ESGALDNS:
TOR-POWERED DISTRIBUTED DNS
FOR TOR HIDDEN SERVICES

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

ABSTRACT

EsgalDNS:

Tor-powered Distributed DNS

for Tor Hidden Services

by

Jesse Victors, Master of Science

Utah State University, 2015

Major Professor: Dr. Ming Li
Department: Computer Science

The Tor network is a second-generation onion routing system that aims to provide anonymity, privacy, and Internet censorship resistance to its users. In recent years it has grown significantly in response to revelations of national and global electronic surveillance, and remains one of the most popular and secure anonymity network in use today. Tor is also known for its support of anonymous websites within its network. Decentralized and secure, the domain names for these services are tied to public key infrastructure (PKI) but have usability challenges due to their long and technical addresses. In response to this difficulty, I propose and partially implement a decentralized DNS system inside the Tor network. The system provides a secure and verifiable mapping between human-meaningful names and traditional Tor hidden service addresses, is backwards- and forwards-compatible with Tor hidden service infrastructure, and preserves the anonymity of the hidden service and its operator.

(42 pages)

This work is dedicated to the developers and community behind the Tor Project and the Tails OS. These individuals work tirelessly to preserve the privacy and security of everyday citizens, journalists, activists, and others around the globe, providing the much-needed service of private conversations in a very public world.

CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The Tor network is a second-generation onion routing system that aims to provide anonymity, privacy, and Internet censorship protection to its users. The Tor client software multiplexes all end-user TCP traffic through a series of relays on the Tor network, typically a carefully-constructed three-hop path known as a *circuit*. Each relay in the circuit has its own encryption layer, so traffic is encrypted multiple times and then is decrypted in an onion-like fashion as it travels through the Tor circuit. As each relay sees no more than one hop in the circuit, in theory neither an eavesdropper nor a compromised relay can link the connection's source, destination, and content. Tor remains one of the most popular and secure tools to use against network surveillance, traffic analysis, and information censorship.

While the majority of Tor's usage is for traditional access to the Internet, Tor's routing scheme also supports anonymous websites, hidden inside Tor. Unlike the Clearnet, Tor does not contain a traditional DNS system for its websites; instead, hidden services are identified by their public key and can be accessed through Tor circuits. A client and the hidden service can thus communicate anonymously.

CHAPTER 2

BACKGROUND

2.1 Tor

The Tor network is a third-generation onion routing system, originally designed by the U.S. Naval Research Laboratory for protecting sensitive government communication. Tor refers both to the client-side multiplexing software and to the worldwide volunteer-run network of over six thousand nodes. The Tor software provides an anonymity and privacy layer to end-users by relaying TCP traffic through a series of relays on the Tor network. Tor sees global widespread use and as of January 2015 has over 2.4 million daily users and passes an average of approximately 6000 MB per second through its network. [**?**] Tor's encryption and routing protocols are designed to make it very difficult for an adversary to correlate an end user to their traffic. Tor has been recognized by the NSA as the "the king of high secure, low latency Internet anonymity". [**?**]

2.1.1 Design

Tor routes encrypted TCP/IP user traffic through a worldwide volunteer-run network of over six thousand relays. Typically this route consists of a carefully-constructed three-hop path known as a *circuit*, which changes over time. These nodes in the circuit are commonly referred to as *guard node*, *middle relay*, and the *exit node*, respectively. Only the first node is exposed to the origin of TCP traffic into Tor, and only the exit node can see the destination of traffic out of Tor. The middle router, which passes encrypted traffic between the two, is unaware of either. The client negotiates a separate TLS connection with each node at a time, and traffic through the circuit is decrypted one layer at a time. As such, each node is only aware of the machines it talks to, and only the client knows the

identity of all three nodes used in its circuit, making traffic correlation much more difficult difficult compared to a VPN, proxy, or a direct TLS connection.

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. [1]

2.1.2 Routing

In traditional Internet connections, the client communicates directly with the server. TLS encryption cannot hide IP and TCP headers, which must be exposed to allow routing. Eeavesdroppers can track end-users by monitoring these headers, easily correlating clients to their activities. Tor combats this by routing end user traffic through a randomized circuit through the network of relays. The client software first queries an authority node or a known relay for the latest consensus document. Next, the Tor client chooses three unique and geographically diverse nodes to use. It then builds and extends the circuit one node at at time, negotiating respective TLS connections with each node in turn. No single relay knows the complete path, and each relay can only decrypt its layer of decryption. In this way, data is encrypted multiple times and then is decrypted in an onion-like fashion as it passes through the circuit.

The client first establishes a TLS connection with the first relay, R_1, using the relay's public key. The client then performs an ECDHE key exchange to negotiate K_1 which is then used to generate two symmetric session keys: a forward key $K_{1,F}$ and a backwards key $K_{1,B}$. $K_{1,F}$ is used to encrypt all communication from the client to R_1 and $K_{1,B}$ is used for all replies from R_1 to the client. These keys are used conjunction with the symmetric cipher suite negotiated during the TLS handshake, thus forming an encrypted tunnel with

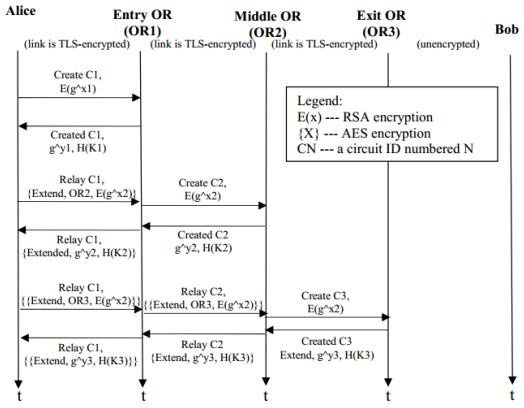

Figure 2.1: Anatomy of the construction of a Tor circuit.

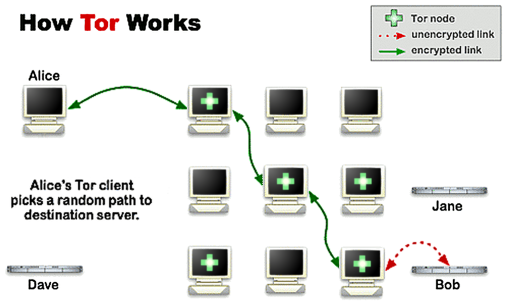

Figure 2.2: A circuit through the Tor network.

perfect forward secrecy. Once this one-hop circuit has been created, the client then sends R_1 the RELAY_EXTEND command, the address of R_2, and the client's half of the Diffie-Hellman-Merkle protocol using $K_{1,F}$. R_1 performs a TLS handshake with R_2 and uses R_2's public key to send this half of the handshake to R_2, who replies with his second half of the handshake and a hash of K_2. R_1 then forwards this to the client under $R_{1,B}$ with the RELAY_EXTENDED command to notify the client. The client generates $K_{1,F}$ and $K_{1,B}$ from K_2, and repeats the process for R_3, [2] as shown in Figure 3. The TCP/IP connections remain open, so the returned information travels back up the circuit to the end user.

Following the complete establishment of a circuit, the Tor client software then offers a Secure Sockets (SOCKS) interface on localhost which multiplexes any TCP traffic through Tor. At the application layer, this data is packed and padded into equally-sized Tor *cells*, transmission units of 512 bytes. As each relay sees no more than one hop in the circuit, in theory neither an eavesdropper nor a compromised relay can link the connection's source, destination, and content. Tor further obfuscates user traffic by changing the circuit path every ten minutes, [3] as shown in Figure 4. A new circuit can also be requested manually by the user.

Tor users typically use the Tor Browser, a custom build of Mozilla Firefox with a focus on security and privacy. The TBB anonymizes and provides privacy to the user in many ways. These include blocking all web scripts not explicitly whitelisted, forcing all

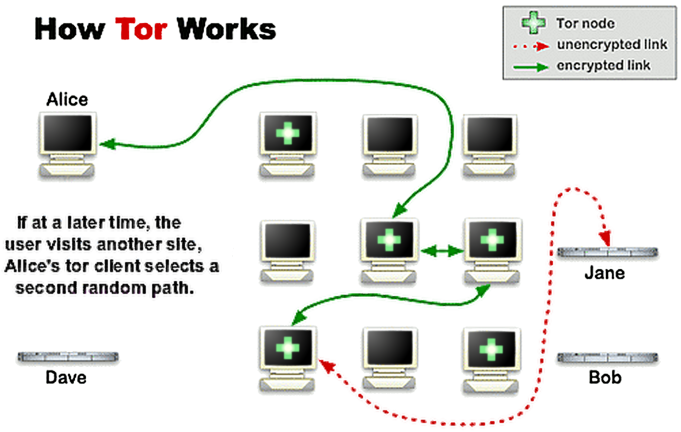

Figure 2.3: A Tor circuit is changed periodically, creating a new user identity.

traffic including DNS requests through the Tor SOCKS port, mimicking Firefox in Windows both with a user agent (regardless of the native platform) and SSL cipher suites, and reducing Javascript timer precision to avoid identification through clock skew. Furthermore, the TBB includes the Electronic Frontier Foundation's HTTPS Everywhere extension, which uses regular expressions to rewrite HTTP web requests into HTTPS for domains that are known to support HTTPS. If this is the case, an HTTPS connection will be established with the web server. If this happens, end-to-end encryption is complete and an outsider near the user would be faced with up to four layers of TLS encryption: $K_{1,F}(K_{2,F}(K_{3,F}(K_{server}(\text{client request}))))$ and likewise $K_{1,B}(K_{2,B}(K_{3,B}(K_{server}(\text{server reply}))))$ for the returning traffic, making traffic analysis very difficult.

2.1.3 Hidden Services

Although Tor's primary and most popular use is for secure access to the traditional Internet, since 2004 Tor also supports anonymous services, such as websites, marketplaces, or chatrooms. These are a part of the Dark Web and cannot be normally 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. This allows for a greater range of communication capabilities. [4]

Tor hidden services are known only by their public RSA key. Tor does not contain a DNS system for its websites; instead the domain names of hidden services are an 80-bit truncated SHA-1 hash of its public key, postpended by the .onion top-level domain (TLD). Once the hidden service is contacted and its public key obtained, this key can be checked against the requested domain to verify the authenticity of the service server. This process is analogous to SSL certificates in the clearnet, however Tor's authenticity check leaks no identifiable information about the anonymous server. If a client obtains the hash domain name of the hidden service through a backchannel and enters it into the Tor Browser, the hidden service lookup begins.

Preceding any client communication, the 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. The server then uploads its public key and the fingerprint identity of these nodes to a distributed hashtable inside the Tor network, signing the result. When a client, Alice, requests contact with Bob, Alice's Tor 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, S.

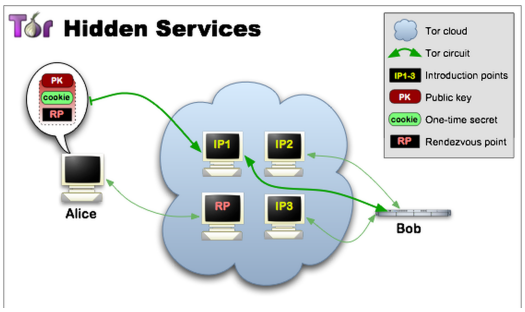

Figure 2.4: Alice uses the encrypted cookie to tell Bob to switch to RP.

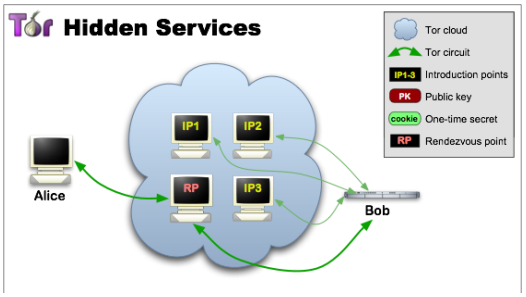

Figure 2.5: Bidirectional communication between Alice and the hidden service.

She then sends to IP_1 an cookie encrypted with B_K, containing RP and S. Bob decrypts this message, builds a circuit to RP, and tells it S_1, enabling Alice and Bob to communicate. 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.

2.2 Motivation

The usability of hidden services is severely challenged by their non-intuitive 16-character base58-encoded domain names. To choose several prominent examples, 3g2upl4pq6kufc4m.onion is the address for the DuckDuckGo hidden service, 33y6fjyhs3phzfjj.onion is the Guardian's SecureDrop service for anonymous document submission, and blockchainbdgpzk.onion is the anonymized edition of blockchain.info. It is rarely clear what service a hidden server is providing by its domain name alone without relying on third-party directories for the correlation, directories which must be updated and reliably maintained constantly. These must be then distributed through backchannels such as /r/onions, the-hidden-wiki.com, or through a hidden service that is known in advance. It is a frequent topic of conversation inside Tor communities. It is clear that this problem limits the usability and popularity of Tor hidden services. Although there have been some workarounds that are partially successful, the issue remains unresolved. It is for these reaons that I propose EsgalDNS as a full solution.

CHAPTER 3

REQUIREMENTS

3.1 Assumptions and Threat Model

One of the primary assumptions that I make in this system is that not all Tor nodes can be trusted. Some of them may be run by malicious operators, curious researchers, or experimenting developers. They may be wiretapped, the Tor software modified and recompiled, or they may otherwise behave in an abnormal fashion. I assume that adversaries have control over some of the Tor network, they have access to large amounts of computational and financial resources, and that they have access to portions of Internet traffic, including portions of Tor traffic. I also assume that they do not have global and total Internet monitoring capabilities and I make no attempt to defend against such an attacker, although it should be noted this is an assumption also made by Tor. I work under the belief that attackers are not capable of cryptographically breaking properly-implemented TLS connections and their modern components, particularly the AES cipher, ECDHE key exchange, and the SHA2 series of digests, and that they maintain no backdoors in the Botan and OpenSSL implementations of these algorithms. Lastly, I assume that adversaries monitor and may attempt to modify the DNS record databases, but I assume that at least 50 percent of the Tor network is trustworthy and behave normally in accordance with Tor and EsgalDNS specifications.

3.2 Design Principles

Tor's high security environment is challenging to the inclusion of additional capabilities, even to systems that are backwards compatible to existing infrastructure. Anonymity, privacy, and general security are of paramount importance. We enumerate a short list of

requirements for any secure DNS system designed for safe use by Tor clients. We later show how existing works do not meet these requirements and how we overcome these challenges with EsgalDNS.

1. The registrations must be anonymous; it should be infeasible to identify the registrant from the registration, including over the wire.

2. Lookups must be anonymous or at least privacy-enhanced; it should not be trivial to determine what hidden services a client is interested in.

3. Registrations must be publicly confirmable; clients must be able to verify that the registration came from the desired hidden service and that the registration is not a forgery.

4. Registrations must be securely unique, or have an extremely high chance of being securely unique such as when this property relies on the collision-free property of cryptographic hashes.

5. It must be distributed. The Tor community will adamantly reject any centralized solution for Tor hidden services for security reasons, as centralized control makes correlations easy, violating our first two requirements.

6. It must remain simple to use. Usability is key as most Tor users are not security experts. Tor hides non-essential details like routing information behind the scenes, so additional software should follow suite.

7. It must remain backwards compatible; the existing Tor infrastructure must still remain functional.

8. It should not be feasible to maliciously modify or falsify registrations in the database or in transit, even though insider attacks.

Several additional objectives, although they are not requirements, revolve around performance: it should be assumed that it is impractical for clients to download the entirety

or large portions of the DNS database in order to verify any of the requirements, a DNS system should take a reasonable amount of time to resolve domain name queries, and that the system should not introduce any significant load on client computers.

CHAPTER 4

CHALLENGES

4.1 Zooko's Triangle

One of the largest challenges is inherent to the difficulty of designing a distributed system that maintains a correlation database of human-meaningful names in a one-to-one fashion. The problem is summarized in Zooko's Triangle, an influential conjecture proposed by Zooko Wilcox-O'Hearn in late 2001. The conjecture states that in a persistent naming system, only two out of the three following properties can be established: [5]

- Human meaningfulness: the names have a quality of meaningfulness and memorability to the users.
- Securely unique: for any name, duplicates do not exist.
- Distributed: the naming system lacks a central authority or database for allocating and distributing names.

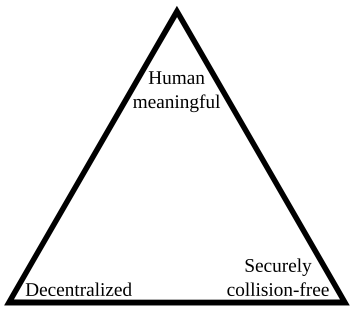

Figure 4.1: Zooko's Triangle.

Tor hidden service .onion domains, PGP keys, and Bitcoin addresses are secure and decentralized but are not human-meaningful; they use the large key-space and the collision-free properties of secure digest algorithms to ensure uniqueness, so no centralized database is needed to provide this property. Tradition domain names on the Clearnet are memorable and provably collision-free, but use a hierarchical structure and central authorities under the jurisdiction of ICANN. Finally, human names and nicknames are meaningful and distributed, but not securely collision-free. [6]

4.2 Communication

4.3 Fault Tolerance

CHAPTER 5

EXISTING WORKS

Several workarounds exist that attempt to alleviate the issue, with one used in practice, but none are fully successful. Other DNS systems already exist, including some that are distributed, but they are either not applicable, or their design makes it extremely challenging to integration into the Tor environment. Two primary problems occur when attempting to use existing DNS systems: being able to prove the correlation of ownership between the DNS name and the hidden service, and the leakage of information that may compromise the anonymity of the hidden service or its operator. As hidden services are only known by their public key and introduction points, it is not easy to use only the hidden service descriptor to prove ownership, and there are privacy and security issues with requiring any more information. Due to these problems, no existing works have been yet integrated into the Tor environment, and the one workaround that is used in practice remains only partially successful.

5.1 Encoding Schemes

In an attempt to address the readability and the memorability of hidden service domain names, two changes to the encoding of domain names have been proposed, with one used in practice.

Shallot, originally created by an anonymous developer sometime between 2004 and 2010, is an application which uses OpenSSL to generate many RSA hidden service keys in an attempt to find one that has a desirable hash, such as one that begins with a meaningful noun. [**?**] By brute-forcing the domain key-space, Shallot will eventually find a domain that is meaningful and partially memorable to the hidden service operator. Shallot was used by Blockchain.info (blockchainbdgpzk.onion), by Facebook (facebookcorewwwi.onion), and by

many other hidden services in an attempt to make their domain name appear less random. However, Shallot only partially successful because the size of the domain key-space is too large to be fully brute-forced in any reasonable length of time. For example, on a 1.5 GHz processor, Shallot is expected to take approximately 25 days to find a key whose hash contains 8 custom letters out of the 16 total. [**?**] Tor Proposal 224 makes this solution even worse as it suggests 32-byte domain names which embed an entire ECDSA hidden service key in base32. [**?**] Although prefixing the domain name with a meaningful word helps identify a hidden service at a glance, it does nothing to alleviate the logistic problems of entering a hidden service domain name, including the remaining random characters, manually into the Tor Browser.

A different encoding scheme was proposed in 2011 by Simon Nicolussi, who suggested that encoding the key hash as a series of words, using a dictionary known in advance by all parties. The list of words would then represent the domain name, rather than base58. While this scheme would improve the memorability of hidden services, the words cannot all be chosen by the hidden service operator, and brute-forcing with Shallot would again only be partially successful due to the large key-space. Therefore this solution could be used to generate some words that relate to the hidden service in a meaningful way, but this scheme is only a partial solution. [4]

These schemes do not change the underlying hidden service protocol, they just attempt to increase the readability of the domain names in Tor Browser. Compared against our original requirements, they meet the anonymity for registration and lookups due to Tor circuits, the confirmability because the domain is the hash of the public key, the uniqueness of domain names due to the collision-free property of SHA-1, the distributed requirement by way of the distributed hashtable stored throughout the Tor network, and resistance to malicious modifications because of the strong association between domain names and the hidden service key. However, it fails to meet the simplicity requirement because although hidden service domain names are entered in the traditional manner into the Tor Browser, the domain names are not entirely human-meaningful nor memorable. The domain names con-

tinue to suffer from usability problems, only partially alleviated by these encoding schemes. These workarounds do not introduce any new DNS systems or change the existing Tor hidden service protocol in any way that allows for customized domain names, but these partial solutions are worthy of mention nevertheless.

5.2 Clearnet DNS

The Internet Domain Name Service (DNS) was originally designed in 1983 as a hierarchical naming system to link domain names to Internet Protocol (IP) addresses and translate one to the other. IP addresses specify the location of a computer or device on a network and domain names identify that resource. Domain names therefore operate as an abstraction layer, allowing servers to be moved to a different IP address without loss of functionality. The names consist of a top-level domain (TLD) that is prefixed by a sequence of labels, delimited by dotes. The labels divide the DNS system into zones of responsibility, where each label can be unique within that zone, but not necessarily across different zones. Each label can consist of up to 63 characters and the domain names can be up to 253 characters in total length. In contrast to IP addresses, domain names are human-meaningful and easily memorized, so DNS is a crucial component to the usability of the Internet.

The Clearnet DNS system suffers from several significant shortcomings that make it inappropriate for use by Tor hidden services. First, responses to DNS queries are not authenticated by digital signatures, so spoofing by a MITM attack is possible. Secondly, queries and responses are not encrypted, so it is easy for anyone wiretapping port 53 to correlate end-users to their activities, even if the communication to those websites is protected by TLS/HTTPS. Third, it suffers from DNS cache poisoning, in which an attacker pretends to be an authoritative server and sends incorrect or malicious records to local DNS resolvers. Finally, owning a TLD specifically for Tor hidden services (such as .tor) is prohibitively expensive. While DNS looks traditionally go through the Tor circuit and are resolved by Tor exit nodes, the shortcomings of the Clearnet DNS system make it unsuitable for our purposes.

The traditional Clearnet DNS system meets only a few of our requirements; registrations require a significant of identifiable and geographic information and thus are far from anonymous, lookups occur without encryption or signatures and are therefore neither anonymous nor privacy-enhanced, registrations are by default not publically confirmable but can be made so through use of an expensive SSL certificate from a central authority, the system is zone-based but is managed by centralized organizations and is therefore only partially distributed, and while it is very simple to use, extremely popular, and backwards compatible with TCP/IP, a Tor-specific TLD does not exist and falsifications of registrations is possible in a number of different ways. For its many security issues we therefore dismiss it as a possible solution.

5.3 Namecoin

Namecoin (NMC) is a decentralized peer-to-peer information registration and transfer system, developed by Vincent Durham in early 2011. It was the first fork of Bitcoin and as such inherits most of Bitcoin's design and capabilities. Like Bitcoin, a digital cryptocurrency created by pseudonymous developer Satoshi Nakamoto in 2008, Namecoin holds name records and transactions in a public ledger known as a blockchain. Users may hold Namecoins, and may prove ownership and authorize transactions with their private secp256k1 ECDSA key. Each transaction is a transfer of ownership of a certain amount of NMC from one public key to another, and as all transactions are recorded into the blockchain by the Namecoin network, all Namecoins can be traced backwards to the point of origin. Purchasing a name-value pair, such as a domain name, consumes 0.01 Namecoins, thus adding value and some expense to names in the system. Durham added two rules to Namecoin not present in Bitcoin: names in the blockchain expire after 36,000 blocks (about every 250 days) unless renewed by the owner and no two unexpired names can be identical. Namecoin domain names use the .bit TLD, which is not in use by ICANN.

The blockchain data structure is Nakamoto's novel answer to the problem of ensuring agreement of critical data across all involved parties. This prevents the double-spending of Bitcoins, or in Namecoin's case the prevention of duplicate names. Starting from an

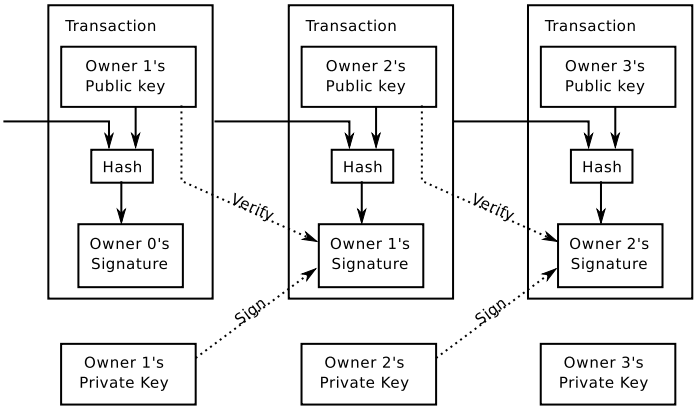

Figure 5.1: Three traditional Namecoin transactions.

initial genesis block, all blocks of data link to one another by referencing the hash of a previous block. Blocks are generated and appended to the chain at a relatively fixed rate by a process known as mining. The mining process is the solving of a proof-of-work (PoW) problem in a scheme similar to Adam Back's Hashcash: find a nonce that when passed through two rounds of SHA256 (SHA256^2) produces a value less than or equal to a target T. This requires a party to perform on average $\frac{1}{Pr[H \leq T]} = \frac{2^{256}}{T}$ amount of computations, but it is easy to verify afterwards that $\text{SHA256}^2(\text{msg}||n) \leq T$. Once the PoW is complete, the block (containing the back-reference hash, a list of transactions, and the PoW variables) is broadcasting to rest of the network, thus forming an append-only chain whose validity everyone can confirm. As each block is generated, the miner receives fresh Namecoins, thus introducing newly-minted Namecoins into the system at a fixed rate. Blocks cannot be modified retrospectively without requiring regeneration of the PoW of that block and all subsequent blocks, so the PoW locks blocks in the chain together. Nodes in the Namecoin network collectively agree to use the blockchain with the highest accumulation of computational effort, so an adversary seeking to modify the structure would need to recompute the proof-of-work for all previous blocks as well as out-perform the network, which is currently considered infeasible. [7]

Each block in the blockchain consists of a header and a payload. The header contains a hash of the previous block's header, the root hash of the Merkle tree built from the

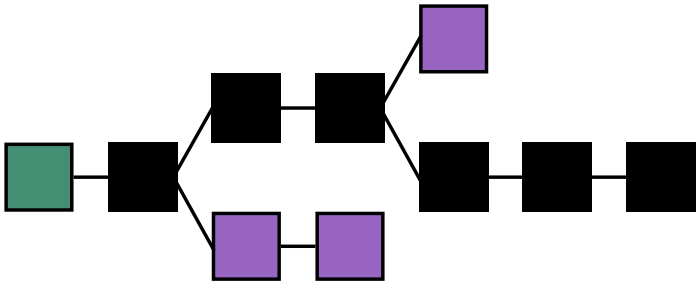

Figure 5.2: A sample blockchain.

transactions in this block, a timestamp, a target T, and a nonce. The block payload consists of a list of transactions. The root node of the Merkle tree ensures the integrity of the transaction vector: verifying that a given transaction is contained in the tree takes $log(n)$ hashes, and a Merkle tree can be built $n * log(n)$ time, ensuring that all transactions are accounted for. The hash of the previous block in the header ensures that blocks are ordered chronologically, and the Merkle root hash ensures that the transactions contained in each block are order chronologically as well. The target T changes every 2016 blocks in response to the speed at which the proof-of-work is solved such that Bitcoin miners take two weeks to generate 2016 blocks, or one block every 10 minutes. The change in target ensures that the difficulty of the proof-of-work remains relatively constant as processing capabilities increase according to Moore's Law. In the event that multiple nodes solve the proof-of-work and generate a new block simultaneously, the forked block becomes orphaned, the transactions recycled, and the network converges to follow the blockchain with the longest path from the genesis node, thus the one with the most amount of PoW behind it. [7]

Namecoin is particularly noteworthy in that it was the first systems to prove Zooko's Triangle false in a practical sense. The blockchain public ledger is held by every node in the distributed network, and human-meaningful names can be owned and placed inside it. The rules of the Namecoin network tell all participants to ensure that there are no name duplicates, and anyone holding the blockchain can verify this property. Names cannot be inserted retroactively due to the PoW, so these properties are ensured. Thus, Namecoin achieves all three properties in Zooko's Triangle. Namecoin is the most commonly-used alternative DNS system and is noted for its security and censorship resistance. In 2014,

Namecoin was recognized by ICANN as the most well-known example of a PKI and DNS system with an emphasis of distributed control and privacy, a growing trend in light of the revelations about the US National Security Agency (NSA) by Edward Snowden. [**?**]

While Namecoin is perhaps the most well-known secure distributed DNS system, it too is not applicable to Tor's environment. One of the main problems is one of practicality: it is unreasonable to require all Tor users to be able to download the entire Namecoin blockchain, which currently stands at 2.05 GB as of January 2015, [**?**] in order to know DNS records and verify the uniqueness of names. While this burden could be shifted to Tor nodes, Tor clients must then be able to trust the nodes to return an accurate record or even the correct block that contained that record. If the client queried the Namecoin network for DNS information through a Tor circuit, they would have no way of verifying the accuracy of the information without holding a complete blockchain to verify it. Secondly, the hidden service operator would have to prove that he owned both the ECDSA private key attached to the Namecoin record and the private RSA key attached to his hidden service in a manner that is both publicaly confirmable and didn't compromise his identity or location. These problems make Namecoin difficult to integrate securely into Tor. While we recognize Namecoin for its novelty and EsgalDNS shares some similarities with Namecoin, Namecoin itself is not a viable solution here.

Namecoin meets only some of our initial requirements; anonymity during registration can be met when the hidden service operator uses a Tor circuit, anonymity or privacy-enhancement during lookup can be met when the Tor client uses a Tor circuit for the query, public confirmability of the name to the hidden service is not easily met, domain name uniqueness is met by the Namecoin network but not easily proven without access to the entire blockchain, the distributed property is met by the nature of the network, simplicity is not fully met by Namecoin and further software would have be developed to accomplish this objective for Tor integration, backwards compatibility is unknown but could theoretically be met by certain designs, and resistance to malicious modifications or falsifications is met by the network but again not easily proven without access to the entire blockchain.

CHAPTER 6

SOLUTION

6.1 Overview

I propose a new DNS system for Tor hidden services, which I am calling EsgalDNS. *Esgal* is a Sindarin Elvish noun from the works of J.R.R Tolkien, meaning "veil" or "cover that hides". [8] EsgalDNS is a distributed DNS system embedded within the Tor network and powered by existing Tor nodes. It exists on top of the existing Tor hidden service infrastructure and is backwards compatible with the current hidden service protocol. EsgalDNS shares some design principles with Namecoin and other DNS systems, and its usage is analogous to the traditional Clearnet DNS system. The system supports a number of command and control operations, known as *records*, including Query, Create, Modify, Move, Renew, and Delete. All records, aside from Query, are authenticated by digital signatures to ensure that only the operator of the hidden service issued them. These commands are processed by a subset of participating Tor nodes, described below.

6.2 Components

6.2.1 Cryptographic Primitives

Our system makes use of cryptographic hash algorithms, digital signatures, and a pseudorandom number generator. As the cryptographic data within our system must persist for many years to come, we select well-established algorithms that we predict will remain strong against cryptographic analysis in the immediate future. In particular importance is the hash algorithm. We choose SHA-384 for most applications for its greater resistance to preimage, collision, and pseudo-collision attacks over SHA-256, which is itself significantly stronger than Tor's default hidden service hash algorithm, SHA-1. Like SHA-512, SHA-384

requires 80 rounds but its output is truncated to 48 bytes rather than the full 64, which saves space. For digital signatures, our default method is EMSA-PSS, (EMSA4) a probabilistic signature scheme defined by PKCS1 v2.1 and republished in 2003's RFC 3447, using a Tor node's 1024-bit RSA key with the SHA-384 digest to form the signature appendix. For signatures inside our proof-of-work scheme, we rely on $EMSA-PKCS1-v1_5$, (EMSA3) defined by 1998's RFC 2315. In contrast to EMSA-PSS, its deterministic nature prevents hidden service operators from bypassing the proof-of-work and brute-forcing the signature to validate the record. Our proof-of-work algorithm is scrypt, a key derivation function which is notable for its large memory and CPU requirements during its operation. In applications that require pseudorandom numbers from a known seed, we use the Mersenne Twister generator. In all instances the Mersenne Twister is initialized from the output of a hash algorithm, negating the generator's weakness of producing substandard random output from certain types of initial seeds.

We use the JSON format to encode records and databases of records. 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.

6.2.2 Quorum

EsgalDNS may have many participants: any Tor nodes that has sufficient storage and bandwidth capacity may perform a full synchronization and mirror the complete DNS system. However, only a subset of them must perform the main duties of the system at any given time. EsgalDNS is primarily powered by a special subset of Tor nodes, called a *quorum*. The M nodes in the quorum may be found deterministically from the signed consensus document published by the Tor authority nodes. This consensus document is generated every hour by the authority nodes and distributed to all Tor nodes and clients. We consider the document published at 00:00 GMT each day; first we take the SHA-384 of the document, then secondly initialize the Mersenne Twister pseudorandom number generator (PRNG) from the output. Third, we construct a numerical list of Tor nodes from

the consensus document, and finally scramble that list using the PRNG. The first M nodes, numbered $1..M$, become the quorum.

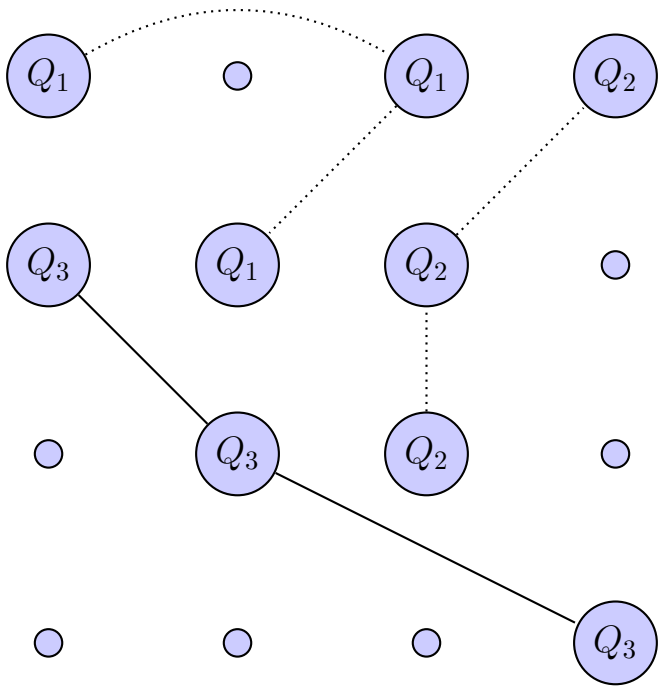

Figure 6.1: Past and present 3-member quorums in a 16-node network, day 3.

Since both clients and Tor nodes hold an authenticated and up-to-date copy of the consensus document, all involved parties are aware and agree on the members of the quorum. As we use the document at the beginning of each day, quorum nodes have an effective lifetime of 24 hours before their primary responsibilities are replaced by a new quorum. Since the health and status of the Tor network cannot be easily determined in advance, the quorum cannot be determined or known in advance until the new consensus document is published. This deterministic but unpredictably random nature makes the system more resilient to attackers attempting to force their node into the quorum for malicious purposes. Furthermore, as the digital signature from the authority nodes is embedded within the consensus document itself and all involved parties have the public keys of the authority nodes, past quorums can be securely derived from archives of the consensus document.

6.2.3 Page

A *page* is long-term JSON-encoded textual database held by quorum nodes. It contains five fields, *prevHash*, *recordList*, *consensusDocHash*, *nodeKey*, and *pageSig*. *prevHash* is the SHA-384 hash of *prevHash*, *recordList*, and *consensusDocHash* of a previous page. *recordList* is an array of records, sorted in some globally deterministic manner. *consensusDocHash* is the SHA-384 of the morning's consensus document. *nodeKey* is the public key of the quorum node, which can hashed to find the node's fingerprint in the consensus document, or used directly to verify *pageSig*, the digital signature of the preceding fields.

Each quorum node holds its page, the pages of the other quorum nodes on that day, and an archive of the consensus document. To generate its page, it first picks a page from the past quorums by the following method:

1. Of all chains of pages in the database, ignore any invalid pages and any pages that form chains using those invalid pages. A page may be invalid if it contains records that do not follow the below specifications, its *consensusDocHash* field does not match the hash of the consensus document on that day, or if *pageSig* does not verify. Invalid pages may suggest that the quorum node was not fully synchronized, that it acted maliciously, or that there was data corruption.

2. From these valid chains, find the chain that has been used by the largest number of Tor quorum nodes (not counting today's quorum) and choose the most recent page.

3. If there are multiple chains that satisfy the second condition, choose from among them the page that contains the largest amount of records.

4. If the third condition cannot be resolved, choose from among them the page that is held by the smallest i in the M quorum nodes.

It then takes the SHA-384 of *prevHash*, *recordList*, and *consensusDocHash* of that page, forming *prevHash*. Secondly, it generates *consensusDocHash* by hashing the morning's consensus document. Finally, it hashes the page and generates a digital signature of that hash, storing the result in *pageSig*. The usage of *recordList* is described below.

6.2.4 Snapshot

Similar to a page, a *snapshot* is JSON-encoded textual database held by quorum nodes, but unlike pages, snapshots are short-term and volatile. They are used for propagating very new records and receiving records from other quorum nodes, and are only held and processed by quorum nodes in the current day. Snapshots contain three fields: *originTime*, *recentRecords*, and *snapshotSig*, where *originTime* is the timestamp when the snapshot was first created, *recentRecords* is a list of DNS records, and *snapshotSig* is the digital signature of *originTime* and *recentRecords*. When a hidden service operator informs a quorum node about a new record, the quorum node first confirms that the record is valid (described below) and if it is, it adds that record into *recentRecords* and updates *snapshotSig*.

Where S_x is the current snapshot at propagation iteration x, at each 15 minute mark each active quorum node Q_j performs the following:

1. Generates a new snapshot, labelled S_{x+1}, sets *originTime* to the current time, creates *snapshotSig*, and sets S_{x+1} to be the currently active snapshot for collecting new records.

2. Randomly selects $M/2$ nodes from the M quorum nodes, defining *swapSet*.

3. With each node N_k in *swapSet*, it swaps S_x and *pageSig* with node N_k, receiving a snapshot $R_{x,k}$ and signature $Sig_{x,k}$, and then verifies the validity of $R_{x,k}$. For efficiency, if N_k in turn selects Q_i, no swap will need to take place.

4. After all swapping is complete, Q_j

   (a) Archives S_x and each $R_{x,k}$ and $Sig_{x,k}$ for all k in *swapSet*.

   (b) Updates its copy of N_k's page by merging in $R_{x-4,k}$, then confirming that $Sig_{x,k}$ verifies against the generated page. If not, it asks N_k for its page so that the discrepancy can be resolved.

   (c) Updates its page by merging the contents of S_{x-4} into *recordList* and regenerates *pageSig*.

(d) Deletes S_{x-4}.

6.2.5 Synchronization

Tor nodes must perform a full synchronization with the quorum nodes or node(s) mirroring the database before they are qualified to be picked as part of a quorum.

up-t

Tor nodes may optionally synchronize with quorum nodes. They may want to mirror the page-chain database or qualify to be a quorum node themselves.

6.2.6 Registration Query

6.2.7 Registration Creation

6.2.8 Record Modification

6.2.9 Ownership Transfer

6.2.10 Domain Renewal

6.2.11 Registration Deletion

6.2.12 Broadcast

Nodes currently in the quorum are responsible for handling transactions such as the Create, Modify, Move, Renew, and Delete commands. They can also respond to Query requests, although other nodes can also respond as well. Hidden service operators can use Tor circuits to give one of these former commands to one or more quorum nodes. The transaction then propagate to the remaining quorum nodes when they synchronize their knowledge bases. Each quorum node is responsible for distributing two databases to other quorum nodes: *pages* and *snapshots*. Pages are long-term and stable collections of records,

whereas snapshots hold volatile records that may be yet be fully propagate across the entire quorum. Pages reference pages from the previous day's quorum, forming a append-only page-chain that grows forward with time. Snapshots, by contrast, do not reference previous pages but their information is merged into the quorum node's page periodically.

A page on day n belonging to quorum member i can be expressed as $P_{n,i}$. This page contains five distinct elements: *prevN*, *prevI*, *prevHash*, *recordList*, *consensusDoc*, and *digSig*, where *prevN* and *prevI* represents the back-reference to the $P_{prevN,prevI}$ page, *prevHash* is the SHA-256 of $P_{prevN,prevI}$, *recordList* is the list of records that were send the quorum on day n, *consensusDoc* is the consensus document digitally signed by the authority nodes, and finally *digSig* is the digital signature from quorum member i on the preceding fields.

At startup, every Tor node participating in EsgalDNS

One of the central elements in my system is a *quarum*, a set of special decision-making Tor nodes. The set of committee nodes changes every day and the set cannot be known in advance.

At a high level, domain registrations are broadcasted through a Tor circuit to all committee nodes. Every node then analyses the registration as well as its knowledge of the chain and makes a decision. If the registration is invalid, the node rejects with an appropriate flag. If the domain name is already taken, the node returns a flag along with the pre-existing registration. Otherwise, it digitally signs its approval and the proposal itself and distributes this to the rest of the committee. It then waits for the rest of the committee votes. Since every Tor node has the up-to-date public keys of all other nodes due to the consensus documents, every committee member can verify the votes of all other committee members. Once the node confirms that all committee nodes received the same proposal and that a significant majority indicate that the domain is both valid and available, it adds the registration to its local storage. More importantly, the registration is recorded to an append-only endless scroll distributed within the Tor network. Thus domain names are consumed in a first-come-first-serve basis.

The scroll is a distributed and highly redundant chained data structure that slowly

rolls through the Tor network. The scroll is N by M in shape and consists of two primary components: *blocks* and *captures*. Blocks contain one or more captures, and blocks are duplicated across the M committee nodes. Each capture is a collection of information from one day; it contains the consensus document from the previous morning, a list of domain registrations approved that day, the approval sign-off digital signatures from the committee nodes on those registrations, the digital signatures from the committee indicating their approval of the integrity of the scroll, and the hash of the previous four captures. In this way, captures are fully verifiable and contain enough information to link to the previous capture, forming a chain. This chain is not held by any single Tor node, rather it is encapsulated within a rolling window of blocks N days deep. As the days progress, the captures in the oldest block are migrated to the current day's block, rolling the structure forward. Thus the scroll is divided across N blocks, with copies of each block held by M Tor nodes. I consider $N = 16, M = 64$ reasonable values, which would involve 1,024 nodes at any given time, although M can be easily changed even while the scroll is in use.

This distributed system provides the ability to confirm a given domain name is not already in use, without relying on a single central authority. Assuming that the committee nodes are honest in their vote and trustworthy in their nature, this achieves all three properties of Zooko's Triangle. Even if the committee nodes are malicious, I believe I can introduce sufficient countermeasures to make it infeasible for an attacker to successfully manipulate the system, assuming that the majority of the Tor network is trustworthy. More research, design, and implementation is needed, but this I believe is a very promising approach.

6.2.13 Domain Registration

A domain registration consists of eight components which are tied together by digital signatures and proof-of-work. The components are *nonce*, *consensusHash*, *time*, *domain*, *subdomains*, *contact*, *digSig*, and *pubKey*.

nonce

An eight-byte number that serves as a source of randomness for the proof-of-work.

consensusHash

A 32-byte value containing the SHA256 hash of the consensus document published by the authority nodes. Consensus documents are generated frequently, so *consensusHash* will be based on the document published at 00:00 GMT that day.

time

A four-byte integer holding the number of seconds since January 1st, 2013.

domain

A null-terminated cstring of the human-meaningful domain that will be correlated with the traditional .onion address of the hidden service. This can be up to 32 characters long. The TLD is .tor

subdomains

Up to 255 bytes of subdomain data, preceded by one byte that indicates the byte length. Each subdomain is null-terminated, so with the null characters 15 subdomains are possible when each is 16 characters long.

contact

16 bytes representing the last 32 base64-encoded bytes of the fingerprint of the service operator's PGP key, if they have one. If they do not, these bytes are zeroed. The purpose of this field is to allow the operator to be contacted securely.

digSig

The digital signature of all preceding fields.

pubKey

The public key of the hidden service.

To generate a registration, *domain*, *subdomains*, and *contact* are determined by the operator, while *consensusHash* and *time* are filled in automatically. The hidden service operator then has to find a value for *nonce* such that the proof-of-work is valid, specifically that that the SHA256 of *digSig* is less than a target value T. I plan to set T such that the

proof-of-works takes a significant amount of time on a modern CPU. This makes registering a domain expensive, thwarting flooding attacks. If *nonce* is found, the registration is valid and ready for broadcast.

Two common proof-of-work systems are hashing, typically double-SHA256 ($SHA256^2$) in the case of Bitcoin, and scrypt, a password-based key derivation function used by Litecoin. I chose the latter here; scrypt is a harder proof-of-work system because it requires large quantities of RAM in addition to CPU time, making brute-forcing significantly more challenging. Finding *nonce* is made even more difficult because for every *nonce*, a new digital signature must be made using the service's private key. This slows mining, complicates porting to GPUs and other specialized hardware, and prevents outsourcing to a outside computational resource. The digital signature ensures that all field are authenticated to the key of the hidden service, verifiable by all.

Once created and finalized, domains are broadcasted through Tor circuit(s) to committee nodes, where it can be approved and added into the scroll.

6.2.14 Registration Query

A client requesting *example.tor* can use a Tor circuit to anonymously query committee nodes, or in fact any node holding the scroll, for the domain name. If the domain is taken, the client will receive the full registration (as specified above) as well as the digital signatures of the committee members who voted on that registration. This consensus and the registration itself can both be validated by the client's machine. The client can then extract *pubKey* from the registration, hash it and truncate it, and look up the .onion in the traditional manner.

6.3 Fault Tolerance

Page wrapping stuff...

CHAPTER 7

ANALYSIS

general analysis...

7.1 Performance

bandwidth, CPU, RAM, latency for clients..

7.1.1 Node Load

demand on participating nodes...

7.2 Security

security breakdown...

CHAPTER 8

RESULTS

8.1 Implementation

implementation...

We use the BSD-licensed Botan library and C++11 for the hash and digest algorithms in our reference implementation, while the Mersenne Twister is implemented in C++'s Standard Template Library.

Our reference implementation uses the libjsoncpp header-only library for encoding and decoding purposes. The library is also available as the libjsoncpp-dev package in the Debian, Ubuntu, and Linux Mint repositories.

8.2 Discussion

discussions...

8.2.1 Guarantees

Guarantees...

CHAPTER 9

FUTURE WORK

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?

CHAPTER 10

CONCLUSION

I have presented EsgalDNS, a distributed DNS system inside the Tor network. The system correlates one-to-one human-meaningful domain names and traditional Tor .onion addresses. Tor nodes and clients can both verify the authenticity, integrity, and uniqueness of registrations. Although this design is nowhere near finalized, so far this system seems both promising and novel. I have more work to do, and I am planning to implement and test this system on a simulated Tor network. If accepted by the Tor community, I believe that EsgalDNS will be a valuable infrastructure that will significantly improve the usability and popularity of Tor hidden services.

REFERENCES

[1] L. Xin and W. Neng, "Design improvement for tor against low-cost traffic attack and low-resource routing attack," *2009 International Conference on Communications and Mobile Computing*, 2009.

[2] Z. Ling, J. Luo, W. Yu, X. Fuc, W. Jia, and W. Zhao, "Protocol-level attacks against tor," *Computer Networks*, 2012.

[3] D. McCoy, K. Bauer, D. Grunwald, T. Kohno, and D. Sicker, "Shining light in dark places: Understanding the tor network," *Department of Computer Science and Engineering, University of Washington, Seattle, WA 98195-2969*, 2008.

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

[5] M. S. Ferdous, A. Jsang, K. Singh, and R. Borgaonkar, "Security usability of petname systems," October 2009.

[6] M. Stiegler, "Petname systems," 2005.

[7] K. Okupski, "Bitcoin developer reference," 2014.

[8] D. Willis, "Hiswelk's sindarin dictionary, edition 1.9.1, lexicon 0.9952."