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.

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 neither convenient nor does it scale well. Given the approx-

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

DOI: 10.475/123_4

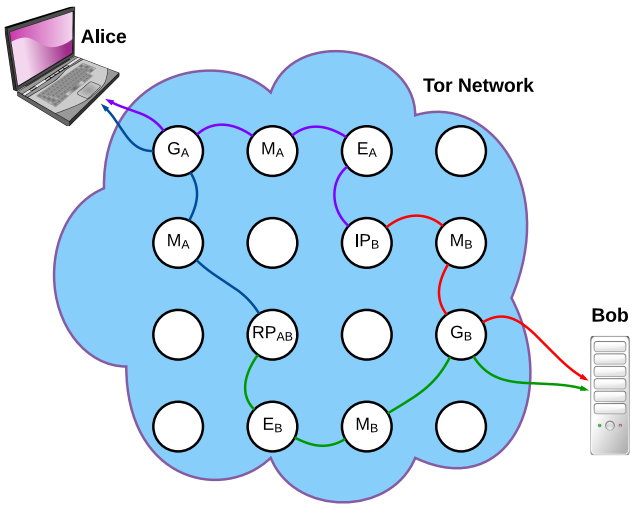

Figure 1: A Tor client, Alice, and a hidden service, Bob, communicate by mating two Tor circuits at a *rendezvous point* (RP). Bob maintains circuits (red) to his *introduction points* (IPs) and advertises his IPs and PK_B to the Tor network. When Alice obtains Bob's address through a backchannel, she 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]

imetly 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

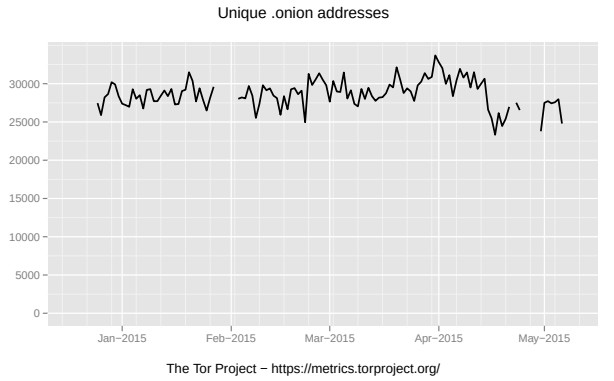

Figure 2: The number of unique .onion addresses seen in the Tor network between December 2014 and May 2015.[11][20]

and 6 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 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.

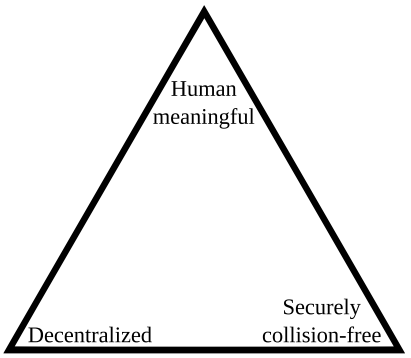

Figure 3: Zooko's Triangle.

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.[24] 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[24] (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.

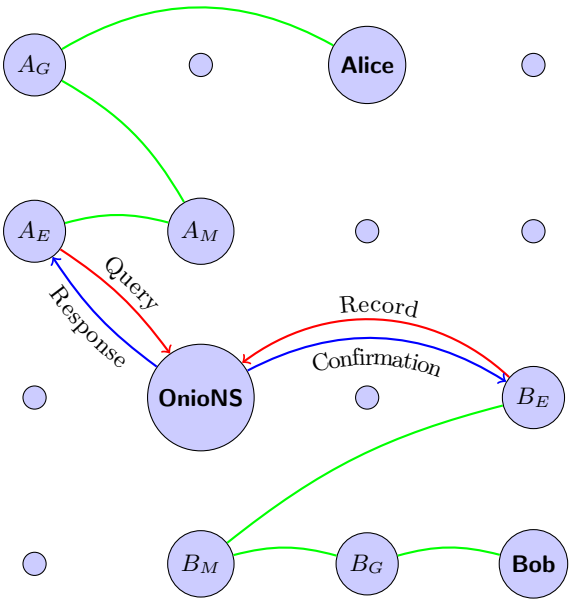

Figure 4: Bob uses a Tor circuit (B_G, B_M, B_E) to anonymously broadcast a record to OnioNS. Alice uses her own Tor circuit (A_G, A_M, A_E) to query the system for a domain name, and she is given Bob's record in response. Then Alice connects to Bob by Tor's hidden service protocol.

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 $S_{RSA}(m, r)$ where $m = nameList \parallel timestamp \parallel consensusHash \parallel nonce \parallel pow$ 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 $S_{ed}(originTime \parallel recentRecords \parallel fingerprint], e)$ where e is the router's private Ed25519 key.

Page
: A *Page* contains *prevHash*, the output of $H(prevHash \parallel recordList \parallel consensusHash)$ 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 $S_{ed}(H(prevHash \parallel recordList \parallel consensusHash)], e)$ 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 maintains its own Page, which has a lifetime of that Quorum. The Quorum is randomly chosen from Quorum Candidates.

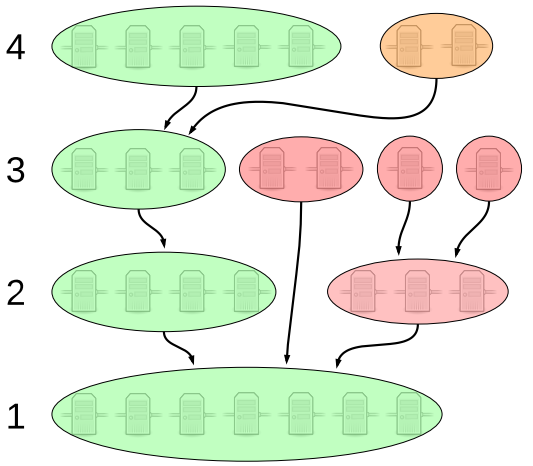

Figure 5: An example Pagechain across four Quorums with three side-chains. The valid master Pagechain from honest Quorum nodes (green) resists corruption from maliciously-colluding nodes (red) and malfunctioning nodes (orange).

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.

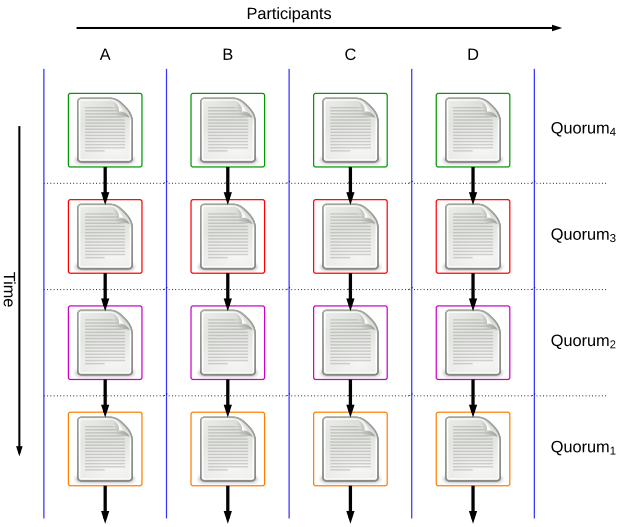

Figure 6: The master Pagechain is one-dimensional but spans across the network as each Mirror holds a copy. Each Page is maintained by a respective Quorum.

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]

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 $S_{RSA}(m, r)$ 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 $H(central \parallel pow \parallel recordSig) \leq 2^{d*c}$ 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 $h = H(prevHash \parallel recordList \parallel consensusHash)$.
   (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 $consensusHash = H(cd)$.
   (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 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.

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 $h = H(prevHash \parallel recordList \parallel consensusHash)$, 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 $\mathcal{O}(\text{size}(T)\Delta T)$. 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 last security assumption.

6. SECURITY ANALYSIS

6.1 Quorum Selection

In section 4, 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.

6.1.1 *Quorum Size*

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.

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, as of May 2015, of Tor routers with the Fast and Stable flags. We provide the statistical calculations in Figure 7.

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

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

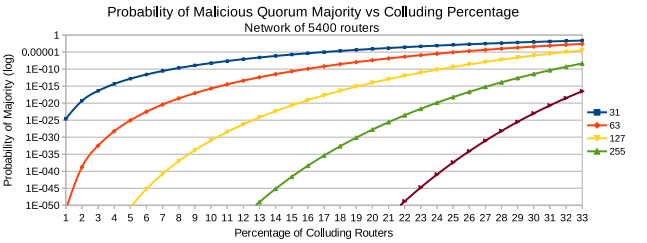

Figure 7: We calculate the PMF of the hypergeometric distribution for a population of 5,400, $f_E \in [1, 33]$, and five selections for L_Q: 31, 63, 127, 255, and 511. $f_E > 33$ represents a complete compromise of the Tor network, so we do not consider values beyond this range.

Figure 7 shows that choices of $L_Q = 31$ and $L_Q = 63$ 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 one percent collusion, $L_Q = 63$ above two percent, and $L_Q = 127$, $L_Q = 255$, and $L_Q = 511$ at ≥ 7.7, ≥ 16.1, and ≥ 24.6 percent, respectively. Therefore we recommend setting $L_Q \geq 127$.

6.1.2 *Quorum Rotation*

In section 4, 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 pe-

riod 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 11.

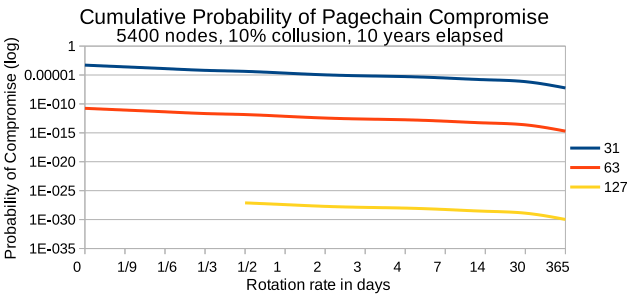

Figure 8: The cumulative probability that Eve controls any Quorum at different rotation rates over 10 years at $f_E = 10$. We do not consider $L_Q = 255$ or $L_Q = 511$ as they produce values less than 10^{-58} and 10^{-134}, respectively, far below our $10^{-38.532}$ threshold.

Figure 11 suggests that although lower values of Δq negatively impact security, the choice of L_Q is more significant. It also supports our earlier conclusion that the choices of $L_Q = 31$ and $L_Q = 63$ are suboptimal. Based on Figure 11 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.

6.1.3 Network Consensus

Periodically, Tor routers upload signed *descriptors* — routing information, cryptographic keys, and other information — to Tor's directory authorities (dirauths). Once per hour, the dirauths aggregate and republish the descriptors back to the network, enabling Tor's network to be dynamic and distributing new information to all parties in an efficient and timely manner. We use Tor's network consensus as a global source of entropy, enabling the entire network to agree on a common Quorum. While Tor provides no guarantee that the network is using the same consensus at the same time, the consensus is timestamped, so we can reference it by their time of publication. We focus on two essential documents that clients assemble from the consensus: *cached-certs* and *cached-microdesc-consensus*. Although these documents contain other elements, for the sake of brevity we briefly describe their essential components.

- **cached-certs** A list of long-term identity and short-term signing RSA keys from each dirauth.
- **cached-microdesc-consensus** Essential information about each router, such as networking information and capabilities.

If cd is $cached\text{–}certs \parallel cached\text{–}microdesc\text{–}consensus$, we initialize $R(s)$ with the output of the $H(cd)$. Eve may desire to find a hash k that generates a desirable Quorum, either one that includes her routers or excludes particular honest routers. As $H(x)$ is SHA-384, SHA-384's strong resistance to preimage attack forces Eve to spend $L_T * 2^{192}$ operations on average to find k. Eve may also try to manipulate her router's descriptors such that $H(cd_{q+1}) = k$, but SHA-384's resistance to second-preimage attacks also requires this to require $L_T * 2^{192}$ operations as well. However, if cd contains sufficient entropy, Eve is unable to predict the next Quorum or influence it to her favor. Since the dirauths publish a new consensus every hour, we must analyse the entropy rate between consecutive hours.

We constructed a first-order Markov model and estimated a lower bound on the entropy of the *cached-microdesc-consensus* document by analysing the transitions for various fields between consecutive consensuses. We also provide analysis on the dynamics of the Tor network by counting routers that enter or leave the Tor network between consecutive consensuses in Figure 9. Here we identity routers by their identity keys; just as Tor does, we consider routers that change their identity keys as two different routers; one leaving and one entering the network. For routers that are present between consecutive pairs, we focus on six critical fields from each router's descriptor inside *cached-microdesc-consensus*: *nickname*, the router's name chosen by its operator or a default name; *publication*, the time when it last published a descriptor; *IP address*, its network address; *ORPort*, the network port for onion routing; *version*, the version of the Tor protocol that this relay is running; and *bandwidth*, as self-reported or as measured by the dirauths. Tor routers change *publication* when either 18 hours has elapsed since the last descriptor publication, its fields have changed, or if its uptime has been reset.

We obtained from collector.torproject.org 5,067 archived versions of *cached-microdesc-consensus* across a seven-month period between September 1, 2014 00:00 GMT and March 31, 2015 23:00 GMT. We observed 21 instances where the dirauths did not publish an hourly consensus due to a race condition bug in Tor's software. Until Tor patches this bug, in the rare event that the Quorum Formation protocol references a missing consensus$_i$, consensus$_{i+1}$ may be used instead. Therefore our analysis includes transitions across these missing consensuses. We construct a Markov model for the six aforementioned fields and illustrate the number of observed transitions for each field in Figure 10.

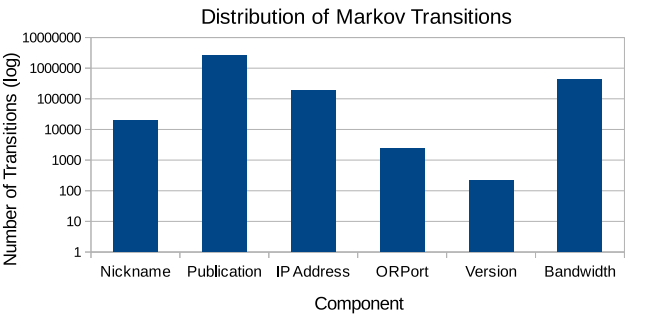

Figure 9: A histogram of the number of routers entering or leaving the network between consecutive consensus across the seven-month period.

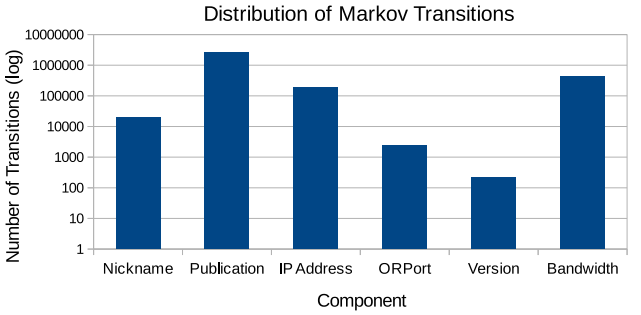

Figure 10: The number of observed transitions for *nickname*, *publication*, *IP address*, *ORPort*, *version*, and *bandwidth* between consecutive consensuses across the seven-month period.

$$\mathrm{H}(\mathcal{S}) = -\sum_i P_i \sum_j P_i(j) \log_2 P_i(j) \tag{3}$$

6.2 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,

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 $C_{0 \le j \le K}$ to Bob.
6. For each Record $C_{0 \le j \le K}$ Bob computes
   (a) Bob sets *pubHSKey* to his public RSA key.
   (b) Bob sets *recordSig* to $S_{RSA}(m, r)$ where $m =$ *central*$\|$ *pow* and r is Bob's private RSA key.
   (c) Bob has found a valid Record if $H($*central* $\|$ *pow* $\|$ *recordSig*$) \le 2^{d*c}$

However, this 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 always places a cost on Bob.

6.3 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.[23] 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.

7. IMPLEMENTATION

We have built a partial reference implementation of OnioNS in C++11, primarily the Record Generation and Domain Query protocols. Botan provides the implementation of our cryptographic operations, Boost Asio provides our networking engine, and Linux libjsoncpp-dev library 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.

We implemented the essential functionality for Bob, Charlie, and Alice. We used the Shallot[12] vanity key generator to find an RSA key that hashed to a hidden service address with the "OnioNS" prefix. Shallot found "onions55e7yam27n.onion". We constructed a website and attached Tor to the nginx web server. Then we loaded the private key into OnioNS and constructed a Record containing an association between "example.tor" and "onions55e7yam27n.onion". Next, we transmit this Record to a fixed server over a Tor circuit and started a TCP server to respond to Domain Queries. Finally, we modified Tor such that when Alice types "example.tor" into the Tor Browser, Tor intercepts the pseudo-TLD and sends it via IPC to the OnioNS client. Then the OnioNS client performs a Domain Query to Charlie and writes "onions55e7yam27n.onion" back to the Tor client over IPC. Finally, Tor rewrites the original "example.tor" lookup to "onions55e7yam27n.onion" and contacts our hidden service over the traditional Tor protocol. Tor performs asynchronously here; it must place the lookup on hold, free its event loop to avoid a deadlock, and asynchronously resume the lookup once it receives the response. From the user's perspective, the content of http://onions55e7yam27n.onion loads transparently under the http://example.tor URL.

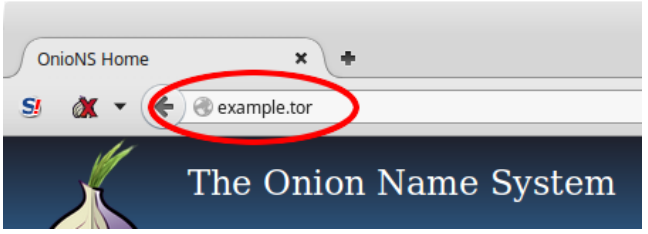

Figure 11: We load the OnioNS' hidden service, onions55e7yam27n.onion, transparently under the "example.tor" domain on Tor Browser 4.5.1, a fork of Firefox ESR 31.7.0.

7.1 Results

We conducted several performance measurements for the Record Generation and Domain Query protocols. 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 archicture) @ 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 hosted our hidden service and TCP server on Machine B.

7.1.1 *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 measured the average CPU wall-time required for both machines to validate the Record.

Description	A (ms)	B (ms)	Samples
Parsing JSON	0.052	0.024	100
Validating scrypt	896.369	589.926	25
$V_{RSA}(m, E)$	0.063	0.027	200
Total Time	901.135	595.643	25

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.

7.1.2 *Latency*

We measure the latency introduced by the Domain Query protocol and compare against loading a hidden service via its traditional .onion addresses. Our experiment measured the time between when Tor received "example.tor" from the Tor Browser and when it first began loading our hidden service. We conducted 25 measurements. We restarted Tor between tests in order to utilize different circuits and also restarted the Tor Browser to flush browser-side caching.

Lookup	Fastest (s)	Slowest (s)	Mean (s)
.onion	6.1	8.5	7.1
.tor	8.5	11.4	9.7

Tor's latency is primarily dominated by circuit construction and negotiation with the hidden service. This time is highly dependent on the circuit's network distance and the speed of each Tor router. Our measurements include the time to build circuits to our TCP server and the Domain Query protocol. which includes a single call to scrypt and $V_{RSA}(m, E)$. We avoid additional latency costs by implementing a DNS cache in order to allow subsequent queries to be resolved locally. In future work we will reduce the latency by recycling existing circuits that Tor constructs on startup and by moving the server to the circuit's exit router, eliminating an unnecessary network hop.

8. FUTURE WORK

In future work we will expand our implementation and pursuit integrating it into Tor. We anticipate this to be straightforward as OnioNS requires few changes to Tor, namely a new .tor pseudo-TLD and Ed25519 router keys, and 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. For example, if hidden services migrated from RSA to Ed25519, OnioNS need only change $S_{RSA}(m, r)$ and $V_{RSA}(m, E)$ to Ed25519.

We use a Merkle tree for authenticated denial-of-existence. Merkle trees with sorted leaves do not support updates, insertions, or deletions without complete rebuilding. In future work we will pursuit more efficient and dynamic data structures such as the hash tables proposed by Papamanthou et el.[18], which are noteworthy for achieving constant query cost and sublinear update cost.

Our implementation currently only supports ASCII characters in domain names. In future work we will explore implementing Punycode to provide support for international character sets. However, 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.

9. CONCLUSION

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.

10. 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. 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.

[24] 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.