The Onion Name System

Tor-powered Distributed DNS for Tor Hidden Services

Jesse Victors
7449 S. Babcock Blvd.
Wasilla, AK 99623
jvictors@jessevictors.com

Ming Li
Utah State University
Logan, UT 84321
ming.li@usu.edu

ABSTRACT

Tor is a third-generation low-latency onion router that provides its users with online privacy, anonymity, and resistance to traffic analysis. In recent years its userbase, network, and community has grown significantly in response to revelations of international electronic surveillance, and it remains one of the most popular anonymity networks in use today. Tor also provides access to anonymous servers known as hidden services – servers of unknown location and ownership who achieve anonymity through Tor circuits. These hidden services can be accessed through any Tor-enabled web browser but they suffer from usability challenges due to the algorithmic generation of their 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 self-authenticating domain name chosen by the hidden service operator. We introduce a new distributed self-healing database 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

ACM CCS '14 October 12-16, 2015, Denver, Colorado, USA

DOI: 10.475/123_4

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. Privacy is not achieved in traditional Internet connections because SSL/TLS encryption cannot hide IP and TCP headers, which must be exposed to allow routing between two parties; eavesdroppers can easily break user privacy by monitoring these headers.[16] 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.

Today, most anonymity tools descend from mixnets. Mixnets have inspired the development of many varied mixnet-like protocols and have generated significant literature within the field of network security.[8] Mixnet descendants can generally be classified into two distinct categories: high-latency and low-latency systems. High-latency networks typically delay traffic packets, increasing their resistance to global adversaries who monitor communication entering and exiting the network. By contrast, low-latency networks do not delay packets and are thus better suited for common Internet activities such as web browsing, instant messaging, or the prompt transmission of email.[7] In this work, we detail and introduce new functionality within low-latency protocols.

Onion routing is the most popular low-latency descendant of mixnets in use today. In onion routing, a user selects a set (typically three) of network nodes, typically called *onion routers* and together a *circuit*, and encrypts the message with the public key of each router. Each encryption layer contains the next destination for the message – the last layer contains the message's final destination. As the *cell* containing the message travels through the network, each of these onion routers in turn decrypt their encryption layer, exposing their share of the routing information. The final recipient receives the message from the last router, but is never exposed to the message's source. The sender therefore has privacy because the recipient does not know the sender's location, and the sender has anonymity if no identifiable or distinguishing information is included in their message.

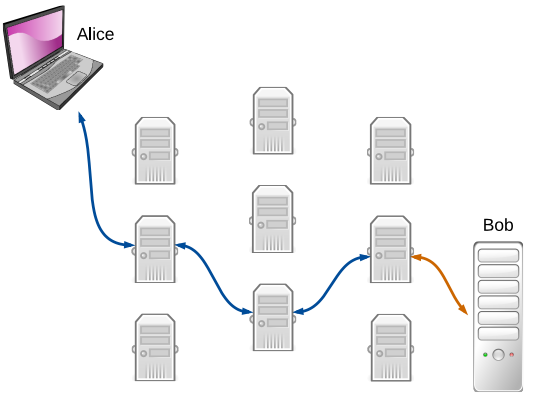

Figure 1: Alice communicates privately to Bob through a Tor circuit. Her communication path consists of three routers, an entry, middle, and exit. Each encrypted Tor link is shown in blue, the final connection from the exit to Bob, shown in orange, is optionally encrypted.

1.1 Tor

Tor[7] is a third-generation onion routing system and is the most popular onion router in use today. Tor's threat model assumes that the capabilities of adversaries are limited to traffic analysis attacks; 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.

Tor assumes a dynamic network topology and introduced a small set of semi-trusted *directory authority* servers to distribute network information and as a form of PKI. Periodically, Tor routers upload digitally-signed *descriptors* – containing routing information, cryptographic keys, bandwidth history, and other information – to these authorities. Once an hour, the directory authorities aggregate, sign, and republish the descriptors as a *network status consensus*. Clients download essential portions of these descriptors from the directory authorities or from known routers redistributing the consensus, forming *microdescriptors*. These are aggregated into three files, *cached-certs*, containing each directory authority's long-term identity RSA key, medium-term signing key chained to the identity key, and validity timeline for these keys; *cached-microdescs*, containing public RSA and Curve25519[3] keys (used for TAP[10] and NTor[11] circuit construction protocols, respectively); and *cached-microdesc-consensus*, containing all microdescriptors. Although Tor provides no guarantee that the entire network has the same consensus at the same time, the consensus can be verified retrospectively, allowing the consensus to be referenced by its time of publication.

1.2 Hidden Services

Tor also supports *hidden services* – anonymous servers that intentionally mask their IP address through Tor circuits and cannot normally be accessed outside the context of Tor. Tor hidden services provide bidirectional anonymity where both parties remain anonymous and never directly communicate with one another. Hidden services are only known by their public key and accessed in any Tor-enabled browser by their 16-character base32-encoded address with the .onion pseudo-TLD; the address is the first 16 bytes of the base32-encoded SHA-1 hash of the server's RSA key. This builds a publicly-confirmable one-to-one relationship between the public key and its address and allows hidden services to be referenced by their address in a distributed environment.

Throughout this work, let Bob be a hidden service and Alice a Tor client. At startup, Bob randomly select several Tor routers and construct Tor circuits to them. He then creates a hidden service descriptor, consisting of his public key B_K and a list of these routers. He signs the descriptor and sends a distributed hashtable within the Tor network, enabling the routers he chose to act as his *introduction points*. When Alice obtains Bob's hidden service address though a backchannel, she queries this hashtable for Bob's descriptor. Alice can confirm that the hash of B_K matches her original address. Alice then builds a circuit to one of the introduction points and simultaneously also selects and builds a circuit to another relay, R_A. She encrypts R_A and a nonce with B_K and gives the result to R_A to send to Bob. Bob decrypts the message and builds a circuit to R_A and sends the nonce to Alice. This confirms Bob's authenticity to Alice and the two begin communication over six Tor nodes: three established by Alice and three by Bob.[19]

Tor hidden service addresses are distributed and globally collision-free, but there is a strong disconnect 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. This suggests the strong need for a more complete and reliable solution.

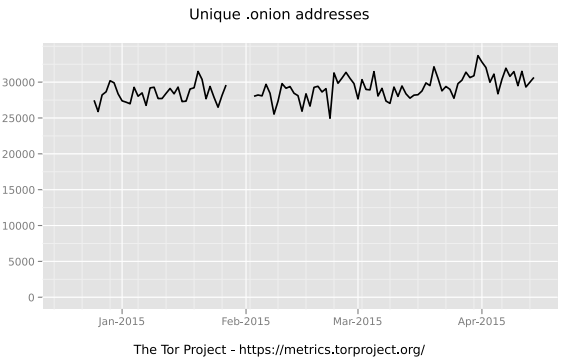

Figure 2: The number of unique .onion addresses seen in Tor's distributed hashtable between January through April 2015.[20][13]

2. DESIGN OBJECTIVES

Tor's privacy-enhanced environment introduces a distinct set of challenges that must be met by any additional functionality. Here we enumerate a list of requirements that must be met by any DNS applicable to Tor hidden services. In Section 3 we analyse several existing prominent naming

systems and show how these systems do not meet these requirements, and in Section 6 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. **Clients must be able to authenticate registrations.** Clients must be able to verify that the domain-destination pairing that they receive from name servers is authentic relative to the authenticity of the destination server.
4. **Domain names must be globally unique.** Any domain name of global scope must point to at most one server. In the case of naming systems that generate names via cryptographic hashes, the domain name key-space must be of sufficient length to be remain resistant to at least collision and second pre-image attacks.
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 user privacy or may not allow anonymous registrations. For these reasons, naming systems with central authorities can safely be considered ill-suited for hidden services.
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 must not introduce significant burdens to clients.** 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 demands.

3. EXISTING WORKS

Vanity key generators (e.g. Shallot[14]) 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.[18] Like vanity key generators, Nicolussi's encoding partially improves the recognition and readability of an address but does nothing to alleviate the logistic problems of manually entering in the address into the Tor Browser. The key-space is not changed and is again too large for hidden service operators to select many meaningful words, making these attempts purely cosmetic and not 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[4] 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[6] 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,[2] constructed a distributed peer-to-peer naming system, but like GNS, made no guarantee that domain names would be globally unique.

Namecoin[1][12] was an early fork of Bitcoin[17] 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 be-

ing a practical alternative DNS for Tor hidden service. However, our work shares some design principles with Namecoin.

4. CHALLENGES

4.1 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,[9][21] he claimed any naming system could only achieve two of these properties. This is illustrated in Figure 3.

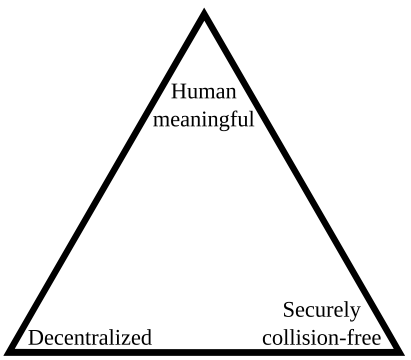

Figure 3: Zooko's Triangle.

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.

Petnames, such as Namecoin and OnioNS, are systems that achieve all three properties of Zooko's Triangle.

4.2 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.

5. ASSUMPTIONS AND THREAT MODEL

1. 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.
2. We assume reliable cryptography; namely that Eve cannot break standard cryptographic primitives such as AES, SHA-2, RSA, Curve25519, Ed25519, the scrypt key derivation function, or Tor protocols derived from these primitives. We assume that Eve maintains no backdoors or knows secret software breaks in the Botan or the OpenSSL implementations of these primitives.
3. 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.
4. 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.
5. If C is a Tor network status consensus, 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.

6. SOLUTION

6.1 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.

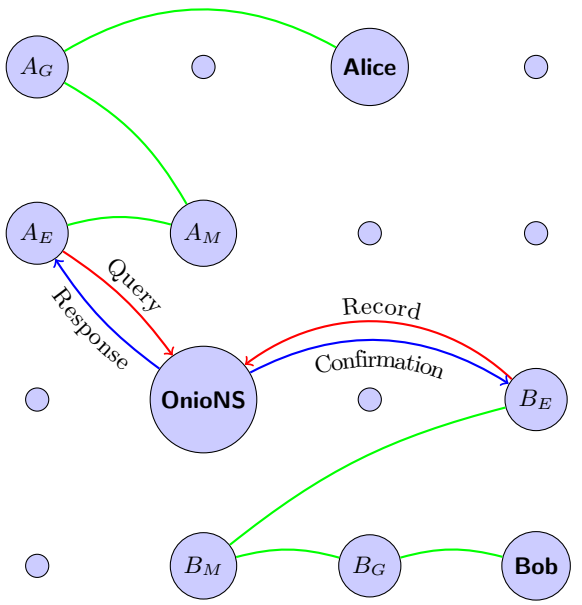

Figure 4: Bob uses a Tor circuit (B_G, B_M, B_E) to anonymously broadcast a record to OnioNS. Alice uses her own Tor circuit (A_G, A_M, A_E) to query the system for a domain name, and she is given Bob's record in response. Then Alice connects to Bob by Tor's hidden service protocol.

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 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.

6.2 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[5] 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 $S_d(m, r)$ use $H(x)$ as a digest function on m in all use cases. In our reference implementation we define $S_d(m, r)$ 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 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. $R(s)$ 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.[15]

6.3 Definitions

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

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 $m = nameList \parallel timestamp \parallel consensusHash \parallel nonce \parallel pow$

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 $S_{ed}(originTime \| recentRecords \| fingerprint], e)$ 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. The primary job of a Mirror is to respond to queries. 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 regular 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.

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.

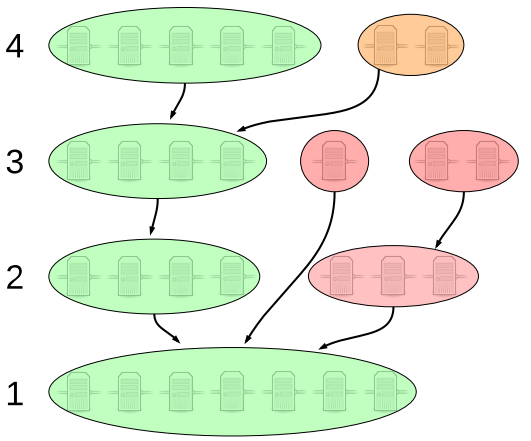

Figure 5: An example Pagechain across four Quorums with three side-chains. The valid master Pagechain from honest Quorum nodes (green) resists corruption from maliciously-colluding nodes (red) and malfunctioning nodes (orange).

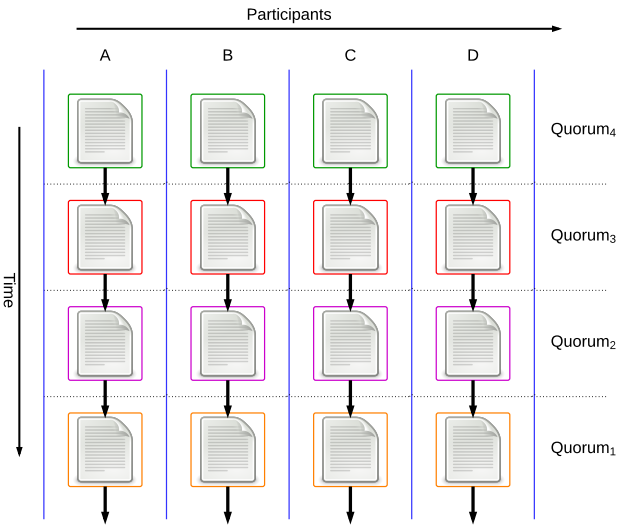

Figure 6: The master Pagechain is one-dimensional but spans across the network as each Mirror holds a copy. Each Page is maintained by a respective Quorum.

6.4 Symbols

- Let L_Q represent size of the Quorum.
- Let L_T represent the number of routers in the Tor network.
- Let L_P represent the maximum number of Pages in the Pagechain.
- Let q be an Quorum iteration counter.
- Let Δq be the lifetime of a Quorum in days: every Δq days q is incremented by one and a new Quorum is chosen.
- Let s be a Snapshot iteration counter.
- Let Δs be the lifetime of a Snapshot in minutes.

6.5 Protocols

We now describe the protocols fundamental to OnioNS functionality.

6.5.1 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 both 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*).

6.5.2 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 from the previous Quorum a Page that is both valid and maintained by the largest number of Quorum members. Mirrors must then verify this selection retrospectively for each Page chronologically in the Pagechain.

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 them.
3. Charlie uses *cd* to calculate the Quorum via the Quorum Derivation protocol.
4. For each Page,
   (a) Charlie checks that *prevHash* references some page from the previous Quorum.
   (b) Charlie calculates $h = H(\textit{prevHash} \parallel \textit{recordList} \parallel \textit{consensusHash})$.
   (c) Charlie checks that *fingerprint* is a member of $Quorum_q$.
   (d) Charlie checks that $V_{cd}(h, E)$ returns true.
5. Charlie sorts the set of Pages by 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 a Page is found valid, *prevHash* must now reference it.

6.5.3 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 Charlie'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]

6.5.4 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.

6.5.5 Quorum Derivation

1. Alice obtains the consensus documents, *cd*, published on day $\lfloor \frac{q}{\Delta q} \rfloor$ at 00:00 GMT.
2. Alice scans *cd* and constructs a list *qc* of Quorum Candidates of Tor routers 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. She can construct *qc* in $\mathcal{O}(L_T)$ time.
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.

6.5.6 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.

1. Alice constructs a Tor circuit to Charlie.
2. If d's highest-level name is "www", Alice removes that name.
3. Alice asks Charlie for the most recent Record r containing d.
4. Charlie searches his local Pagechain and returns r to Alice.
5. Alice checks the validity of r. If r is invalid, Charlie is acting dishonestly.
6. If d in r points to a domain with a .tor pseudo-TLD, d becomes that destination and Alice jumps back to step 2.
7. Since the destination uses a .onion pseudo-TLD, Alice contacts Bob by the traditional hidden service protocol.
8. Alice extracts Bob's key from his hidden service descriptor and verifies that it matches r's *pubHSKey*. Alice throws an assertion error is this is not correct.
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. Since p's *pageSig* is a signature on $H(prevHash \parallel recordList \parallel consensusHash)$ she can also ask for these hashes and verify that p is in the largest set and that p authenticates against multiple Quorum nodes. 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 8 Mbits,[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.

6.5.7 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.

6.6 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 work to authenticate denial-of-existence claims on domains en-masse.

A simple solution to authenticated denial-of-existence is to enumerate all existing domain names, have a trusted party sign the list, and pass the list to clients to prove non-existence. However, as each domain may be up to 128 characters long, this leads to a list up to 3.2 MB in size for the $\approx$ 25,000 hidden services currently advertising on the Tor network, which would take approximately eight seconds to send over a Tor circuit.[20] We utilize a hashtable, a bitset, and an AVL to reduce our space requirement and decrease our verification time to $\mathcal{O}(1)$ on average and $\mathcal{O}(\log(n))$ in the worst case.

Let n be the number of domain names in the Pagechain, s be a space scaling factor, m a security scaling factor, and z a network load scaling factor. Then each Quorum node

1. Constructs and zeros a bitset *bset* of size $L = s * n * (m + 2)$ bits.
2. Constructs an empty array list *arr*.
3. For each domain name d in each Record r in the Pagechain,
   (a) If bit 0 in $bset(H(d))$ is "1", sets bit 1 to "1" and add $H(d)$ to *arr*.
   (b) If bit 0 in $bset(H(d))$ is "0", sets bit 0 to "1" and set the last m bits of the $bset(H(d))$ bucket to the last m bits of $H(d)$.
4. Sorts *arr* in numerical order.
5. Divides *bset* into z sections, signs each section via $S_{ed}(m, e)$, and generates $S_{ed}(arr, e)$.

Then if Alice requests a domain name that Charlie claims does not exist,

1. Charlie returns back the section t of *bset* that contains $bset(H(d))$ and t's signature.
2. Alice verifies that $V_{ed}(t, E)$ returns true.
3. If bit 0 in $bset(H(d))$ is "0", Alice knows d does not exist.
4. If bit 0 is "1", and the last m bits of $bset(H(d))$ match $H(d)$, Alice knows d does not exist.
5. If bit 0 and bit 1 are both "1" and $H(d)$ does not exist in *arr*, Alice knows d does not exist.
6. Otherwise, either Charlie or the Quorum nodes who signed *bset* are dishonest.

Note that a Bloom filter with k hash functions could be used instead of a compact hashtable, but a Bloom filter would require sending up to k sections of buckets to the client. Therefore, we use a simple hashtable scheme, which is effectively a Bloom filter with $k = 1$.

7. REFERENCES

[1] Namecoin. https://namecoin.info/, May 2015.

[2] 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.

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

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

[5] 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.

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

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

[8] M. Edman and B. Yener. On anonymity in an electronic society: A survey of anonymous communication systems. *ACM Computing Surveys (CSUR)*, 42(1):5, 2009.

[9] 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.

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

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

[12] F. Jacobs. Providing better confidentiality and authentication on the internet using namecoin and minimalt. *arXiv preprint arXiv:1407.6453*, 2014.

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

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

[15] 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.

[16] B. Miller, L. Huang, A. D. Joseph, and J. D. Tygar. I know why you went to the clinic: Risks and realization of https traffic analysis. In *Privacy Enhancing Technologies*, pages 143–163. Springer, 2014.

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

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

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

[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.