COLBAC: Shifting Cybersecurity from Hierarchical to Horizontal Designs

Kevin Gallagher
kevin.gallagher@tecnico.ulisboa.pt
DCentral/DEI
Instituto Superior Técnico, Universidade de Lisboa
Lisboa, Portugal
Department Of Computer Science and Engineering
New York University
Brooklyn, New York, USA
Defend Our Privacy Association (PrivacyLX)
Lisbon, Portugal

Santiago Torres-Arias
santiagotorres@purdue.edu
Department of Electrical and Computer Engineering
Purdue University
West Lafayette, Indiana, USA

Nasir Memon
memon@nyu.edu
Department of Computer Science and Engineering
New York University
Brooklyn, New York, USA

Jessica Feldman
jfeldman@aup.edu
Department of Communication, Media, and Culture
American University of Paris
Paris, France

ABSTRACT

Cybersecurity suffers from an oversaturation of centralized, hierarchical systems and a lack of exploration in the area of horizontal security, or security techniques and technologies which utilize democratic participation for security decision-making. Because of this, many horizontally governed organizations such as activist groups, worker cooperatives, trade unions, not-for-profit associations, and others are not represented in current cybersecurity solutions, and are forced to adopt hierarchical solutions to cybersecurity problems. This causes power dynamic mismatches that lead to cybersecurity and organizational operations failures. In this work we introduce COLBAC, a collective based access control system aimed at addressing this lack. COLBAC uses democratically authorized capability tokens to express access control policies. It allows for a flexible and dynamic degree of horizontality to meet the needs of different horizontally governed organizations. After introducing COLBAC, we finish with a discussion on future work needed to realize more horizontal security techniques, tools, and technologies.

CCS CONCEPTS

• **Security and privacy** → *Access control*; **Authorization**; **Usability in security and privacy**; **Social aspects of security and privacy**.

NSPW '21, October 25–28, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8573-2/21/10.
https://doi.org/10.1145/3498891.3498903

KEYWORDS

security, democracy, activism, horizontality, participation, participatory design, distributed systems, authorization, access control

ACM Reference Format:
Kevin Gallagher, Santiago Torres-Arias, Nasir Memon, and Jessica Feldman. 2021. COLBAC: Shifting Cybersecurity from Hierarchical to Horizontal Designs. In *New Security Paradigms Workshop (NSPW '21), October 25–28, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3498891.3498903

1 INTRODUCTION

The struggle of power between centralized, hierarchical structures and decentralized, horizontal structures has existed for millennia. This struggle has expressed itself in all forms of life: politics, economics, social interaction, and more. Recent events all around the globe have demonstrated the continuance of this struggle: while wealth inequality increases and political power centralizes in the hands of political families, economic elites, or dictators, other elements of society demonstrate a desire for more inclusive, participatory, and fair social structures. Examples of this can be seen in the "Arab Spring" movements, which demanded more democratic participation in governments of the Arab world, and the Occupy Movement, which demanded more democratic participation in economic and political policies. More recently, in Hong Kong, recent protests have demanded the right to participate in their own government through democratic vote. In the Movement for Black Lives, some Black Lives Matter chapters are demanding more than just increased police oversight and funding reallocation, but community control of police, including access to relevant local, state, and federal law enforcement agency information [26]. In the economic sector, worker cooperatives and worker participation in their place of employment is becoming more prevalent, and research is beginning to demonstrate economic advantages to cooperative organization of the workplace [19, 25, 43]. In the tech sector, Free Open Source

Software (FOSS) projects practice a range of democratic governance mechanisms [1]. Even some schools are beginning to include the democratic participation of the school children to their decision making processes, encouraging engagement and investment in the child's own education [30]. In many sectors of life, the centralization and strict hierarchy of current organization is being challenged by a demand for more participatory structures.

Despite the increasing demand for a changed structure of organization, Cybersecurity tools, technologies and techniques continue to remain highly hierarchical. Security policies are written by a security team or imported from pre-existing (and sometimes external) templates and imposed on the rest of the organization, with at most the limited oversight and approval of a manager or higher-up. Even still, not every member of a security team participates in deciding what these policies are meant to achieve. This decision is often made by a manager, who ensures that the policies do not get in the way of the visions of the individuals on the top of the hierarchies. In developing security policies and implementing security technologies, the resulting technologies and policies entrench the political dynamics in place during their creation [41]. The resulting policies and technologies then themselves have political implications that affect the individuals, organizations, and societies that use them, intensifying intertia and preventing change. Since most security tools have their historical roots in hierarchical entities such as military or corporate business, these tools tend to reflect the assumptions of hierarchy that exist in those communities. While this may make sense for the military or corporate sectors, these technologies are not the best fit for horizontal and participatory sectors such as worker cooperatives, activist groups, and more horizontally run nonprofits, as well as cities and states that are experimenting with direct democracy initiatives.

One example of this is the access to organization-controlled secrets. Secrets such as organizational passwords to third-party services are often trusted to one individual or a small group in an organization, but easily recovered by an administrator or other individual higher in the hierarchy in the case that the individual responsible leaves the organization or performs a violation. However, when an organization becomes less hierarchical, the decision of who should have access to such secrets becomes more difficult to make. If everyone should have access to the secrets, such as passwords, then any insider can change the passwords and cause denial of service for the organization. If only a few people have access to the password, then a hierarchy forms based on one's access to confidential organizational material. This mismatch between governance structure and assumed governance structures entrenched in technologies leads to events like password wars [21], or issues like the "digital vanguard problem", discussed below. [13].

A case in point: one of the organizations we have observed in our ethnographic research was a democratic trade union, which had recently transitioned from a more hierarchical and bureaucratic organizational structure to one which involved more transparent decision-making and meetings, with direct participation by members (meetings were open to all, membership became less exclusive, and contract bargaining was open to all members, among other reforms.) This change came about through years of grass-roots organizing within the union, and was implemented through a democratic election in which a new slate of representatives took power within the union. In this model – neither purely direct nor purely representative democracy – a committee affiliated with the ruling caucus was charged with handling communications to the members, through consensus-based decision-making. For this purpose, access to the union's email contacts and digital archives had to be transferred to the committee.

Here is where the trouble began. Although a democratic process had put the committee in power, and although the committee shared decision-making power equally across all members, access to digital communications and member lists was controlled by single passwords. In the past, these communications had been handled by one or a few people who had the passwords and who worked within a hierarchical structure, implementing the wishes of their superiors. These actors refused to share the passwords with the new committee, and a great struggle ensued, for weeks. During this period, communications to the membership were sent illegitimately and by those who previously held power. A great deal of time and energy was spent coming to a compromise, and forcing the former caucus to share access. Here we see a clear case wherein the digital security design does not respond to or mirror changes in the political structure of a democratic collective. This can be the site of injustice, struggle, and a place where a lot of time, energy, and credibility is lost. A more distributed version of access control could have averted this situation.

Similar stories arise in other research at the intersection of communications and social movements: Anastasia Kavada, in her research on the Occupy movement, describes multiple cases, in different cities, of administrators being locked out of Facebook pages or hijacking the group's twitter account to tell their own narrative [20] Similarly, Paulo Gerbaudo theorizes what he calls a "digital vanguard" in non-hierarchical social movements, wherein those in control of the social media accounts become the default voice of, and ideological leaders, of the group. Gerbaudo describes

> the emergence of new forms of power stratification embedded in the hierarchy of content management systems used by activists, and the explosion of power struggles for the control of social media accounts [13].

However, the issue does not stop at secrets or accounts. Indeed, these cases are specific instances of a more general problem: a mismatch between the governance structures of an organization and the governance structures assumed by technologies. In this specific case, it is the mismatch between horizontally governed organizations and hierarchical access control.

In current access control systems, whether or not an entity should be allowed to access a resource is decided by a hierarchical process. In discretionary access control the owner of an object decides whether or not others can read from, write to, or execute that object. With the recognition that some files are more important to operations than others, it is easy to see that this creates a hierarchy. This access control information itself must be stored in an object, and that object must have an owner. Thus, the owner of that object has the ability to change the system and reallocate permissions, putting themselves on top of that hierarchy. Similarly, in Role Based Access Control at least one role needs to be able to assign and remove roles, allowing them to grant themselves whatever role they need to access to any object they wish. More generally,

the authorization process of access control is centralized at the top of the hierarchies of each individual organization.

In his work *The Moral Character of Cryptographic Work*, Rogaway states "Cryptography can be developed in directions that tend to benefit the weak or the powerful." [34]. However, there is no reason why this must stop at cryptography, and cannot generalize to all the fields of security. Why, then, have we not developed **more** technologies that benefit the community by limiting the privileges of the powerful within an organization via democratic participation? Why have no access control methods been developed to allow for authorization via horizontal, participatory protocols?

In this work we address this gap. In the following sections we propose COLBAC, a collective based system that allows for access control via flexible and dynamic democratic participation to fit the needs of the organization using it. We start by discussing the current hierarchical state of access control in Section 2.1 and discussing previous attempts at hierarchical security in Section 2.2. We then discuss COLBAC's threat model in Section 4 and introduce the design goals of COLBAC in Section 5.1. We informally describe the COLBAC system design in Sections 5.2, 5.3 and 5.4, and formally define COLBAC in Section 5.5. We then discuss the properties of COLBAC in Section 5.6 and discuss the intersection of Democracy and COLBAC in Section 3. In Section 6 we perform a security analysis of COLBAC and in Section 7 we discuss its usability and scalability. We then discuss COLBAC's limitations in Section 8. Finally, we discuss open research and future work in Section 9 and in Section 10 we conclude.

2 ENVISIONING HORIZONTAL SECURITY

Central to the issue of Horizontal Security are the notions of *authorization* (i.e., who is allowed to access a particular resource) and *authentication* (i.e., determining who a given individual is). If an authorized party reads, modifies, or destroys information, the definitions imply that there is no breach of policy[1]. Thus, whether or not the security of an organization has been breached depends highly on whether or not an entity is authorized to perform an action. This decision as to whether or not an entity is authorized is heavily informed by an organization's security policy.

However, as is natural with the notion of authorization, two fundamental pieces must exist: authorization *policies* that represent collective-based authorization, and an access control system that can *enforce* such policies. Though *authentication* is also a vital piece of this process, in this work we assume that *authentication* is performed by another system and do not consider it further in this work.

2.1 Hierarchy and Access Control

Access control is commonly both encoded and enforced in a top-down fashion. The CISO of an organization holds institutional power to define access control policies within an organization. Further lower-level administrators may have some decision making power in how to define subsets of a policy, yet, they typically do not have decision making power to change the direction of the policy completely.

[1]though there may still be a breach of trust from other individuals in the organization

This concept typically holds true not just with access control and authorization, but with all aspects of an organization's security policy. Decisions about how infringements are dealt with or the scope of the policies are also made in this manner. Decisions about how these policies are enforced, and what security mechanisms are used, are also made up top, with entities at the bottom left to enact the desires passed down to them.

We see, then, that hierarchy introduces itself in multiple dimensions of security. These dimensions include the different decision-making processes of security. However, in democratic institutions security can be multilateral [32], and thus hierarchical security decision making is not appropriate. Instead, it is possible that horizontal security may be achieved by shifting policy-making powers, implementation decisions, enforcement powers, and other dimensions of security away from individuals and towards democratic, participatory processes[2].

However, organizations that attempt to create their access control policies horizontally may run into problems when they move to implement them. Currently, access control systems force the creation of a hierarchy. In access control systems such as MAC, DAC, or RBAC, certain entities in the organization have the power to implement the access control rules that were democratically created by the organization. Thus, if the individuals who have the ability to enforce the rules decide not to, the newly created access control policies become ineffective. This forms a hierarchy with the entities capable of enforcing access control sitting at the top. Similarly, if an organization were to attempt to temporarily act hierarchically and to implement that using tools that assumed horizontality, this may cause the hierarchy to lose any potential efficiency benefits. Thus, a question arises: can we develop technology that allows an organization to be flexible and dynamic in its horizontality, being participatory or hierarchical depending on the needs at hand, without jeopardizing the ability to return to horizontality when desired?

2.2 Previous Attempts at Horizontal Security

Having explored the principles that unlock horizontality, we can move on to creating specific systems that allow entities to organize in such a way. Achieving horizontality is not trivial. In fact, many projects and organizations have attempted to do so with varying degrees of success. In this section, we argue that these failings are a consequence of a missing building block: Collective-based Access Control (or COLBAC).

When laying out known horizontal software systems, perhaps the most widely-known example is Bitcoin [28]. This cryptocurrency was developed to allow for financial transactions between two individuals without the need for a trusted third party to detect and prevent double spending. By design, the network prevents the modification of past blocks of the Blockchain unless there exists collusion of a large group of entities with computing [4] (or to a lesser extent, network [14]) resources. This decision was made specifically to make it difficult for malicious actors in the network to perform

[2]Further questions arise when we consider the structures of organizations with flexible memberships, who operate on principles of open participation (plein-air citizens assemblies, etc.), but these will be explored in future work

malicious activities, such as stealing bitcoin or deleting past transactions. The implication of this design is that Bitcoin cannot be centrally regulated [40], at least in theory. Thus, a core element of cryptocurrencies, the consensus protocol is an instantiation of a democratically-based access control policy.

Yet there exist many shortcomings in the Bitcoin/cryptocurrency model. First is identifying *who* is part of the collective: are only miners stakeholders, or are users stakeholders as well, are developers stakeholders? This question raises the fundamental limitation of Bitcoin. Although it can be considered "network-horizontal" it does not provide horizontality mechanisms for other elements of the ecosystem. This is apparent when considering historical events in Blockchain-based systems, such as the DAO hack [38], the Bitcoin Classic fork [18] and more. Indeed, it seems that developers of a Blockchain, those blessed with public visibility (or even just commit access) can have more impact on the nature of the Blockchain than a large mining cartel.

Another issue with this approach is its competitive nature. Rather than cooperating miners attempting to create, implement, and enforce transaction policies, we have competing miners interested in their own profit. One implication of this competitive focus is that there is no mechanism to vote in Bitcoin; rather, what is referred to as "regulation" in this work is seen by the Bitcoin community as an attack. However, the design property of requiring a majority of participating members to agree in order to make a change on the system may still be useful to inspire future horizontal system designs. In the case of cryptocurrencies, the competitive design tends to centralize the network over time. Bitcoin's Proof of Work algorithm tends to favor those who can afford highly specialized hardware, eventually leading to a centralization of the network into the hands of those who are wealthy enough to control large amounts of machines specifically made for mining Bitcoins. Proof of Stake, another consensus algorithm for Blockchain, bypasses this middle step and directly benefits those who control larger amounts of monetary stake in the network.

Other, more voluntary (and less profit-driven) designs also attempt to achieve horizontality. Perhaps the most well known example of a security system meant to be horizontal by design is the PGP Web of Trust (WoT) design. This ecosystem attempts to allow entities to authenticate (i.e., tie public key material to emails, names, or identities) and exchange signed or encrypted messages without a central authority. This is done through a transitive trust process: known-good entities can sign trust relationships with other entities and various WoT algorithms will compute resulting trust levels based on a personal security policy that ranges from marginal trust to full trust. There is however no global notion of policing or enforcement in the PGP ecosystem. As such, it is impossible for entities in a group to agree access control policies in a distributed ecosystem.

In addition to the Blockchain approach used by Bitcoin, and to the web of trust used by PGP, many other attempted approaches towards horizontal security already exist. These include other consensus algorithms such as Practical Byzantine Fault Tolerance (PBFT) [7], Paxos [23], Raft [29], and more. However, most of these protocols do not allow for flexible horizontality; rather, they have a set percentage of entities in the system that must agree. More, these systems are more concerned with fault tolerance rather than access control. However, using these solutions for the distribution of horizontal security technologies may be beneficial in future projects.

In addition to consensus algorithms, secret sharing schemes like Shamir Secret Sharing [36], Blakley's Secret Sharing [3], and Verifiable Secret Sharing [8] schemes can be used for distributed secret management in organizations. This allows for individuals to recover access to a resource if a set fraction of the participants agree and submit their secret share. However, these approaches are inefficient and are also rigid in their horizontality.

Other solutions to small horizontal security problems exist. Applications like OAuth [24] can be used to grant minimal and ephemeral access to organization accounts to members of the organization who need it, without revealing a password. However, someone in the organization still must control the password, causing hierarchy issues. Capability [10] based schemes and systems can be used to allow for transfer of control over resources in a more flexible way, but is not sufficient alone: a collective authorization scheme is required to determine whether or not to issue capability tokens.

Previous attempts of expanding access control to fill some of these gaps have been made, but each of them has its flaw. Most work centers around the idea of multi-ownership access control or multi-party access control (MPAC), where different files in a system are owned by multiple different users who need to agree on who may see a given resource and who may not. These schemes are usually specifically designed for online social networks (OSNs), and generally use either majority voting ([15, 17], etc.) secret sharing techniques ([16], etc.), or conflict resolution protocols ([5, 6, 33], etc.) to determine with whom files should be shared. Other approaches rely on the relationships between the owners of a resource and other users within a community or (OSN) to determine who gets to access a file. However, these approaches are different from ours in many ways. Primarily, none of the MPAC solutions focus on flexible horizontality: instead, they assume either majority voting or some form of conflict resolution, rather than begin flexible enough to cover multiple forms of horizontal structure. More, these systems focus on specific issues related to OSNs, such as privacy rights and concerns for photo sharing, and as such these systems emphasize read decisions over write or execute decisions, making them insufficient for the purpose of access control and authorization on a general-purpose system. Finally, previous attempts to fill this gap consider resource sharing and authorization within a complex community where individuals who belong to the community own files and make decisions regarding those files. They do not consider the problem of access control and authorization in the name of the community or collective as a whole, as our solution does.

To address these gaps we present COLBAC, a collective based access control system to provide the fundamental basis for more democratic and participatory security software and systems.

3 DEMOCRACY AND COLBAC

Democracy is colloquially defined as "government by the people." The term has its root in the Ancient Greek *demos*, which meant both the village (*demos*, the smallest possible administrative unit in the state), and "the people" (*Demos*, which, in ancient Athens, meant anyone who could participate in the governing Assembly – adult men, not slaves, who had completed their military training, and whose parents were from Athens.) In Athenian democracy,

the Assembly practiced direct democracy, wherein the Assembly made decisions through deliberative processes and without elected representatives [2]. Forms of governance in which no one has more political or managerial power than any other are sometimes called horizontal, drawing from the Spanish *horizontalidad*, which was first used to describe new political practices in Argentina after a popular rebellion in 2001 [37]. In most contemporary democratic states, most governance at the national level is performed by representative democracy, wherein the citizens vote for leaders who are entrusted with the work of running the state. These two forms — representative and direct/horizontal democracy – can be seen as the most basic types of democracy. Many, many democratic practices and groups exist which mix these two forms in different ways, and which use different techniques for decision-making, and the distribution of power and resources. Furthermore, a plentitude of solutions and debates exist around the democratic "boundary problem" [22], wherein the definition of who constitutes the *Demos* is differently decided, and, certainly, has changed over time.[3]

While we do not have the space here to delve into a history of democracy and debates over its various forms, a few key concepts are important to outline. Decision-making within democracies can take many forms. *Consensus* refers to a process of decision-making wherein the group comes to a shared agreement on a decision (everyone agrees (unanimity), or, more commonly, a certain threshold of agreement is reached (75% of the group agrees, for example), or, no one disagrees enough to block a decision in going forward.)[4] *Deliberation* is a process of discussion and argument that is usually considered essential for consensus and healthy for all modes of democracy. Deliberation ideally involves authentic, nonhierarchical conversations, a plurality of participants, and the incorporation of relevant knowledge and information on the subject under debate, undergirded by the idea that such discussions allow for people to learn and to change their minds, and for a general wisdom to emerge that is agreeable to the *Demos* and greater than the sum of its parts (that is, individual positions). Neither consensus nor deliberation necessarily requires voting, but sometimes they employ it. *Voting* can be defined as a process by which an individual registers their preferences (on an issue, for a representative, regarding where to order lunch, etc.) via some system for counting (a ballot, a stone, a raised hand, an online form). *Election* is the selection of political representatives through a voting process. *Sortition* is a process for choosing representatives or assembly members through randomization (lottery). Sortition was used in ancient Athens to choose the city council and juries, and is currently used by many Citizens' assemblies in the EU and UK, and to form juries in many countries.

In all these cases, technologies have always performed an important role. Tools for counting, randomization, and deliberation (sortition machines, amplification architectures) were important to ancient democracies. Technologies and political practices have changed together; voting machines and mass media have been key to practicing (and perhaps compromising or warping) representative democracy. More cutting-edge developments (such as decidim[5], democracy.os[6]) use online platforms and AI to aid in projects like participatory budgeting, referenda, and direct decision-making, and have been adopted by organizations, social movements, municipal and even national governments (including Barcelona, the French National Assembly referendum platform, New York City, and many others.) While solutions exist for sharing power at the application layer, very little has been done to implement horizontality at the security/access control layer.

4 THREAT MODEL

Collective management of resources opens an organization to a collection of both internal and external threats. As such, it is expected that attackers may be outsiders impersonating members of a community as well as disgruntled insiders that are trying to subvert the system. We expect, however, that at least a simple majority of users will not collude, and will attempt to use the system to represent the collective goal of the broader community.

However, we also expect attackers to attempt to inject accounts in attempts to dilute the democratic process (e.g., a Sybil attack [11, 39]). While these types of attacks are often performed in the wild [42], we assume this can be solved by a strong authentication scheme, implemented as each organization sees fit, and thus we consider this outside of our threat model. One example of how this can be solved separately from the authorization is the membership authentication of the Debian project, for which membership is often gated through physically-present PGP key signing processes, forming a strong authentication method. This has been widely documented in work by Coleman [9].

Malicious users may abuse other elements beyond raw voting power. They could, for example, gather a group colluding users, who could abuse time-zones (e.g., making decisions and calling votes when many other people are not online), system availability (e.g., network outages, net-splits) to carry out exceptional votes. This can likely be mitigated by ensuring the vote parameters are adjusted to the nature of the community (e.g., by adjusting simple majority to relative majority or set a necessary threshold for quorum).

5 COLBAC: COLLECTIVE BASED ACCESS CONTROL

As discussed in previous sections, the choice of certain software, protocols, or techniques such as X509 certificates or PGP web of trust has implications in the level of horizontality possible for systems built on top of those software, protocols, or techniques. How, then, can we build a foundation such that organizations of different horizontality can use the same foundation and arrive at much differently structured organizations?

In this section we describe Collective Based Access Control, or COLBAC: our approach to a dynamic horizontality access control system that performs authorization and execution of access control

[3] For example, almost all self-identifying democratic states now allow women to vote. The question of citizenship and national borders as definitive of a *Demos* is becoming contested in current times of increased migration, and given the problem of governing objects of a global nature, such as communication technologies, economies, and climate change.

[4] Consensus is used in Quaker communities, by the Haudenosaunee (Iroquois) Confederacy Grand Council, in Indonesia native cultures. It was used in the 2015 United Nations climate change conference, and the 2020 French Citizen's Assembly for the Climate, and has also been used by many political groups and movements such as the SNCC, the alter-globalization movement of the 1990s, and the Occupy and 15M movements. In the world of technology, the Internet Engineering Task Force (IETF) also uses consensus.

[5] https://decidim.org

[6] https://worldjusticeproject.org/our-work/programs/democracyos

policies via democratic processes. We describe its requirements in Section 5.1, focusing on the dynamic horizontality that separates COLBAC from other access control systems. We then define the system itself in Sections 5.2, 5.3, 5.4 and 5.5. Finally, we discuss the properties of COLBAC in Section 5.6. We discuss the limitations of this system later in the paper in Section 8. Though this is, to our knowledge[7], the first attempt at defining a collective based access control system with dynamic and flexible horizontality, it is far from the only potential approach. Section 9 discusses some potential future work to improve upon it.

Democracy is a complex concept, and COLBAC only incorporates some aspects of democracy. For example, sortition is not utilized at all in COLBAC. For our definition of the *Demos*, we consider all members of a system using COLBAC do be part of the *Demos* and thus eligible to vote. Our design of COLBAC also assumes there exists an external channel for deliberation that is used by all members of the *Demos*, and that there may exist external channels for other forms of decision-making. However, every decision made in these external channels that affects the system must eventually be reflected by a vote within COLBAC, be it an Action Token Petition or a Delegation Token Petition.

Given these assumptions and the design of COLBAC, we can see that COLBAC allows for a large variety of different democratic governance schemes. More direct forms of democracy can be done using Action Tokens, and elective representation can be done using Delegation Tokens (described in Section 5.3). The spectrum between full consensus and majority decision-making is represented through the security parameter f and the decision of how much participation is required per vote and, therefore, how difficult it is to create a quorum can be fine-tuned using the security parameter m. The amount of time a petition remains open to votes is specificied by t. These security parameters are described in Section 5.2.

5.1 System Requirements

COLBAC is aimed at addressing a novel requirement in Cybersecurity research: access control and authorization given an organization with dynamic levels of horizontal control. Though previous approaches exist for hierarchical access control (such as MAC, RBAC, etc.), to our knowledge no access control model exists for organizations of dynamic and flexible horizontality. To realize this, our solution must be able to be flexible in terms of horizontality. Said another way, our system must not assume or define a pre-determined threshold for horizontal control, i.e., it must not assume majority, or super-majority, or full consensus is the preferred method of democratic participation. Instead, the threshold must be configured by the collective itself, and must be able to be changed when necessary. This allows for rapid temporary centralization of the system to respond to crises, or to perform a task that requires expertise that few members of the collective contain. However, these moments of centralization must be quick to expire and easy to override in order to prevent abuse of centralized power. Said another way, it must always be easier for the collective to return to more horizontal control than for a centralized entity to maintain control over the system.

However, it makes no sense to have horizontal or democratic control without transparency. An individual cannot meaningfully vote or otherwise decide on a practice, or place confidence in a representative, without understanding what action is going to be taken, and what actions have been taken in the past. More, if authority is abused, the collective must be able to notice the abuse, and remove the powers that allowed the abuse to take place. Thus, we can see that any horizontal access control system requires transparency. Towards this end, our system must have a method of logging information about the actions of individuals and the collective that is immutable and available.

Finally, our system is an authorization system, not an authentication system. We assume that there is a working authentication system for the system, and that all COLBAC users are already authenticated when they interact with the COLBAC reference monitor.

5.2 System Design

COLBAC presents a solution to access control that relies on the collective. However, as will be discussed later in this section, not all objects on the system will need to be collectively controlled or administered. As such, we define three distinct *spheres* of the system, or areas that require different approaches to access control. These spheres are the **Collective Sphere**, the **User Spheres**, and the **Immutable Sphere**.

In order to achieve different degrees of horizontality, there must be a portion of the system that is controlled not by any individual user of the system, but by some democratic process of the users of the system. We call this portion of the system the **Collective Sphere**, as it contains programs, files, and other resources only accessed based on collective authorization. In any horizontal system, the administrative functions of the operating system would need to exist within this portion of the system to allow for true collective control. Additionally, all services that the collective offers, or official sources of collective information, must also exist in this sphere.

However, not everything should be directly managed by the collective. Individual users may have their own files and programs, which they intend to use only in ways that do not affect other users of the system or the resources of the collective. This sphere, called the **User Sphere**, can use traditional DAC systems without affecting the horizontality of the system as a whole.

Finally, to have meaningful control of the system we must have transparency. To achieve this, a system must have an **Immutable Sphere**, or a portion of the file system and programs that cannot be altered once written to, including by democratic control. This allows for the system to provide append-only logs that are vital to maintaining collective control, as described later in this section. Additionally, this section can hold a list of inalienable rights that each participant in the system has, such as the right to a vote.

When the system is first installed, a *Registration Phase* will occur. During this phase, at least 2 users will be signed up in the system. These users will need to supply their public keys, which correspond to offline private keys, since they will be needed for later interactions with the system. In addition to supplying their public keys, the users will need to decide on an initial fraction f, a minimum

[7] This lack of democratic access control systems was identified through a review of all systemization of knowledge or survey papers in the ACM Computing Surveys Journal mentioning access control. Out of the 288 entries, none explicitly aimed at creating a flexibly horizontal access control system. The closest access control systems are described in Section 2.2

fraction m that will be used for action and delegation petitions, and the voting timeout parameter t, as explained later.

After the system is set up, users can interact with objects in the User Sphere as they would in any other system. However, to interact with any objects in the Collective Sphere, users would need to follow a specific authorization procedure consisting of three phases: the *Draft Phase*, the *Petition Phase*, and the *Authorization Phase*.

In the Draft Phase, depending on the action the user wishes to perform, they will write the code or commands that will interact with objects in the Collective Sphere, or will identify the permissions they will need to accomplish their task or tasks. At the end of this process, the user will have a draft Action or Delegation Token, as described later in Sections 5.3 and 5.4.

After the user has completed their token, they move on to the Petition Phase. In this phase the user sends their draft token to the reference monitor running on the system. This reference monitor will then forward the draft token to all other members of the system and ask for a vote. The users will then vote one of the following ways: yes, no, or abstain[8]. The system will wait for a pre-configured amount of time before marking all individuals who did not vote as abstaining.

After all votes are collected or the response period has timed out, the system enters the authorization phase. During the action phase the reference monitor counts all of the votes. If the number of votes divided by the total number of users is greater than or equal to m, and the fraction of yes votes to the total number of votes is greater than or equal to f, the petition is considered successful and the token is returned to the petitioning user. In addition to taking the action, the system logs the action, the edited file, and the permissions used in log files held in the immutable sphere. However, if the number of all votes divided by the total number of users is less than m or if the number of yes votes divided by the total number of votes is less than f, the petition fails, and the attempted action is logged in the immutable sphere.

5.3 Types of Tokens

As the previous section demonstrates, the token is the main form of interacting with the Collective Sphere. Users who wish to affect the Collective Sphere do so by creating a token which is then voted on by the users of the system. In order to facilitate easy interaction with the COLBAC system, there are multiple types of tokens.

The first token is called the **Action Token**. This token allows a single command, a small script, or a program to be run in the Collective Sphere, and to enjoy Collective Sphere access. This is the most straightforward type of token, since everything the token will allow to occur is known during the Petition Phase. However, this type of token is rigid and inflexible. If there is an error in the command, or in the code, a user would need to re-enter the Draft Phase, fix the error, and re-enter the Petition Phase to authorize a new token. The more a single Action Token attempts to do, the more likely there will be errors, causing frustration for both the users drafting the tokens and the users voting on them.

To avoid these issues, COLBAC also allows for **Delegation Tokens**, which allow the drafting user to act in the Collective Sphere for a given set of time, and with a given set of restrictions. The procedure is obtaining a Delegation Token is the same as that for an Action Token. However, the information that would be put in the token would be slightly different. The format for Action Tokens and Delegation Tokens are shown in Section 5.4.

Consider the example of the democratically run trade union introduced in Section 1. In this example, a new communication committee was put in power though a slate of democratic elections. However, former individuals who had access to read the emails that arrived to the collective's inbox did not allow members of the new committee to read or send emails. If this trade union had been using COLBAC, after the elections a Delegation Token would have been drafted to grant the new committee access to both read and send emails. This Delegation Token would then gone through a Petition Phase, where presumably it would pass[9]. Thus, the individuals previously in charge of email would be incapable of denying access to the new committee.

There are some instances in which one needs to respond to emergency situations as soon as possible, and cannot wait for a slow authorization process by a potentially large set of users. To accommodate these situations, COLBAC has an **Emergency Token**. These tokens allow users to run short scripts, single commands, or small programs in the Collective Sphere without immediate authorization. However, this action is immediately logged and all users are informed by the Reference Monitor that an Emergency Token was used. In the case that an Emergency Token was incorrectly used (say, to overwrite the result of a democratically made decision), a member can create a new Action Token to undo the actions of the Emergency Token, which will be pushed to a Petition Phase for democratic decision making. To avoid large-scale misuse of the Emergency Token, and to avoid Emergency Token Wars[10], each member only has a small number of Emergency Tokens for a given period of time. Additionally, there are limits placed on what can be done with Emergency Tokens.

5.4 Token Format

Each token contains three sections, a header, a body, and a footer. Different types of tokens (Action Tokens, Delegation Tokens, or Emergency Tokens) contain different fields in their body sections. However, the headers and footers of all token types contain the same fields. For a graphical representation of the token format, please see Figure 1.

The first portion of a token is called the header. The header contains the following fields:

(1) **Nonce/ID**:
    An integer used to both identify the token and avoid replay attacks.
(2) **Token Type**:
    The type of token. Can only be Action, Delegation, or Emergency.

[8] It is important to mention that this is not the only way of performing democratic participation. However, other models of democratic participation are left to future work.

[9] This assumption is based off of the results of the democratic elections that occurred before.

[10] We define Emergency Token Wars as instances in which different members in the organization use Emergency Tokens to undo the actions taken in the name of the collective.

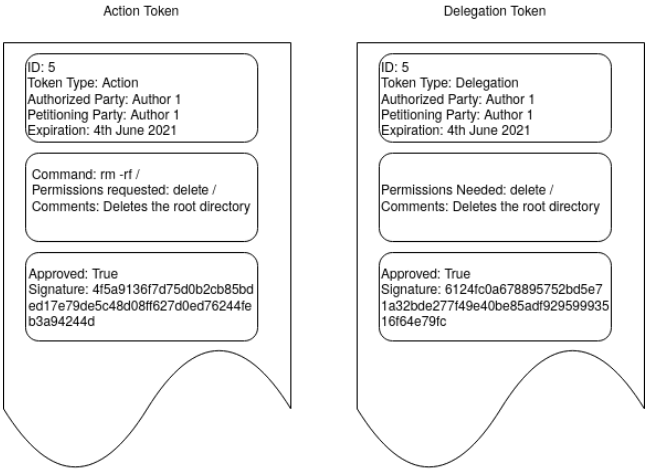

Figure 1: COLBAC Token Structures

(3) **Party(s) Being Authorized**:
A single entity or set of entities requesting authorization to perform an action or set of actions in the Collective Sphere.
(4) **Petitioning Party**:
The single entity petitioning for authorization. This user usually also exists in the list of parties being authorized.
(5) **Token Expiration**:
The expiration time of the token in Unix format.

For Action Tokens or Emergency Tokens, the token body contains the following fields:

(1) **Code to be Run**:
The command, script, or program to be run in the Collective Sphere.
(2) **Permissions Requested**:
A set of permissions requested to complete the task. These permissions include negative and positive permissions, of which negative permissions take precedence. The inclusion of both negative and positive permissions make it easier for permissions to be specified. For example, if a folder in the Collective Sphere contains 950 files, and the code running needs read access to 940 of them, it is easier to specify positive read permissions on the whole folder and negative read permissions on the 10 files, rather than specify 940 positive read permissions and no negative read permissions.
(3) **Comments (optional)**:
Comments explaining what the code included in the token does, and why it is necessary. This field is similar to messages included in version control systems like a Git commit message.

For Delegation Tokens, the body contains the following fields:

(1) **Permissions Needed**:
Similar to the set of permissions requested in the Action or Emergency Token body.
(2) **Comments (optional)**:
Similar to the comments section of the Action or Emergency Token body.

As we can see, the Delegation Token body is similar to the Action or Emergency Token body, except in that it does not specify the code that is running. This design is ideal for sessions that require troubleshooting, or tasks that may require back-and-forth between the system and the individual or group performing the task. However, when using Delegation Tokens it is more important to ensure that the Permissions Needed section follows the principle of least privilege. If not, individuals can use the privileges they gain from the Delegation Token to perform actions that were not originally intended for their task(s). Though these actions will be logged into the Immutable sphere, it still requires time and effort of the system users to undo the actions of an individual who abused the power granted to them through Delegation Tokens.

Each token, regardless of type, ends with a footer. The footer simply contains one field, a field which states if it is approved or denied, along with a verifiable authentication value that is difficult to predict and easy to verify, such as an HMAC of the permission token keyed by a secret value known only to the Reference Monitor.

5.5 Formalizing COLBAC

In this section we formalize COLBAC, a collective based access control system. To begin, we define important sets. We then go on to discuss which permissions are possible in COLBAC, what abbreviations are used in our notation, and what functions we rely on in our formalization. Finally, we introduce different algorithms used by the reference monitor in our proposed model. This formalization serves as a basis for our access control model.

Definition 1 - Spheres of COLBAC.

In COLBAC, a **Sphere** is a set which contains both subjects (users, processes, etc.) and objects (files, etc.).

Let U be the User Sphere, I be the Immutable Sphere, and C be the Collective Sphere. Let ξ be the set of all subjects and objects in the system. Let $\emptyset$ be the Empty Set. These sets have the following properties:

$$U \cup I \cup C = \xi$$
$$U \cap I = \emptyset$$
$$U \cap C = \emptyset$$
$$I \cap C = \emptyset$$

In addition to the sets above, our access control model also requires a set of permissions, similar to the sets of permissions used in other access control systems. These permissions will be used later when the reference monitor is deciding whether or not to permit an action.

Definition 2 - Permissions in COLBAC.

Create allows the creation of an object.
Append allows a subject to append to the end of an object.
Write allows arbitrary rights to an object.
Read allows a subject to read from an object.
Delete allows a subject to delete an object.

Execute allows a subject to run an object as a program.

The first important function we need to define is the *GetSphere* function. When passed a subject or object, the *GetSphere* function will return the Sphere that the subject or object belongs to. Given the properties mentioned in Definition 1, we can see that a subject or object will be in exactly one Sphere, meaning that this function will never return *NULL* or more than one value.

Definition 3 - GetSphere.

$GetSphere(o$: Subject or Object$) \rightarrow U$ iff $o \in U$, C iff $o \in C$, I iff $o \in I$

A very important building block of COLBAC is the **token**, which is a primitive taken from Capability based access control. A token will allow a subject which exists in U to perform an action on an object that exists in C. These tokens can be of type **Action**, denoted by a sub-scripted a, **Delegation**, denoted by a sub-scripted d, or **Emergency**, denoted by a sub-scripted e. Before authorization, a token is first created by a $DraftToken$ function, which calls one of the following three functions depending on the type of token the subject wishes to create. In addition, COLBAC requires a function to get the type of a given token.

Definition 4 - Token Functions.

Let u be a user of the system.
Let o be the object or set if objects the user is attempting to access.
Let p be the set of permissions the user is requesting.
Let τ be the type of token the user is attempting to create.
Let e be the proposed expiration time of the token.
Let c be the comment attached to a token, or in the case it doesn't exist, let c be *NULL*.
Let a be the action (command, script, or program) the user wants to run in the Collective Sphere, or in the case that it doesn't exist, let a be *NULL*.
Let d be the set of delegates the user is proposing, or in the case where it doesn't exist, let d be *NULL*.
$DraftToken(u, o, p, e, a, c, \tau) \rightarrow T$ of type T_a or T_d or T_e
$DraftToken_a(u, o, p, e, a, c) \rightarrow T_a = (u, o, p, e, a, c)$
$DraftToken_d(u, o, p, e, d, c) \rightarrow T_d = (u, o, p, e, d, c)$
$DraftToken_e(u, o, p, a, c) \rightarrow T_e = (u, o, p, a, c)$
$GetTokenType(\tau$: token$) \rightarrow$ {the type of token τ from *Action*, *Delegation*, or *Emergency*}

Unlike other capability based access control systems, COLBAC does not rely on a centralized authority or resource owner in the system to grant capability tokens. Instead, when a token is drafted by the user requesting it, the token enters a *Petition* function, which sends the token to all other users[11] of the system for a participatory decision-making process on whether or not the token should be authorized. This process lasts t seconds, as per the parameter of the system. The *Petition* function returns a set of votes on the token T, referred to as V_T, and can only be called on tokens of type *action* or *delegation*.

Let's consider again the example of the democratic trade union presented in Section 1. As mentioned in Section 5.3, in COLBAC this change of power would occur through a Delegation Token. Relating to the notation in Definition 4, if u were a member of the committee, o was the set of files, folders, and programs needed to read, write, and send emails, p was the necessary set of permissions to perform these actions[12], e was the expiration date of the token[13], d was the list of all members of the committee, and c was any human-readable comment the token drafter deemed necessary. Then, the token drafter would compute $T_d = DraftToken_d(u, o, p, e, d, c)$, and run $Petition(T, t)$ as follows.

Definition 5 - Petition Function.

Let T be a token created through $DraftToken$.
Let t be the system parameter that specifies how long voting is open.
$Petition(T, t) \rightarrow$ {set of votes V_T on authorizing T iff T is of type T_a or T_d, else *NULL*}.

This Petition Phase would collect votes from all of the members on the system after waiting t seconds. The returned set of votes, V, can then be split into more useful sets, such as the set of votes that affirm the authorization of T, $V_{(T,Yes)}$, the set of votes that negate the authorization of T, $V_{(T,No)}$, and the set of abstentions, $V_{(T,Blank)}$. After votes are sorted into their respective sets, the process of authorizing the votes comes down to a simple task of comparing the number of votes to fractions of voters initially defined at system registration. Said another way, a function called *AuthorizeToken* will count the votes and compare the results to two security parameters, f, or the fraction of yes votes required to authorize a token, and m, the fraction of voters required to participate in order for a vote to be considered. These two security parameters, along with the time a petition remains open to votes, are chosen at the initialization of the system, and can be changed by a successful Action Token[14]. If the vote is successful, the token is then authorized by the addition of a signature field that is difficult to predict or replicate, but easy for the reference monitor to later verify. One example of this is a keyed HMAC.

Definition 6 - Authorization in COLBAC.

Let V be the set of all votes on token T.
Let $V_{(T,Yes)} = \{v \in V \text{ s.t. } v = True\}$
Let $V_{(T,No)} = \{v \in V \text{ s.t. } v = False\}$
Let $V_{(T,Blank)} = \{v \in V \text{ s.t. } v = NULL\}$
$AuthorizeToken(T, V_T) \rightarrow True$ iff $\frac{|V_{(T,Yes)}|}{|V|} > f \wedge$

[11] Here we mean human users of the system, not subjects or user accounts on the system that don't correspond to humans.

[12] Such as the $Execute$ permission for the email program, the $Create$ and $Write$ permission for temporary files, the $Read$ permission for the files the emails are stored in, etc.

[13] Which should be set to the last day of the mandate of the elected body.

[14] These parameters cannot be changed by a delegate or by an Emergency Token. This is discussed more in Section 5.6.

$\frac{|V_{(T,Yes)} \cup V_{(T,No)}|}{|V|} \geq m$, else $False$

The process of determining authorization in COLBAC occurs in one or two phases, depending on which Sphere contains the object the subject wants to access. If the object is in the User Sphere, the reference monitor simply performs a traditional DAC check, like one would see on a typical Unix-like operating system.

If the object is in the Collective Sphere, the subject is expected to provide a previously authorized token or draft a new token for the reference monitor. If the subject provides a previously authorized token, the validity of that token is checked, and if it is valid, the action is taken. If the subject does not have a valid token, they instead provide a draft token. The reference monitor then takes this draft token and enters the Petition Phase, where all users of the system are given the opportunity to vote on the token. If the petition succeeds, the token is authorized and returned to the subject. If the petition fails, the token is not returned. In either case, the submission of the token to the reference monitor is logged in the Immutable Sphere. In the example provided in Section 1, all of the data required to access the collective email exists in the Collective Sphere.

If the object the subject is trying to access is in the Immutable Sphere, the reference monitor performs a very simple access control check. If the type of access on the resource in the Immutable Sphere is a *read* operation, it always succeeds. This is to provide transparency on the different operations that are attempted in the Collective Sphere, since the Immutable Sphere mainly contains logs of what was done in the Collective Sphere. If the type of access on the resource in the Immutable Sphere is a *write* operation, it always fails, since *write* operations can arbitrarily write to any portion of a file. Likewise, *delete* operations always fail. If the operation is an *execute*, *create* or *append* operation, the reference monitor must check that the subject has a valid token to do so. If not, they may not perform the action. The reference monitor, however, may always execute, append or create in the Immutable Sphere.

An algorithmic representation of these access control decisions are included in Algorithm 1. To continue with our example from Section 1, imagine that a member of the committee wants to read an email in the Collective Sphere. We can see that $GetSphere(o)$ would return C, since the email program exists in the Collective Sphere. Thus, the block of code from lines 14 to 20 would be executed, and the token would be checked. Assuming that the previous vote on the Petition Phase for the Delegation Token has passed, line 14 would return the non-NULL Delegation Token that u has as a member of the elected committee. Then, since the action that they wish to perform (read an email) is covered by the permissions of the Delegation Token T, the action will be logged on line 17 and then performed on line 18. However, if u were not a member of the committee, then the $GetToken$ function would return NULL, or would return tokens that do not have the relevant permissions. As such, lines 22 – 32 or line 19 would be executed, respectively.

Perhaps the most important part of this logic lies in the IsValid function. This function must ensure that the token itself has been issued by the reference monitor, that the token has not expired, and that the permissions of the actions that the token grants are consistent with what the action is attempting to do. In the case that one of these three conditions is not true, the reference monitor must not perform an action. We present the logic of the IsValid function in Algorithm 2.

To understand this Algorithm we still need to introduce one more function definition. In order to be able to decide whether

Algorithm 1 The main decision making process of COLBAC

1: Let u be a user of the system.
2: Let s be the subject attempting access.
3: Let o be the object or set if objects the user is attempting to access.
4: Let p be the set of permissions the user is requesting.
5: Let τ be the type of token the user is attempting to create.
6: Let e be the proposed expiration time of the token.
7: Let c be the comment attached to a token, or in the case it doesn't exist, let c be $NULL$.
8: Let a be the action the user wants to run in the Collective Sphere, or in the case that it doesn't exist, let a be $NULL$.
9: Let d be the set of delegates the user is proposing, or in the case where it doesn't exist, let d be $NULL$.
10: Let f, m and t be the system parameters reflecting approval fraction, quorum fraction, and petition voting time.
11: Let $DAC(s : \text{Subject}, o : \text{Object})$ be a Discretionary Access Control system.
12: **if** $GetSphere(o) = U$ **then**
13: Return $DAC(s, o)$
14: **else if** $GetSphere(o) = C$ **then**
15: $T = GetToken(u)$
16: **if** $T \neq NULL$ **then**
17: **if** IsValid(C, T, a) **then**
18: LogAction(a, u)
19: PerformAction(a)
20: **else**
21: LogFailedAction(a, u)
22: **else**
23: $T = DraftToken(u, o, p, \tau, e, c, a, d)$
24: **if** $GetTokenType(T) \in (T_a, T_d)$ **then**
25: $V_T = Petition(T, t)$
26: $V_{(T,Yes)} = \{v \in V \text{ s.t. } v = True\}$.
27: $V_{(T,No)} = \{v \in V \text{ s.t. } v = False\}$.
28: $V_{(T,Blank)} = \{v \in V \text{ s.t. } v = NULL\}$.
29: **if** $AuthorizeToken(T, V_{(T,Yes)}, V_{(T,No)}, V_{(T,Blank)}, f, m)$ **then**
30: LogSuccess(T)
31: Return T
32: **else**
33: LogFailure(T)
34: **else**
35: $T = GetToken(u)$
36: **if** IsValid(I, T, a) **then**
37: LogAction(a, u)
38: PerformAction(a)
39: **else**
40: LogFailedAction(a, u)

or not a given action should be allowed, the reference monitor must be able to see what permissions the action requires. Thus, we introduce the GetRequiredPermissions function as follows. This function works on the level of commands, instead of actions, to allow for granular identification of which portion of the action is causing issues (if indeed the action is longer than one command).

Definition 7.

GetRequiredPermissions(c : Command) $\rightarrow$ {the set of permissions, p' needed to complete the proposed command.}
PermissionType(p : Permission) $\rightarrow$ {the type of access the permission is requesting among the permissions listed in Definition 2}

Algorithm 2 The IsValid function of COLBAC.

```
1:  procedure IsValid(S: Sphere, T: Token, a: Action)
2:      Let C be the Collective Sphere.
3:      Let I be the Immutable Sphere.
4:      if S = C then
5:          if GetTokenType(T) ≠ T_e then
6:              for c ∈ a do          ▷ For each command in the action
7:                  p = GetRequiredPermissions(c)
8:                  if p ∉ T.p then
9:                      Return False
10:             Return True
11:         else
12:             p = GetRequiredPermissions(c)
13:             if CheckEmergencyPermissions(p) ≠ True then
14:                 Return False
15:             Return True
16:     else if S = I then
17:         for c ∈ a do              ▷ For each command in the action
18:             p = GetRequiredPermissions(c)
19:             if PermissionType(p) ∈ (write, delete, execute)
    then
20:                 Return False
21:             else if PermissionType(p) ∈ (create, append) then
22:                 if p ∉ T.p then
23:                     Return False
24:                 Return True
25:             else                  ▷ The permission is read
26:                 Return True
```

In this paper we leave CheckEmergencyPermissions as yet undefined, but discuss some possibilities for this function in Section 5.6.

5.6 Properties of COLBAC

In Section 5.2 we introduced the design of COLBAC, an access control system meant to serve as a primitive that can be used to achieve a more horizontal form of security. In Section 5.5 we presented a more formal definition of the system, filling in some of the details of how the system worked.

As stated in Section 5.1, the goals of the system design were to provide flexible and dynamic horizontality, and to provide transparency such that individuals participating in the system could reasonably participate in the decision-making process. Through the design of COLBAC we have created an access control system that meets these goals. We achieve flexible and dynamic horizontality in our design through the modification of our two security parameters, f and m. We achieve transparency through the use of the Immutable Sphere, and the logging that our reference monitor does during its operation (as described in Algorithm 1.

Though our design does provide for flexible and dynamic horizontality, it does not allow for every level of horizontality. For example, in our design a strictly-hierarchical approach is not possible: even if f is set to $\frac{1}{n}$, where n is the number of users, that simply means that one person needs to vote in favor to authorize an action, but it does not say *which* individual has that power, meaning that any single vote from any user is capable of authorizing an action. This is far from a dictatorial structure, in which all power is fixed in the hands of one individual or a small set of individuals.

This design does allow for many different expressions of horizontal structures, however. For example, because of the existence of Delegation Tokens which expire, our design allows for an authorization structure that reflects those of representative democracies, where elected individuals preside over certain responsibilities in the system. However, if the collective does not approve of how the elected individual is performing their duties, our solution allows for users to call for a democratic vote that invalidates the token of the delegate, thus forcing a new delegate to be chosen, or bringing the responsibility back into the collective.

More, our system allows for Action Tokens to modify the values of f and m, thus making it possible for an organization to adjust how many individuals in the collective must agree on authorizing a token in order for the token to be authorized. This allows the organization to decide if they want to work with a full consensus based approach, a super majority approach, a majority approach, or something else entirely. A consequence of this system is that it is easier to require **more** consensus than it is to require **less** consensus. This property is obvious when one realizes that if an organization calls a vote to switch from f to f' where $f > f'$, that action token still needs f votes, the larger number, to become authorized. However, if the organization wants to require **more** consensus, in other words a user petitions a token to change from f to f' where $f < f'$, then that vote requires f votes, the smaller number of the two, to be authorized. A similar claim can be made about the amount of interaction required, which would be affected by changing the security parameter m.

Another interesting property of this system is that it contains in it the inalienable right to vote. Whenever a vote is called, the reference monitor does not send it to a small set of individuals who are marked as having a right to vote. Instead, it sends it to all individuals who are a part of the system. Thus, in the COLBAC system the right to vote is inalienable as long as one remains part of the system. Likewise, because the reference monitor grants all read access to any object in the Immutable Sphere and disallows any deletions or arbitrary writes in the Immutable Sphere, transparency of the system is guaranteed so long as the reference monitor behaves according to the system design.

However, in order to avoid the equivalent of a coup d'état on the system, we must set some limitations of what certain tokens can do.

For example, if an individual user of the system wishes to perform a coup d'état, one approach would be to use an emergency token to set f and m to $\frac{1}{n}$ where n is the number of users in the system, then create and authorize action tokens to remove users from the system until only user accounts that are loyal to that user remain. To avoid this, there must be limitations on what Emergency Tokens and delegation tokens can do. For this case specifically, emergency and Delegation Tokens must not be able to alter the values of f and m. Similarly there must be a limitation on the number of emergency tokens an individual has, and individuals with access to delegation tokens must not be able to remove many members. It is likely that even more restrictions are required to maintain the safety of the system, but we leave the identification of those limitations, and further analysis of the properties of COLBAC, to future work.

6 SECURITY ANALYSIS OF COLBAC

In this section, we explore the security properties of COLBAC in two possible scenarios. First, Section 6.1 explores some of the possible technical attacks against the COLBAC system at different system endpoints. After, we elaborate on some well-known attacks against democracy, and how they fit into and are mitigated by COLBAC's design.

6.1 COLBAC failure modes and technical attack surfaces

COLBAC's ability to carry out collective decision making is dependent on the attackers' ability to affect foundational elements of Information Security: Confidentiality, Availability, Integrity and Non-repudiation. Although COLBAC is still in the "theoretical" space, we are able to explore the failure modes within the system and derive a taxonomy of potential attacks by malicious users (as per the categorization outlined in Section 4). In this taxonomy, the potential threats we face are:

- **Collective bypass**: a cadre of malicious users is able to perform actions bypassing the democratic process.
- **Sybil attack**: a malicious user is able to impersonate legitimate voters. Though we consider authentication as out of scope for this work, we do acknowledge that because of the democratic nature of COLBAC Sybill attacks can cause sustem takeover, which is a threat unique to democratic authorization systems.
- **System disruption**: a cadre of malicious users is able to halt the system from operating by disrupting voting operations.

To explore the attack surface, we enumerate the states and interfaces of COLBAC described in Sections 5.2 through 5.5 and explore the impact if these interfaces were compromised/mis-implemented.

DraftToken
Failure in the DraftToken function may materialize by a failure to authenticate the originator. As a consequence, it may allow malicious users to create tokens that impersonate actions on behalf of other users. This could lead to a *Collective bypass* or a *Sybil* attack.

PetitionToken
Within *DraftToken*, the *Petition* endpoint may fail as well. In this case, it is likely that a system stalled waiting for peer input (i.e., votes) may halt in a similar way as a SYN-flood attack. As such, the *Petition* function could be abused to cause a *System Disruption*. Similarly, *Vote* can be abused by attackers to flood the system with bogus votes, with a similar consequence.

AuthorizeToken
Finally, *AuthorizeToken* likely serves as a gatekeeper for actions running on the system. An attacker in control of this endpoint is able to bypass any democratic process (i.e., a *Collective Bypass*). However, in contrast to failure in *DraftToken* functions, it is likely that a compromised *AuthorizeToken* endpoint could leave no trace of compromise. With this in mind, we envision this functionality of COLBAC to run as part of a hardened environment (e.g., a hardware enclave [35]).

6.2 Democratic Attacks and COLBAC

In addition to the technical attacks enumerated above, there are also political attacks that can occur against the system. Each of these attacks are not specific to our system, but rather are attacks against democratic forms of organization and management.

Malicious Representative
One such attack is the concept of a malicious representative, or an entity in the organization that has gained access to a resource through a Delegation Token but then does not perform the promised actions, or, worse, performs other actions that are not in the interest of the organization, causing a *Collective bypass*. Unfortunately, no technical solution exists to the problem of malicious representatives. However, previous democracies have mitigated this problem through accountability in the form of *checks and balances*.

In order to have checks and balances, a democratic system must have a group that is responsible for over-ruling the decisions made by potentially malicious representatives, and that group must have transparency into the actions of the representatives and the ability to undo the actions these representatives have done. In COLBAC, the group responsible for over-ruling the decisions made by the malicious representatives are the users themselves. Transparency is achieved through the existence of COLBAC's Immutable Sphere, where the elected representative cannot remove traces of their actions, nor can they deny members of the system the right to see the Immutable Sphere. If the users have decided that the representative has acted maliciously, they can call a Petition Phase for an Action Token that will revoke the representative's rights. They can also undo any actions that the representative has made through additional Action Tokens, or elect a new representative to do that via a new Delegation Token.

Misuse of Emergency Tokens
Similar to the Malicious Representative attack, individuals in the organization can use Emergency Tokens to perform an action that is against the collective will, causing a *Collective bypass* attack. Similar to the Malicious Representative attack, these attacks are mitigated through checks and balances within the system.

Artificial Quorum through Collusion
An Artificial Quorum attack occurs when a minority group in the

organization attempts to manipulate who is present at a vote to increase the chances of an unpopular outcome. This can result in a *Collective bypass*. This attack is likely to occur for organizations that have a low f value, and thus it can be mitigated through selecting a large enough value f to ensure a large group is needed for quorum.

Quorum Denial through Collusion
In direct contrast to an Artificial Quorum attack, a Quorum Denial attack occurs when a minority group in the organization refuses to log into the system to vote on a popular token, potentially denying quorum and resulting in a *Collective bypass*. This attack is likely to occur for organization that have a high f value, and thus it can be mitigated through selecting a large enough value f to ensure a small group cannot deny quorum.

7 USABILITY OF COLBAC

One of the largest potential issues that COLBAC faces is lack of usability. However, before we can discuss methods of improving COLBAC's usability, we first must analyze it.

The most obvious usability concern of the COLBAC system is the amount of time that a given petition phase may take. However, in order to reason about the usability of COLBAC's petition phase, we first need to define its factors. These include:

(1) The number of members of a COLBAC based system, m.
(2) The amount of time COLBAC petition phases are open, t.
(3) The average amount of time it takes for members to vote, a.
(4) Fraction of people required to form a quorum, f.

In order to measure the total time for a Petition Phase, we studied the users' access patterns to a running server. Our goal is to identify how often k different users log in to a system, assuming they would vote after logging in. To do this, we used nearly 10 years of entries from the Wikimedia database and event log dump system [27]. We obtained the actions performed by all administrators of the English Wikipedia site, including edition, creation and deletion events. With this information in hand, we measured the time it takes for a server to receive k consecutive accesses from unique administrator ids. In other words, we assumed that if a user were to start a Petition Phase at any point in the log the k subsequent actions performed by unique ids would be preceded by a successful login and by our assumption, a vote. After this, we normalized our results using the quorum value of votes needed to perform a Petition Phase. Using this data we can derive the vote times for a certain number of users. For example, for a quorum of 4, the Petition Phase times ranged from 60 seconds, to under 4 minutes in the 99^{th} percentile. Likewise, a COLBAC system configured with a quorum of 8 should finish a Petition Phase in around 80 seconds on average. Thus, on average, for each individual required to make a quorum, the average time for the Petition Phase increases by 10 seconds. It is worth mentioning that the number of active administrators oscillated around 5,000 at all times; however, the average vote time decreased from ≈24 seconds to ≈10 seconds.

However, the amount of time for the petition phase of a token is not the only usability or user experience concern of the COLBAC system. Additionally, issues may arise with voting fatigue. Specifically, large amount of votes on action tokens could cause users to become fatigued, and thus pay less attention to the details of each vote. To avoid this, voting can be simplified through the use of simplified democratic structures like elected representatives. Here we can have seats for administrating different functions of a system, and these seats can be elected by democratic vote. This can be done via long-lived delegation tokens with permissions relevant only to the elected representative's role within the system. If the representative misbehaves, their current tokens could be revoked with an action token, and new elections can be held. Similarly to how workers in some worker cooperatives hire their managers, or how union members elect their stewards, members in COLBAC can elect their administrators.

For organizations that prefer more direct democratic approaches, issues with voting fatigue can be avoided using non-technological organizational practices. For example, many more horizontally run organizations (such as some of the *Mondragon*[15] cooperatives in Spain or the democratic free school *Escola da Ponte* [16] in Portugal) have regular meetings where organizational decisions are made. Organizations that run COLBAC can also use these meetings to decide aspects of system administration. This alleviates voting fatigue, since results of these discussions can be batched into a few action tokens and the participants will know exactly what they are voting on before the Petition Phase of COLBAC even begins.

8 LIMITATIONS

Though COLBAC provides a first step towards democratic technology and horizontal security, it still suffers from some setbacks. Like most access control systems, COLBAC does not aim to solve issues regarding account takeover or fake accounts. Rather, solving these problems is left to outside solutions such as salted, peppered, and hashed passwords and identity confirmation by the organization running COLBAC. However, if identity verification and account security are not well handled they could lead to Sybill attacks against the COLBAC system. This could lead to tokens that would have failed to move past the Petition Phase to pass instead, or tokens that would have passed to fail because of compromised or fake accounts.

Additionally, while our work is, to our knowledge, the first to apply democratic processes to access control, it does not provide any new defenses to these democratic processes. If a democratic process is weak to an attack (for example, a group of individuals refusing to vote to deny the quorum needed to authenticate a token, or a Petition Phase being held when the opponents of the petition are likely to be busy and unable to vote), then COLBAC will also be vulnerable to that attack. Future work, including ethnographic and technical work, can explore how to defend against attacks aimed at democratic systems.

Moreover, in this work we do not cover all participatory organizational or governance schemes. Some governance schemes (such as *plein-air* citizens assemblies, or representative democracies that do not permit referendums) cannot trivially be represented with COLBAC. Future work will explore how to expand the set of governance schemes COLBAC can represent.

Other limitations of the COLBAC system fall in the area of usability and user experience. Democratic processes can be difficult for people to learn and participate in at first [31], and while COLBAC

[15] https://www.mondragon-corporation.com/en/
[16] https://www.escoladaponte.pt/

does bring democratic processes to technology, it does not provide a method of making democratic processes any simpler. This is a social problem, and is outside of the scope of this paper. Further, there will be a learning curve for individuals moving to the COLBAC system, even if they are familiar with democratic processes. In order to effectively use the COLBAC system, one must know what it is they are voting for, and what the potential outcomes of these votes are. Future work will explore methods of presenting this information to users such that they can make an informed decision while voting on a token. Such potential solutions could include static or dynamic analysis of the code proposed to be run, or performing "dry runs" of the software on a virtualized environment to determine its effects.

Ultimately, difficulties introduced by purely horizontal democratic processes may cause some organizations, specifically larger organizations, to adopt a more hierarchical elected representative system. However, this is not a limitation of the scalability of COLBAC, but rather a limitation of scalability of participatory structures themselves. However, COLBAC addresses this difficulty by providing the ability for organizations' systems to operate as representational democracies, allowing for organizations to choose the level of horizontality that works for them at a given time.

Though COLBAC does have limitations, no system is perfect when first created. Hierarchical access control systems are built on top of a monumental amount of previous work and lessons learned from previous adoptions. However, COLBAC is the first attempt at constructing a democratic access control system. As such, we will build upon the proposed system in future work to solve some of the security and usability issues, and we expect to arrive at a more usable and secure democratic access control system.

9 FUTURE WORK

A collective based access control system is a step in the direction of horizontally controlled technology: it provides an access control foundation that other democratically controlled technologies can build on top of. However, COLBAC's limitations leads to some interesting questions. Specifically, COLBAC's inability to represent all methods of democratic and participatory design leads us to wonder an access control method could be created that covers more modes of organization and governance. In order to achieve this, it would be necessary to first understand a few key points. How do horizontally run communities organize themselves? How do they develop trust? How do they view and interact with technologies, and how would they interact with more horizontal security technologies like COLBAC?

Answering these questions requires a mixture of ethnographic research, theoretical computer security research, systems construction with iterative design, and user experience and usability research. Thus, the creation of more horizontal security does not end with this work, but rather this work takes a small step into a much larger world of horizontal technologies.

The concept of horizontality of security introduces many opportunities for future work. The first, and possibly most obvious, opportunity is determining the horizontality of security in different types of organizations, including typical corporations, worker cooperatives, trade unions, activist groups, government agencies, and more. Through ethnographic research we can determine how they organize, develop trust, make security decisions in the physical world, and how they interact with centralized organizations.

After the ethnographic work begins to generate insights, we can begin to develop tools, techniques, and technologies that facilitate or encourage horizontal security. One potential positive outcome of this ethnographic research is adjusting COLBAC to be flexible enough to fit the democratic practices of potential COLBAC users. This will allow us to address the some of the well-known HCI issues of infrastructure [12] by making *deep* changes to COLBAC to reflect the users' mental models of democracy. Through this, we can mitigate the problem of *constrained possibilities* [12] by adjusting COLBAC's design to incorporate more forms of democracy. However, this seems to necessitate more configuration options for COLBAC, potentially requiring more *unmediated interactions* [12]. We leave solving this problem to future work.

Another potential positive outcome of this ethnographic research is observing how these organizations work with centralized organizations and systems, and using these established practices to generate methods of designing interfaces between COLBAC and more centralized, hierarchical systems. For example, how would a COLBAC-based system handle a request issues by law enforcement using a warrant? Can exceptions such as these be included in COLBAC without creating an easy path for those who which to turn the system strictly hierarchical?

Perhaps the most obvious previous work created by the COLBAC access control model is the creation of a COLBAC access control system, both as a single machine and in a distributed system. This will allow us to measure the impacts of different parameter choices for COLBAC and their implicaitons on the system, both in terms of security and usability.

After improving COLBAC based on our findings, we can begin to solve other current horizontal security problems, such as password wars [21], and other aspects of horizontal secret sharing. However, in addition to the small advancements, one cannot lose sight of the bigger picture. Larger advances must also be made to allow for a large-scale shift to horizontal technology. Taking what we learn from our ethnographic work, we can begin to implement other technology that reflects the organization, trust development, and security decision making they display in physical matters. We can also measure how the introduction of horizontal security techniques affect the organization of the communities, and reflect these changes in new technologies as well.

More, we must ensure that the solutions we develop are usable. The creation of a horizontal security system, or any system, is only useful if people are willing to use it, and are able to use it effectively. Thus, research into the user experience and usability of these solutions is a requirement for their continued growth and development.

Finally, the ultimate aim of our work would be a general system. Can we create a concrete and parseable language to express different methods of organization with varying levels of horizontality? If so, can this language cover all possible forms of governance and organization, or only a subset of them? If this language were created, could we create a system that takes as input a description of a governance system in this language, and dynamically adjusts the access control needs to fit the governance structure? If so, how would this system operate? What would it output? And can it, itself,

be secured from misuse? Future research examines these questions and more.

10 CONCLUSIONS

In this work we presented the concept of horizontal security and presented COLBAC, a collective based access control system with dynamic and flexible horizontality to fit the needs of the organization. We both informally and formally described the system, discussed its properties, and outlined its limitations. We then discussed the potential impact of the introduction of COLBAC, and concluded with several opportunities for interdisciplinary future work, including ethnographic research, theoretical computer security research, system design research, user experience research, and more.

ACKNOWLEDGMENTS

The authors with to thank Brendan Dolan-Gavitt for his input and feedback, New Security Paradigm Workshop's anonymous peer reviewers for their insightful feedback, and Shamal Faily for shepherding, assistance, and comments on this article. Jessica Feldman gratefully acknowledges the Ford Foundation and Alfred P. Sloan Foundation for their support of her research into Critical Digital Infrastructures, which informed this paper. Feldman's research with this project was also supported by *l'Institut français du Monde associatif*, under the aegis of the Foundation for the University of Lyon, and by *l'Institut National pour la Jeunesse et l'Éducation Populaire*. This research work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no 952226.

REFERENCES

[1] [n.d.]. Free and Open Source Governance. https://fossgovernance.org/
[2] Christopher W Blackwell. 2003. Athenian democracy: an overview. *Dēmos: Classical Athenian Democracy* (2003), 1–57.
[3] GR Blakley and GA Kabatianskii. 1993. Linear algebra approach to secret sharing schemes. In *Workshop on Information Protection*. Springer, 33–40.
[4] Joseph Bonneau. 2018. Hostile blockchain takeovers (short paper). In *International Conference on Financial Cryptography and Data Security*. Springer, 92–100.
[5] Glenn Bruns and Michael Huth. 2011. Access control via belnap logic: Intuitive, expressive, and analyzable policy composition. *ACM Transactions on Information and System Security* 14, 1 (May 2011), 1–27. https://doi.org/10.1145/1952982.1952991
[6] Barbara Carminati and Elena Ferrari. 2011. Collaborative Access Control in On-line Social Networks. In *Proceedings of the 7th International Conference on Collaborative Computing: Networking, Applications and Worksharing*. IEEE. https://doi.org/10.4108/icst.collaboratecom.2011.247109
[7] Miguel Castro, Barbara Liskov, et al. 1999. Practical byzantine fault tolerance. In *OSDI*, Vol. 99. 173–186.
[8] Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. 1985. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)*. IEEE, 383–395.
[9] E Gabriella Coleman. 2012. *Coding freedom*. Princeton University Press.
[10] Jack B Dennis and Earl C Van Horn. 1966. Programming semantics for multiprogrammed computations. *Commun. ACM* 9, 3 (1966), 143–155.
[11] John R. Douceur. 2002. The Sybil Attack. In *Peer-to-Peer Systems*, Peter Druschel, Frans Kaashoek, and Antony Rowstron (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 251–260.
[12] W Keith Edwards, Mark W Newman, and Erika Shehan Poole. 2010. The infrastructure problem in HCI. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 423–432.
[13] Paolo Gerbaudo. 2017. Social media teams as digital vanguards: the question of leadership in the management of key Facebook and Twitter accounts of Occupy Wall Street, Indignados and UK Uncut. *Information, Communication & Society* 20, 2 (2017), 185–202.
[14] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. 2015. Eclipse attacks on bitcoin's peer-to-peer network. In *24th USENIX Security Symposium (USENIX Security 15)*. 129–144.
[15] Hongxin Hu, Gail-Joon Ahn, and Jan Jorgensen. 2013. Multiparty Access Control for Online Social Networks: Model and Mechanisms. *IEEE Transactions on Knowledge and Data Engineering* 25, 7 (Jul 2013), 1614–1627. https://doi.org/10.1109/TKDE.2012.97
[16] Panagiotis Ilia, Barbara Carminati, Elena Ferrari, Paraskevi Fragopoulou, and Sotiris Ioannidis. 2017. SAMPAC: Socially-Aware collaborative Multi-Party Access Control. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*. ACM, 71–82. https://doi.org/10.1145/3029806.3029834
[17] Panagiotis Ilia, Iasonas Polakis, Elias Athanasopoulos, Federico Maggi, and Sotiris Ioannidis. 2015. Face/Off: Preventing Privacy Leakage From Photos in Social Networks. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 781–792. https://doi.org/10.1145/2810103.2813603
[18] Investopedia. [n.d.]. A History of Bitcoin Hard Forks. https://www.investopedia.com/tech/history-bitcoin-hard-forks/.
[19] Robert Jackall. 1984. 6. Paradoxes of Collective Work: A Study of the Cheeseboard, Berkeley, California. In *Worker cooperatives in America*. University of California Press, 109–136.
[20] Anastasia Kavada. 2015. Creating the collective: social media, the Occupy Movement and its constitution as a collective actor. *Information, Communication and Society* 18, 8 (2015), 872–886.
[21] Anastasia Kavada and Thomas Poell. 2020. From Counterpublics to Contentious Publicness: Tracing the Temporal, Spatial, and Material Articulations of Popular Protest Through Social Media. *Communication Theory* (2020).
[22] Jonathan Kuyper. 2016. Global Democracy. In *The Stanford Encyclopedia of Philosophy* (Winter 2016 ed.), Edward N. Zalta (Ed.). Metaphysics Research Lab, Stanford University.
[23] Leslie Lamport. 2019. The part-time parliament. In *Concurrency: the Works of Leslie Lamport*. 277–317.
[24] Barry Leiba. 2012. Oauth web authorization protocol. *IEEE Internet Computing* 16, 1 (2012), 74–77.
[25] Frank Lindenfeld. 1982. *Workplace democracy and social change*. Extending Horizons Books.
[26] M4BL. [n.d.]. Policy Platforms: Community Control. https://m4bl.org/policy-platforms/community-control/.
[27] Meta. 2021. Data dumps — Meta, discussion about Wikimedia projects. https://meta.wikimedia.org/w/index.php?title=Data_dumps [Online; accessed 20-August-2021].
[28] Satoshi Nakamoto. 2019. *Bitcoin: A peer-to-peer electronic cash system*. Technical Report. Manubot.
[29] Diego Ongaro and John Ousterhout. 2013. In search of an understandable consensus algorithm (extended version).
[30] José Pacheco. 2008. Escola da ponte. *Formação e Transformação da Educação. Sl* (2008).
[31] Francesca Polletta. 2012. Freedom is an endless meeting. In *Freedom Is an Endless Meeting*. University of Chicago Press.
[32] Kai Rannenberg. 2000. Multilateral security a concept and examples for balanced security. In *Proceedings of the 2000 workshop on New security paradigms - NSPW '00*. ACM Press, 151–162. https://doi.org/10.1145/366173.366208
[33] Prathima Rao, Dan Lin, Elisa Bertino, Ninghui Li, and Jorge Lobo. 2011. Fine-grained integration of access control policies. *Computers & Security* 30, 2–3 (Mar 2011), 91–107. https://doi.org/10.1016/j.cose.2010.10.006
[34] Phillip Rogaway. 2015. The Moral Character of Cryptographic Work. *IACR Cryptol. ePrint Arch.* 2015 (2015), 1162.
[35] Carlos Rozas. 2013. Intel® Software Guard Extensions (Intel® SGX).
[36] Adi Shamir. 1979. How to share a secret. *Commun. ACM* 22, 11 (1979), 612–613.
[37] Marina A Sitrin. 2012. *Everyday revolutions: Horizontalism and autonomy in Argentina*. Bloomsbury Publishing.
[38] Cryptopedia Staff. [n.d.]. What was the DAO? https://www.gemini.com/cryptopedia/the-dao-hack-makerdao.
[39] Zied Trifa and Maher Khemakhem. 2014. Sybil Nodes as a Mitigation Strategy Against Sybil Attack. *Procedia Computer Science* 32 (2014), 1135–1140. https://doi.org/10.1016/j.procs.2014.05.544 The 5th International Conference on Ambient Systems, Networks and Technologies (ANT-2014), the 4th International Conference on Sustainable Energy Information Technology (SEIT-2014).
[40] Kevin V Tu and Michael W Meredith. 2015. Rethinking virtual currency regulation in the Bitcoin age. *Wash. L. Rev.* 90 (2015), 271.
[41] Langdon Winner. 1980. Do artifacts have politics? *Daedalus* (1980), 121–136.
[42] Philipp Winter, Roya Ensafi, Karsten Loesing, and Nick Feamster. 2016. Identifying and Characterizing Sybils in the Tor Network. In *25th USENIX Security Symposium (USENIX Security 16)*. USENIX Association, Austin, TX, 1169–1185. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/winter
[43] Chris Wright. 2014. *Worker cooperatives and revolution: History and possibilities in the United States*. BookLocker.