ONIONS:
TOR-POWERED DISTRIBUTED DNS
FOR ANONYMOUS SERVERS

by

Jesse Victors

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Computer Science

Approved:

Dr. Ming Li Major Professor	Dr. Nicholas Flann Committee Member
Dr. Daniel Watson Committee Member	Dr. Mark R. McLellan Vice President for Research and Dean of the School of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2015

Copyright © Jesse Victors 2015

All Rights Reserved

CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Onion Routing

As the prevalence of the Internet and other communication has grown, so too has the development and usage of privacy-enhancing systems. These are tools and protocols that provide privacy by obfuscating the link between a user's identification 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. A closely related property is anonymity – a part of privacy where user activities cannot be tracked and their communications are indistinguishable from others. Tools that provide these systems hold a user's identity in confidence, and privacy and anonymity are often provided together. Following a general distrust of unsecured Internet communications and in light of the 2013-current revelations by Edward Snowden of Internet mass-surveillance by the NSA, GCHQ, and other members of the Five Eyes, users have increasingly turned to these tools for their own protection. Privacy-enhancing and anonymity tools may also be used by the military, researchers working in sensitive topics, journalists, law enforcement running tip lines, activists and whistleblowers, or individuals in countries with Internet censorship. These users may turn to proxies or VPNs, but these tools often track their users for liability reasons and thus rarely provide anonymity. Furthermore, they can easily voluntarily or be forced to break confidence to destroy user privacy. More complex tools are needed for a stronger guarantee of privacy and anonymity.

Today, most anonymity tools descend from mixnets, an early anonymity system invented by David Chaum in 1981. [1] In a mixnet, user messages are transmitted to one

or more mixes, who each partially decrypt, scramble, delay, and retransmit the messages to other mixes or to the final destination. This enhances privacy by heavily obscuring the correlation between the origin, destination, and contents of the messages. Mixnets have inspired the development of many varied mixnet-like protocols and have generated significant literature within the field of network security. [2] [3]

Mixnet descendants can generally be classified into two distinct categories: high-latency and low-latency systems. High-latency networks typically delay traffic packets and are notable for their greater resistance to global adversaries who monitor communication entering and exiting the network. However, high-latency networks, due to their slow speed, are typically not suitable for common Internet activities such as web browsing, instant messaging, or the prompt transmission of email. Low-latency networks, by contrast, do not delay packets and are thus more suited for these activities, but they are more vulnerable to timing attacks from global adversaries. [4] In this work, we detail and introduce new functionality within low-latency protocols.

Onion routing is a technique for enhancing privacy of TCP-based communication across a network and is the most popular low-latency descendant of mixnets in use today. It was first designed by the U.S. Naval Research Laboratory in 1997 for military applications [5] [6] but has since seen widespread usage. In onion routing with public key infrastructure (PKI), a user selects a set 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 like an onion, 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. [3] 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.

The first generation of onion routing used circuits fixed to a length of five, assumed a

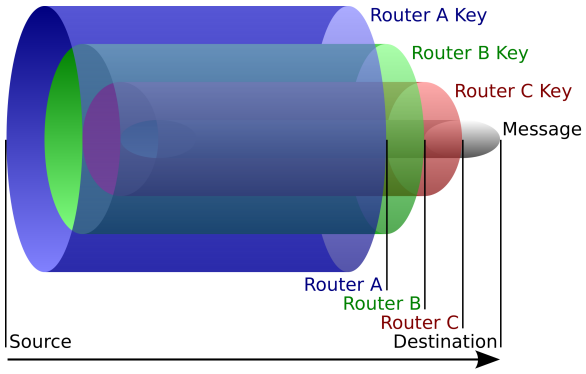

Figure 1.1: An example cell and message encryption in an onion routing scheme. Each router "peals" off its respective layer of encryption; the final router exposes the final destination.

static network topology, and most notably, introduced the ability to mate two circuits at a common node or server. This last capability enabled broader anonymity where the circuit users were anonymous to each other and to the common server, a capability that was adopted and refined by later generation onion routers. Second generation introduced variable-length circuits, multiplexing of all user traffic over circuits, exit policies for the final router, and assumed a dynamic network by routing updates throughout the network. A client, Alice, in second-generation onion routers also distributed symmetric keys through the cell layers. If routers remember the destinations for each message they received, the recipient Bob can send his reply backwards through the circuit and each router re-encrypts the reply with their symmetric key. Alice unwraps all the layers, exposing the Bob's reply. The transition from public-key cryptography to symmetric-key encryption significantly reduced the CPU load on onion routers and enabled them to transfer more packets in the same amount of time. However, while influential, first and second generation onion routing networks have fallen out of use in favor of third-generation systems. [3]

1.2 Tor

Tor is a third-generation onion routing system. It was invented in 2002 by Roger Dingledine, Nick Mathewson, and Paul Syverson of the Free Haven Project and the U.S. Naval Research Laboratory [4] and is the most popular onion router in use today. Tor inherited many of the concepts pioneered by earlier onion routers and implemented several

key changes: [3] [4]

- **Perfect forward secrecy:** Rather than distributing keys via onion layers, Tor clients negotiate ephemeral symmetric encryption keys with each of the routers in turn, extending the circuit one node at a time. These keys are then purged when the circuit is torn down; this achieves perfect forward secrecy, a property that ensures that the encryption keys will not be revealed if long-term public keys are later compromised.

- **Circuit isolation:** Second-generation onion routers mixed cells from different circuits in realtime, but later research could not justify this as an effective defence against an active adversary. [3] Tor abandoned this in favor of isolating circuits from each other inside the network. Tor circuits are used for up to 10 minutes or whenever the user chooses to rotate to a fresh circuit.

- **Three-hop circuits:** Previous onion routers used long circuits to provide heavy traffic mixing. Tor removed mixing and fell back to using short circuits of minimal length. With three relays involved in each circuit, the first node (the *guard*) is exposed to the user's IP address. The middle router passes onion cells between the guard and the final router (the *exit*) and its encryption layer exposes it to neither the user's IP nor its traffic. The exit processes user traffic, but is unaware of the origin of the requests. While the choice of middle and exits can be routers can be safely random, the guard nodes must be chosen once and then consistently used to avoid a large cumulative chance of leaking the user's IP to an attacker. This is of particular importance for circuits from hidden services. [7] [8]

- **Standardized to SOCKS proxy:** Tor simplified the multiplexing pipeline by transitioning from application-level proxies (HTTP, FTP, email, etc) to a TCP-level SOCKS proxy, which multiplexed user traffic and DNS requests through the onion circuit regardless of any higher protocol. The disadvantage to this approach is that Tor's client software has less capability to cache data and strip identifiable information out of a protocol. The countermeasure was the Tor Browser, a fork of Mozilla's open-source

Firefox with a focus on security and privacy. To reduce the risks of users breaking their privacy through Javascript, it ships with the NoScript extension which blocks all web scripts not explicitly whitelisted. The browser also forces all web traffic, including DNS requests, through the Tor SOCKS proxy, provides a Windows-Firefox user agent regardless of the native platform, and includes many additional security and privacy enhancements not included in native Firefox. The browser also utilizes the EFF's HTTPS Everywhere extension to re-write HTTP web requests into HTTPS whenever possible; when this happens the Tor cell contains an additional inner encryption layer.

- **Directory servers:** Tor introduced a set of trusted directory servers to distribute network information and the public keys of onion routers. Onion routers mirror the digitally signed network information from the directories, distributing the load. This simplified approach is more flexible and scales faster than the previous flooding approach, but relies on the trust of central directory authorities. Tor ensures that each directory is independently maintained in multiple locations and jurisdictions, reducing the likelihood of an attacker compromising all of them. [3] We describe the contents and format of the network information published by the directories in section 1.2.3.

- **Dynamic rendezvous with hidden services:** In previous onion routers, circuits mated at a fixed common node and did not use perfect forward secrecy. Tor uses a distributed hashtable to record the location of the introduction node for a given hidden service. Following the initial handshake, the server and the client then meet at a different onion router chosen by the client. This approach significantly increased the reliability of hidden services and distributed the communication load across multiple rendezvous points. [4] We provide additional details on the hidden service protocol in section 1.2.4 and our motivation for addition infrastructure in section 1.3.

As of March 2015, Tor has 2.3 million daily users that together generate 65 Gbit/s of traffic. Tor's network consists of nine authority nodes and 6,600 onion routers in 83 countries. [9] In a 2012 Top Secret U.S. National Security Agency presentation leaked by

Edward Snowden, Tor was recognized as the "the king of high secure, low latency Internet anonymity". [10] [11] In 2014, BusinessWeek claimed that Tor was "perhaps the most effective means of defeating the online surveillance efforts of intelligence agencies around the world." [12]

1.2.1 Design

Tor's design focuses on being easily deployable, flexible, and well-understood. Tor also places emphasis on usability in order to attract more users; more user activity translates to an increased difficulty of isolating and breaking the privacy of any single individual. Tor however does not manipulate any application-level protocols nor does it make any attempt to defend against global attackers. Instead, its threat model assumes that the capabilities of adversaries are limited to observing fractions of Tor traffic, that they can actively delay, delete, or manipulate traffic, that they may attempt to digitally fingerprint packets, that they may run onion routers themselves, and that they may compromise a fraction of other existing routers. Together, most of the assumptions may be broadly classified as traffic analysis attacks. Tor's final focus is defending against these types of attacks. [4]

1.2.2 Circuit Construction

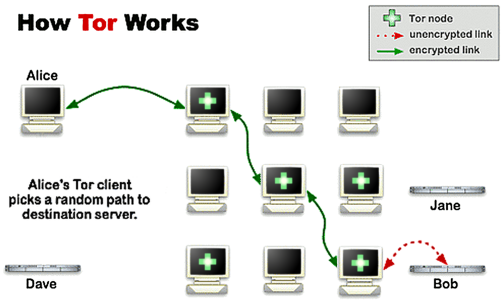

Figure 1.2: A circuit through the Tor network.

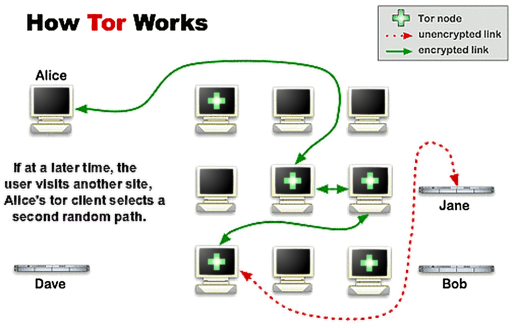

Figure 1.3: A Tor circuit is changed periodically, creating a new user identity.

Overview

Let Alice be a client who wishes to enhance her privacy by using Tor but does not run a Tor router herself. Her basic procedure for initializing her own circuit is as follows:

1. Alice selects a Tor router R_1 and sets up an authenticated encrypted connection to it.
2. Alice selects a second router R_2 and tells R_1 to build a TCP link to it.
3. Alice uses her R_1 tunnel to establish an authenticated encrypted connection with R_2.
4. Alice selects an R_3 and uses the $Alice \leftrightarrow R_1 \leftrightarrow R_2$ tunnel to tell R_2 to connect to R_3.
5. Alice establishes an authenticates encrypted connection with R_3 over the $Alice \leftrightarrow R_1 \leftrightarrow R_2$ tunnel.

Following the successful circuit construction, she can then send messages out of R_3 through the $Alice \leftrightarrow R_1 \leftrightarrow R_2 \leftrightarrow R_3$ tunnel.

As a defense against traffic analysis using packet size, Alice packs and pads circuit traffic into equally-sized Tor *cells* of 512 bytes and changes R_2 and R_3 every 10 minutes. [13]

todo: Explain why Alice must choose her guard node carefully and then stick with it.

TAP

The Tor Authentication Protocol (TAP) is a critical piece of Tor networking infrastructure as it allows Alice to verify the authenticity of the onion routers that she selects for her circuit while still remaining anonymous herself. Namely it prevents active spoofing attacks, which if successful would deanonymize Alice and compromise her privacy.

First, TAP assumes that the following is established:

- Alice has a reliable PKI that can securely distribute the identity, IP addresses, and public keys of all Tor routers. Tor accomplishes this through consensus documents from authority nodes, described in section 1.2.3.
- Let $\mathcal{E}_B$ mean encryption and $\mathcal{D}_B$ mean decryption under B's public-private keypair for some party B.

- p is a prime such that $q = \frac{q-1}{2}$ is also prime and let g be a generator of the subgroup of $\mathbb{Z}_p^*$ of order q.
- Let R_L be a generator that returns uniformly random L-bit values in the interval $[1, \min(q, 2^L) - 1]$.
- Define f as SHA-1 which takes input from $\mathbb{Z}_p$ and returns a L_f-bit value.

Todo: I'd like to include that TAP diagram [14] [4] and find out about the forward-backwards keys.

Then TAP proceeds as:

1. (a) Alice selects a Tor router, R_i.
   (b) Alice generates $x = R_L$ and computes $s = g^x \bmod p$.
   (c) Alice sends $c = \mathcal{E}_{R_i}(s)$ to R_i.
2. (a) R_i computes $m = \mathcal{D}_{R_i}(c)$ and confirms that $1 < m < p - 1$.
   (b) R_i generates $y = R_L$ and computes $a = g^y \bmod p$ and $b = f(m^y \bmod p)$.
   (c) R_i sends (a, b) to Alice.
3. Alice confirms that $1 < a < p - 1$ and that $b = f(a^x \bmod p)$.
4. If the assertions pass, then Alice has confirmed R_i's identity and now Alice and R_i can use $a^x = m^y$ as a shared symmetric encryption key and encrypt messages under AES.
5. Alice repeats steps 1 – 4 until the circuit is of the desired length.

Thus for each router in the circuit, Alice perform one public-key encryption and her half of a Diffie-Hellman-Merkle (DHE) handshake, and each router in turn performs one private-key decryption and their half of a DHE handshake. Under the assumptions that RSA is one way, AES remains unbroken, and f is strong and acts as a random oracle, an attacker has only a negligible chance of being able to impersonate R_i and read messages that she tunnels through the circuit. [15]

NTor

In mid 2013, TAP was superseded by "NTor", a new protocol invented by Goldberg, Stebila, and Ustaoglu. [16] Compared to TAP, NTor replaced the computational cost associated with DHE exponentiation with significantly more efficient elliptic curve cryptography (ECC). Its implementation uses Curve25519, [17] a Montgomery curve designed by Bernstein to be used for high-speed ECDHE key exchanges. All Curve25519 operations are implemented to run in $\mathcal{O}(1)$ time.

NTor is initialized by defining the following,

- Let $H(x, k)$ be HMAC-SHA256, which accepts a message x and yield a 32-byte MAC output. Let k_1, k_2, and k_3 be some fixed secret keys for the HMAC algorithm, $k_1 \neq k_2 \neq k_3$.
- Let $C_{gen}()$ generate a Curve25519 keypair. This function requires 32 bytes from a cryptographically secure source (such as a CSPRNG) to generate the private key, then it derives the public key.
- Let $C_{sec}(pvt, pub)$ yield a 32-byte common secret given a Curve25519 private and public key, *pvt* and *pub*, respectively.
- Let each Tor router R_j generate $b_j, B_j = C_{gen}()$ and publish B_j through the consensus document.
- Let *id* by the router's identification fingerprint.

Then NTor proceeds as:

1. (a) Alice selects a Tor router, R_i.
   (b) Alice generates $x, X = C_{gen}()$ and sends X to R_i.
2. (a) R_i generates $y, Y = C_{gen}()$.
   (b) R_i sets $sec = C_{sec}(y, X) \parallel C_{sec}(b, X) \parallel id \parallel X \parallel Y$.
   (c) R_i computes $seed = H(sec, k_1)$.

(d) R_i computes $auth = H(H(sec, k_2) \parallel id \parallel B \parallel Y \parallel X, k_3)$.

(e) R_i sends Y and $auth$ to Alice.

(f) R_i deletes the y, Y keypair.

3. (a) Alice sets $sec = C_{sec}(x, Y) \parallel C_{sec}(x, B) \parallel id \parallel X \parallel Y$.

   (b) Alice computes $seed = H(sec, k_1)$.

   (c) Alice confirms that $auth = H(H(sec, k_2) \parallel id \parallel B \parallel Y \parallel X, k_3)$.

4. Alice and R_i now have a shared secret *seed* that is then used indirectly for symmetric key encryption.

1.2.3 Consensus Documents

Todo: this section is incomplete. Here I plan to detail the consensus documents that I will be using, since I use them in my Solution section.

The Tor network is maintained by nine authority nodes, who each vote on the status of nodes and together hourly publish a digitally signed consensus document containing IPs, ports, public keys, latest status, and capabilities of all nodes in the network. The document is then redistributed by other Tor nodes to clients, enabling access to the network. The document also allows clients to authenticate Tor nodes when constructing circuits, as well as allowing Tor nodes to authenticate one another. Since all parties have prior knowledge of the public keys of the authority nodes, the consensus document cannot be forged or modified without disrupting the digital signature. [18]

cached-certs

This section is incomplete. See https://collector.torproject.org/formats.html

cached-microdescs

This section is incomplete. See Tor's dir-spec.txt, section 3.3

cached-microdesc-consensus

This section is incomplete. See https://collector.torproject.org/formats.html

1.2.4 Hidden Services

Although Tor's primary and most popular use is for secure access to the traditional Internet, Tor also supports *hidden services* – anonymous servers hosting services such as websites, marketplaces, or chatrooms. These servers intentionally mask their IP addresses through Tor circuits and thus cannot normally be accessed outside the context of Tor. In contrast to Tor-anonymized web requests where the client is anonymous but the server is known, Tor hidden services provide bidirectional anonymity where both parties remain anonymous and never directly communicate with one another. [19]

Tor does not contain a DNS system for its websites; instead hidden service addresses are algorithmically generated by generating the SHA-1 hash of their public RSA key, truncating to 80 bits, and converting the remainder to base58. Ignoring the possibility of SHA-1 collisions, this builds a one-to-one relationship between a hidden service's public key and its address which can be confirmed without requiring any identifiable information or central authorities. The address is the appended with the .onion Top-Level Domain (TLD).

A hidden server, Bob, first builds Tor circuits to several random relays and enables them to act as *introduction points* by giving them its public key, B_K. He then then uploads his public key and the fingerprint identity of these nodes to a distributed hashtable inside the Tor network, signing the result. He then publishes his hidden service address in a backchannel. When Alice obtains this address and enters it into a Tor-enabled browser, her software queries this hashtable, obtains B_K and Bob's introduction points, and builds a Tor circuit to one of them, ip_1. Simultaneously, the client also builds a circuit to another relay, rp, which she enables as a rendezvous point by telling it a one-time secret, sec. During this procedure, hidden services must continue to use their same entry node in order to avoid leaking its IP address to possibly malicious onion routers. [7] [8]

She then sends to ip_1 a cookie encrypted with B_K, containing rp and sec. Bob decrypts this message, builds a circuit to rp, and tells it sec, enabling Alice and Bob to communicate.

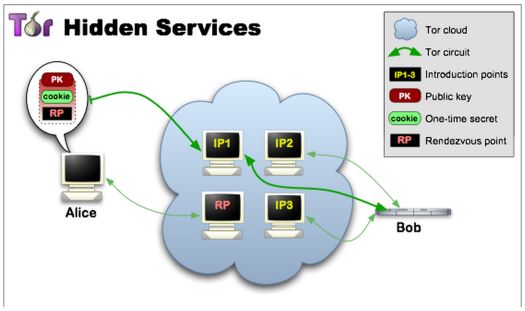

Figure 1.4: Alice uses the encrypted cookie to tell Bob to switch to *rp*.

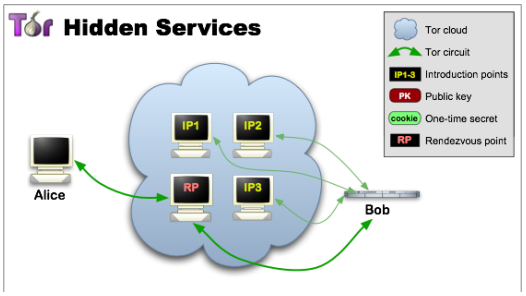

Figure 1.5: Bidirectional communication between Alice and the hidden service.

Their communication travels through six Tor nodes: three established by Alice and three by Bob, so both parties remain anonymous. From there traditional HTTP, FTP, SSH, or other protocols can be multiplexed over this new channel.

As of March 2015, Tor hosts approximately 25,000 unique hidden services that together generate around 450 Mbit/s of traffic. [9]

1.3 Motivation

As Tor's hidden service addresses are algorithmically derived from the service's public RSA key, there is at best limited capacity to select a human-meaningful name. Some hidden service operators have attempted to work around this issue by finding an RSA key by brute-force that generates a partially-desirable hidden service address (e.g. "example0uyw6wgve.onion") and although some alternative encoding schemes have been proposed, (section) the problem generally remains. The usability of hidden services is severely challenged by their non-intuitive and unmemorable base58-encoded domain name. For example, 3g2upl4pq6kufc4m.onion is the address for the DuckDuckGo search engine, suw74isz7wqzpmgu.onion is a WikiLeaks mirror, and 33y6fjyhs3phzfjj.onion and vbmwh445kf3fs2v4.onion are both SecureDrop instances for anonymous document submission. These addresses maintain strong privacy as the strong association between its public key and its address significantly breaks the association between the service's purpose and its address. However, this privacy comes at a cost: it is impossible to classify or label a hidden service's purpose in advance, a fact well known within Tor hidden service communities.

Over time, third-party directories – both on the Clear and Dark Internets – appeared in attempt to counteract this issue, but these directories must be constantly maintained and furthermore this approach is not convenient nor does it scale well. This suggests the strong need for a more complete and reliable solution.

1.4 Contributions

Our contribution to this problem is six-fold:

- We introduce a novel distributed naming system that enables Tor hidden service operators to choose a unique human-meaningful domain name and construct a strong association between that domain name and their hidden service address, squaring Zooko's Triangle (section 3.1).
- We described a new distributed, publicly confirmable, and partially self-healing transactional database.
- We provide OnioNS as a Tor plugin and rely on the existing Tor network and other infrastructure, rather than introduce a new network. This simplifies our assumptions and reduces our threat model to attack vectors already known and well-understood on the Tor network.
- We introduce a novel protocol for proving the non-existence of any domain name.
- We enable Tor clients to verify the authenticity of a domain name against its corresponding hidden service address with minimal data transfers and without requiring any additional queries.
- We preserve the privacy of both the hidden service and the anonymity of Tor clients connecting to it.

CHAPTER 2

PROBLEM STATEMENT

2.1 Assumptions and Threat Model

OnioNS' basic design and protocols rely on several assumptions and expected threat vectors.

1. Let Alice is a Tor user and let Bob be a recipient. If Alice constructs a three-router Tor circuit via NTor and sends a message m to Bob, we assume that the Tor circuit preserves Alice's privacy and provides her with anonymity from Bob's perspective. Namely, we assume that out of Alice's identity, location, and knowledge of Bob, an outside attacker Eve can obtain at most one, and that Bob can know no more about Alice than the contents of m. The exception to this is if Alice chooses to disclose her identity only to Bob, but then Bob does not know Alice's location. We assume that neither Bob nor Eve can force Alice into breaking her privacy involuntarily. The protection of a Tor circuit relies on Tor's assumption that Eve only has access to some Tor traffic; no defence is made by Tor nor by OnioNS to defend against a global adversary.

2. Adversaries cannot break standard cryptographic primitives.

   (a) We assume that they cannot efficiently break AES ciphers, SHA-384 hashes, RSA-1024, Curve25519 ECDHE, Ed25519 digital signatures, or have hardware-level attacks against the scrypt key derivation function.

   (b) We assume that they maintain no backdoors or other software breaks in the Botan or the OpenSSL implementations of the primitives.

(c) We assume that they are not capable of breaking cryptographic protocols built on the primitives such as TAP and NTor.

3. We assume that not all Tor nodes are honest. We assume that at least some Tor nodes are run by malicious operators, curious researchers, experimenting developers, or government organizations. They may also be wiretapped, exploited, broken into, or become unavailable. This assumption is also made by Tor's developers and therefore we must conclude that any new functionality added to existing Tor nodes must also fall under their assumption. However, we assume that the majority of Tor nodes are honest and reliable. We consider this reasonable because it is a prerequisite for the security of Tor circuits.

4. We assume that the percentage of dishonest routers within the Tor network 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 Tor traffic is purposely kept secret and is therefore much more valuable to Eve. OnioNS uses public data structures so we don't consider the inclusion of OnioNS a motivating factor to Eve. Moreover, we assume that Eve cannot predict in advance the identities of the Tor routers that act as authorities for OnioNS.

5. Let C be the set of Tor nodes that have the Fast and Stable flags and let Q be an M-sized set chosen randomly from C. Q may be under the influence of one or more adversaries who may cause subsets of Q to collude, but our final assumption is that the largest subset of Q is acting honestly according to specifications.

2.2 Design Objectives

Tor's high security environment introduces a distinct set of challenges that must be met by additional functionality. Privacy and anonymity are of paramount importance. Here we enumerate a list of requirements for any security-enhanced DNS applicable to Tor hidden services. In Chapter we analyse several existing prominent naming systems and show how

these systems do not meet these requirements. In Chapters and we demonstrate how we overcome them with OnioNS.

1. **The system must support anonymous registrations.** It must be hard for any party to infer the identity or location of the registrant, unless the registrant chooses to disclose that information. Hidden services are privacy-enhanced servers and the registration of a domain name should not compromise that privacy.

2. **The system must support privacy-enhanced lookups.** The resolver should not be able to identify a client nor track their lookups. In other words, clients should ideally have anonymity and be indistinguishable from other clients from the perspective of a resolver.

3. **Clients must be able to authenticate registrations.** The users of the system must be able to verify that the information they received from the service has not been forged and is authenticate relative to the authenticity of the server they will connect to.

4. **Domain names must be provably or have a near-certain chance of being 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 easily compromise user privacy or may not allow anonymous registrations, although this is system dependent. For these reasons, naming systems with central authorities can safely be considered dangerous and ill-suited for hidden services.

6. **The system must be simple and relatively easy to use.** It should be assumed that users are not security experts or have technical backgrounds. Tor's developers go to great lengths to ensure that Tor tools are straightforward, and the success of Tor's low-latency onion routing model demonstrates the effectiveness of convenience within high-security settings. Any naming system used by Tor must be equivalently simple, resolve protocols with minimal input from the user, and hide non-essential details.

7. **The system must be backwards compatible with existing protocols.** Just as the Internet's DNS did not introduce any changes in the TCP/IP layer, naming systems for Tor must preserve the original Tor hidden service protocol. DNS is optional, not required.

Additionally, we specify several performance-enhancing objectives, although these are not requirements.

1. **The system should not require clients to download the entire database.** It can safely assumed that in most realistic environments clients have neither the network capacity nor the storage to hold the system's database. While they may choose to download the database, it should not be required for normal functionality or to meet any of the above objectives.

2. **The system should not introduce significant burdens to the clients.** Despite the rapid and exponential growth of consumer-grade hardware, in most environments not every client is on a high-end machine. Therefore the system should not burden user computers with significant computation or memory demands.

3. **The system should have low latency** and resolve lookup queries within a reasonable amount of time.

CHAPTER 3

CHALLENGES

3.1 Zooko's Triangle

Our primary objective with OnioNS is to provide human-meaningful domain names Tor hidden services, but we list distributed and securely unique domain names in our design requirements. Achieving all three objectives is not easy; a naming scheme can use a central root zone or authority to ensure that meaningful domains remain unique but then it is not distributed, it can achieve a distributed nature by generating domain names with cryptographic hash function but then domains are no longer human-meaningful, or it can allow peers to provide meaningful names to each other but then these names are not guaranteed to be globally unique. This problem is illustrated in Figure 3.1 and summarized by Zooko's Triangle, a conjecture proposed by Zooko Wilcox-O'Hearn in late 2001. The conjecture states a persistent naming system can achieve at most two of these properties: it can provide unique and meaningful names but not be distributed, it can be distributed and provide unique names that are not meaningful, or it can be distributed and provide meaningful names that are not guaranteed to be unique. [20] [21]

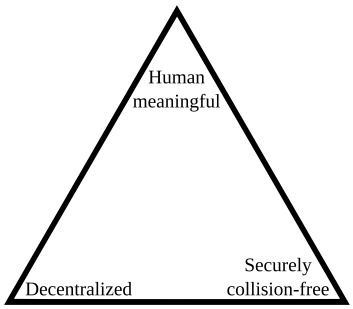

Figure 3.1: Zooko's Triangle.

Petnames are systems that achieve all three properties of Zooko's Triangle. Some examples of naming systems that achieve only two of these properties include:

- **Securely unique and human-meaningful** — Clearnet domain names are memorable and provably collision free, but they use DNS with a hierarchical structure and central authorities under the jurisdiction of ICANN.
- **Decentralized and human-meaningful** — Human names and nicknames are legal and social labels for each other, but we provide no protection against name collisions.
- **Securely unique and decentralized** — Tor hidden service .onion domains, PGP keys, and Bitcoin/Namecoin addresses use the large key-space and the collision-free properties of cryptographic hash algorithms to ensure uniqueness, but do not use meaningful names.

We consider the securely unique and human-meaningful properties the most important objectives, and like Namecoin (discussed in section **??**) we provide a mechanism that enables the network to agree on a common database and largely prevents it from fragmenting. Additionally, all participants can confirm for themselves the integrity of the database and the uniqueness of domain names contained within it. This holds the securely unique property within the distributed environment.

3.2 Non-existence Verification

In our design requirements we specify that clients of a naming system should be able to verify the authenticity of domain names. On the Internet, this is achieved through SSL certificates: clients first receive from the destination server a digital certificate, they verify that the certificate matches the domain name they requested to their DNS resolver, and finally they check the certificate against Certificate Authorities via a Chain of Trust. Of equal importance, however, is the capability to verify the non-existence of domain names. Specifically a resolver may falsely claim that a domain name cannot be resolved because it does not exist, but a client has no mechanism by which to verify this claim besides changing

resolvers. This is a weakness often overlooked in other DNSs and resolving this problem is not easy. However, we in section **??** we introduce a data structure and a protocol that allows a resolver to proof non-existence in constant time.

CHAPTER 4

ANALYSIS

4.1 Security

Now we examine and compare OnioNS's central protocols against our security assumptions and expected threat model.

4.1.1 Hidden Service Protocols

Record Generation

Tor hidden services are identified by their public keys, and assuming a hidden service Bob does not provide a PGP key in his Record, we leak no identifiable information

Although this approach does not entirely thwart Sybil attack, this attack vector is difficulty to impossible to counter in a privacy-enhanced environment, and trading anonymity for defence is highly undesirable.

Let Bob create the Record r.

The proof-of-work scheme is carefully designed to limit Eve to the same capabilities as legitimate users, thus significantly deterring this attack. The use of scrypt makes custom hardware and massively-parallel computation expensive, and the digital signature in every record forces the hidden service operator to resign the fields for every iteration in the proof-of-work. While the scheme would not entirely prevent the operator from outsourcing the computation to a cloud service or to a secondary offline resource, the other machine would need the hidden service private key to regenerate *recordSig*, which the operator can't reveal without compromising his security. However, the secondary resource could perform the scrypt computations in batch without generating *recordSig*, but it would always perform more than the necessary amount of computation because it would could not generate the

SHA-384 hash and thus know when to stop. Furthermore, offloading the computation would still incur a cost to the hidden service operator, who would have to pay another party for the consumed computational resources. Thus the scheme always requires some cost when claiming a domain name.

Record Broadcast

Tor circuit preserves privacy.

4.1.2 OnioNS Server Protocols

Page Selection

Synchronization

Quorum Qualification

The quorum nodes hold the greatest amount of responsibility and control over OnionNS out of all participating nodes in the Tor network, therefore ensuring their security and limiting their attack capabilities is of primary importance.

In section 2.1, we assumed that an attacker, Eve, already controls a fraction of Tor routers. For a dynamic network size, a fixed fraction of dishonest nodes, and a fixed quorum size, every time a quorum is selected there is a probability that more than half of the quorum is dishonest and is colluding together. If this occurs, records may be dropped rather than archived. *Here we explore the optimal quorum rotation rate, given the balance between high overhead and high security risk if the quorum is rotated quickly, and long-term implications and possible stability issues associated with very slow rotation.*

If Eve controls some Tor nodes (who may be assumed to be colluding with one another), the attacker may desire to include their nodes in the quorum for malicious manipulation, passive observation, or for other purposes. Alternatively, Eve may wish to exclude certain legitimate nodes from inclusion in the quorum. In order to carry out either of these attacks,

Eve must have the list of qualified Tor nodes scrambled in such a way that the output is pleasing to Eve. Specifically, the scrambled list must contain at least some of Eve's malicious nodes for the first attack, or exclude the legitimate target nodes for the second attack. We initialize Mersenne Twister with a 384-bit seed, thus Eve can find k seeds that generates a desirable scrambled list in 2^{192} operations on average, or 2^{384} operations in the worst case. The chance of any of those seeds being selected, and thus Eve successfully carrying out the attack, is thus $\frac{2^{384}}{k}$.

Eve may attempt to manipulate the consensus document in such a way that the SHA-384 hash is one of these k seeds. Eve may instruct her Tor nodes to upload a custom status report to the authority nodes in an attempt to maliciously manipulate the contents of the consensus document, but SHA-384's strong preimage resistance and the unknown state and number of Tor nodes outside Eve's control makes this attack infeasible. The time to break preimage resistance of full SHA-384 is still 2^{384} operations. This also implies that Eve cannot determine in advance the next consensus document, so the new quorum cannot be predicted. If Eve has compromised at least some of the Tor authority nodes she has significantly more power in manipulating the consensus document for her own purposes, but this attack vector can also break the Tor network as a whole and is thus outside the scope of our analysis. Therefore, the computation required to maliciously generate the quorum puts this attack vector outside the reach of computationally-bound adversaries.

OnionNS and the Tor network as a whole are both susceptible to Sybil attacks, though these attacks are made significantly more challenging by the slow building of trust in the Tor network. Eve may attempt to introduce large numbers of nodes under her control in an attempt to increase her chances of at least one of the becoming members of the *quorum*. Sybil attacks are not unknown to Tor; in December 2014 the black hat hacking group LizardSquad launched 3000 nodes in the Google Cloud in an attempt to intercept the majority of Tor traffic. However, as Tor authority nodes grant consensus weight to new Tor nodes very slowly, despite controlling a third of all Tor nodes, these 3,000 nodes moved 0.2743 percent of Tor traffic before they were banned from the Tor network. The Stable

and Fast flags are also granted after weeks of uptime and a history of reliability. As nodes must have these flags to be qualified as a *quorum candidate*, these large-scale Sybil attacks are financially demanding and time-consuming for Eve.

Flooding

4.1.3 Tor Clients

Quorum Derivation

Domain Query

Tor circuit preserves privacy.

OnionNS records are self-signed and include the hidden service's public key, so anyone — particularly the client — can confirm the authenticity (relative to the authenticity of the public key) and integrity of any record. This does not entirely prevent Sybil attacks, but this is a very hard problem to address in a distributed environment without the confirmation from a central authority. However, the proof-of-work component makes record spoofing a costly endeavour, but it is not impossible to a well-resourced attacker with sufficient access to high-end general-purpose hardware.

Hidden service .onion addresses will continue to have an extremely high chance of being securely unique as long the key-space is sufficiently large to avoid hash collisions.

As we have stated earlier, falsely claiming a negative on the existence of a record is a problem overlooked in other domain name systems. One of the primary challenges with this approach is that the space of possible names so vast that attempting to enumerate and digitally sign all names that are not taken is highly impractical. Without a solution, this weakness can degenerate into a denial-of-service attack if the DNS resolver is malicious towards the client. Our counter-measure is the highly compact hashtable bitset with a Merkle tree for collisions. We set the size of the hashtable such that the number of collisions

is statistically very small, allowing an efficient lookup in $\mathcal{O}(1)$ time on average with minimal data transferred to the client.

Onion Query

4.2 Design Objectives

OnioNS achieves all of our original requirements:

1. **The system must support anonymous registrations** — OnioNS Records do not contain any personal or location information. The PGP key field is optional and may be provided if the hidden service operator wishes to allow others to contact him. However, the operator may be using an email address and a Web of Trust disassociated from his real identity, in which case no identifiable information is exposed.

2. **The system must support privacy-enhanced lookups** — OnioNS performs Domain and Onion Queries through Tor circuits, and under our original assumption that circuits provide strong guarentees of client privacy and anonymity, resolvers cannot sufficiently distinguish users to track their lookups.

3. **Clients must be able to authenticate registrations** — OnioNS Records are self-signed, enabling Tor clients to verify the digital signature on the domain names and check the public key against the server's key during the hidden service protocol. This ensures that the association has not been modified in transit and that the domain name is authentic relative to the authenticity of the destination server.

4. **Domain names must be provably or have a near-certain chance of being unique** — Tor hidden services .onion addresses are cryptographically generated with a key-space of $58^{16} \approx 2^{93.727695922}$ and domain names within OnioNS are provably unique by anyone holding a complete copy of the Pagechain.

5. **The system must be distributed** — The responsibilities of OnioNS are spread out across many nodes in the Tor network, decreasing the load and attack potential

for any single node. The Pagechain is likewise distributed and locally-checked by all Mirrors, and although its head is managed by the Quorum, these authoritative nodes have temporary lifetimes and are randomly selected. Moreover, Quorum nodes do not answer queries, so they have limited power.

6. **The system must be simple and relatively easy to use** — Domain Queries are automatically resolved and require no input by the user. From the user's perspective, they are taken directly from a meaningful domain name to a hidden service. Users no longer have to use unwieldy .onion addresses or review third-party directories, OnioNS introduces memorability to hidden service domains.

7. **The system must be backwards compatible with existing protocols** — OnioNS does not require any changes to the hidden service protocol and existing .onion addresses remain fully functional. Our only significant change to Tor's infrastructure is the mechanism for distributing hashes for the Quorum Qualification protocol, but our initial technique for using the Contact field minimizes any impact. We also hook into Tor's TLD checks, but this change is very minor. Our reference implementation is provided as a software package separate from Tor per the Unix convention.

Finally, we meet our optional performance objectives:

1. **The system should not require clients to download the entire database** — Only Mirrors hold the Pagechain, and clients do not need to obtain it themselves to issue a Domain Query. Therefore clients rely on their existing and well-established trust of Tor routers when resolving domain names. However, clients may optionally obtain the Pagechain and post Domain Queries to localhost for greater privacy and security guarantees.

2. **The system should not introduce significant burdens to the clients** — Record verification should occur in sub-second constant time in most environments, and Ed25519 achieves very fast signature confirmation so verifying Page signatures at level 1+ takes trivial time. However, clients also verify the Record's proof-of-work,

so for some scrypt parameters the client may spend non-trivial CPU time and RAM usage confirming the one scrypt iteration required to check. We must therefore choose our parameters carefully to reduce this burden especially on low-end hardware.

3. **The system should have low latency** — Domain Queries without any packet delays over Tor low-latency circuits. Its exact performance is largely dependent on circuit speed and the client's verification speed.

We therefore believe that we have squared Zooko's Triangle; OnioNS is distributed, enables hidden service operators to select human-meaningful domain names, and domain names are guaranteed unique by all participants.

4.3 Performance

bandwidth, CPU, RAM, latency for clients to be determined...

demand on participating nodes to be determined...

Unlike Namecoin, OnionNS' *page*-chain is of L days in maximal length. This serves two purposes:

1. Causes domain names to expire, which reduced the threat of name squatting.

2. Prevents the data structure from growing to an unmanageable size.

4.3.1 Proof-of-work

4.3.2 Broadcast

4.3.3 Flood

4.3.4 Query

4.4 Reliability

Tor nodes have no reliability guarantee and may disappear from the network momentarily or permanently at any time. Old *quorums* may disappear from the network without consequence of data loss, as their data is cloned by current *mirrors*. So long as the *quorum* nodes remain up for the Δi days that they are active, the system will suffer no loss of functionality. Nodes that become temporarily unavailable will have out-of-sync *pages* and will have to fetch recent records from other *quorum* nodes in the time of their absence.

CHAPTER 5

FUTURE WORK

We introduce no changes to Tor's hidden service protocol and also note that the existence of a DNS system introduces forward-compatibility: developers can replace hash functions and PKI in the hidden service protocol without disrupting its users, so long as records are transferred and OnionNS is updated to support the new public keys.

web of trust for .onion keys to thwart Sybil attack

Some open problems that I need to address include:

1. How frequently should domains expire? Are there any security risk in sending a Renew request?
2. How should unreachable or temporarily down nodes be handled? I'd like to know the percentage of the Tor network that is reachable at any given time.
3. How many nodes in the Tor network should be assumed to be actively malicious? What are the implications of increasing this percentage?
4. What attack vectors are there from the committee nodes, and how can I thwart the attacks?
5. What are the implications of a node intentionally voting the opposite way, or ignoring the request altogether?
6. What would happen if a node was too slow or did not have enough storage space to do its job properly?
7. What other open problems are there?
8. What related works are there in the literature that relate to the concepts I have created here?

REFERENCES

[1] D. Chaum, "Untraceable electronic mail, return addresses and digital pseudonyms," in *Secure electronic voting*. Springer, 2003, pp. 211–219.

[2] M. Edman and B. Yener, "On anonymity in an electronic society: A survey of anonymous communication systems," *ACM Computing Surveys (CSUR)*, vol. 42, no. 1, p. 5, 2009.

[3] P. Syverson, "A peel of onion," in *Proceedings of the 27th Annual Computer Security Applications Conference*. ACM, 2011, pp. 123–137.

[4] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," DTIC Document, Tech. Rep., 2004.

[5] P. F. Syverson, D. M. Goldschlag, and M. G. Reed, "Anonymous connections and onion routing," in *Security and Privacy, 1997. Proceedings., 1997 IEEE Symposium on*. IEEE, 1997, pp. 44–54.

[6] M. G. Reed, P. F. Syverson, and D. M. Goldschlag, "Anonymous connections and onion routing," *Selected Areas in Communications, IEEE Journal on*, vol. 16, no. 4, pp. 482–494, 1998.

[7] K. Bauer, D. McCoy, D. Grunwald, T. Kohno, and D. Sicker, "Low-resource routing attacks against tor," in *Proceedings of the 2007 ACM workshop on Privacy in electronic society*. ACM, 2007, pp. 11–20.

[8] L. Overlier and P. Syverson, "Locating hidden servers," in *Security and Privacy, 2006 IEEE Symposium on*. IEEE, 2006, pp. 15–pp.

[9] T. T. Project, "Tor metrics," https://metrics.torproject.org/, 2015, accessed 4-Feb-2015.

[10] S. Landau, "Highlights from making sense of snowden, part ii: What's significant in the nsa revelations," *Security & Privacy, IEEE*, vol. 12, no. 1, pp. 62–64, 2014.

[11] R. Plak, "Anonymous internet: Anonymizing peer-to-peer traffic using applied cryptography," Ph.D. dissertation, TU Delft, Delft University of Technology, 2014.

[12] D. Lawrence, "The inside story of tor, the best internet anonymity tool the government ever built," *Bloomberg BusinessWeek*, January 2014.

[13] D. McCoy, K. Bauer, D. Grunwald, T. Kohno, and D. Sicker, "Shining light in dark places: Understanding the tor network," in *Privacy Enhancing Technologies*. Springer, 2008, pp. 63–76.

[14] Z. Ling, J. Luo, W. Yu, X. Fu, W. Jia, and W. Zhao, "Protocol-level attacks against tor," *Computer Networks*, vol. 57, no. 4, pp. 869–886, 2013.

[15] I. Goldberg, "On the security of the tor authentication protocol," in *Privacy Enhancing Technologies*. Springer, 2006, pp. 316–331.

[16] I. Goldberg, D. Stebila, and B. Ustaoglu, "Anonymity and one-way authentication in key exchange protocols," *Designs, Codes and Cryptography*, vol. 67, no. 2, pp. 245–269, 2013.

[17] D. J. Bernstein, "Curve25519: new diffie-hellman speed records," in *Public Key Cryptography-PKC 2006*. Springer, 2006, pp. 207–228.

[18] L. Xin and W. Neng, "Design improvement for tor against low-cost traffic attack and low-resource routing attack," in *Communications and Mobile Computing, 2009. CMC'09. WRI International Conference on*, vol. 3. IEEE, 2009, pp. 549–554.

[19] S. Nicolussi, "Human-readable names for tor hidden services," 2011.

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

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