DNSonChain: Delegating Privacy-Preserved DNS Resolution to Blockchain

Lin Jin
University of Delaware
linjin@udel.edu

Shuai Hao
Old Dominion University
shao@odu.edu

Yan Huang
Indiana University
yh33@indiana.edu

Haining Wang
Virginia Tech
hnw@vt.edu

Chase Cotton
University of Delaware
ccotton@udel.edu

***Abstract*—Domain Name System (DNS) is known to present privacy concerns. To this end, decentralized blockchains have been used to host DNS records, so that users can synchronize with the blockchain to maintain a local DNS database and resolve domain names locally. However, existing blockchain-based solutions either do not guarantee a domain name is controlled by its "true" owner; or have to resort to DNSSEC, a not yet widely adopted protocol, for verifying ownership. In this paper, we present DNSonChain, a new blockchain-based naming service compatible with DNS. It allows domain owners to claim their domain ownership on the blockchain where DNS records are hosted. The core function of DNSonChain is to validate the domain ownership in a decentralized manner. We propose a majority vote mechanism that randomly selects multiple participants (i.e., voters) in the system to vote for the authority of domain ownership. To provide resistance to attacks from fraudulent voters, DNSonChain requires two rounds of voting processes. Our security analysis shows that DNSonChain is robust against several types of security failures, able to recover from various attacks. We implemented a prototype of DNSonChain as an Ethereum decentralized application and evaluate it on an Ethereum Testnet.**

I. Introduction

The Domain Name System (DNS) provides vital mappings between domain names and their numerical IP addresses to direct users to Internet services. However, DNS protocol was originally designed as an unencrypted protocol that allows eavesdroppers to sniff the domain that a user is going to visit, raising a well-known privacy concern.

One possible way to mitigate this privacy problem is to encrypt DNS traffic. However, encrypted DNS traffic is still vulnerable to traffic analysis, and resolvers are required to be trustworthy. Alternatively, researchers and practitioners are leveraging decentralized blockchain technology to host DNS records, so that users can download DNS records from a blockchain to their local storage to build their own DNS databases. As a result, the name resolution becomes local DNS lookup without generating any network traffic, and hence preserves DNS privacy. However, existing blockchain-based naming systems [1], [15], [32] are incompatible with DNS as either domain owners may not be able to hold their own domain names on blockchains or an ownership verification requires DNSSEC [4], which is currently not widely adopted due to limited registrar support [11] and dynamic DNS mappings [23].

978-1-6654-4131-5/21/$31.00 ©2021 IEEE

In this paper, we propose DNSonChain, a new blockchain-based naming system, to address the DNS's privacy issues. It is compatible with the current DNS system and bridges the Internet users who have concerns on their DNS privacy and the domain owners who would like to provide privacy benefits to their visitors. DNSonChain is a decentralized system realized as smart contracts running on Ethereum. It allows a domain owner to claim its domain ownership on the blockchain without the support from DNSSEC, and host its DNS records on the blockchain thereafter. To achieve this goal, DNSonChain needs to validate the domain ownership on DNS. However, as a decentralized system, DNSonChain has no central point of authority, thereby the ownership validation has to be done in a decentralized manner. To this end, we propose a majority vote mechanism in which multiple participants (i.e., voters) in the system are randomly selected to validate the domain ownership by performing normal DNS lookups, and then vote for its correctness. To verify domain ownership, a domain owner stores its blockchain ID in a `TXT` record that can be retrieved by voters. Once the ownership is established, the domain owner is allowed to update its DNS records on the blockchain. Also, the domain owner can associate multiple IP addresses to its domain for load balancing purposes.

The development of DNSonChain faces three major challenges. First, the majority vote mechanism must be able to resist Sybil attacks, where adversaries can employ a large number of pseudonymous identities so that they have an unfairly high probability to be selected. We defend against such a Sybil attack with stake-based committee selection where each voter's voting weight is determined in proportion to its stake (or asset). Second, domain ownership recorded by DNSonChain may be inconsistent with ordinary DNS due to a domain takeover attack or domain expiration. We introduce a cleanup procedure to address these issues by incentivizing participants who notice the questionable ownership to rectify the ownership records. Third, adversaries may maneuver votes through DNS manipulation as voters validate the domain ownership with a normal DNS lookup. DNSonChain provides resistance against such attacks by requiring two separate rounds of voting processes with a prescribed interval. The interval raises the bar for adversaries to attack a domain on DNSonChain, as they have to either succeed in the nameserver attacks twice or compromise a nameserver for a lasting period longer than the interval.

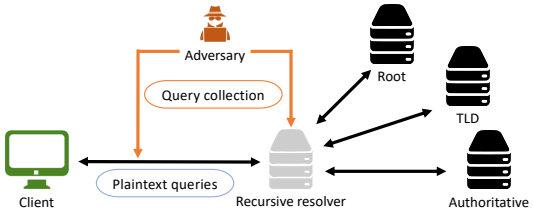

Fig. 1. DNS Privacy Issue.

We implement a prototype of DNSonChain and deploy it to the Ropsten network, an Ethereum Testnet. We demonstrate that one voting process can be done within several minutes in real scenarios. Moreover, we show that a domain owner can incentivize the voters to vote for its domain with a small cost.

The remainder of this paper is organized as follows. Section II surveys the background and other related work. We present the system overview of DNSonChain in Section III and detail the majority vote mechanism in Section IV. Section V describes the ownership management on DNSonChain. Section VI provides the security analysis of DNSonChain. Implementation and evaluation of DNSonChain are described in Section VII, and we conclude the paper in Section VIII.

II. Background and Related Work

DNS Privacy. As shown in Figure 1, to conduct a DNS resolution, a client issues a DNS query to a recursive resolver which will then traverse the DNS hierarchical database and return the requested records to the client. As DNS is an unencrypted protocol, adversaries on the path between the client and its resolver including the resolver can sniff the client's DNS queries, allowing adversaries to profile the client's browsing activities that are considered private information.

To address such a privacy concern, the idea of encrypted DNS (e.g., DNS-over-HTTPS (DoH) [25] and DNS-over-TLS (DoT) [27], [36]) has been proposed to secure the communication between the client and resolver. However, existing studies [26], [33], [34] have demonstrated that encrypted DNS traffic is still vulnerable to traffic analysis, which could lead to privacy leakage. More importantly, an encrypted DNS provider still has the capability to examine and manipulate its clients' DNS queries [29]. As a result, the encrypted DNS techniques do not fundamentally resolve the DNS privacy issues for clients, instead, it shifts the information collector from, usually, the client's local ISP to the encrypted DNS providers. In comparison, DNSonChain completely addresses such issues by eliminating the network traffic when conducting DNS lookups, and hence no on-path adversaries or third-party resolvers could get involved. We notice that, except for DNS, there are multiple channels such as HTTP(S) that can be leveraged to leak user privacy, and DNSonChain only focuses on addressing DNS privacy issues.

Blockchain and Smart Contract. Blockchain is a public ledger that permanently and unalterably records all the transactions sent by users. Blockchain systems leverage consensus protocols to achieve agreement on the next block. Bitcoin [7] and Ethereum [18] adopt the proof-of-work protocol where miners repeatedly guess a nonce so that the hash of the nonce and block data satisfies a requirement. Algorand [21] proposes a Byzantine agreement protocol that different sets of users are selected to propose and vote for blocks.

A smart contract is a piece of code programmed by users and stored in the blockchain. Variables and functions are defined in the smart contract. Users interact with smart contracts by sending blockchain transactions. All miners execute the functions of the smart contract when they are called and update the variables accordingly. With the ability of decentralized computing, smart contracts enable decentralized applications, which is the way that DNSonChain is built upon.

TABLE I
Summary of Blockchain-based Naming Systems

	Separate TLD	Registrar	DNS Compatibility
Namecoin	•	•	
Blockstack	•	•	
ENS	•	•	Limited†
DNSonChain			•

† DNSSEC is required for domain ownership validation.

Blockchain-based Naming System. Namecoin [32] is a Bitcoin-based system that attempts to decentralize DNS. Since it does not have a mechanism to validate domain ownership, all the domains in Namecoin under `.bit` top-level domain, which is not in the DNS namespace. Kalodner *et al.* [30] thoroughly analyzed the Namecoin system and revealed that most of the registered domain names are squatted names.

Ali *et al.* [1] designed Blockstack, a naming and storage system built upon Bitcoin. In Blockstack, the data can be stored off the blockchain so that there is no limit on data size, and it is flexible to update the data value without making a new transaction on the underlying blockchain. However, like Namecoin, it cannot validate the domain ownership of DNS.

Ethereum Name Service (ENS) [15] is another blockchain-based naming system built on Ethereum [18]. To claim the domain ownership on ENS, the domain needs to enable DNSSEC [9], [16], which unfortunately has an extremely low adoption. According to [10], [11], less than 1% of second-level domains with `.com` TLD are properly signed, and one-third of the signed domains are misconfigured.

In summary, all these Blockchain-based naming systems are not well-compatible with DNS. In contrast, our DNSonChain is compatible with DNS by enabling all domain owners to hold their same domain names on the blockchain. Table I summarizes the current blockchain-based naming systems.

III. System Overview

High-level Idea. DNSonChain provides DNS privacy protection by hosting DNS records on a blockchain. Users who synchronize the blockchain can build a local DNS database to enable local DNS lookup. DNSonChain's core function is to validate and manage the ownership of domains in a decentralized fashion, implemented as smart contracts running over a blockchain.

System Participants. In DNSonChain, each participant is also a user of the underlying blockchain with a pair of public and private keys, and its blockchain ID is determined by its public key. The participant communicates with DNSonChain by sending blockchain transactions. The DNSonChain involves three types of roles: the domain owners, the voters, and the cleaners. Note that a participant can have multiple roles.

(1) *Domain owners* are those Internet users who have the authorization to modify the DNS records of a domain. A domain owner only needs to claim the ownership of its apex domain (e.g., `example.com`). After that, the domain owner controls the apex domain as well as all its subdomains. To claim and hold the domain ownership on DNSonChain, domain owners need to show their capability to control their domains on DNS. Domain owners are motivated to participate in DNSonChain to offer privacy benefits to their domain visitors.

(2) *Voters* are those users who deposit money (i.e., the cryptocurrency) into the system as their stakes so that they have a non-zero probability to be selected as committee members. The committee members help to validate the ownership of domain owners. Voters are incentivized to validate domain ownerships for earning rewards from their honest votes.

(3) *Cleaners* can be any participants who help DNSonChain clean up the domain ownership records that are inconsistent with the traditional DNS. Cleaners are incentivized to clean the ownership by receiving rewards.

Majority Vote. DNSonChain validates domain ownership through a majority vote mechanism. First, the domain owner uploads its blockchain ID to a TXT record of a pre-defined domain, e.g., `_dnsonchain_.example.com`. Then, a selected committee of voters retrieve and verify the TXT records before casting their votes.

When a voting request is received, each voter first determines whether itself is being selected as a voting committee member. The selection is done in a *Proof-of-Stake* manner to defend against Sybil attacks. Therefore, a voter's influence is in proportion to its stakes, and the impact of malicious voters is bounded by their total stakes. Next, the selected committee members send DNS queries to retrieve the TXT record of the corresponding domain and validate whether an expected blockchain ID is stored. Their votes of domain ownership are then cast, and the majority vote is regarded as the final voting result.

To incentivize the voters to behave honestly, DNSonChain rewards the voters whose votes are in line with the final voting result. Following the same idea, one can also prevent participants from abusing the system by penalizing dishonest voters.

DNSonChain Operation. DNSonChain manages the domain ownership based on the voting results. It includes granting the ownership to domain owners after validating votes. Since one claim/cleanup voting process may be accidentally dominated by malicious participants, DNSonChain requires two rounds of claim/cleanup voting processes to successfully claim/cleanup domain ownership. Moreover, there is a requirement of the interval between the two claim/cleanup voting processes. The interval raises the bar for adversaries to attack a domain on DNSonChain. In addition, it provides opportunities for a potential victim domain owner to avoid fraudulent ownership cleanups.

IV. Majority Vote

A. Committee Selection

In each voting process, DNSonChain randomly selects a subset of voters as a committee qualified to vote. In order to counter a Sybil attack, every voter is selected with a probability proportional to its stake in the system.

Stake pool. DNSonChain manages a stake pool in which anyone can deposit money (i.e., the *stake*) to become a voter. The stake can be withdrawn later. The stake pool records the most updated amount of stake held by each voter, as well as the time of the latest stake update. When a voter updates its stake by either making a deposit or withdrawal, it loses its eligibility to vote for all voting processes launched prior to the update. We set this restriction because (1) a voter's stake is the selection algorithm's input that determines the selection result, and (2) DNSonChain does not keep a history of stake update due to high storage cost. Without such a restriction, a voter can obtain unfair influence over the selection result by manipulating its stakes during the voting process.

Selection Procedures. Voters conduct self-selection locally to learn their qualifications to vote. In particular, each voter finds its tickets for a voting process through the ticket generation algorithm shown in Algorithm 1. A voter is selected as a committee member if it finds at least one ticket and the total number of tickets it finds is its voting weight. However, if no ticket is found, then the voter is not selected as a committee member for this voting process.

Algorithm 1 Ticket Generation

```
1:  procedure TICKETS(seed, V_id, s, S, len, W)
2:      tickets ← []
3:      p ← s/S
4:      threshold ← p * (2^len − 1)
5:      index ← 0
6:      while index < W do
7:          hash_value ← hash(seed, V_id, index)
8:          if hash_value < threshold then
9:              append(index, tickets)
10:         index++
11:     return tickets
```

Concretely, when a voting process is launched, DNSonChain specifies an expected voting weight W for the voting process. The W indicates the expected number of total tickets that can be found by all voters. In the meantime, DNSonChain records the total number of stakes S and generates a seed by hashing (1) the current block hash, (2) the blockchain ID of the voting requester, and (3) the requested domain. Also,

each voter calculates its stake fraction $p = s/S$, where s represents the number of stakes held by the voter. Then, each voter generates W hash values by hashing the seed, the voter's blockchain ID (V_{id}), and an index $i, i = 0, 1, ..., W-1$. A threshold of each voter is set to $p * (2^{len} - 1)$, where len is the length of the hash value. As such, if a hash value is smaller than the threshold, the index that generates the hash value serves as the voter's ticket. Figure 2 illustrates the ticket generation results.

For each voter, the ticket generation algorithm essentially performs W independent Bernoulli trials with parameter p. Therefore, the result of ticket generation follows the binomial distribution $Binomial(W, p)$, and the expected voting weight of a voter is Wp, which is proportional to its stake. As such, the expected voting weight of adversaries would not increase by distributing stakes to multiple pseudonymous identities.

Algorithm 2 Determine Voting Weight

```
1:  procedure WEIGHT(seed, V_id, s, S, len, W, tickets)
2:      tickets ← filter_duplicate_and_illegal(tickets)
3:      p ← s/S
4:      threshold ← p * (2^len − 1)
5:      weight ← 0
6:      for ticket ∈ tickets do
7:          hash_value ← hash(seed, V_id, ticket)
8:          if hash_value < threshold then
9:              weight++
10:     return weight
```

When a voter finds its tickets, it submits the tickets to DNSonChain for verification. Once the tickets are verified, DNSonChain obtains the voting weight of the voter. Algorithm 2 describes how DNSonChain verifies a voter's tickets and determines its voting weight in detail. First, DNSonChain filters out duplicate and illegal tickets. The illegal tickets are those that are not in the range of $[0, W-1]$. Then, DNSonChain recomputes the threshold for the voter. The threshold is calculated using the voter's stake s and total stake S. For an eligible voter, its stake at the time when DNSonChain verifies its tickets must be the same as its stake at the beginning of the voting process. Although DNSonChain does not record every voter's stake at the beginning of the voting process, it checks the time of the latest stake update to verify the eligibility of a voter. Thus, DNSonChain obtains the eligible voter's stakes s from the stake pool at the time of verification. As the total stake S is already recorded at the beginning of the voting process, the threshold is recomputed. Furthermore, DNSonChain recomputes the hash values by hashing the seed, the voter's blockchain ID, and filtered tickets. Finally, DNSonChain determines the voting weight by counting the number of recomputed hashes that are smaller than the threshold.

B. Cast Votes

When a voter has been selected as a committee member, it retrieves the TXT record of the pre-defined subdomain under the requested domain. Voters determine whether to cast an approval vote or a disapproval vote based on the type of the voting process, i.e., the claim voting process and cleanup voting process. In the claim voting process, if the content in the TXT record is the blockchain ID of the voting requester, voters cast an approval vote, and vice versa. In the cleanup voting process, if the content in the TXT record is *not* the blockchain ID of the current domain owner on DNSonChain, voters cast an approval vote, and vice versa.

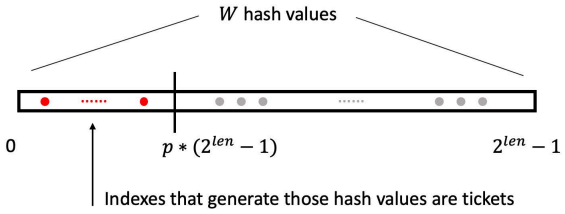

Fig. 2. Ticket Generation Illustration.

However, there is no guarantee that voters will follow the above procedures that honestly cast their votes based on what they retrieved from the TXT record, resulting in the ownership on DNSonChain being unreliable. To tackle this problem, DNSonChain first incentivizes voters to behave honestly by rewarding the voters who vote for the final decision (i.e., only voters who in the majority camp receive rewards). Then, as we have the assumption that the majority of stakes are held by the good voters, the probability that good voters dominate the voting process approaches 1 (Section VI-A).

Nevertheless, such an incentive is still not sufficient to motivate voters to behave honestly. One subtle implication is that voters could simply follow others' votes to become the majority at a high chance. This is because the votes, essentially the blockchain transactions, sent by voters will be available to all other voters once the transactions are being executed. Therefore, one strategy for the voters to become the majority is to wait for enough votes till there is a clear trend to infer what is the final decision of the voting process, and then simply submit the majority vote to DNSonChain.

To discourage such lazy followers, we design a two-stage voting strategy called `Commit & Fulfill`, where each voter first makes a commitment to its vote at the `Commit` stage and then reveals its vote at the `Fulfill` stage. In particular, When a voting request is received by DNSonChain, the voting stage is set to the `Commit` stage. Each committee member generates a nonce and calculates a commitment by hashing the nonce and its vote. Then, they submit the commitments to DNSonChain and wait for the `Fulfill` stage. Note that the commitment can be sent in the same transaction as the tickets. DNSonChain records the commitment for each voter and sets a threshold τ, which controls how much voting weight it needs.

Once the total received voting weight reaches τW, DNSonChain stops receiving commitments and sets the voting stage to the `Fulfill` stage.[1] After that, each voter whose commitment

[1]This also means that commitments received after the `Commit` stage will be ignored, which encourages voters to commit as early as possible.

has been added to DNSonChain sends its vote and nonce together to DNSonChain to fulfill its commitment. DNSonChain calculates the hash of the nonce and vote, and then verifies if it matches the commitment received at the `Commit` stage. Only those votes that have their commitments matched will be counted towards the final decision. Note that each commitment received at the `Commit` stage must be unique, otherwise a voter may still be able to follow others' vote by copying both the commitment at the `Commit` stage and the vote and nonce at the `Fulfill` stage. Therefore, DNSonChain rejects duplicate commitments received at the `Commit` stage. The commitment will always be unique as long as voters do not leak their nonces before the `Fulfill` stage and the length of the nonce and commitment are long enough, e.g., 256 bits. In this manner, voters are not able to follow others' votes. Hence, to be the camp of the majority, a good voter is incentivized to perform the domain validation and cast a vote honestly as other good voters will do the same. Note that both the `Commit` and `Fulfill` stage will be terminated if any of them cannot receive enough legitimate votes in a certain period, resulting in the abandonment of the voting process. However, assuming that most of the participants in DNSonChain are benign users, the two stages should be able to successfully converge due to the reward incentive.

Once a vote is revealed, DNSonChain adds its voting weight to the corresponding camp (approval or disapproval). Note that the voting weight is calculated at the `Commit` stage. DNSonChain determines the final voting decision as a success if the total voting weight of approval votes is greater than $\tau W/2$, and vice versa. DNSonChain finally manages the ownership based on the final voting decision, and we describe the details in Section V.

C. Rewards and Penalties

Participants who communicate with DNSonChain incur costs, since they need to send transactions to the underlying blockchain and usually blockchain transactions require costs, such as transaction fees. To encourage participation, DNSonChain follows the same convention of decentralized systems where rewards are presented to the miners/validators.

In DNSonChain, voters are incentivized by earning rewards from voting requesters. Specifically, along with each voting request, the voting requester (i.e., a domain owner or a cleaner) presents a reward offer for the voters. The reward offer states how much money it would pay for committee members who vote for the final decision, and is escrowed at DNSonChain. No matter what the voting result is, the rewards will be assigned to the corresponding committee members once the voting process is completed. However, it is possible that malicious voters commit their votes but do not fulfill them so that the voting process cannot be completed. In this case, DNSonChain would apply penalties to those misbehavioral voters. To enforce this penalty, DNSonChain locks a voter's stakes once it commits a vote, so that the voter cannot withdraw its stake from the stake pool before it fulfills the vote or is penalized. Once the penalty stakes are applied, it goes to the voters who fulfilled their votes, and voting requesters could also take their reward offers back.

Domain owners incentivize voters for their ownership on DNSonChain, and cleaners incentivize voters for their cleanup rewards. The cleanup rewards are generated from the system. However, a malicious cleaner may launch cleanup voting processes to legitimate domains, not for taking away its ownership, but for abusing cleanup rewards. To prevent such abuse behaviors, DNSonChain only rewards the cleaners who succeeded in a cleanup voting process. The offer the cleaners made for voters will be seen as penalties for the cleaners if the cleanup voting process fails. On the other hand, if the cleaner is honest, the chance that a voting process fails will be negligible.

V. DNSonChain Operation

DNSonChain acts as a decentralized third-party to manage the domain ownership and host DNS records for Internet users. It provides two functions: ownership claim and ownership cleanup. They synchronize the domain ownership between DNS and DNSonChain. We define the states of ownership in Table II and illustrate the ownership lifecycle in Figure 3.

Ownership Claim. A domain owner claims the ownership by sending claim voting requests (i.e., transactions) to DNSonChain. A claim voting request initiates the claim voting process which validates the ownership of the requester through the majority vote. However, adversaries may manipulate votes to compromise a voting process, e.g., by poisoning a committee member's DNS response. To provide resistance against such attacks, DNSonChain requires two rounds of claim voting processes with a required interval.

The ownership claim works as follows. According to our definition in Table II, a domain's ownership is implicitly in the `Wild` state at the initial of DNSonChain. A participant launches the first claim voting process on DNSonChain, and the success of such a claim voting process sets the participant as the candidate of the domain and the domain ownership transits to its `Claiming` state (❶). Then, it has to wait for the required interval. The interval raises the bar for adversaries as it is not feasible to compromise a nameserver for a relatively long time (Section VI-B). After that, the candidate launches the second claim voting process, and its success finally grants the ownership to the candidate (i.e., the voting requester) (❷), and now the domain ownership enters the `Established` state. Once domain ownership enters the `Established` state, the ownership claim completes and the claim voting process associated with the domain cannot be launched.

It is possible that an adversary accidentally succeeds in its first claim voting process, but does not make the second one after the required interval. If this situation happens, the domain ownership remains its `Claiming` state, and the adversary does not control the domain since it is still the candidate, not the owner. Then, when the real owner joins, it would directly launch the claim voting process that overrides the candidate (❸). Still, after a required interval the real owner launches the

TABLE II
OWNERSHIP STATES

States	Explanation
`Wild`	A domain does not have an owner or a candidate.
`Claiming`	A domain does not have an owner, but has a candidate.
`Established`	A domain has an owner, but does not have a cleanup flag.
`Cleaning`	A domain has an owner, and has a cleanup flag.

second claim voting process, and its success would make the transition ❷ happen since it is now the candidate.

Ownership Cleanup. There are two situations that domain ownership could have an inconsistency between DNS and DNSonChain. First, the current owner of a domain on DNSonChain used to be the actual owner of the domain on DNS and claimed the ownership on DNSonChain, but then the domain is expired on DNS and the domain owner does not actively give up the domain ownership on DNSonChain. Second, an adversary might accidentally and successfully pass two claim voting processes to obtain domain ownership. If those situations happen, DNSonChain needs to clean the ownership, which will reset the domain configurations.

DNSonChain allows cleaners to clean the inconsistent ownership via cleanup voting processes that validate if the current domain owner on DNSonChain is the rightful domain owner on DNS through the majority vote. Recall that in order to hold the ownership of a domain, the domain owner needs to have its blockchain ID stored in the TXT record. Cleaners regularly check if the TXT record is correctly configured. If the domain owner fails to present its blockchain ID, it is considered that the domain owner on DNSonChain has lost the control of the domain on DNS. Then, its ownership on DNSonChain should be cleaned. For the same security consideration of the ownership claim, the ownership cleanup also requires two cleanup voting processes with the same required interval.

In particular, the first successful voting process sets a cleanup flag on the domain, and the domain ownership enters the `Cleaning` state (❹). Then, the cleaner waits for a required interval. During the interval, the domain owner can defend itself by launching a claim voting process, and its success will clear the cleanup flag, and the domain ownership returns to its `Established` state (❻). Otherwise, after the required interval, the cleaner can launch the second cleanup voting process to clean the ownership of the domain so that the domain ownership becomes `Wild` again (❺).

DNSonChain Usage. Once domain ownership is established, a domain owner publishes its DNS records to DNSonChain by sending one blockchain transaction. The new records will be permanently recorded on the blockchain once the transaction is mined. Although the blockchain is a decentralized system, the blockchain database is a central place that hosts all DNS records. Therefore, DNSonChain users can synchronize DNS records from the blockchain to its local storage to build their local DNS databases in advance. Note that DNSonChain users do not require encrypted traffic to build DNS databases. This is because eavesdroppers cannot figure out meaningful information (i.e., which website a client visits) from the synchronization except knowing that DNSonChain is used. Once the local DNS database is built, the domain resolution would not generate any network traffic.

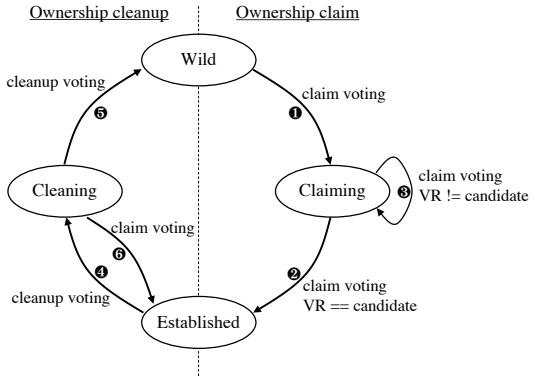

Fig. 3. Ownership Lifecycle. Claim/cleanup voting process must succeed to make the transition happen. VR stands for voting requester.

Note that, to take advantage of the privacy benefits provided by DNSonChain, a client could store a complete copy of the DNS records of all domains locally, but it is not a must. The implication is that a client may not visit all domains in DNSonChain. Therefore, the client could customize its local DNS database. To do so, the client still receives record updates of all domains from DNSonChain so that adversaries cannot reverse-engineer the client's customized DNS database, but only keeps the DNS records that the client is interested in.

VI. SECURITY ANALYSIS

The security of DNSonChain is the key to whether participants and users would like to get involved. Technically, adversaries may compromise the voting processes on DNSonChain by either attacking the committee selection or DNS infrastructure, resulting in the denial of service in voting processes or the fraud of ownership conferment.

To analyze the security of DNSonChain, we first describe our assumptions. First, we assume that miners/validators of the underlying blockchain have no bias on DNSonChain transactions. Also, both good and bad voters can access the blockchain and their transactions can be added to the blockchain within a reasonable time. In addition, we further make specific assumptions for two particular attacks. For the attacks on committee selection (Section VI-A), we assume that adversaries can only possess a small fraction (e.g., up to 25%) of stakes in the system. For the attacks on DNS (Section VI-B), we assume that the nameservers of a domain are well-maintained so that they can be recovered within a reasonable time (e.g., 6 hours) when under attack.

A. Attacks on Committee Selection

Threat Model. As the randomness nature of the committee selection procedures, the number of tickets (i.e., the voting

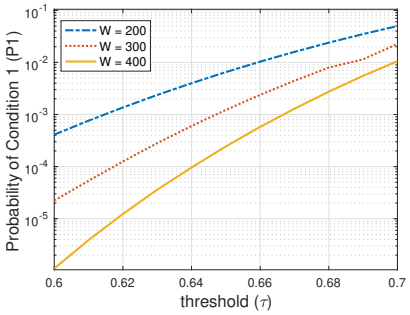

Fig. 4. Impact of the expected total voting weight W and threshold τ on the probability of condition 1. $G = 1000$ and $b_{total} = 0.2$.

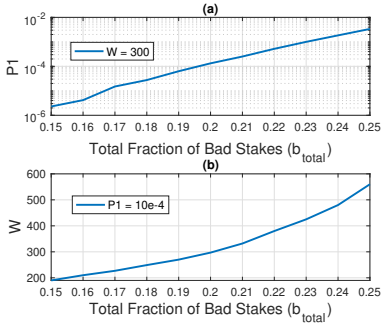

Fig. 5. Impact of the total fraction of bad stakes b_{total} on the probability of condition 1. $G = 1000$ and $\tau = 0.62$.

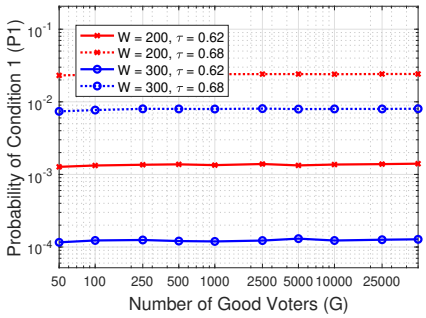

Fig. 6. Number of good voters does not impact the probability of condition 1. $b_{total} = 0.2$.

weight) held by good and bad voters varies. By exploiting such variance, adversaries may launch (i) DoS attacks and (ii) domain takeover attacks. In a DoS attack, adversaries choose not to commit if they are selected so that the voting process would not enter the `Fulfill` stage as the total voting weight of good voters may not be greater than or equal to τW. It nullifies the voting process. The domain takeover attack happens when adversaries dominate two consecutive claim voting processes. For each of them, adversaries must satisfy two conditions, (1) the total voting weight is greater than $\tau W/2$, and (2) votes with at least $\lfloor \tau W/2 + 1 \rfloor$ voting weight are committed before the end of the `Commit` stage. However, in our analysis, we consider the worst case where the second condition is always true. Here, we do not consider that adversaries could take down a domain through two cleanup voting processes because a victim domain owner can defend itself when the domain is in the `Cleaning` state.

Probability of Successful Attacks. Recall that in the committee selection, the voting weight of each voter follows a binomial distribution. Let G and B represent the total number of good voters and bad voters in the system, respectively. The stake percentage of each good voter and bad voter are denoted as $g_i, i = 0, 1, ..., G-1$ and $b_i, i = 0, 1, ..., B-1$, respectively. Therefore, the total fraction of bad stakes is $b_{total} = \sum_{i=0}^{B-1} b_i$. The probability mass function (PMF) of the sum of good voters' voting weight is $P(G_s) = \sum_{i=0}^{G-1} Binomial(W, g_i)$, and the PMF of the sum of bad voters' voting weight is $P(B_s) = \sum_{i=0}^{B-1} Binomial(W, b_i)$, where G_s and B_s denote the voting weight held by good and bad voters, respectively. Here we calculate the probability of two conditions for successful attacks as follows.

- *Condition 1*: $G_s < \tau W$
- *Condition 2*: $B_s > \tau W/2$

The first condition allows adversaries to launch the DoS attack and the second condition allows adversaries to compromise one voting process. The success of a domain takeover attack needs to satisfy the second condition twice. In the analysis, we use $P1$ and $P2$ to denote the probability of condition 1 and condition 2, respectively.

The distribution of G_s and B_s are the sum of independent binomial distributions. To the best of our knowledge, the sum of the independent and identical binomial distributions follows a binomial distribution, but the sum of the independent but non-identical binomial distributions does not follow a well-known distribution. In consequence, if the percentages of stakes among good or bad voters are equal, G_s or B_s follows a binomial distribution, thereby the probability of conditions can be directly calculated. In other cases, we run simulations to study the probability.

Condition 1. To emulate the real scenarios, the stakes of good voters are randomly distributed. In the simulation, we set $G = 1000$, and later we will show that the G does not affect the analysis results in real cases.

First, Figure 4 presents the relationship between $P1$ and the expected voting weight W and threshold τ. It shows that $P1$ decreases as W increases, and decreases as τ decreases. When we set $W = 300$ and $\tau = 0.62$,[2] $P1$ can be as low as 10^{-4}, which means that the adversaries have a negligible chance to succeed in a DoS attack against a voting process.

We then study the impact of the b_{total} on $P1$. Figure 5(a) shows that the $P1$ increases as the b_{total} increases. If we keep the setup of $W = 300$ and $\tau = 0.62$, the $P1$ increases to $3.4 * 10^{-3}$ when b_{total} reaches 0.25. However, we could increase the W to keep the $P1$ to 10^{-4} as shown in Figure 5(b).

Finally, we show that the number of G does not affect the analysis results. In Figure 6, the G spans from 50 to 50,000, but it does not result in non-trivial changes on $P1$. As we consider that the number of good voters in the system should always be greater than 50, we conclude that the G has no significant impact on the $P1$ in real cases.

Condition 2. Here, we envision that all adversaries collude together, trying to distribute the stakes to multiple pseudonyms. Note that such behavior does not change the mean value of B_s, but it affects $P2$ to some extent. So, we first study the impact of the number of pseudonyms on $P2$. Here, we consider two strategies to distribute the stakes, even distribution ($b_i = b_{total}/B, i = 0, 1, ..., B-1$) and random distribution. For even distribution, B_s follows

[2]We empirically set $\tau = 0.62$ for both condition 1 & 2 since it shows proper effectiveness of voting process.

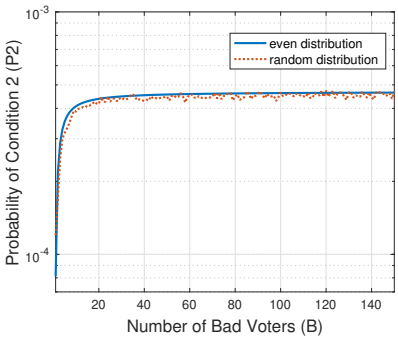

Fig. 7. Impact of number of bad voters on the probability of condition 2 under even and random distribution. $W = 200, b_{total} = 0.2$ and $\tau = 0.62$.

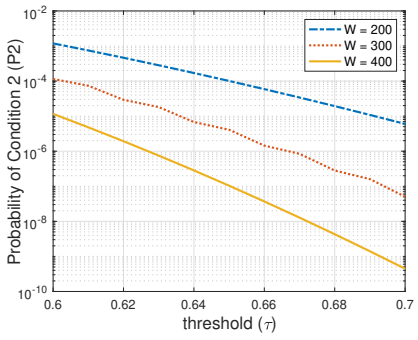

Fig. 8. Impact of the expected total voting weight W and threshold τ on the probability of condition 2. $B = 100$ and $b_{total} = 0.2$.

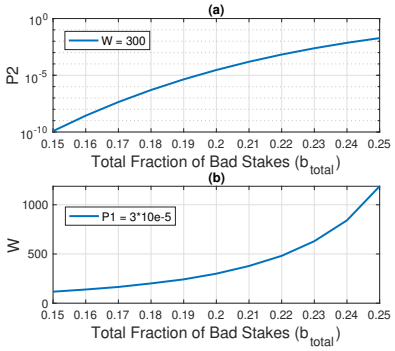

Fig. 9. Impact of the total fraction of bad stakes b_{total} on the probability of condition 2. $B = 100$ and $\tau = 0.62$.

$Binomial(BW, b_{total}/B)$, hence we calculate the $P2$ from its PMF. For random distribution, we run the simulations. In Figure 7, we can see that $P2$ increases quickly when B is very small. This means that when the system has a very limited number of pseudonyms controlled by adversaries, increasing the pseudonyms could increase the probability of a successful attack. However, $P2$ converges to its upper bound very quickly and would not have a non-trivial increase when $B > 20$. Also, the even distribution has a higher $P2$ but the difference is not significant. Therefore, in the following analysis, we set B to 100 and use even distribution as bad voters' stake distribution, which represents the worst case in our simulation.

Then, we study the impact of the expected voting weight W and threshold τ on $P2$. Figure 8 shows that $P2$ decreases as both W and τ increase. When $W = 300$ and $\tau = 0.62$, $P2$ reaches $3 * 10^{-5}$, which means that adversaries have a negligible probability of $(3 * 10^{-5})^2 = 9 * 10^{-10}$ to take over a domain.

Moreover, we analyze the relationship between b_{total} and $P2$. Figure 9(a) shows that $P2$ increases as b_{total} increases. However, even if the $P2$ increases to 0.25, the success of a domain takeover attack only has a probability of $3.5 * 10^{-4}$. Figure 9(b) presents the needed W when $P2 = 3 * 10^{-5}$ is guaranteed. As b_{total} increases, the needed W increases.

In summary, raising the expected voting weight W would reduce the probability of both the DoS and domain takeover attacks. On the other hand, a higher W generally requires more voters to be selected as committee members, hence requires more cost to complete the voting process (cost evaluation will be provided in Section VII-B). In addition, the selection of threshold shows a trade-off that a higher threshold leads to a higher probability of domain takeover attacks but a lower probability of DoS attacks. To achieve a safe low probability for both attacks, we will set $\tau = 0.62$ for evaluation.

B. Impact of DNS Attacks on DNSonChain

Threat Model. The voters retrieve TXT records by sending DNS queries. For domains without correct DNSSEC configuration, voters are not able to verify the integrity of the records. However, the adoption of DNSSEC is very low [10]. Thus, adversaries may compromise a voting process by poisoning the DNS responses. To takeover/takedown domain ownership, adversaries need to compromise two voting processes.

Analysis. We consider two ways of attacks that the adversaries could try to compromise a voting process, the off-path DNS poisoning and nameserver compromise.

Generally, for a successful off-path poisoning attack in DNSonChain, an adversary needs to correctly guess the transaction ID and the source port of DNS queries sent from voters who in total hold more than half of the voting weight. The transaction ID and the source port are both 16 bits random numbers, and it is not feasible to correctly guess them. Recent works [8], [24] present the defragmentation cache poisoning attack which does not need to guess the transaction ID and source port. However, it needs to guess the IP ID value in the IP fragmentation, which is still a very difficult challenge.

Then, we consider a scenario that an adversary compromises a nameserver of a domain, and it can craft fake DNS responses and send them back to the voters. However, DNS is designed to be a robust system in which each zone should maintain at least two nameservers with geographic/topological diversity [2], [14], [31]. Therefore, a strategy for voters to mitigate such an attack is to request TXT records from replica nameservers and submit the majority one. To study how feasible this method is, we conduct a measurement on the Alexa top 1M domains to analyze the deployment of their nameservers. Here, we make a rough but fair assumption that the nameservers located in different /24 subnets are logically isolated. In doing so, we collect the nameservers of these domains and resolve their IP addresses. We first filter out roughly 5.7% of the domains which do not have a nameserver. Next, we find nameservers that are using anycast routing by matching their IP addresses with the anycast prefix list provided in [3], [6], [12]. We highlight anycast addresses because anycast-based nameservers are running on multiple locations, hence they are immune to such attacks. Finally, we group IP addresses into /24 subnets and present the results in Figure 10. In total, 27% of the domains are using anycast-based nameservers. For

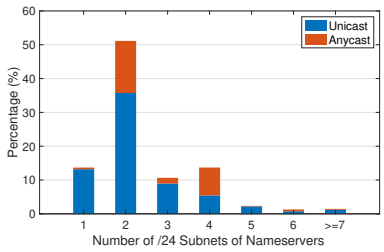

Fig. 10. Distribution of located /24 subnets of nameservers

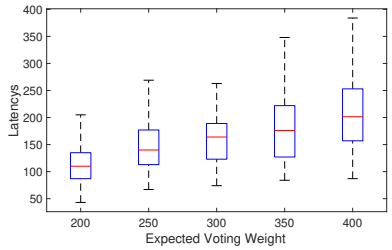

Fig. 11. Latency of voting processes with different expected voting weight. $\tau = 0.62$.

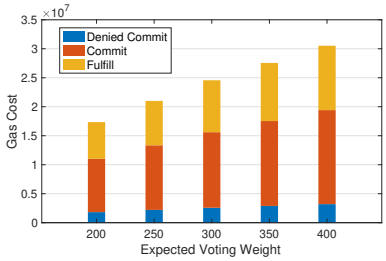

Fig. 12. Gas cost of voting processes with different expected voting weight. $\tau = 0.62$.

the rest of the domains, 13% of them have their nameservers located in one /24 subnets, which are vulnerable to such attacks. In addition, for the domains deploying their nameservers in two /24 subnets, the adversary may manipulate the DNS responses from one /24 subnet, resulting in a small chance to compromise over half of the weighted voters. However, for the domains that deploy their nameservers in more than two /24 subnets, we consider that such an attack can be naturally mitigated as voters can distinguish the authentic TXT record.

Note that compromising one voting process does not mean a successful domain takeover/takedown since the adversary still needs to wait for a required interval and then succeed in the second voting process. The interval and two rounds of voting process pose a challenge to adversaries since they have to compromise a nameserver twice or compromise a nameserver for a period that is longer than the required interval.

C. Recovery

Although with our design, adversaries are infeasible to launch successful attacks, there is always a possibility that the domain takeover/takedown can be successful. DNSonChain allows the damage of such attacks to be recovered once the attack is mitigated as the domain owners can take their ownership back through claim voting processes and the cleaners can remove the stale ownership with cleanup voting processes. The time to recover the damage highly depends on the required interval. The longer the required interval is, the harder the adversary can succeed. However, it also requires more time for recovery. We consider six hours as a safe period for the voting interval while also acceptable for recovery.

VII. Implementation and Evaluation

A. Implementation

We implement DNSonChain on the Ethereum blockchain. Ethereum is the most popular blockchain that supports smart contracts. `Ether` is the cryptocurrency on Ethereum. Ethereum introduces a concept of `gas` to quantify the cost of transactions, and the gas cost will be eventually converted to `Ether`. The gas cost comes from sending a transaction to the blockchain and executing smart contracts, and it is paid by the transaction sender.

DNSonChain consists of two smart contracts, the `DomainToken` contract and the `DNSonChain` contract. The contracts are written in Solidity language. The `DomainToken` contract manages the tokens of participants in DNSonChain. It complies with the ERC20 standard [35] which defines the interfaces that operate the token contract. The `DNSonChain` contract enforces the majority vote mechanism, manages domain ownership, and hosts DNS records. We use the `keccak256` for all hash operations in the `DNSonChain` contract since it is natively supported by Ethereum and has the cheapest gas cost among other supported hash functions.

Ethereum provides APIs that facilitate interactions between smart contracts and users. All DNSonChain participants run applications written in JavaScript to interact with the `DomainToken` and `DNSonChain` contract, and we use the `web3.js` library to handle these API calls.

Ethereum has one Mainnet and multiple Testnets [5]. The Mainnet is where decentralized applications will finally be deployed, and the transactions over the Mainnet cost real money. Conversely, Testnets are used for testing decentralized applications, and `Ether` on the Testnets can be obtained from Testnet faucets for free. Among all Testnets, the Ropsten Testnet [19] is the best emulation for the Mainnet since they are running the same consensus protocol (i.e., proof-of-work), and their average block time is close. Therefore, we deploy our system on the Ropsten Testnet for evaluation, and the implementation can be found at [28].

B. Evaluation

We evaluate the performance of DNSonChain by answering three questions: (1) what is the voting latency of DNSonChain? (2) what are the costs of the participants? and (3) comparing to DNS, what is the performance trade-off that DNSonChain can balance?

Experiment Setup. To evaluate the performance of DNSonChain, we set 1000 voters and consider all of them are good voters. Their stakes are pre-funded and randomly chosen from the range of [1, 100,000]. To simulate the geolocation diversity of the voters, we employ 1000 globally distributed and long-persistence open resolvers extracted from Censys [13], and assign each voter an open resolver for its DNS queries.

Note that the ownership voting process and cleanup voting process have almost the same operations except how voters determine the approval vote, which does not affect the evaluation result. Therefore, we only launch ownership voting processes

in the experiments for evaluation and the results also represent the evaluation for cleanup voting processes.

Voting Latency. Note that Ethereum's throughput is theoretically bounded by its block time and global gas limit. However, the throughput is shared by all Ethereum users, and the Ethereum miners may not form a block that achieves the gas limit for each block. Therefore, the actual throughput available to DNSonChain will be affected by the congestion level of the Ethereum network and miners' strategy, which is out of our control. Therefore, the latency varies at different times. Hence, for each test, we run one voting process every five minutes with a total of 100 votings processes to examine the distribution of the voting latency in real scenarios.

Figure 11 shows the relationship between the latency of voting processes and the expected voting weight (W). The latency increases as W grows. This is because a voting process with a higher W selects more voters, and hence more transactions are needed to finish the voting process. As a result, more time is needed to mine those transactions. Overall, the latencies remain at a low level (i.e., several minutes).

In addition to the number of transactions, the gas price also affects latency. Gas price is the amount of `Ether` that will be paid for a gas unit. For every transaction sent in our experiment, we set the gas price to 1 `Gwei` (1 `Gwei` = 10^9 `Wei`, and 1 `Wei` = 10^{-18} `Ether`.), which is the minimum recommended gas price. Transactions with a higher gas price are likely to be mined faster as miners may prioritize transactions with a higher gas price. Therefore, the results in Figure 11 represent an upper bound on the voting latency.

Gas Cost. Gas cost is what participants would pay for interacting with DNSonChain. We extract the gas cost of all transactions we sent in the latency experiments and find that the gas cost of a successful `Commit` transaction and `Fulfill` transaction is almost fixed, at 83K gas and 57K gas, respectively, making a total of 140K gas to successfully cast a vote. Note that a voter's `Commit` transaction may get denied if enough voting weight is received, but it still costs 27K gas for the transactions.

At the time of our experiment, the standard gas price is 3 `Gwei` [17]. Using this value, the the monetary cost for a voter to complete both `Commit` and `Fulfill` stage would be $4.2{*}10^{-4}$ `Ether`, and monetary cost for the voter that have a denied commitment is $8.1{*}10^{-5}$ `Ether`. In addition, the gas cost used by domain owners to launch a voting process is about 157K gas ($4.7 * 10^{-4}$ `Ether`) per request. Note that the gas cost is also significantly affected by the CPU and storage usage when executing the smart contracts, optimized implementation may further lower the cost. Therefore, our prototype presents an upper bound of the gas cost of DNSonChain operations.

Recall that initiating a voting process requires the requester to present a reward offer to incentivize voters. To figure out how much the offer should be, we examine the gas cost of all voters for each voting process and show the results in Figure 12. Since the voting processes with different W's result in different sizes of the committee, it would affect the cost of a voting process. As a result, the gas cost of a voting process increases linearly with the increase of W. Overall, the gas cost is at a low level. For example, when $W = 300$, the gas cost of a voting process is $2.5 * 10^7$, which equals 0.075 `Ether` (given the gas price of 3 `Gwei`). Consequently, a domain owner can make a reward offer greater than such a small cost to incentivize voters.

Storage Cost. To use DNSonChain, a user needs to synchronize DNS records to its local disk. As the local DNS database can be customized, different users would have different storage costs. To provide a baseline of the storage cost, we resolve Alexa top 1M domains to obtain their IP addresses and store them in a JSON file. The size of such a file is only 43MB, and it increases linearly with the number of domains. Therefore, the storage cost is negligible for a modern device.

Lookup and Record Propagation Delay. Users of DNSonChain perform domain lookup by querying their local databases built in advance. Compared to the lookup latency of 10 to 300 ms in DNS, the latency of local lookup in DNSonChain is essentially negligible.

Record propagation delay is the period from the time when a domain owner updates a record to the time when the updated record is available to users. In DNSonChain, a domain owner updates DNS records by sending a single blockchain transaction, and the records will be available to all users once the transaction is being mined. The median latency for a transaction to be mined on Ethereum is 29 seconds [17]. However, in DNS, users query DNS records from their recursive resolvers, and the resolvers would cache the stale DNS records until they expire. Therefore, the record propagation delay is significantly impacted by the TTL of DNS records. According to [20], [22], the TTLs of A records are usually in the range of 5 to 60 minutes. Thus, DNSonChain can provide a much shorter latency for propagating the record updates to its users.

VIII. Conclusion

This paper presents a new blockchain-based naming service called DNSonChain, which is compatible with the existing DNS namespace, to fully address DNS privacy issues. DNSonChain allows domain owners to hold their same domain names on a blockchain, and thus domain owners can publish DNS records on the blockchain. As a result, DNSonChain users can synchronize DNS records from the blockchain to build their local DNS databases. In DNSonChain, the name resolutions are no longer at any privacy risks. DNSonChain introduces the majority vote mechanism to validate the domain ownership in a decentralized fashion. We conducted a security analysis of DNSonChain and demonstrated that DNSonChain has a negligible security failure probability and can recover from various malicious attacks. Finally, we implemented a prototype of DNSonChain and deployed it on the Ropsten network. Our evaluation results show that the voting process can be done within several minutes in real scenarios and the cost of a domain owner for incentivizing the voters is small.

ACKNOWLEDGMENT

We would like to thank our shepherd Toru Hasegawa and the anonymous reviewers for their detailed and insightful comments. This work was supported in part by NSF grant DGE-1821744.

REFERENCES

[1] Muneeb Ali, Jude Nelson, Ryan Shea, and Michael J. Freedman. Blockstack: A Global Naming and Storage System Secured by Blockchains. In *USENIX Annual Technical Conference (ATC'16)*, 2016.

[2] Mark Allman. Comments on DNS Robustness. In *ACM Conference on Internet Measurement Conference (IMC'18)*, 2018.

[3] Anycast Datasets. https://anycast.telecom-paristech.fr/dataset/, 2017.

[4] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. DNS Security Introduction and Requirements. *IETF RFC 4033*, 2005.

[5] Wil Barnes. Ethereum 101 - Part 6 - Mainnet & Testnets. https://kauri.io/article/3eba08b801a44776a07607b9e046dd08/ethereum-101-part-6-mainnet-and-testnets, 2019.

[6] Rui Bian, Shuai Hao, Haining Wang, Amogh Dhamdere, Alberto Dainotti, and Chase Cotton. Towards Passive Analysis of Anycast in Global Routing: Unintended Impact of Remote Peering. In *ACM SIGCOMM Computer Communication Review (CCR)*, 2019.

[7] Bitcoin. https://bitcoin.org.

[8] Markus Brandt, Tianxiang Dai, Amit Klein, Haya Shulman, and Michael Waidner. Domain Validation++ For MitM-Resilient PKI. In *ACM Conference on Computer Communication Security (CCS'18)*, 2018.

[9] Brantly Millegan. Step-by-Step Guide to Importing a DNS Domain Name to ENS. https://medium.com/the-ethereum-name-service/step-by-step-guide-to-importing-a-dns-domain-name-to-ens-d2d15feb03e8, 2021.

[10] Taejoong Chung, Roland van Rijswijk-Deij, Balakrishnan Chandrasekaran, David Choffnes, Dave Levin, Bruce M. Maggs, Alan Mislove, and Christo Wilson. A Longitudinal, End-to-End View of the DNSSEC Ecosystem. In *USENIX Security Symposium (USENIX Security'17)*, 2017.

[11] Taejoong Chung, Roland van Rijswijk-Deij, David Choffnes, Dave Levin, Bruce M. Maggs, Alan Mislove, and Christo Wilson. Understanding the Role of Registrars in DNSSEC Deployment. In *ACM Conference on Internet Measurement Conference (IMC'17)*, 2017.

[12] Danilo Cicalese, Jordan Augé, Diana Joumblatt, Timur Friedman, and Dario Rossi. Characterizing IPv4 Anycast Adoption and Deployment. In *ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT'15)*, 2015.

[13] Zakir Durumeric, David Adrian, Ariana Mirian, Michael Bailey, and J. Alex Halderman. A Search Engine Backed by Internet-Wide Scanning. In *ACM Conference on Computer and Communications Security (CCS'15)*, 2015.

[14] R. Elz, R. Bush, S. Bradner, and M. Patton. Selection and Operation of Secondary DNS Servers. *IETF RFC 2182*, 1997.

[15] ENS. https://ens.domains/.

[16] ENS. https://docs.ens.domains/dns-registrar-guide.

[17] ETH Gas Station. https://ethgasstation.info/.

[18] Ethereum. https://www.ethereum.org/.

[19] Ethereum. Ropsten testnet PoW chain. https://github.com/ethereum/ropsten, 2018.

[20] Hongyu Gao, Vinod Yegneswaran, Yan Chen, Phillip Porras, Shalini Ghosh, Jian Jiang, and Haixin Duan. An Empirical Reexamination of Global DNS Behavior. In *ACM SIGCOMM Conference (SIGCOMM'13)*, 2013.

[21] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling Byzantine Agreements for Cryptocurrencies. In *ACM Symposium on Operating Systems Principles (SOSP '17)*, 2017.

[22] Shuai Hao, Haining Wang, Angelos Stavrou, and Evgenia Smirni. On the DNS Deployment of Modern Web Services. In *IEEE International Conference on Network Protocols (ICNP'15)*, 2015.

[23] Shuai Hao, Yubao Zhang, Haining Wang, and Angelos Stavrou. End-Users Get Maneuvered: Empirical Analysis of Redirection Hijacking in Content Delivery Networks. In *the 27th USENIX Security Symposium (USENIX Security'18)*, 2018.

[24] Amir Herzberg and Haya Shulman. Fragmentation Considered Poisonous, or: One-domain-to-rule-them-all.org. In *IEEE Conference on Communications and Network Security (CNS'13)*, 2013.

[25] P. Hoffman and P. McManus. DNS Queries over HTTPS (DoH). *IETF RFC 8484*, 2018.

[26] Rebekah Houser, Zhou Li, Chase Cotton, and Haining Wang. An Investigation on Information Leakage of DNS over TLS. In *ACM International Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, 2019.

[27] Z. Hu, L. Zhu, J. Heidemann, A. Mankin, D. Wessels, and P. Hoffman. Specification for DNS over Transport Layer Security (TLS). *IETF RFC 7858*, 2016.

[28] Lin Jin. https://github.com/lin-jin/decentralize-dns.

[29] Lin Jin, Shuai Hao, Haining Wang, and Chase Cotton. Understanding the Impact of Encrypted DNS on Internet Censorship. In *The Web Conference (WWW)*, 2021.

[30] Harry Kalodner, Miles Carlsten, Paul Ellenbogen, Joseph Bonneau, and Arvind Narayanan. An Empirical Study of Namecoin and Lessons for Decentralized Namespace Design. In *Workshop on the Economics of Information Security (WEIS'15)*, 2015.

[31] P. Mockapetris. Domain Names - Concepts and Facilities. *IETF RFC 1034*, 1987.

[32] Namecoin. https://namecoin.org/.

[33] Haya Shulman. Pretty Bad Privacy: Pitfalls of DNS Encryption. In *Workshop on Privacy in the Electronic Society (WPES'14)*, 2014.

[34] Sandra Siby, Marc Juarez, Claudia Diaz, Narseo Vallina-Rodriguez, and Carmela Troncoso. Encrypted DNS –> Privacy? A Traffic Analysis Perspective. In *Network and Distributed System Security Symposium (NDSS)*, 2020.

[35] Fabian Vogelsteller and Vitalik Buterin. EIP 20: ERC-20 Token Standard. https://eips.ethereum.org/EIPS/eip-20, 2015.

[36] Liang Zhu, Zi Hu, John Heidemann, Duane Wessels, Allison Mankin, and Nikita Somaiya. Connection-Oriented DNS to Improve Privacy and Security. In *IEEE Symposium on Security and Privacy (S&P'15)*, 2015.