ibtimes.co.uk/what-greece-can-learn-bitcoin-adoption-latin-america-1511183.

[67] Ivan Martinovic, Frank A. Zdarsky, Matthias Wilhelm, Christian Wegmann, and Jens B. Schmitt. Wireless Client Puzzles in IEEE 802.11 Networks: Security by Wireless. In *Proceedings of the First ACM Conference on Wireless Network Security*, WiSec '08, pages 36–45, 2008.

[68] Silvio Micali. ALGORAND: The Efficient and Democratic Ledger. *CoRR*, abs/1607.01341, 2016.

[69] Andrew Miller, James Litton, Andrew Pachulski, Neil Spring, Neal Gupta, Dave Levin, and Bobby Bhattacharjee. Discovering Bitcoin's Public Topology and Influential Nodes, 2015. `http://cs.umd.edu/projects/coinscope/coinscope.pdf`.

[70] Aziz Mohaisen and Scott Hollenbeck. Improving Social Network-Based Sybil Defenses by Rewiring and Augmenting Social Graphs. In *Proceedings of the* 14^{th} *International Workshop on Information Security Applications (WISA)*, pages 65–80, 2014.

[71] Aziz Mohaisen and Joongheon Kim. The Sybil Attacks and Defenses: A Survey. *Smart Computing Review*, 3(6):480–489, 2013.

[72] Hector Garcia Molina, Frank Pittelli, and Susan Davidson. Applications of Byzantine Agreement in Database Systems. *ACM Transactions on Database Systems (TODS)*, 11:27–47, 1986.

[73] Diogo Mónica, Jo ao Leitão, Luís Rodrigues, and Carlos Ribeiro. On the Use of Radio Resource Tests in Wireless Ad-Hoc Networks. In *Proceedings of the 3rd Workshop on Recent Advances on Intrusion-Tolerant Systems*, pages F21–F26, 2009.

[74] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System, 2008. `http://bitcoin.org/bitcoin.pdf`.

[75] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[76] Till Neudecker, Philipp Andelfinger, and Hannes Hartenstein. Timing analysis for inferring the topology of the bitcoin peer-to-peer network. In

Proceedings of the 13th IEEE International Conference on Advanced and Trusted Computing (ATC), July 2016.

[77] Till Neudecker and Hannes Hartenstein. Could network information facilitate address clustering in bitcoin? In *International Conference on Financial Cryptography and Data Security*. springer, 2017.

[78] James Newsome, Elaine Shi, Dawn Song, and Adrian Perrig. The Sybil Attack in Sensor Networks: Analysis & Defenses. In *Proceedings of the 3rd International Symposium on Information Processing in Sensor Networks*, IPSN '04, pages 259–268, 2004.

[79] Oceanstore. The Oceanstore Project, 2011. http://oceanstore.cs.berkeley.edu.

[80] National Institute of Standards and Technology. Secure Hash Standard (SHS). `http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf`, 2015.

[81] Luke Parker. Blockchain Tech Startup Chain Inc Hits the Wall Street Motherload, 2015. http://bravenewcoin.com/news/blockchain-tech-startup-chain-inc-hits-the-wall-street-motherload/.

[82] Bryan Parno, Dan Wendlandt, Elaine Shi, Adrian Perrig, Bruce Maggs, and Yih-Chun Hu. Portcullis: Protecting Connection Setup from Denial-of-Capability Attacks. *ACM SIGCOMM Computer Communication Review*, 37(4):289–300, 2007.

[83] Nuno Preguiça, Rodrigo Rodrigues, Christóvão Honorato, and João Lourenço. Byzantium: Byzantine-Fault-Tolerant Database Replication Providing Snapshot Isolation. In *Proceedings of the Fourth Conference on Hot Topics in System Dependability*, page 9. USENIX Association, 2008.

[84] Jothi Rangasamy, Douglas Stebila, Lakshmi Kuppusamy, Colin Boyd, and Juan Gonzalez Nieto. *Efficient Modular Exponentiation-Based Puzzles for Denial-of-Service Protection*, pages 319–331. Springer Berlin Heidelberg, 2012.

[85] Sean Rhea, Patrick Eaton, Dennis Geels, Hakim Weatherspoon, Ben Zhao, and John Kubiatowicz. Pond: The OceanStore prototype. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies*, pages 1–14, 2003.

[86] Christian Scheideler and Stefan Schmid. *A Distributed and Oblivious Heap*, pages 571–582. 2009.

[87] Micah Sherr, Matt Blaze, and Boon Thau Loo. Veracity: Practical Secure Network Coordinates via Vote-based Agreements. In *Proceedings of the 2009 Conference on USENIX Annual Technical Conference*, USENIX'09, pages 13–13, 2009.

[88] SINTRA. SINTRA - Distributed Trust on the Internet, 2017. http://www.zurich.ibm.com/security/dti/.

[89] K. Srinathan and C. Pandu Rangan. Efficient asynchronous secure multiparty distributed computation. In *INDOCRYPT 2000, Lecture Notes in Computer Science*, volume 1977, pages 117–129. Springer-Verlag, 2000.

[90] Daniel Stutzbach and Reza Rejaie. Understanding Churn in Peer-to-peer Networks. In *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, IMC '06, pages 189–202, New York, NY, USA, 2006. ACM.

[91] Melanie Swan. *Blockchain: Blueprint for a new economy*. " O'Reilly Media, Inc.", 2015.

[92] Florian Tegeler and Xiaoming Fu. SybilConf: Computational Puzzles for Confining Sybil Attacks. In *Proceedings of the IEEE Conference on Computer Communications Workshops (INFOCOM)*, pages 1–2, 2010.

[93] XiaoFeng Wang and Michael K. Reiter. Defending Against Denial-of-Service Attacks with Puzzle Auctions. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, page 78, 2003.

[94] Brent Waters, Ari Juels, J. Alex Halderman, and Edward W. Felten. New Client Puzzle Outsourcing Techniques for DoS Resistance. In *Proceedings of the 11th ACM Conference on Computer and Communications Security (CCS)*, pages 246–256, 2004.

[95] Wei Wei, Fengyuan Xu, Chiu C. Tan, and Qun Li. SybilDefender: A Defense Mechanism for Sybil Attacks in Large Social Networks. *IEEE Transactions on Parallel & Distributed Systems*, 24(12):2492–2502, 2013.

[96] Waloddi Weibull. Wide applicability. *Journal of applied mechanics*, 103(730):293–297, 1951.

[97] Zhi Yang, Christo Wilson, Xiao Wang, Tingting Gao, Ben Y. Zhao, and Yafei Dai. Uncovering Social Network Sybils in the Wild. In *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, IMC '11, pages 259–268, 2011.

[98] Haifeng Yu. Sybil Defenses via Social Networks: A Tutorial and Survey. *SIGACT News*, 42(3):80–101, October 2011.

[99] Haifeng Yu, Michael Kaminsky, Phillip B. Gibbons, and Abraham Flaxman. SybilGuard: Defending Against Sybil Attacks via Social Networks. *Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, 36:267–278, August 2006.

[100] Wenbing Zhao. A Byzantine Fault Tolerant Distributed Commit Protocol. In *Proceedings of the* 3^{rd} *IEEE International Symposium on Dependable, Autonomic and Secure Computing*, pages 37–46, 2007.