The Onion Name System:
Tor-powered Distributed DNS for Tor Hidden Services

[Anonymous submission]

***Abstract*—Tor hidden services are anonymous servers of unknown location and ownership who can be accessed through any Tor-enabled web browser. They have gained popularity over the years, but still suffer from major usability challenges due to their cryptographically-generated non-memorable addresses.**

In response to this difficulty, in this work we introduce the Onion Name System (OnioNS), a privacy-enhanced distributed DNS that allows users to reference a hidden service by a meaningful globally-unique verifiable domain name chosen by the hidden service operator. We introduce a new distributed self-healing public ledger and construct OnioNS as an optional backwards-compatible plugin for Tor on top of existing hidden service infrastructure. We simplify our design and threat model by embedding OnioNS within the Tor network and provide mechanisms for authenticated denial-of-existence with minimal networking costs. Our reference implementation demonstrates that OnioNS successfully addresses the major usability issue that has been with Tor hidden services since their introduction in 2002.

I. Introduction

As the prevalence of the Internet and other communication has grown, so too has the development and usage of privacy-enhancing systems. These are protocols that provide privacy by obfuscating the link between a user's identity or location and their communications. Following a general distrust of unsecured Internet communications and in light of the 2013-current revelations by Edward Snowden of international Internet mass-surveillance, users have increasingly turned to these tools for their own protection.

Tor[8] is a third-generation onion routing system and is the most popular low-latency anonymous communication network in use today. In Tor, users construct a layered encrypted communications circuit over three onion routers in order to mask their identity and location. As messages travel through the circuit, each onion router in turn decrypts their encryption layer, exposing their respective routing information. This provides end-to-end communication confidentiality of the sender.

Tor users interact with the Internet and other systems over Tor via the Tor Browser, a security-enhanced fork of Firefox ESR. This achieves a level of usability but also security: Tor achieves most of its application-level sanitization via privacy filters in the Tor Browser; unlike its predecessors, Tor performs little sanitization itself. Tor's threat model assumes that the capabilities of adversaries are limited to traffic analysis attacks on a restricted scale; they may observe or manipulate portions of Tor traffic, that they may run onion routers themselves, and that they may compromise a fraction of other existing routers. Tor's design centers around usability and defense against these types of attacks.

A. Motivation

Tor also supports *hidden services* – anonymous servers that intentionally mask their IP address through Tor circuits. They utilize the .onion pseudo-TLD, preventing hidden services from being accessed outside the context of Tor. Hidden services are only known by their public RSA key and typically referenced by their address, 16 base32-encoded characters derived from the SHA-1 hash of the server's key. This builds a publicly-confirmable one-to-one relationship between the public key and its address and allows hidden services to be accessed via the Tor Browser by their address within a distributed environment.

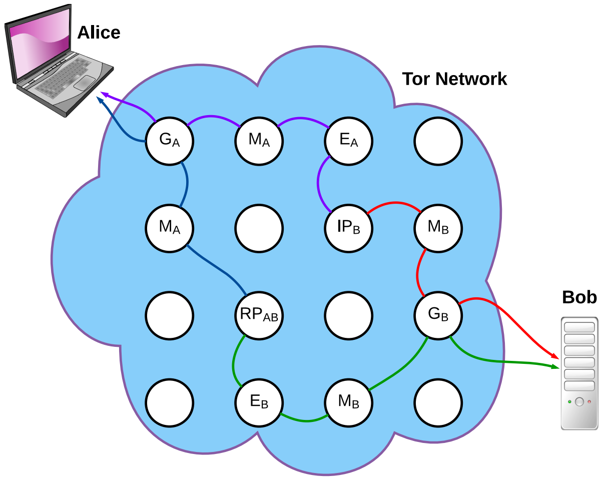

Fig. 1. A Tor client, Alice, and a hidden service, Bob, first mate two Tor circuits (purple and red) at one of Bob's long-term *introduction points* (IP). They then renegotiate and communicate over another pair of Tor circuits (blue and green) at an ephemeral *rendezvous point* (RP). This achieves communication with bi-directional anonymity[22].

Tor hidden service addresses are distributed and globally collision-free, but there is a strong discontinuity between the address and the service's purpose. For example, a visitor cannot determine that 3g2upl4pq6kufc4m.onion is the DuckDuckGo search engine without visiting the hidden service.

NDSS '15, 8-11 February 2015, San Diego, CA, USA
Copyright 2015 Internet Society, ISBN ¡TBD¿
http://dx.doi.org/10.14722/ndss.2015.23xxx

Generally speaking, it is currently impossible to categorize or fully label hidden services in advance. Over time, third-party directories – both on the Clearnet and Darknet – have appeared in attempt to counteract this issue, but these directories must be constantly maintained and the approach is neither convenient nor does it scale well. Given the approximately 27,000 hidden services on the Tor network, (Figure 2) this suggests the strong need for a more complete solution to solve the usability issue.

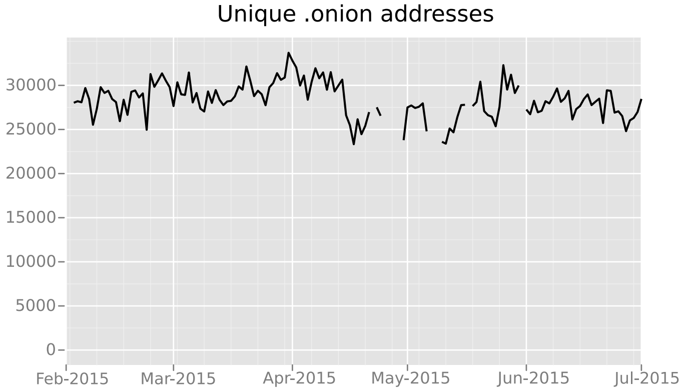

Fig. 2. The number of unique .onion addresses seen in the Tor network between February 2015 and July 2015[14][24].

B. Contributions

In this paper, we present the design, analysis and implementation of the Onion Name System, (OnioNS) a distributed, secure, and usable domain name system for Tor hidden services. Any hidden service can anonymously register an association between a meaningful human-readable domain name and a target hidden service and clients can query against OnioNS in a privacy-preserving and verifiable manner. OnioNS is powered by a random subset of nodes within the existing infrastructure of Tor, significantly limiting the additional attack surface. We devise a distributed DNS database using a blockchain-based data structure that is tamper-proof, self-healing, resistant to node compromise, and provides authenticated denial-of-existence. We design OnioNS as a backwards-compatible plugin to the Tor software. Our prototype implementation demonstrates the high usability and performance of OnioNS. To the best of our knowledge, this is the first alternative DNS for Tor hidden services which is distributed, secure, and usable at the same time.

Paper Organization: This paper is divided into four main sections. In section II we define our design objectives and explain why existing works do not meet our goals. We also define our threat model, which closely matches Tor's model with one additional assumption. In section V, we describe the system overview and define several key protocols. In section VI we analyse the security of our assumptions and examine other attack vectors. Last, in section VI we evaluate our implementation prototype and show that it can access a hidden service under a meaningful domain name.

II. Problem Statement

To integrate with Tor, we must provide a secure system, preserve user privacy, and avoid compromising other areas of the Tor network. Additionally, we seek to achieve all three properties of Zooko's Triangle (section II-B1) and to providing a mechanism for authenticated denial-of-existence (section II-B2).

A. Design Objectives

Here we enumerate a list of requirements that must be met by any DNS applicable to Tor hidden services. In Section III we analyse existing works and show how these systems do not meet these requirements and in Section V we demonstrate how we overcome them with OnioNS.

1. **The system must support anonymous registrations.** The system should not require any personally-identifiable or location information from the registrant. Tor hidden services publicize no more information than a public key and Introduction Points.

2. **The system must support privacy-enhanced queries.** Clients should be anonymous, indistinguishable, and unable to be tracked by name servers.

3. **Registrations must be authenticable.** Clients must be able to verify that the domain-address pairing that they receive from name servers is authentic relative to the authenticity of the hidden service.

4. **Domain names must be globally unique.** Any domain name of global scope must point to at most one server. For naming systems that generate names via cryptographic hashes, the key-space must be of sufficient length to resist cryptanalytic attack.

5. **The system must be distributed.** Systems with root authorities have distinct disadvantages compared to distributed networks: specifically, central authorities have absolute control over the system and root security breaches could easily compromise the integrity of the entire system. Root authorities may also be able to compromise the privacy of both users and hidden services or may not allow anonymous registrations.

6. **The system must be relatively easy to use.** It should be assumed that users are not security experts or have technical backgrounds. The system must resolve protocols with minimal input from the user and hide non-essential details.

7. **The system must be backwards compatible.** Naming systems for Tor must preserve the original Tor hidden service protocol, making the DNS optional but not required.

8. **The system should be lightweight.** In most realistic environments clients have neither the bandwidth nor storage capacity to hold the system's entire database, nor the capability of meeting significant computation burdens.

B. Challenges

1) Zooko's Triangle: In 2001, Zooko Wilcox-O'Hearn described three desirable properties for any persistent naming system: distributed design, assignment of human-meaningful names, and globally unique names. In a statement now known as Zooko's Triangle,[10][26] he claimed any naming system could only achieve two of these properties. This is illustrated in Figure 3.

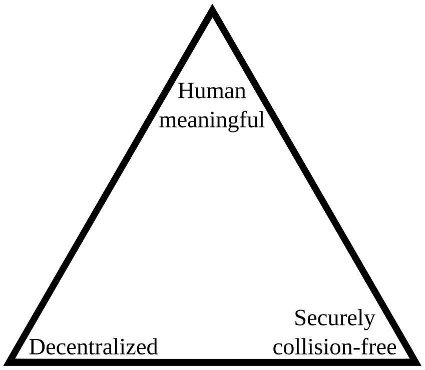

Fig. 3. Zooko's Triangle.

Some examples of naming systems that achieve only two of these properties include:

- **Securely unique and human-meaningful** — Internet domain names and GNS.
- **Decentralized and human-meaningful** — Human names and nicknames.
- **Securely unique and decentralized** — Tor hidden service .onion addresses.

2) Authenticated Denial-of-Existence: If a naming system provides authentication, clients should be able to verify the authenticity of existing domain names and authenticate a denial-of-existence claim by their name server. On the Internet, the former is addressed by SSL certificates and a chain of trust to root Certificate Authorities, while the latter remains a possible attack vector. DNSSEC includes an extension for Hashed Authenticated Denial of Existence (NSEC3) which provides signed non-existence claims on a per-domain basis. However, DNSSEC has not seen widespread use, storing per-domain denial-of-existence records introduces significant storage requirements, and to our knowledge no alternative DNS provides mechanisms for authenticated denial-of-existence. Closing this attack vector is not easy; the naïve solution of generating proof individually or en-masse for every non-existent domain is infeasible since the number of possible domain names is likely too large to practically enumerate.

III. Related Works

Vanity key generators (e.g. Shallot[15]) attempt to find by brute-force an RSA key that generates a partially-desirable hash. Vanity key generators are commonly used by hidden service operators to improve the recognition of their hidden service, particularly for higher-profile services[27]. For example, a hidden service operator may wish to start his service's address with a meaningful noun so that others may more easily recognize it. However, these generators are only partially successful at enhancing readability because the size of the domain key-space is too large to be fully brute-forced in any reasonable length of time. If the address key-space was reduced to allow a full brute-force, the system would fail to be guaranteed collision-free. Nicolussi suggested changing the address encoding to a delimited series of words, using a dictionary known in advance by all parties[21]. Like vanity key generators, Nicolussi's encoding partially improves the recognition and readability of an address but does nothing to counter the large key-space nor alleviate the logistic problems of manually entering in the address into the Tor Browser. These attempts are purely cosmetic and do not qualify as a full solution.

The Internet DNS is another one candidate and is already well established as a fundamental abstraction layer for Internet routing. However, despite its widespread use and extreme popularity, the Internet DNS suffers from several significant shortcomings and fundamental security issues that make it inappropriate for use by Tor hidden services. Generally speaking, the Internet DNS by default does not use any cryptographic primitives. DNSSEC is primarily designed to prevent forgeries and DNS cache poisoning from intermediary name servers and it does not provide any degree of query privacy[29]. Additional extensions and protocols such as DNSCurve[3] have been proposed, but DNSSEC and DNSCurve are optional and have not yet seen widespread full deployment across the Internet. The lack of default security in Internet DNS and the financial expenses involved with registering a new TLD casts significant doubt on the feasibility of using it for Tor hidden services. Cachin and Samar[5] extended the Internet DNS and decreased the attack potential for authoritative name servers via threshold cryptography, but the lack of privacy in the Internet DNS and the logistical difficulty in globally implementing their work prevents us from using their system for hidden services.

The GNU Name System[29] (GNS) is another zone-based alternative DNS. GNS describes a hierarchical zones of names with each user managing their own zone and distributing zone access peer-to-peer within social circles. While GNS' design guarantees the uniqueness of names within each zone and users are capable of selecting meaningful nicknames for themselves, GNU does not guarantee that names are *globally* unique. Furthermore, the selection of a trustworthy zone to use would be a significant challenge for using GNS for Tor hidden services and such a selection no longer makes the system distributed. Awerbuch and Scheideler,[1] constructed a distributed peer-to-peer naming system, but like GNS, made no guarantee that domain names would be globally unique.

Namecoin[6] is an early fork of Bitcoin[19] and is noteworthy for achieving all three properties of Zooko's Triangle. Namecoin holds information transactions in a distributed ledger known as a blockchain. Storing textual information such as a domain registration consumes some Namecoins, a unit of currency. While Namecoin is often advertised as capable of assigning names to Tor hidden services, it has several practical issues that make it generally infeasible to be used for that purpose. First, to authenticate registrations, clients must be able to prove the relationship between a Namecoin owner's secp256k1 ECDSA key and the target hidden service's RSA key: constructing this relationship is non-trivial. Second, Namecoin generally requires users to pre-fetch the blockchain which introduces significant logistical issues due to high bandwidth, storage, and CPU load. Third, although Namecoin supports anonymous ownership of information, it is non-trivial to anonymously purchase Namecoins, thus preventing domain registration from being truly anonymous. These issues prevent Namecoin from being a practical alternative DNS for Tor hidden service. However, our work shares some design principles with Namecoin.

IV. Assumptions and Threat Model

We assume that Tor provides privacy and anonymity; if Alice constructs a three-hop Tor circuit to Bob with modern Tor cryptographic protocols and sends a message m to Bob, we assume that Bob can learn no more about Alice than the contents of m. This implies that if m does not contain identifiable information, Alice is anonymous from Bob's perspective, regardless of if m is exposed to an attacker, Eve. Identifiable information in m is outside of Tor's scope, but we do not introduce any protocols that cause this scenario. The security of Tor circuits is also dependent on the honesty of directory authorities: we also assume that more than fifty percent of Tor directory authorities are at least semi-honest; they may wiretap but are not capable of violating protocols.

Let R be the set of Tor routers with the Fast, Stable, Running, and Valid flag and let Q be an M-sized set randomly chosen from R with selection probability $P(Q_i) = \frac{Q_i(w)}{\sum_{j=0}^{\|R\|} R_j(w)}$ where $Q_i(w)$ is Q_i's consensus weight as determined by Tor directory authorities. We then assume that Q is under the influence of one or more adversaries and that largest subset of agreeing routers in Q are at least semi-honest.

We assume that Eve controls some percentage of dishonest colluding Tor routers as well as semi-honest routers, however this percentage is small enough to avoid violating our first assumption. We assume a fixed percentage of dishonest and semi-honest routers; namely that the percentage of routers under an Eve's control does not increase in response to the inclusion of OnioNS into Tor infrastructure. This assumption simplifies our threat model analysis but we consider it realistic because while Tor traffic is purposely secret as it travels through the network, we consider OnioNS information public so we don't consider the inclusion of OnioNS a motivating factor to Eve.

Finally, we assume secure cryptographic primitives; namely that Eve cannot break standard cryptographic primitives such as AES, SHA-2, RSA, Curve25519, Ed25519, and the scrypt key derivation function. We assume that Eve maintains no backdoors or knows secret software breaks in the Botan or the OpenSSL implementations of these primitives.

V. Solution

A. Overview

We propose the Onion Name System (OnioNS) as an abstraction layer to hidden service addresses and introduce ".tor" as a new pseudo-TLD for this purpose. First, Bob generates and self-signs a *Record*, containing a meaningful second-level domain name and his public key $\mathcal{P}_B$. Without loss of generality, let this be "example.tor → onions55e7yam27n.onion". We introduce a proof-of-work scheme that requires Bob to expend computational and memory resources to claim "example.tor", a more privacy-enhanced alternative to financial compensation to a central authority. Proof-of-work systems are noteworthy for their asymmetry: they require the issuer to spend effort to find an answer to a moderately hard computational problem, but once solved can be easily verified correct by any recipient. The requirement of proof-of-work fulfils three main purposes:

1) Significantly reduces the threat of DoS flood attack.
2) Introduces a barrier-of-entry that encourages the utilization of domain names and the availability of the underlying hidden services.
3) Increases the difficulty of domain squatting.

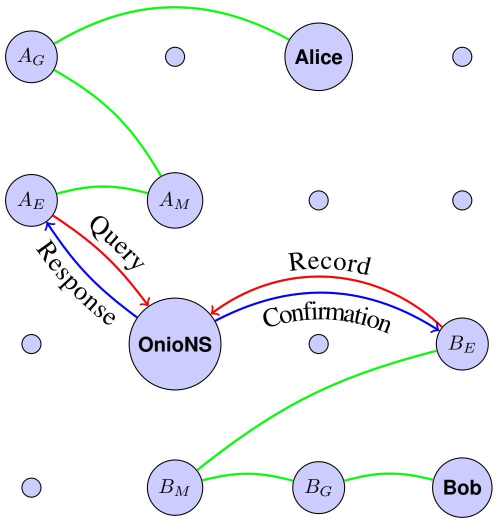

Fig. 4. Bob uses a Tor circuit (B_G, B_M, B_E) to anonymously broadcast a record to OnioNS. Alice uses her own Tor circuit (A_G, A_M, A_E) to query the system for a domain name, and she is given Bob's record in response. Then Alice connects to Bob by Tor's hidden service protocol.

Second, Bob uses a Tor circuit to anonymously transmit his Record to an authoritative short-lived random subset of OnioNS servers, known as the Quorum, inside the Tor network. The Quorum archive Bob's Record in a sequential public ledger known as a Pagechain, of which each OnioNS node holds their own local copy. Bob's Record is received by all Quorum nodes and share signatures of their knowledge with each other, so they maintain a common database. Quorum nodes are not name servers, so let Charlie be a name server outside the Quorum and assume that Charlie stays synchronized with the Quorum.

Third, Alice, uses a Tor client to anonymously connect to Charlie, then asks Charlie for "example.tor". Alice receives Bob's Record, verifies its signature and proof-of-work, and uses $\mathcal{P}_B$ to contact Bob via Tor's hidden service protocol. As Bob's Record is self-signed and contains $\mathcal{P}_B$, Alice can verify the Record's authenticity relative to $\mathcal{P}_B$. In this way, Alice does not have to resort to using "onions55e7yam27n.onion", rather that Bob can be successfully referenced by "example.tor". We illustrate the OnioNS overview in Figure 4.

B. Cryptographic Primitives

OnioNS makes use of a number of cryptographic primitives, notably a proof-of-work algorithm and a global source of randomness. Additionally, we require that Tor routers generate an Ed25519[4] keypair and distribute the public key via the consensus document. We note that because the Ed25519 elliptic curve is birationally equivalent to Curve25519[2] and because it is possible to convert Curve25519 to Ed25519 in constant time, we can theoretically use existing NTor[11] keys for digital signatures. However, we refrain from this because to our knowledge there is no formal analysis that demonstrates that this a cryptographically secure operation. Therefore we require Tor to introduce Ed25519 keys to all Tor routers. If this is infeasible, Ed25519 can be substituted with RSA in all instances.

- Let $\mathcal{H}(x)$ be a cryptographic hash function. In our reference implementation we define $\mathcal{H}(x)$ as SHA-384.
- Let $S_{RSA}(m, r)$ be a deterministic RSA digital signature function that accepts a message m and a private RSA key r and returns a digital signature. Let $S_{RSA}(m, r)$ use $\mathcal{H}(x)$ as a digest function on m in all use cases. In our reference implementation we define $S_{RSA}(m, r)$ as EMSA PKCS1 v1.5. (EMSA3)
- Let $V_{RSA}(m, R)$ validate an RSA digital signature by accepting a message m and a public key R, and return true if and only if the signature is valid.
- Let $S_{ed}(m, e)$ be an Ed25519 digital signature function that accepts a message m and a private key e and returns a 64-byte digital signature. Let $S_{ed}(m, e)$ use $\mathcal{H}(x)$ as a digest function on m in all use cases.
- Let $V_{ed}(m, E)$ validate an Ed25519 digital signature by accepting a message m and a public key E, and return true if and only if the signature is valid.
- Let $\mathrm{PoW}(k)$ be a one-way function that accepts an input key k and returns a deterministic output. Our reference implementation uses the scrypt[23] key derivation function with a fixed salt.
- Let $\mathcal{G}(t)$ be a cryptographically-secure generator of random or pseudorandom timestapped numbers. $\mathcal{G}(t)$ deterministically returns a value for time t in the present or past.
- Let $\mathcal{R}(s)$ be a pseudorandom number generator that accepts an initial seed s and returns a list of pseudorandom numbers. In our design, $\mathcal{R}(s)$ uses $\mathcal{G}(t)$, so $\mathcal{R}(s)$ does not need to be cryptographically secure. We suggest MT19937, commonly known as the Mersenne Twister. This generator is widely used throughout most programming languages and is well known for its speed, long period, and the high quality of its pseudorandom output[17].

C. Definitions

domain name The syntax of OnioNS domain names mirrors the Internet DNS; we use a sequence of name-delimiter pairs with a .tor pseudo-TLD. The Internet DNS defines a hierarchy of administrative realms that are closely tied to the depth of each name. By contrast, OnioNS makes no such distinction; we let hidden service operators claim second-level names and then control all names of greater depth under that second-level name.

A **Record** contains *type*, the purpose of this Record; *name*, a meaningful second-level domain name with the .tor pseudo-TLD; *subdomains*, a one-to-one map of .tor domains of level three or higher and .tor or .onion destinations; *contact*, Bob's PGP key fingerprint if he chooses to disclose it; *rand*, the output of $\mathcal{G}(t)$ at the current time; *nonce*, bytes used as a source of randomness for the proof-of-work; *pow*, the output of $\mathrm{PoW}(i)$; *signature*, the output of $S_{RSA}(m, r)$; and *pubHSKey*, Bob's public RSA key.

A **Page** contains *prevHash*, set to $\mathcal{H}(prevHash \parallel recordList \parallel rand)$ of some previous Page; *recordList*, a deterministically-sorted array list of Records; *rand*, the output of $\mathcal{G}(t)$ at the current time; *fingerprint*, the Tor fingerprint of the router maintaining this Page; and *pageSig*, the output of $S_{ed}(\mathcal{H}(prevHash \parallel recordList \parallel rand), e)$ where e is the router's private Ed25519 key.

prevHash links Pages over time, forming an append-only public ledger known as a Pagechain. In contrast to existing cryptocurrencies such as Namecoin, we bound the Pagechain to a finite length, forcing hidden service operators to renew their domain periodically to avoid it being dropped from the network. In correspondence with our security assumptions, *prevHash* must reference a Page that is both valid and maintained by the largest number of Quorum members whom we assume are at least semi-honest, as illustrated in Figure 5. As *prevHash* does not include the router-specific *fingerprint* and *pageSig* fields, *prevHash* is equal across all Quorum members maintaining that Page.

A **Mirror** is any name server that holds a complete copy of the Pagechain and maintains synchronization against the Quorum. Mirrors respond to queries but must provide signatures from Quorum nodes to prevent Mirrors from falsifying responses. We note that Mirrors may be outside the Tor network, but in this work we do not specify any protocols for this scenario.

Quorum Candidate are *Mirrors* that provide proof in the network status consensus that they are an up-to-date Mirror in the Tor network and that they have sufficient CPU and bandwidth capabilities to handle OnioNS communication in addition to their Tor duties.

A **Quorum** is a subset of Quorum Candidates who have active responsibility over maintaining the master Pagechain. Each Quorum node actively maintains its own Page, which has a lifetime of that Quorum. The Quorum is randomly chosen from Quorum Candidates as described in section V-D3.

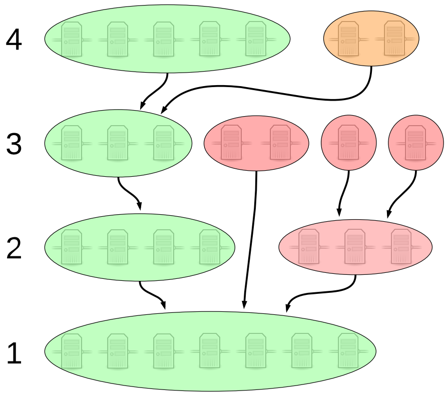

Fig. 5. An example Pagechain across four Quorums with three side-chains. The valid master Pagechain from honest/semi-honest Quorum nodes (green) resists corruption from dishonest and colluding nodes (red) and malfunctioning nodes (orange).

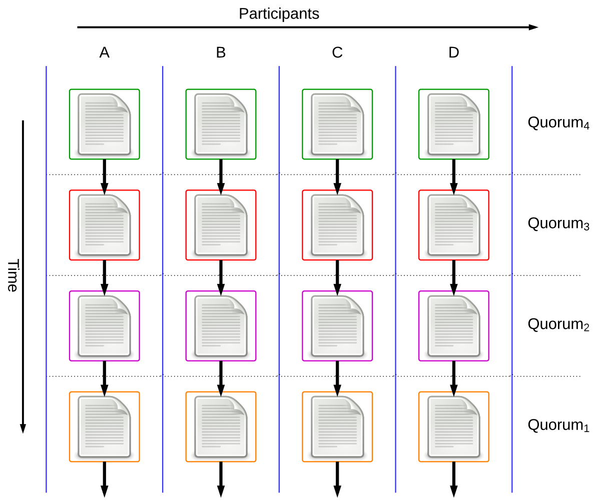

Fig. 6. The master Pagechain is one-dimensional but spans across the network as each Mirror holds a copy. Each Page is maintained by a respective Quorum.

L_Q	the size of the Quorum
L_T	the number of routers in the Tor network
L_P	the maximum number of Pages in the Pagechain
q	the Quorum iteration counter
Δq	the lifetime of a Quorum in days

TABLE I. FREQUENTLY USED NOTATIONS

D. Protocols

We now describe the protocols fundamental to OnioNS functionality.

1) Random Number Generation: We use $\mathcal{G}(t)$ as a basis for Quorum selection, although we note that $\mathcal{G}(t)$ has applications in Tor beyond OnioNS. One straightforward definition of $\mathcal{G}(t)$ is the SHA-384 hash of Tor's consensus documents. If the Tor network is dynamic enough to provide significant amounts of entropy into the consensus documents, then $\mathcal{G}(t)$ may be considered cryptographically secure. However, this assumption does not hold because current router descriptors are publicly available before the consensus documents are published, allowing $\mathcal{G}(t)$ under this approach to be easily manipulated by a few malicious Tor routers. The attack becomes significantly easier in the final moments before the directory authorities publish the consensus.

Instead, we suggest implementing $\mathcal{G}(t)$ as the commitment scheme proposed by Goulet and Kadianakis[13]. Their algorithm modifies the consensus voting protocol that is run once an hour by Tor directory authorities. In their scheme, each authority commits a SHA-256 hash of a secret key x into each consensus vote across a 12 hour period. Then each directory authority reveals x across the next set of 12 consensuses. Finally, the revealed values are hashed together to create a single random number, which is then embedded in the consensus documents so that it is efficiently distributed to both Tor router and end users. A different random number thus appears in the consensus every 24 hours. We reiterate their security analysis in section VI-A3.

2) Quorum Qualification: Quorum Candidates must prove that they are both up-to-date Mirrors and that they sufficient capabilities to handle the increase in communication and processing from OnioNS protocols.

The naïve solution to demonstrating the first requirement is to simply ask Mirrors for their Page, and then compare the recency of its latest Page against the Pages from the other Mirrors. However, this solution does not scale well; Tor has ≈ 2.1 million daily users[24]: it is infeasible for any single node to handle queries from all of them. Instead, let each Mirror first calculate $t = \mathcal{H}(pc \parallel \lfloor \frac{m-15}{30} \rfloor)$ where pc is the Mirror's Pagechain and m is the number of minutes elapsed in that day, then include t in the Operator Contact field in his relay descriptor. Tor's consensus documents are published at the top of each hour; we manipulate m such that t is consistent at the top of each hour even with at most a 15-minute clock-skew. We suggest placing t inside a new field within the router descriptor in future work, but our use of the Contact field eases integration with existing Tor infrastructure. OnionNS would not be the first system to embed special information in the Operator Contact field: PGP keys and BTC addresses commonly appear in the field, especially for high-performance routers.

Tor's infrastructure already provides a mechanism for demonstrating the latter requirement; Quorum Candidates must also have the Fast, Stable, Running, and Valid flags. As of July 2015, out of the ≈ 7,000 nodes participating in the Tor network, ≈ 5,400 of these node have these flags and meet the latter requirement[24]. Tor routers with larger CPU or bandwidth capabilities relative to their peers also receive a proportionally larger consensus weight from the directory authorities. Thus we can benefit from this infrastructure by deriving the selection probability from consensus weight.

3) Quorum Formation: Quorum Candidates and Tor clients can derive a Quorum in $\mathcal{O}(L_T)$ time. Without loss of generality, let Alice run this algorithm.

1) Alice obtains the consensus documents, cd, published on day $\lfloor \frac{q}{\Delta q} \rfloor$ at 00:00 UTC.
2) Alice extracts $g = \mathcal{G}(t)$ from cd.
3) Alice scans cd and constructs a list l of Quorum Candidates that have the Fast, Stable, Running, and Valid flags and that are in the largest set of Tor routers that publish an identical time-based hash.
4) Alice computes $s = \sum_{j=0}^{\|R\|} R_j(w)$ where $R_j(w)$ is R_j's consensus weight as determined by Tor directory authorities.
5) Alice uses $\mathcal{R}(g)$ to select $\min(\text{size}(qc), L_Q)$ Quorum nodes from l with selection probability $P(Q_i) = \frac{Q_j(w)}{s}$.

This procedure may be performed on any cd in the past because Tor consensus documents contain timestamps and digital signatures from Tor directory authorities and thus may be retroactively verified regardless of where they are archived.

4) Record Generation: Bob must first generate a valid Record to claim a second-level domain name for his hidden service. The validity of his Record is checked by Mirrors and clients, so Bob must follow this protocol to ensure that his Record is accepted.

1) Bob sets *type* to “Create”.
2) Bob provides the *name* and *subdomains* fields by specifying a second-level domain name and subdomain-destination pairs for his hidden service, respectively.
3) Bob optionally sets *contact* to his PGP key fingerprint.
4) Bob sets *rand* to the value from $\mathcal{G}(t)$ as published in the consensus documents on day $\lfloor \frac{q}{\Delta q} \rfloor$ at 00:00 UTC.
5) Bob fills *nonce* with zeros.
6) Let c be $type \,\|\, name \,\|\, subdomains \,\|\, contact \,\|\, rand \,\|\, nonce$.
7) Bob sets *pow* as $\mathrm{PoW}(c)$.
8) Bob sets *signature* as the output of $S_{RSA}(m, r)$ where $m = c \,\|\, pow$ and r is Bob's private RSA key.
9) Bob saves the PKCS.1 DER encoding of his RSA public key in *pubHSKey*.

The Record is valid when $\mathcal{H}(c \| pow \| signature) \leq d$ where d is a fixed constant that specifies the work difficulty. This also requires Bob to increment *nonce* and resign his Record at every iteration of $\mathrm{PoW}(c)$. Once it is valid, Bob can send his Record to all Quorum nodes.

5) Record Operations: The Onion Name System also supports common operations on Records in a manner similar to Namecoin. So long as Bob retains exclusive knowledge of his private key, he may manipulate his own Records at any later date.

Modification Although Bob cannot change the Record's *name* field, he can change the *subdomains* or *contact* fields by reissuing another Record with these manipulations in place. This modification requires a difficulty of $\frac{d}{4}$.

Deletion To relinquish his claim, he may issue a deletion record, releasing his domain and allowing it to be claimed by another. There is no difficulty in doing this, so his request can be issued immediately, which may be beneficial in some cases.

Renew Bob must also periodically renew his claim by reissuing his Record every $L_P * \Delta q$ days to ensure that the Page containing it does not drop off the Pagechain. This requires the same difficulty as generating a new Record.

Transfer Bob may also choose to change the ownership of his Record. He can do this by issuing a transfer Record, which includes a new field, *recipientKey*, the public RSA key of the recipient. However, unlike Namecoin, here Bob also retains the ability to issue a deletion Record. This property not only discourages the economic benefits of domain squatting, but also helps provides a defence against phishing attacks. If Eve gains access to Bob's private key and issues a transfer Record to her key before Bob notices the breach, Bob can still release the domain to minimize the impact.

6) Record Processing: A Quorum node Q_j listens for new Records from hidden service operators. Mirrors subscribe to Quorum nodes by maintaining a network session with them and sending a subscription request. This allows Records to traverse the network quickly in a peer-to-peer fashion. When a Record r is received, Q_j

1) Q_j rejects r's *name* already exists in Q_j's Pagechain.
2) Q_j rejects r if the Record is not valid according to the above protocol.
3) Q_j rejects r if Bob's hidden service or any destination in *subdomains* does not exist.
4) Q_j informs Bob that r has been accepted.
5) Q_j sends r to all of its subscribers.

Quorum nodes must also periodically regenerate and resign the Merkle tree described in V-E and send the signature to all Quorum nodes and to all subscribers.

7) Page Selection: New Quorum nodes must select a Page from the previous Quorum to reference when generating a fresh Page. To reduce the chances of compromise, we select Pages according to our security assumptions. Let Charlie be a Mirror.

1) Charlie obtains and authenticates the consensus *cd* issued on day $\lfloor \frac{q}{\Delta q} \rfloor$ at 00:00 UTC and authenticates it.
2) Charlie derives the Quorum via the procedure described in section V-D3.
3) Charlie obtains the set of Pages maintained by $Quorum_q$.
4) For each Page,
   a) Charlie asserts that $fingerprint \in Quorum_q$.
   b) Charlie asserts that *prevHash* references Page_{q-1} found by this protocol.
   c) Charlie calculates $h = \mathcal{H}(prevHash \,\|\, recordList \,\|\, rand)$.
   d) Charlie asserts that $V_{ed}(h, E)$ returns true.
5) Charlie sorts the set of Pages by the number of routers that have signed h.
6) For each Page in each h,
   a) Charlie checks that *rand* is contained in *cd*.
   b) Charlie checks the validity of each Record in *recordList*.
7) If the validation of a Page fails, Charlie continues to the next h.
8) If the Page is valid, Charlie selects this page and aborts the procedure.

8) Domain Query: Alice needs only Bob's Record to contact Bob by his meaningful domain name. Let Alice type a domain d into the Tor Browser.

1) Alice constructs a Tor circuit to Charlie.
2) Alice asks Charlie for the most recent Record r containing d.
3) Charlie finds and returns r to Alice and the components of the Merkle tree T described in section V-E.
4) If r is not found, Alice asserts that the second-level name of d is not found in T, as otherwise Charlie is dishonest.
5) If r is found, Alice asserts that r is valid and contained in T, as otherwise Charlie is dishonest.
6) If d in r points to a domain with a .tor pseudo-TLD, d becomes that destination and Alice jumps to step 2.
7) Alice asserts that the destination uses a .onion pseudo-TLD and contacts Bob by the traditional hidden service protocol.

8) Alice extracts Bob's key from his hidden service descriptor and asserts that it matches r's *pubHSKey*.
9) Alice sends the original d to the hidden service.

The Merkle tree also prevents phishing attacks from Charlie. While the Quorum provides the verification that r is authentic and that d is unique, Alice may become certain by performing a synchronization against the OnioNS network and checking the Pagechain herself, but this is impractical in most environments. Tor's median circuit speed is often less than 4 Mbit/s,[24] so for the sake of convenience data transfer must be minimized. Therefore Alice can simply fetch minimal information and rely on her existing trust of members of the Tor network.

9) Onion Query: OnioNS also supports reverse-hostname lookups. In an Onion Query, Alice issues a hidden service address *addr* to Charlie and receives back all Records that have *addr* as either the owner or as a destination in their *subdomain*. Alice may obtain additional verification on the results by issuing Domain Queries on the source .tor domains. We do not anticipate Onion Queries to have significant practical value, but they complete the symmetry of lookups and allow OnioNS domain names to have Forward-Confirmed Reverse DNS matches. We suggest caching destination hidden service addresses in a digital tree (trie) to accelerate this lookup; a trie turns the lookup from $\mathcal{O}(n)$ to $\mathcal{O}(1)$, while requiring $\mathcal{O}(n)$ time and $\mathcal{O}(n)$ space to pre-compute the cache.

E. Authenticated Denial-of-Existence

In any system that serves authenticable names, a name server can prove a claim on the existence of a name by simply returning it. An often overlooked problem is ensuring that name servers cannot claim false negatives on resolutions; clients must be able to authenticate a denial-of-existence claim. Extensions to DNSSEC attempt to close this attack vector, but DNSSEC is not widely deployed, and we are not aware of any alternative DNS that addresses this. Although Alice may download the entire Pagechain and prove non-existence herself, we do not consider this approach practical in most realistic environments. Instead, we introduce a mechanism for authenticating denial-of-existence with minimal networking costs. To our knowledge this represents the first alternative DNS to authenticate denial-of-existence claims on domains en-masse.

We suggest reducing the networking costs with a Merkle tree[18] T. Let each Quorum node

1) Construct an array list *arr*.
2) For each second-level domain c in each Record r in the Pagechain, add $c \,\|\, \mathcal{H}(r)$ to *arr*.
3) Sort *arr*.
4) Construct a Merkle tree T from *arr*.
5) Generate $sig_T = S_{ed}(t \,\|\, r, e)$ where t is a timestamp and r is the root hash of T.

As Records contain both second-level domains and their subdomains, T needs only contain c to reference all domains in r, which further saves space. Then during a Domain Query Alice may use T to authenticate a domain d and verify non-existence for a Record r.

1) Alice extracts the second-level name c from d.
2) If r exists, Charlie returns the leaf node containing r and all the tree nodes from leaf r to the root and their sibling nodes, so that Alice can verify authenticity of r by recomputing the root hash and verify that the largest subset of Quorum nodes signed the same root hash.
3) If Charlie claims non-existence of c, he returns two adjacent leaves a and b (and the nodes on their paths and siblings) such that $a < c < b$, or in the boundary cases that a is undefined and b is the left-most leaf or b is undefined and a is the right-most leaf.
4) If either assertion fails, Charlie is dishonest.

The Quorum must regenerate T every ΔT hours to include new Records. Then Alice needs only fetch the signatures on T at least every ΔT hours to ensure that she can authenticate new Records during the Domain Query. Thus ΔT is the primary factor in the speed of Record propagation: Alice cannot authenticate or verify denial-of-existence claims on Records newer than ΔT. Alice must also fetch the L_Q signature from all Quorum nodes and assert that T is signed by the largest set of nodes maintaining the same Page, in correspondence with our security assumptions.

We note that a sorted Merkle tree does not support efficient dynamic record updates. The tree needs to be rebuilt to handle updates. To support efficient record update in $\mathcal{O}(\log(n))$ time, we can adopt the skip list data structure proposed in [12]. Proof of existence and non-existence still works in a similar way.

VI. Security Analysis

In this section, we analyse the security of the OnioNS system with regard to our security goals. First, the registrations and client queries are anonymous because they occur over a Tor circuit, which we assume provides privacy and anonymity. No identifiable information is leaked from the data contents as well. Second, registrations are authenticable, which can be reduced to the security of the Merkle hash tree and to the assumption that the largest subset of Quorum nodes are honest. We verify this assumption in section VI-A and VI-A1. Domain uniqueness also stems from the above assumption, relying on honesty of Quorum nodes to avoid collisions. Quorum selection is a random process assuming security of the commitment algorithm run by the directory authorities, which we analyse in section VI-A3. We also highlight the potential for the leakage of the .tor pseudo-TLD on the Internet DNS.

A. Quorum Selection

In section IV, we assume that an attacker, Eve, controls some fixed f_E fraction of routers on the Tor network. Quorum selection may be considered as an L_Q-sized random sample taken from an L_T-sized population without replacement, where the population contains a subset of $L_Q * f_E$ entities that we assume are compromised and colluding. If our selection includes L_E Eve-controlled routers, then Eve controls the Quorum if either $> \frac{L_Q - L_E}{2}$ honest Quorum nodes disagree or if $L_E > \frac{L_Q}{2}$. The former scenario is hard to model theoretically or in simulation, but the probability of the latter can be statistically calculated. The Quorum is short-lived and changed every Δq days, so we must consider the implications of selections for both L_Q and Δq.

The probability that Eve controls L_E Quorum nodes is given by the hypergeometric distribution, whose probability mass function (PMF) is shown in Equation 1.

$$\Pr(L_E) = \frac{\binom{f_E}{k}\binom{N-f_E}{L_Q-k}}{\binom{N}{L_Q}} \tag{1}$$

Quorum selection is heavily skewed by the distribution of consensus weight. The influence of consensus weight provides a strong defence against Sybil attacks by making malicious subversion significantly more expensive.

1) Quorum Size: If all Tor routers have equal consensus weight, then the probability that $L_E > \frac{L_Q}{2}$, a complete compromise of the Quorum, is given by Equation 2. Odd choices for L_Q prevents the network from splintering in the event that the Quorum is evenly split across two Pages. We assume that L_Q will be selected from a pool of 5,400 Quorum Candidates – the number of Tor routers that have Fast, Stable, Running, and Valid flags as of July 2015[24]. We provide the statistical calculations in Figure 7.

$$\Pr(L_E > \frac{L_Q}{2}) = \sum_{x=\lceil \frac{L_Q}{2} \rceil}^{L_Q} \frac{\binom{f_E}{k}\binom{N-f_E}{L_Q-k}}{\binom{N}{L_Q}} \tag{2}$$

However, in practice Tor routers do not have equal consensus weight. We provide a histogram of weight distribution across July 2015 in Figure 8. This significantly skews the hypergeometric distribution as shown in Figure 9.

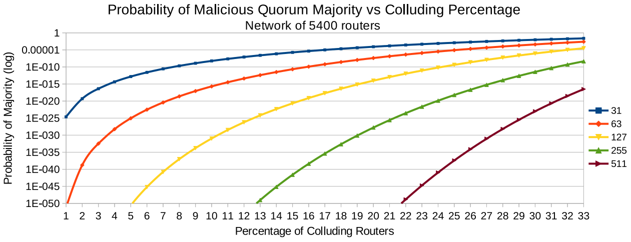

Fig. 8. PLACEHOLDER IMAGE

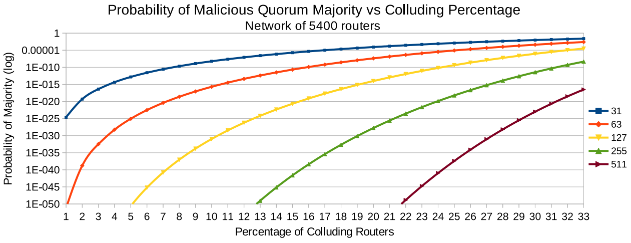

Fig. 9. PLACEHOLDER IMAGE

Figure 7 shows that choices of $L_Q = XX$ and $L_Q = YY$ are suboptimal, they quickly grow for even low levels of f_E. Small Quorums are also more susceptible to DDoS attacks. We feel it safe to discard threats that have probabilities $\leq \frac{1}{2^{128}} \approx 10^{-38.532}$ — the probability of Eve randomly guessing a 128-bit AES key. $L_Q = 31$ exceeds this with more than XX percent collusion, $L_Q = 63$ above YY percent, and $L_Q = 127$, $L_Q = 255$, and $L_Q = 511$ at $\geq AA$, $\geq BB$, and $\geq CC$ percent, respectively. Therefore we recommend setting $L_Q \geq ZZ$.

2) Quorum Rotation: In section IV, we assume that f_E is fixed and does not increase in response to the inclusion of OnioNS on the Tor network. If we also assume that L_T is fixed, then we can examine the impact of choices for Δq and calculate the probability of Eve compromising *any* Quorum over a long period of time t. Smaller values for Δq implies that we select more Quorums over that time period and thus increase our cumulative chances of compromise, but it also reduces the disruption timeline for a malicious Quorum. Eve's cumulative chances of compromising any Quorum is given by $1-(1-\frac{f_E}{L_T})^t$. We estimate this over 10 years in Figure 10.

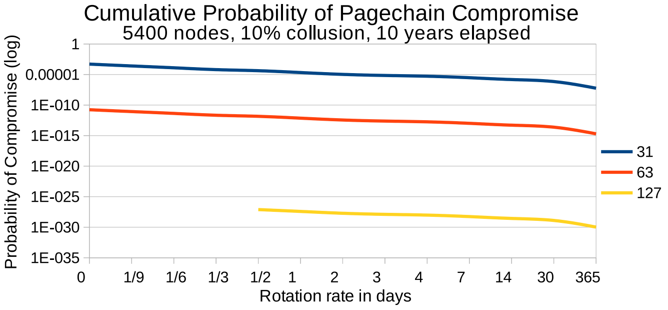

Fig. 10. The cumulative probability that Eve controls any Quorum at different rotation rates over 10 years at $f_E = 10$. We do not consider $L_Q = 255$ or $L_Q = 511$ as they produce values less than 10^{-58} and 10^{-134}, respectively, far below our $10^{-38.532}$ threshold.

Figure 10 suggests that although lower values of Δq negatively impact security, the choice of L_Q is more significant. Based on Figure 10 we further reiterate our recommendation of $L_Q \geq 127$ and suggest $\Delta q \geq 1$. Although a malicious Quorum would have the capabilities to deploy a variety of attacks on the network, the proper selections of L_Q and Δq reduces the likelihood of this occurring to near-zero probabilities. We consider this a stronger solution than introducing countermeasures to specific Quorum-level attacks.

3) Global Randomness: We implement $\mathcal{G}(t)$ using a commitment scheme run by Tor directory authorities[13]. Commitment protocols have been studied in other works[25][20], have many interesting applications, and are well understood. If all parties are at least semi-honest then the commitment protocols generally display correctness, privacy, and binding. However, if some participants are malicious, they demonstrate known weaknesses.

Namely, while reveals must demonstrably match commits, each participant may choose to reveal or not. If they do not reveal, their value is lost and the protocol produces a different output. If Eve controls b participants, she can make this choice with each participant in turn, allowing 2^b different outcomes. Goulet and Kadianakis' directory authority commitment scheme is particularly vulnerable to this attack because the reveals are public for approximately 12 hours before the final random value is published. However, as there

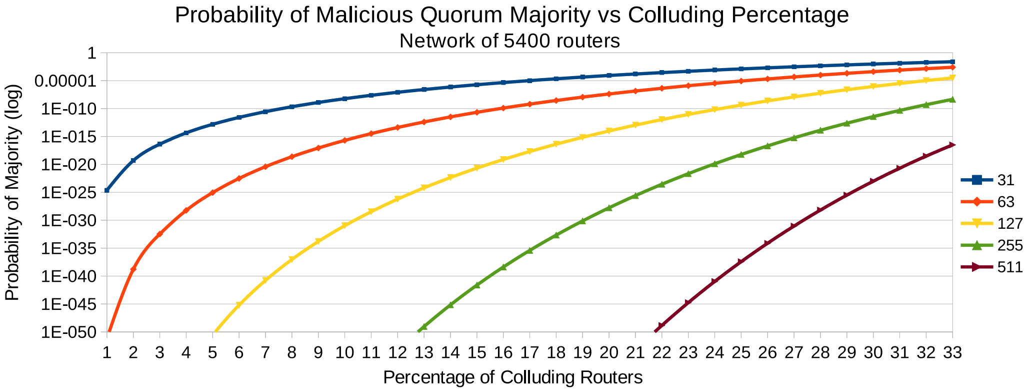

Fig. 7. We calculate the PMF of the hypergeometric distribution for a population of 5,400, $f_E \in [1, 33]$, and five selections for L_Q: 31, 63, 127, 255, and 511. $f_E > 33$ represents a complete compromise of the Tor network, so we do not consider values beyond this range.

are currently only nine directory authorities and the security of the Tor network rests on the assumption that five or more of them are at least semi-honest, the commitment scheme has at most $2^4 = 16$ different outcomes in the worst case without violation of our assumptions.

Given the statistical calculations and the safety margin introduced by the recommendations in section VI-A1, we do not consider this a significant threat to our system and conclude that the Quorum Formation protocol is secure under our design assumptions. The unpredictability of the reveals provides the strongest defence against Quorum-level attacks.

B. Outsourcing Record Generation

In the Record Generation protocol, we intentionally included the scrypt calculation inside the signing step in order to require Bob to run scrypt himself. However, our protocol does not entirely prevent Bob from outsourcing this expensive computation to a secondary resource, Craig, in all cases. We assume that Craig does not have Bob's private key. Then,

type, the purpose of this Record; *name*, a meaningful second-level domain name with the .tor pseudo-TLD; *subdomains*, a one-to-one map of .tor domains of level three or higher and .tor or .onion destinations; *contact*, Bob's PGP key fingerprint if he chooses to disclose it; *rand*, the output of $\mathcal{G}(t)$ at the current time; *nonce*, bytes used as a source of randomness for the proof-of-work; *pow*, the output of $\mathrm{PoW}(i)$; *signature*, the output of $S_{RSA}(m, r)$; and *pubHSKey*

1) Bob creates an initial Record R and completes the *type*, *name*, *subdomains*, *contact*, and *rand* fields.
2) Bob sends R to Craig.
3) Let c be $type \parallel name \parallel subdomains \parallel contact \parallel rand$.
4) Craig generates a random integer K and then for each iteration j from 0 to K,
    a) Craig increments *nonce*.
    b) Craig sets *PoW* as $\mathrm{PoW}(c \parallel nonce)$.
    c) Craig saves the new R as C_j.
5) Craig sends all $C_{0 \leq j \leq K}$ to Bob.
6) For each Record $C_{0 \leq j \leq K}$ Bob computes
    a) Bob sets *pubHSKey* to his public RSA key.
    b) Bob sets *signature* to $S_{RSA}(m, r)$ where $m = c \parallel nonce \parallel pow$ and r is Bob's private RSA key.
    c) Bob has found a valid Record if $\mathcal{H}(c \| nonce \| pow \parallel signature) \leq d$ where d is the difficult level.

However, this ensures that Craig will with high probability compute more scrypt iterations than necessary; Craig cannot generate *signature* and thus cannot compute if the hash is below the threshold. Moreover, the scrypt work incurs a cost onto Craig that must be compensated financially by Bob. Thus the Record Generation protocol always places a cost on Bob.

C. DNS Leakage

Accidental leakage of .tor lookups over the Internet DNS via human mistakes or misconfigured software may compromise user privacy. This vulnerability is not limited to OnioNS and applies to pseudo-TLD; Mohaisen and Thomas observed .onion lookups on root DNS servers at a frequency that corresponded to external global events and highlighting the human factor in those leakages[28]. Closing this leakage is difficult; arguably the simplest approach is to introduce whitelists or blacklists into common web browsers to prevent known pseudo-TLDs from being queried over the Internet DNS. Such changes are outside the scope of this work, but we highlight the potential for this attack.

VII. Evaluation

A. Implementation

We have build a reference implementation of the Onion Name System in C++11. We implemented the essential protocols for Bob, Charlie, and Alice. We divided our software into three parts: OnioNS-client, OnioNS-server, and OnioNS-HS, with OnioNS-common as a shared library dependency and built packages for several Linux distributions. We utilize several common libraries; Botan[7] provides the implementation of our cryptographic operations, Boost Asio[16] provides our networking engine, Stem provides abstraction for Tor control, and jsoncpp[9] provides implementations of JSON encoding and decoding. We encode all the data structures in JSON; JSON is significantly more compact than XML, but retains user readability and its support of basic primitive types is highly applicable to our needs. Our code is available under the Modified BSD License, matching Tor.

OnioNS-HS is a small command-line utility that provides the capability for Bob to define a Record, make it valid, and then transmit it over a Tor circuit to a Quorum node. We provide flags to allow Bob to run the proof-of-work while inside a small VM or other restricted environment. OnioNS-HS then communicates with Tor's SOCKS port to send the Record to Quorum nodes.

OnioNS-server provides the main OnioNS functionality and is designed to run in the background with minimal configuration. We introduce a flag to specify whether the server is a Quorum node or a traditional Mirror. As a Mirror, it maintains a network connection and subscribes to OnioNS servers running as Quorum nodes. It utilizes OnioNS-common to distributes networking information and public keys by installing several JSON files.

The client-side software consists of three distinct pieces: onions-client, which connects to a Tor circuit over a Mirror and then listens on a TCP port on localhost for .tor domains; a Python script which waits for Tor stream requests, intercepts .tor domains, and sends them to the IPC port for resolution; and onions-tbb, which launches the Tor executable, onions-client, and then the Python script as child processes when the Tor Browser starts. The Python script reconfigures Tor to prevent it from auto-attaching network streams to circuits; instead, it filters stream requests and resolves .tor domains before letting Tor attach them, while instantly manually attaching all other domains.

1) Integration Test: First, we started two OnioNS servers, a Quorum node and a Mirror. We created a hidden service for our project, "onions55e7yam27n.onion", and generated a Record to claim "example.tor". Once the Record was complete, OnioNS-HS transmitted it over a Tor circuit to the Quorum node. The Mirror, as a subscriber to the Quorum node, also received, verified, and cached our Records.

Finally, we installed our client software into Tor Browser 5.5, a fork of Firefox 38.2.0 ESR. Once we launched the Tor Browser, the prepackaged Tor binary, onions-client, and the Python script started in the background. We entered "example.tor" into the Tor Browser, which caused Tor to fire a network stream event over its controller port. The Python script intercepted the .tor pseudo-TLD and sent the domain over IPC to onions-client. The client then issued a Domain Query to the Mirror for "example.tor" and resolved it to "onions55e7yam27n.onion". The Python script rewrote the stream request to "onions55e7yam27n.onion" before letting Tor attach it to a circuit. Tor then loaded our hidden service and returned our webpage back to the Tor Browser.

This process resulted in the Tor Browser transparently loading our hidden service under "example.tor" without requiring any further user interaction. We illustrate this result in Figure 11. The software performed asynchronously and allowed normal browsing to both the Internet and other hidden services, even while the OnioNS domain was resolving.

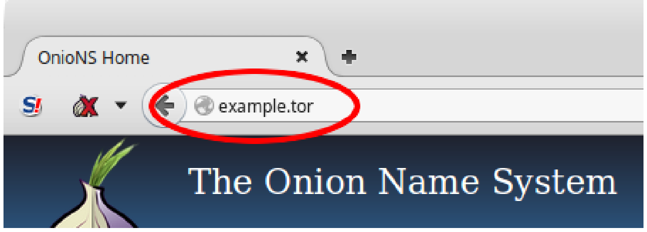

Fig. 11. We load the OnioNS' hidden service, onions55e7yam27n.onion, transparently under the "example.tor" domain. The OnioNS software launches with the Tor Browser.

B. Results

1) Performance: We selected the parameters of scrypt such that it consumed 128 MB of RAM during operation. We consider this an affordable amount of RAM for low-end consumer-grade computers. We created a multi-threaded implementation of the Record Generation protocol and used all eight virtual CPU cores on Machine B to generate our Record. As expected, our RAM consumption scaled linearly with the number of scrypt instances executed in parallel; we observed approximately 1 GB of RAM consumption during Record Generation.

Then we conducted several performance measurements for the Domain Queries. Our experiment involves two machines, A and B. Both were hosted on 1 Gbit connections on a university campus. Machine A has an Intel Core2 Quad Q9000 (Penryn architecture) @ 2.00 GHz CPU from late 2008 and Machine B has an Intel i7-2600K (Sandy Bridge architecture) @ 4.3 GHz CPU from 2011, representing low-end and medium-end consumer-grade computers, respectively. We measured and averaged 200 samples of the CPU wall-time required for both machines to validate the Record.

Description	A (ms)	B (ms)
Parsing JSON	5.21	2.42
Validating scrypt	448.184	294.963
$V_{RSA}(m, E)$	6.35	2.74
Total Time	459.744	300.123

As expected, Machine B outperformed Machine A in all instances and we observed that single iteration of scrypt dominated the total validation time. This is a CPU cost introduced to Tor clients, Mirrors, and Quorum nodes for each Record.

Mirrors must also check the Page signatures from all L_Q Quorum nodes. We use Ed25519 to reduce the signature space requirements and CPU time required for verification: $S_{ed}(m, e)$ signatures fit into 64 bytes and may be verified in batch form. Bernstein et al. reports that a quad-core Westmere-era CPU can generate 109,000 signatures per second and verify 71,000 signatures per second with 134,000 CPU cycles per signature[4]. Therefore, even with large Quorums, we anticipate Mirrors being able to verify Page signatures from all Quorums in sub-second time on moderate hardware.

2) Latency: Although Tor is a low-latency network, as Domain Queries occur over a Tor circuit the three-hop path still introduces some latency into the communications between the client and a Mirror. The time is highly dependent on the circuit's network distance and the speed of each Tor router. This adds an additional delay between the time that a user enters an OnioNS domain into the Tor Browser and when Tor begins loading the hidden service. Fortunately, latency and load times across Tor circuits have been well studied. Domain Queries transfer both a Record, a Merkle subtree, and a collection of ed25519 signatures. We estimate the size of this information to be approximately 50 KB. We provide the distribution of circuit performance in Figure 12.

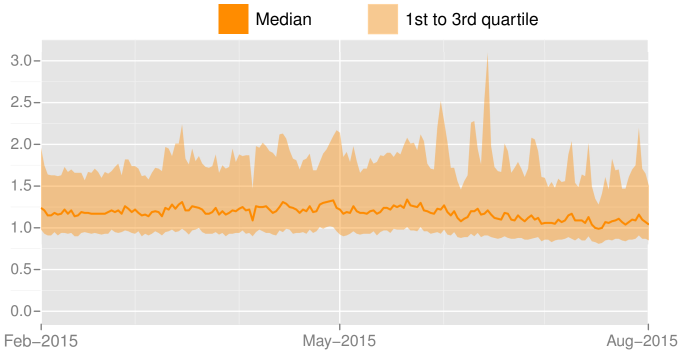

Fig. 12. The average performance to download 50 KB files over Tor circuits, as measured by Tor bandwidth authorities. The first and third quartiles are shown with the median time shown in orange[24].

This network latency is supplemental to the time to verify the Record and Merkle subtree. We avoid additional latency costs by implementing a client-side cache in order to allow subsequent queries to be resolved locally. We also reduce the expense of circuit construction by building the circuit on startup.

VIII. Conclusions and Future Work

We have presented the Onion Name System (OnioNS), a distributed, secure, and usable alternative DNS that maps globally-unique and meaningful .tor domains to .onion hidden service addresses, and achieve all three properties of Zooko's Triangle. We enable any hidden service operator to anonymously claim a human-readable name for their server and clients to query the system in privacy-enhanced manner. We introduce a distributed self-healing blockchain-based database and mechanisms that let clients authenticate and verify denial-of-existence claims. Additionally, we utilize the existing and semi-trusted infrastructure of Tor, which significantly narrows our threat model to already well-understood attack surfaces and allows our system to be integrated into Tor with minimal effort. Our reference implementation demonstrates high usability and shows that OnioNS successfully addresses the major usability issue that has been with Tor hidden services since their introduction in 2002.

In future work we will expand our implementation and pursuit integrating it into Tor. OnioNS requires a few changes to Tor, namely a new .tor pseudo-TLD and Ed25519 router keys, but we introduce no changes to Tor's hidden service protocol. Should Tor's developers introduce changes to the hidden service protocol, OnioNS can become forwards-compatible with a few changes. Additionally, our implementation currently only supports ASCII characters in domain names, so in future work we will explore implementing Punycode to provide support for international character sets. Unlike the Internet DNS, we will disallow digits zero and one (similar to base32 encoding) in order to reduce the threat of phishing attacks from spoofed domains with indistinguishable characters.

We will also explore the possibilities of hosting the OnioNS servers as hidden services. The Tor protocol provides defences against several network-level attacks and when combined with end-to-end authentication this approach may prove beneficial to our system.

Acknowledgements

We would like to thank Roger Dingledine, George Kadianakis, Yawning Angel, and Nick Mathewson for their support, commentary, and assistance with Tor technical support.

References

[1] B. Awerbuch and C. Scheideler. Group spreading: A protocol for provably secure distributed name service. In *Automata, Languages and Programming*, pages 183–195. Springer, 2004.

[2] D. J. Bernstein. Curve25519: new diffie-hellman speed records. In *Public Key Cryptography-PKC 2006*, pages 207–228. Springer, 2006.

[3] D. J. Bernstein. Dnscurve: Usable security for dns, 2009.

[4] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang. High-speed high-security signatures. In *Cryptographic Hardware and Embedded Systems–CHES 2011*, pages 124–142. Springer, 2011.

[5] C. Cachin and A. Samar. Secure distributed dns. In *Dependable Systems and Networks, 2004 International Conference on*, pages 423–432. IEEE, 2004.

[6] R. Castellucci. Namecoin. https://namecoin.info/, 2015.

[7] B. Developers. Botan: Crypto and tls for c++11. http://botan.randombit.net/, 2015. accessed August 10, 2015.

[8] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. Technical report, DTIC Document, 2004.

[9] C. Dunn. Jsoncpp – c++ library for interacting with json. https://github.com/open-source-parsers/jsoncpp, 2015. accessed August 10, 2015.

[10] M. S. Ferdous, A. Jøsang, K. Singh, and R. Borgaonkar. Security usability of petname systems. In *Identity and Privacy in the Internet Age*, pages 44–59. Springer, 2009.

[11] I. Goldberg, D. Stebila, and B. Ustaoglu. Anonymity and one-way authentication in key exchange protocols. *Designs, Codes and Cryptography*, 67(2):245–269, 2013.

[12] M. T. Goodrich, R. Tamassia, and A. Schwerin. Implementation of an authenticated dictionary with skip lists and commutative hashing. In *DARPA Information Survivability Conference &amp; Exposition II, 2001. DISCEX'01. Proceedings*, volume 2, pages 68–82. IEEE, 2001.

[13] D. Goulet and G. Kadianakis. Random number generation during tor voting. https://gitweb.torproject.org/torspec.git/tree/proposals/250-commit-reveal-consensus.txt, 2015. accessed August 10, 2015.

[14] G. Kadianakis and K. Loesing. Extrapolating network totals from hidden-service statistics. *Tor Technical Report*, page 10, 2015.

[15] katmagic. Shallot. https://github.com/katmagic/Shallot, 2012. accessed May 9, 2015.

[16] C. Kohlhoff. Boost asio. http://www.boost.org/doc/libs/1_59_0/doc/html/boost_asio.html, 2015. accessed August 10, 2015.

[17] M. Matsumoto and T. Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8(1):3–30, 1998.

[18] R. C. Merkle. A digital signature based on a conventional encryption function. In *Advances in Cryptology-CRYPTO'87*, pages 369–378. Springer, 1988.

[19] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Consulted*, 1(2012):28, 2008.

[20] M. Naor. Bit commitment using pseudo-randomness. In *Advances in CryptologyCRYPTO89 Proceedings*, pages 128–136. Springer, 1990.

[21] S. Nicolussi. Human-readable names for tor hidden services. 2011.

[22] L. Overlier and P. Syverson. Locating hidden servers. In *Security and Privacy, 2006 IEEE Symposium on*, pages 15–pp. IEEE, 2006.

[23] C. Percival and S. Josefsson. The scrypt password-based key derivation function. 2012.

[24] T. T. Project. Tor metrics. https://metrics.torproject.org/, 2015. accessed May 9, 2015.

[25] R. Rivest. Unconditionally secure commitment and oblivious transfer schemes using private channels and a trusted initializer. *Unpublished manuscript*, 1999.

[26] M. Stiegler. Petname systems. *HP Laboratories, Mobile and Media Systems Laboratory, Palo Alto, Tech. Rep. HPL-2005-148*, 2005.

[27] P. Syverson and G. Boyce. Genuine onion: Simple, fast, flexible, and cheap website authentication. 2014.

[28] M. Thomas and A. Mohaisen. Measuring the leakage of onion at the root. *In the Proceedings of the 12th Workshop on Privacy in the Electronic Society*, 2014.

[29] M. Wachs, M. Schanzenbach, and C. Grothoff. A censorship-resistant, privacy-enhancing and fully decentralized name system. In *Cryptology and Network Security*, pages 127–142. Springer, 2014.