The Onion Name System

Tor-powered Distributed DNS for Tor Hidden Services

[three authors redacted]

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 the cryptographically-generated non-human-readable 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.

CCS Concepts

•Information systems → Block / page strategies; •Networks → Naming and addressing; •Security and privacy → *Distributed systems security;* Cryptography; Security protocols;

Keywords

Tor, onion, hidden service, anonymity, privacy, network security, petname

1. 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[7] 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 defence against these types of attacks.

1.1 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.

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. 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

ACM CCS '15 October 12-16, 2015, Denver, Colorado, USA

DOI: 10.475/123_4

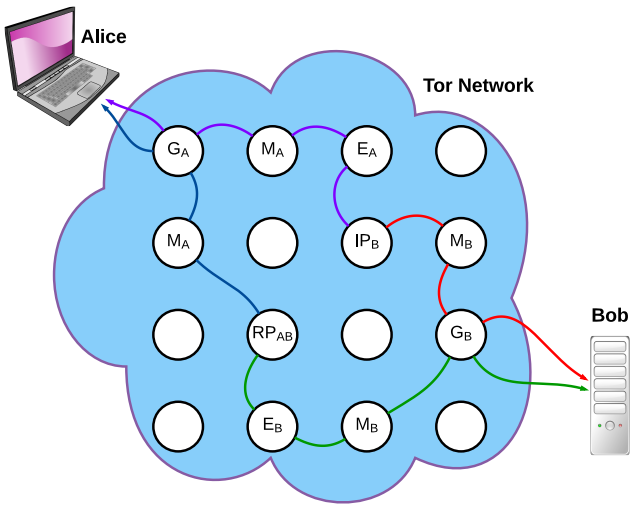

Figure 1: A Tor client, Alice, and a hidden service, Bob, communicate by mating two Tor circuits at a *rendezvous point* (RP). Bob maintains circuits (red) to his *introduction points* (IPs) and advertises his IPs and PK_B to the Tor network. When Alice obtains Bob's address through a backchannel, she selects and builds a circuit (blue) to an RP and other circuit (purple) to one of Bob's IPs. She then tells $\mathcal{E}_{\mathrm{PK}_B}(RP)$ to IP_B. Then Bob construct a circuit (green) to the *RP* and Alice and Bob can communicate with bi-directional anonymity.[17]

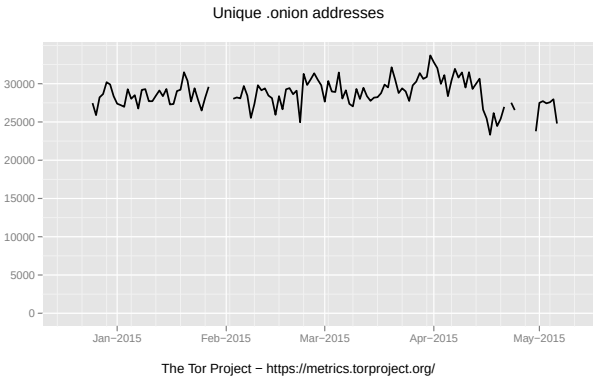

Figure 2: The number of unique .onion addresses seen in the Tor network between December 2014 and May 2015.[11][20]

neither convenient nor does it scale well. Given the approximetly 25,000 hidden services on the Tor network, (Figure 2) this suggests the strong need for a more complete solution to solve the usability issue.

1.2 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 its .onion address 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 are tamper-proof, self-healing, resistant to node compromise and achieves 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.

2. DESIGN OBJECTIVES

Tor's privacy-enhanced environment introduces a distinct set of challenges that must be met by any additional infrastructure. Here we enumerate a list of requirements that must be met by any DNS applicable to Tor hidden services. In Section 3 we analyse existing works and show how these systems do not meet these requirements and in Section 5 and **??** 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. **Authenticable registrations.** 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 or memory burdens.

2.1 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,[8][21] he claimed any naming system could only achieve two of these properties. This is illustrated in Figure 3.

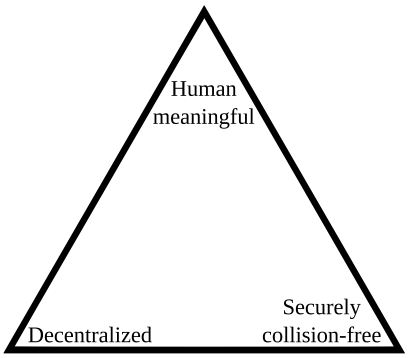

Figure 3: Zooko's Triangle.

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.

2.2 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.

3. RELATED WORKS

Vanity key generators (e.g. Shallot[12]) 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.[22] 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.[16] 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. With the exception of extensions such as DNSSEC, 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.[23] 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[23] (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[15] 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.

4. 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.

We assume secure cryptographic primitives; namely that Eve cannot break standard cryptographic primitives such as AES, SHA-2, RSA, Curve25519, Ed25519, 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.

We assume that not all Tor routers are honest; that Eve controls some percentage of Tor routers such that Eve's routers may actively collude. Routers may also be semi-honest; wiretapped but not capable of violating protocols. However, the percentage of dishonest and semi-honest routers 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 OnionNS 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.

If C is a Tor network status consensus, (section 5.1) Q is an M-sized set randomly but deterministically selected from the Fast and Stable routers listed in C, and Q is under the influence of one or more adversaries, we assume that the largest subset of agreeing routers in Q are at least semi-honest.

5. SOLUTION

5.1 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 an association between a meaningful second-level domain name and his .onion address. Without loss of generality, let this be "example.tor → example0uyw6wgve.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 denial-of-service 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.

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 it signature and proof-of-work, and follows the association to "example0uyw6wgve.onion". As Bob's Record is self-signed using Bob's private key, Alice can verify the Record's authenticity. Finally, Alice uses this address and the Tor hidden service protocol to contact Bob. Note that Alice does not have to resort to using "example0uyw6wgve.onion", rather that Bob can be successfully referenced by "example.tor". We illustrate the OnioNS overview in Figure 4.

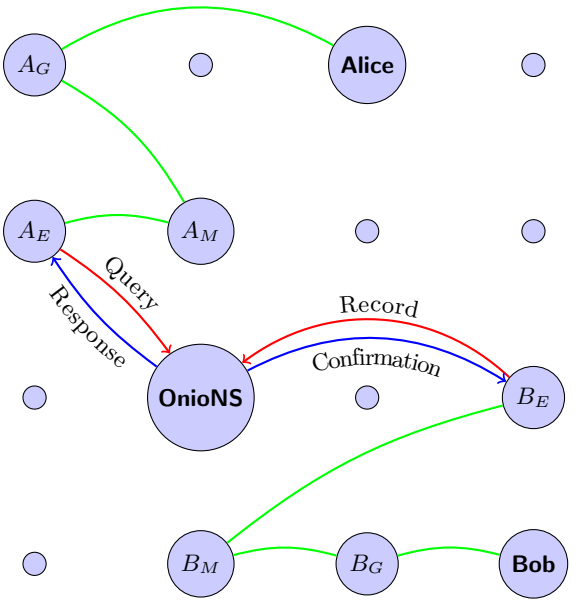

Figure 4: Bob uses a Tor circuit (B_G, B_M, B_E) to anonymously broadcast a record to OnioNS. Alice uses her own Tor circuit (A_G, A_M, A_E) to query the system for a domain name, and she is given Bob's record in response. Then Alice connects to Bob by Tor's hidden service protocol.

5.2 Cryptographic Primitives

OnionNS makes use of cryptographic hash algorithms, digital signatures, proof-of-work, and a pseudorandom number generator. 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 and because it is possible to convert Curve25519 to Ed25519 in constant time, we can theoretically use existing NTor 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 $H(x)$ be a cryptographic hash function. In our reference implementation we define $H(x)$ 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 an RSA digital signature. Let $S_{RSA}(m, r)$ use $H(x)$ as a digest function on m in all use cases. In our reference implementation we define $S_{RSA}(m, r)$ as EMSA-PSS, (EMSA4) a probabilistic signature scheme defined by PKCS1 v2.1 and republished in 2003's RFC 3447.
- Let $V_{RSA}(m, E)$ 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 $H(x)$ 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 PoW(k) be a one-way collision-free function that accepts an input key k and returns a deterministic output. Our reference implementation uses the scrypt[19] key derivation function with a fixed salt.
- Let $R(s)$ be a pseudorandom number generator that accepts an initial seed s and returns a list of numerical pseudorandom numbers. s is unpredictable in our design, so $R(s)$ 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.[13]

5.3 Definitions

network status consensus
: In Tor's existing infrastructure, a small set of semi-trusted *directory authority* servers digitally sign and distribute the network information, status, and cryptographic keys of all Tor routers to the network every hour. This allows a dynamic network topology and introduces a PKI. The consensus is primarily useful for circuit construction via the TAP[9] or NTor[10], the latter of which utilizes Curve25519[2] keys for fast ECDHE. In this work, we utilize the consensus as an information distribution system and as a global source of agreed-upon entropy.

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.

Record
: A *Record* contains *nameList*, a one-to-one map of .tor pseudo-TLD domain names and .tor or .onion pseudo-TLD destinations; *contact*, Bob's PGP key fingerprint if he chooses to disclose it; *consensusHash*, the hash of the consensus document that generated the current Quorum; *nonce*, four bytes used as a source of randomness for the proof-of-work; *pow*, the output of PoW(i); *recordSig*, the output of $S_d(m, r)$ where $m =$ *nameList* $\|$ *timestamp* $\|$ *consensusHash* $\|$ *nonce* $\|$ *pow* and r is the hidden service's private RSA key; and *pubHSKey*, Bob's public hidden service RSA key.

Snapshot
: A *Snapshot* contains *originTime*, the Unix time when the snapshot was first created; *recentRecords*, an array list of Records in reverse chronological order; *fingerprint*, the Tor fingerprint of the router maintaining this Snapshot; and *snapshotSig*, the output of $S_{ed}($*originTime*$\|$ *recentRecords* $\|$ *fingerprint*$], e)$ where e is the router's private Ed25519 key.

Page
: A *Page* contains *prevHash*, the output of $H($*prevHash*$\|$ *recordList*$\|$*consensusHash*) of a previous Page; *recordList*, a deterministically-sorted array list of Records; *consensusHash*, the hash of the consensus document that generated the current Quorum; *fingerprint*, the Tor fingerprint of the router maintaining this Page; and *pageSig*, the output of $S_{ed}(H($*prevHash* $\|$ *recordList* $\|$ *consensusHash*$)], e)$ where e is the router's private Ed25519 key.

  prevHash links Pages over time, forming an append-only public ledger known as an *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 last security assumption, *prevHash* must reference a Page that is both valid and maintained by the largest number of Quorum members, 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.

Mirror
: 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

A *Quorum Candidates* 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.

Quorum

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

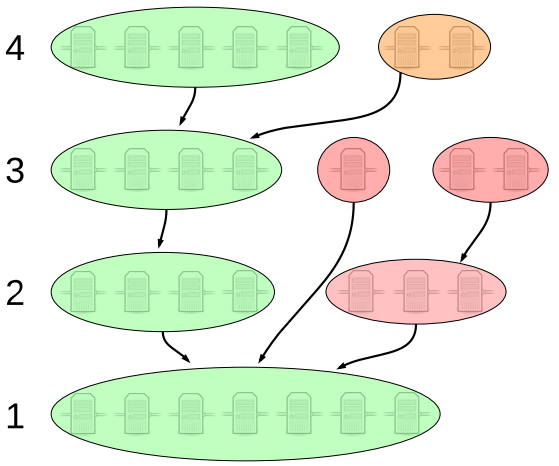

Figure 5: An example Pagechain across four Quorums with three side-chains. The valid master Pagechain from honest Quorum nodes (green) resists corruption from maliciously-colluding nodes (red) and malfunctioning nodes (orange).

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
s	the Snapshot iteration counter
Δs	the lifetime of a Snapshot in minutes

Table 1: Frequently used notations

5.4 Protocols

We now describe the protocols fundamental to OnioNS functionality.

5.4.1 *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 $\approx$ 2.25 million daily users[20]: it is infeasible for any single node to handle queries from all of them. Instead, let each Mirror first calculate $t = H(pc \parallel \lfloor \frac{m-15}{30} \rfloor)$ 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 February 2015, out of the $\approx$ 7,000 nodes participating in the Tor network, $\approx$ 5,400 of these node have these flags and meet the latter requirement.[20]

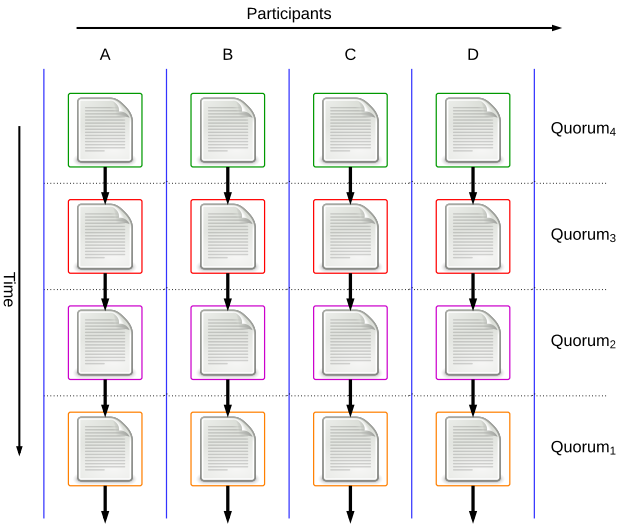

Figure 6: The master Pagechain is one-dimensional but spans across the network as each Mirror holds a copy. Each Page is maintained by a respective Quorum.

5.4.2 *Quorum Formation*

Quorum Candidate Tor routers can determine in $\mathcal{O}(L_T)$ time if they have been chosen as part of the current Quorum. By the same procedure, clients can derive the Quorum in the past or present. Let Charlie be a Quorum Candidate.

1. Charlie obtains the consensus documents, cd, published on day $\lfloor \frac{q}{\Delta q} \rfloor$ at 00:00 GMT.
2. Charlie scans cd and constructs a list qc 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.
3. Alice constructs $f = R(H(cd))$.
4. Alice uses f to randomly scramble qc.
5. The first $\min(\text{size}(qc), L_Q)$ routers are the Quorum.

5.4.3 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 Quorum Nodes, Mirrors, and clients, so Bob must follow this protocol.

1. Bob constructs the *nameList* domain-destination associations. He must include at least one second-level domain name. Each domain name can be up to 128 characters and contain any number of names.
2. Bob optionally provides his PGP key fingerprint in *contact*.
3. Bob sets *consensusHash* to the output of $H(x)$, where x is the consensus documents published at 00:00 GMT on day $\lfloor \frac{q}{\Delta q} \rfloor$.
4. Bob initially defines *nonce* as four zeros.
5. Let *central* be *type* $\|$ *nameList* $\|$ *contact* $\|$ *timestamp* $\|$ *consensusHash* $\|$ *nonce*.
6. Bob sets *pow* as PoW(*central*).
7. Bob sets *recordSig* as the output of $S_d(m, r)$ where $m =$ *central* $\|$ *pow* and r is Bob's private RSA key.
8. Bob saves the PKCS.1 DER encoding of his RSA public key in *pubHSKey*.

The Record is valid when $H(\textit{central} \parallel \textit{pow} \parallel \textit{recordSig}) \leq 2^{d*c}$ where d is a fixed constant that specifies the work difficulty and c is the number of second-level domain names claimed in the Record. This also requires Bob to increment *nonce* and resign his Record at every iteration of PoW(*central*).

5.4.4 Record Processing

A Quorum node Q_j listens for new Records from hidden service operators. When a Record r is received, Q_j

1. Q_j rejects r if the Record is not valid.
2. Q_j rejects r if any destination .onion addresses have no matching hidden service descriptor.
3. Q_j rejects r if any of its second-level domains already exist in Q_j's Pagechain.
4. Q_j informs Bob that r has been accepted.
5. Q_j merges r into its current Snapshot.
6. Q_j regenerates *snapshotSig*.

Then every Δs minutes each Quorum node floods its Snapshot to all other Quorum nodes, merges in Snapshots from other Quorum nodes into its Page, and generates a fresh Snapshot.

5.4.5 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 based on our last security assumption. Let Charlie be a Mirror.

1. Charlie obtains the set of Pages maintained by $Quorum_q$.
2. Charlie obtains the consensus *cd* issued on day $\lfloor \frac{q}{\Delta q} \rfloor$ at 00:00 GMT and authenticates it.
3. Charlie uses *cd* to derive the Quorum.
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 $h = H(\textit{prevHash} \parallel \textit{recordList} \parallel \textit{consensusHash})$.
   (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 $\textit{consensusHash} = H(cd)$.
   (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, the next *prevHash* references it.

5.4.6 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, and let Alice pre-fetch from Charlie, a Quorum Candidate, the Merkle tree T described in section 5.5.

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, or an error if d cannot be found.
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 assets 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.

Alice may also request additional information from Charlie, providing her with more authenticity verification at the expense of additional networking and processing costs. Alice may ask for the Page p containing r, which she can verify and authenticate against a single Quorum node, but she cannot check the last security assumption. Since p's *pageSig* is a signature on $h = H(\textit{prevHash} \parallel \textit{recordList} \parallel \textit{consensusHash})$, she may also ask for all hs and all *pageSig*s and assert that the Page Selection protocol derives p. However, she does not have enough information to verify the integrity of p's *prevHash*. Lastly, Alice may become certain that r is authentic and that d is unique 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,[20] 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.

5.4.7 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 a destination in their *nameList*. 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.

5.5 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[14] 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 $c \parallel H(r)$ to *arr*.
3. Sort *arr*.
4. Construct a Merkle tree T from *arr*.
5. Generate $sig_T = S_{ed}(t \parallel r, e)$ where t is a timestamp and r is the root hash of T.

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 Charlie returns r, Alice finds the leaf l in T containing c and then asserts that $H(r) \in l$.
3. If Charlie claims non-existence, Alice asserts that c does not exist in T.
4. If either assertion fails, Charlie is dishonest.

As the leaves of T are sorted, Alice may locate l in $\mathcal{O}(\log(n))$ time via a binary search; she simply needs to find two leaves a and a 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. 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.

The Quorum must regenerate T every ΔT hours to include new Records. Then Alice needs only pre-fetch T and its sig_Ts 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. However, the networking cost is $\mathcal{O}(n\Delta T)$. Alice must also fetch the $L_Q sig_T$s from all Quorum nodes and assert that T is signed by the largest set of nodes maintaining the same Page, in correspondence with our last security assumption.

6. SECURITY ANALYSIS

6.1 Quorum Selection

The optimal selection of L_Q and Δq is dependent on both security and performance analysis; our security analysis introduces a lower bound on both L_Q and Δq. For the following evaluations, we feel it safe to discard threats that have probabilities at or below $\frac{1}{2^{128}} \approx 10^{-38.532}$ — the probability of Eve randomly guessing a 128-bit AES key, a threat that would violate our security assumptions.

Our security analysis assumes that L_Q will be selected from a pool of 5,400 Quorum Candidates — the number, as of April 2015, of Tor routers with the Fast and Stable flags, whom we assume are all up-to-date Mirrors. Let L_E be the number of Quorum nodes under Eve's control. Then Eve controls the Quorum if the L_E routers become the largest agreeing subset in the Quorum, which can occur if either more than $\frac{L_Q - L_E}{2}$ honest Quorum nodes disagree or if $L_E > \frac{L_Q}{2}$. The second scenario can be statistically modelled.

Quorum selection may be considered as an L_Q-sized random sample taken from an N-sized population without replacement, where the population contains a subset of f_E entities that we assume are compromised and colluding. Then the probability that Eve controls k Tor routers in the Quorum is given by the hypergeometric distribution, whose probability mass function (PMF) is $\frac{\binom{f_E}{k}\binom{N-f_E}{L_Q-k}}{\binom{N}{L_Q}}$. Then the probability that $L_E > \frac{L_Q}{2}$ is given by $\sum_{x=\lceil \frac{L_Q}{2} \rceil}^{L_Q} \frac{\binom{f_E}{k}\binom{N-f_E}{L_Q-k}}{\binom{N}{L_Q}}$. Odd choices for L_Q prevents the possibility of network disruption when the Quorum is evenly split in terms of the current Page. We examine the probability of Eve's success for increasing amounts of f_E in Figure 7.

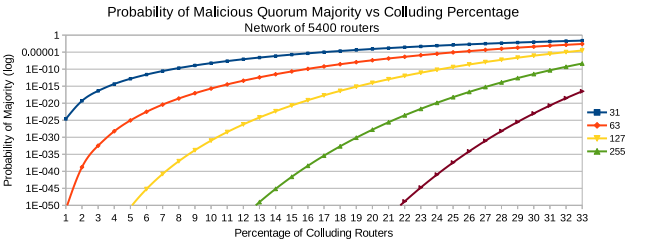

Figure 7: The probability that Eve controls the majority of the Quorum is given by the PMF of the hypergeometric distribution. We fix N at 5,400 nodes and graph Eve's success probability as a function of an increasing percentage of Eve-controlled colluding routers. We examine five selections for L_Q: 31, 63, 127, 255, and 511. We do not consider percentages beyond 33 percent as 33 percent represents a complete compromise of the Tor network: it is near 100 percent that the three routers selected during circuit construction are under Eve's control, a violation of our security assumptions.

Figure 7 shows that a choice of $L_Q = 31$ is suboptimal: the probabilities are above the $10^{-38.532}$ threshold for even small levels of collusion. $L_Q = 63$ likewise fails with approximately two percent collusion, although choices of 127, 255, and 511 fail at levels above approximately 8, 16, and 25 percent, respectively. The figure also suggests that larger Quorums are superior with respect to security. Small Quorums are also less resilient to DDOS attacks at the Quorum in general.

If we assume that Eve controls 10 percent of the Tor network, then we can examine the impact of the longevities of Quorums; over a fixed period of time, slower rotations suggests a lower cumulative chance of selecting any malicious Quorum. If w is Eve's chance of compromise, then her cumulative chances of compromising any Quorum is given by

$1-(1-2)^t$. This gives us a bound estimate on Δq. We estimate this over 10 years in Figure 8.

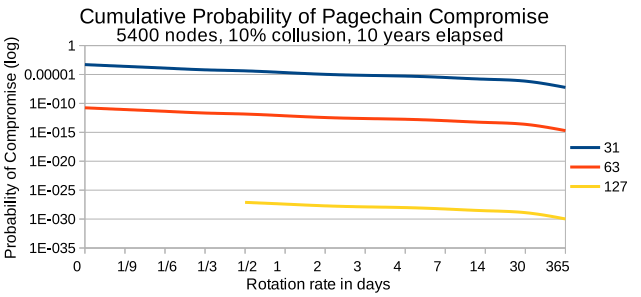

Figure 8: The cumulative probability that Eve controls any Quorum at different rotation rates. We assume 10 percent collusion in a network of 5400 Tor routers, and view across 10 years. We do not graph L_Q values of 255 or 511 as they generate probabilities far below our $10^{-38.532}$ threshold; $L_Q = 255$ and $L_Q = 511$ produce values less than 10^{-58} and 10^{-134}, respectively.

Figure 8 suggests that while slow rotations (i.e a period of 7 days) generates orders of magnitude less chance than fast rotations, the choice of L_Q is far more significant. Like Figure 7, it also shows that $L_Q = 31$ and $L_Q = 61$ are relatively poor choices.

If a selected Quorum is malicious, fast rotation rates will minimize the duration of any disruptions, as shown in Figure **??**. This figure suggests that fast rotations are optimal in that respect, a contradiction to Figure 7. However, given the very low statistical likelihood of selecting a malicious Quorum, we consider this a minor contribution to the decision.

Although a malicious Quorum would have the capabilities to deploy a variety of attacks on the network, the proper selections of $L_Q \geq 127$ and $\Delta q \geq 1$ reduces the likelihood of this occurring to near-zero probabilities. We consider this a stronger solution than introducing countermeasures to those attacks. Based on our security analysis, we suggest $L_Q \geq 127$ and $\Delta q \geq 1$. However, the networking and performance load scales linearly with Quorum size. Based on this balance and our above analysis, we suggest 127 or 255 for values of L_Q and 7 or 14 for Δq.

6.2 Consensus Entropy

The *cached-microdesc-consensus* document describes the network status and is our main source of entropy. In the header, the *vote-digest* header is the hash of a directory authority's status vote. Each directory operates independently and there are nine directory authorities, so we consider this value unpredictable. Router descriptors follow the header in the body of the document.

The r field in a router's descriptor contains routing information and time of last restart. Tor routers have no guarantee of availability and routers may restart for a variety of reasons. As of April 2015 Tor's network consists of approximately 7,000 routers [20] and assuming that a given router restarts every 60 days, statistically the number of unpredictable r fields each day is given by $\frac{7000}{60} \approx 116.66$. Tor displays the restart time down to the second, so each restart adds approximately six bytes of entropy, for an estimated total of $\frac{7000*6}{60} = 700$ bytes of short-term entropy from the r field.

The v field describes the version of Tor that the router is running. Although the versions of Tor are publicly known and the official and unofficial Linux repositories are publicly accessible, Eve cannot predict when an administrator will upgrade their router to the next available Tor version. Although new versions of Tor are not frequently released and routers are upgraded infrequently as shown in Figure **??**, the v field still introduces a degree of medium-term entropy into the document.

The w field contains the router's estimated bandwidth capacity as calculated by bandwidth authorities. Clients use this field during circuit construction; routers with a higher bandwidth capacity relative to the rest of the network have a higher probability of being included in a circuit. Although it is likely that a given router will have similar bandwidth measurements between consecutive consensus documents, Eve cannot predict the exact performance of a router from the perspective of the bandwidth authority. Eve's capacity to predict the performance of all routers falls outside of Tor's and OnioNS' security assumptions; Eve must therefore be a global attacker who would be capable of compromising the Tor network as a whole anyway.

We note that significant amounts of additional entropy could be trivially added into *cached-microdesc-consensus* if each router added a single random byte to their descriptor or if the directory authorities each contributed entropy.

6.3 Outsourcing Record Generation

We designed the Record Generation protocol with the objective of requiring the hidden service operator to also perform the scrypt proof-of-work. However, our protocol does not entirely prevent the operator from outsourcing the computation to secondary resource in all cases.

Let Bob be the hidden service operator, and let Craig be a secondary computational resource. We assume that Craig does not have Bob's private key. Then,

1. Bob creates an initial Record R and completes the *type*, *nameList*, *contact*, *timestamp*, and *consensusHash* fields.
2. Bob sends R to Craig.
3. Let *central* be *type* $\|$ *nameList* $\|$ *contact* $\|$ *timestamp* $\|$ *consensusHash* $\|$ *nonce*.
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 PoW(*central*).
   (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 *recordSig* to $S_d(m, r)$ where $m =$ *central*$\|$ *pow* and r is Bob's private RSA key.
   (c) Bob has found a valid record if $H($*central* $\|$ *pow* $\|$ *recordSig*$) \leq 2^{difficulty*count}$

Our protocol ensures that Craig must always compute more scrypt iterations than necessary; Craig cannot generate *recordSig* 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 places a lower bound on the cost paid by Bob.

6.4 DNS Leakage

Accidental leakage of .tor lookups over the Internet DNS via human mistakes or misconfigured software is one vector that can 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.[**?**] 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 and we offer no defence, but we highlight the potential for this attack.

7. IMPLEMENTATION

Alongside this publication, we provide a reference implementation of OnioNS. We utilize C++11, the Botan cryptographic library, the Standard Template Library's (STL) implementation of Mersenne Twister, and the libjsoncpp-dev library for JSON encoding. We develop in Linux Mint and compile for Ubuntu Vivid, Utopic, Trusty, and their derivatives. Our software is built on Canonical's Launchpad online build system and is available online at https://github.com/Jesse-V/OnioNS.

We have developed an OnioNS prototype that implements the Domain Query protocol. In this initial prototype, we use a static server, a single fixed Create Record, and a hidden service that we deployed at onions55e7yam27n.onion. Although our implementation is primarily a separate software package, we made necessary modifications to the Tor client software to intercept the .tor pseudo-TLD, pass the domain to the OnioNS client over inter-process communication (IPC), and receive and lookup the returned hidden service address.

All textual databases are encoded in JSON. JSON is significantly more compact than XML, but retains readability. Its support of basic primitive types is highly applicable to our needs. Additionally, we consider the JSON format safer than byte-level encoding.

7.1 Analysis

We created a rudimentary form of the Record Generation and Client Verification protocols and conducted performance measurements on two different machines. We tested these protocols on Machine A, which has an Intel Core2 Quad Q9000 @ 2.00 GHz; and machine B, which has an Intel i7-2600K @ 4.3 GHz. We chose machines with this hardware in order to more accurately determine the performance with out target audience; Machines A and B represent low-end and medium-end consumer-grade computers, respectively. To minimize latency on our end, both machines were hosted on 1 Gbits connections at Utah State University. As the Record Generation protocol requires a hidden service private key, we created a hidden service and hosted it on Machine B, using Shallot, a vanity key generator, to generate a recognizable hidden service address, http://onions55e7yam27n.onion. Then on Machine A, we measured the time required to connect to our hidden service over OnioNS and over the more direct hidden service protocol.

We selected the parameters of scrypt such that it consumed 128 MB of RAM during operation. This is an affordable amount of RAM by even low-end consumer-grade machines. This RAM consumption scales linearly with the number of scrypt instances executed in parallel. We used all eight cores on Machine B to generate a Record for our hidden service and observed approximately 1 GB of RAM consumption, matching our expectations. We set the difficulty of the Create Record such that the Record Generation protocol took only a few minutes on average to conduct, but in the future we may change it so that the protocol takes several hours instead.

We measured the CPU wall time required for different parts of client-side protocols. We measured how long it takes the client to build a Record from a JSON-formatted textual string, which involves parsing and assembly of the various fields; the time to check the proof-of-work PoW, and the time to check the *recordSig* digital signature.

Description	A Time (ms)	B Time (ms)	Samp
Parsing JSON into a Record	0.052	0.0238	10
Scrypt check	896.369	589.926	25
Check of $S_d(m, r)$ RSA signature	0.06304	0.0267	20

Machine B, as expected, performed much faster than Machine A at all of these tasks. Parsing and signature checks both took trivial time, though the total time was dominated by the single iteration of scrypt. Record Validation protocol is single threaded and consumes 128 MB of RAM due to scrypt.

We compared the load latency between an OnioNS domain name with a traditional hidden service address. Our tests measured the time between when a user entered in "example.tor" into the Tor Browser to the time when the browser first began to load our hidden service webpage. We also tested http://onions55e7yam27n.onion, the destination of "example.tor". We performed our experiment 15 times with a different client-side Tor circuit for each by restarting Tor at each iteration. To prevent browser-side caching, we restarted the Tor Browser between tests as well.

Lookup	Fastest Time	Slowest Time	Mean Time
.tor	6.1	8.5	7.1
.onion	9.3	12.2	10.2

The latency is circuit-dependent and heavily depends on the speed of each Tor router and the distance between them. To avoid the latency cost whenever possible, we implemented a DNS cache into the OnioNS client-side software to allow subsequent queries to be resolved locally, avoiding unnecessary remote lookups.

8. FUTURE WORK

However, Merkle trees do not support updates without complete rebuilding. Future work may explore more efficient data structures such as the hash tables proposed by

Papamanthou et el.[18], which is noteworthy for achieving constant query cost and sublinear update cost.

In future work we will expand our implementation of OnioNS and develop the remaining protocols. While our implementation functions with a fixed resolver, we will deploy our implementation onto larger and more realistic simulation environments such as Chutney and PlanetLab. When we have completed dynamic functionality and the remaining protocols, we will pursuit integrating our implementation into Tor. We expect this to be straightforward as OnioNS is designed as a plugin for Tor, introduces no changes to Tor's hidden service protocol, and requires very few changes to Tor's software. In future developments, Tor's developers may also make significant changes to Tor's hidden services, but OnioNS' design enables our system to become forwards-compatible after a few minor changes.

Additionally, several questions need to answered in future studies:

We support the same character set as the Internet DNS but disallow the digits zero and one (similar to base32 encoding) in order to reduce the threat of phishing attacks via spoofed domains with indistinguishable characters. In future work we will explore implementing Punycode to provide support for international character sets.

- Should the Quorum expire registrations that point to non-existent hidden services, and if so, how can this be done securely?
- How can we reduce vulnerability to phishing/spoofing attacks? Can OnioNS be adapted to include a privacy-enhanced reputation system?
- How can OnioNS support domain names with international encodings? A naïve approach to this is to simply support UTF-16, though care must also be taken to prevent phishing attacks by domain names that use Unicode characters that visually appear very similar.
- What other networks can OnioNS apply to? We require a fully-connected networked and an global source of entropy. We encourage the community to adapt our work to other systems that fit these requirements.

9. CONCLUSION

We have introduced OnioNS, a Tor-powered distributed DNS that maps unique .tor domain names to traditional Tor .onion addresses. It enables hidden service operators to select a human-meaningful domain name and provide access to their service through that domain. We preserve the privacy and anonymity of both parties during registration, maintenance, and lookup, and furthermore allow Tor clients to verify the authenticity of domain names. Moreover, we rely heavily upon existing Tor infrastructure, which simplifies our design assumptions and narrows our threat model largely to attack vectors already well-understood throughout the Tor literature.

We use the Pagechain distributed data structure to prevent disagreements from forming within the network. Furthermore, every participant can verify the uniqueness of domain names. The Pagechain also has a fixed maximal length, which places an upper bound on the networking, computational, and storage requirements for all participants, a valuable efficiency gain especially noticeable long-term.

OnioNS achieves all three properties of Zooko's Triangle: it is distributed, allows hidden service operators to select meaningful domain names, and all parties can confirm for themselves the uniqueness of domain names in the database. We provide a reference implementation in C++ that should enable Tor developers to deploy OnioNS into the Tor network with minimal effort. We believe that OnioNS will be a useful abstraction layer that will significantly enhance the usability and the popularity of Tor hidden services.

10. ACKNOWLEDGEMENTS

We thank Mark Ellzey and Tor developers Yawning Angel and Nick Mathewson for their assistance with libevent and Tor technical support.

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

[8] 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.

[9] I. Goldberg. On the security of the tor authentication protocol. In *Privacy Enhancing Technologies*, pages 316–331. Springer, 2006.

[10] 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.

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

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

[13] 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.

[14] R. C. Merkle. A digital signature based on a conventional encryption function. In *Advances in*

Cryptology-CRYPTO'87, pages 369–378. Springer, 1988.

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

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

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

[18] C. Papamanthou, R. Tamassia, and N. Triandopoulos. Authenticated hash tables. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 437–448. ACM, 2008.

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

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

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

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

[23] 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.