# Resisting Traffic Analysis on Unclassified Networks*

**Roger Dingledine**
The Free Haven Project
arma@freehaven.net

**Nick Mathewson**
The Free Haven Project
nickm@freehaven.net

**Catherine Meadows**
Naval Research Laboratory
meadows@itd.nrl.navy.mil

**Paul Syverson**
Naval Research Laboratory
syverson@itd.nrl.navy.mil

**Abstract**

While the need for data and message confidentiality is well known, the need to protect against traffic analysis on networks, including unclassified networks, is less widely recognized. Tor is a circuit-based low-latency anonymous communication service that resists traffic analysis. This second-generation Onion Routing system adds to the first-generation design with perfect forward secrecy, congestion control, directory servers, integrity checking, variable exit policies, and a practical design for rendezvous points. Tor works on the real-world Internet, requires no special privileges or kernel modifications, requires little synchronization or coordination between nodes, and provides a reasonable tradeoff between anonymity, usability, and efficiency.

## 1 Introduction

It is well known that encryption hides the content of communication but does nothing to hide who is communicating with whom. Indeed, Whit Diffie, an inventor of public-key cryptography, has noted that traffic analysis, not cryptanalysis, is the backbone of signals intelligence [9]. The military has many reasons to communicate over open networks without revealing its communications partners. Communicating in this way assists intelligence gathering from open Internet sources, rapid formation of dynamic coalitions without an existing shared private infrastructure between members, and private communication with vendors to help conceal procurement patterns. Finally, it is sometimes not the communicants that are sensitive but their location: a server whose physical or logical location is known may be vulnerable to physical attack and denial of service.

* This work supported by DARPA and ONR.

*Paper presented at the RTO IST Symposium on "Adaptive Defence in Unclassified Networks", held in Toulouse, France, 19 - 20 April 2004, and published in RTO-MP-IST-041.*

Onion Routing is an overlay network concept for making anonymous connections resistant to eavesdropping and traffic analysis. It permits low-latency TCP-based communication such as web traffic, secure shell remote login, and instant messaging. The current design and implementation, Tor, improves on the original [13, 16, 18, 19] by providing perfect forward secrecy (see Section 2), interfacing to unmodified applications via SOCKS, multiplexing application connections on Onion Routing circuits, adding congestion control adding integrity checking, and including a rendezvous points design that protects the responder of a connection in addition to the initiator.

Onion Routing may be used anywhere traffic analysis is a concern. Because Onion Routing is an overlay network, it can exist on top of public networks such as the Internet without any modification to the underlying routing structure or protocols. In addition to protecting data confidentiality and integrity, the Onion Routing protocol hides the endpoint of each transmission. An intelligence analyst surfing a web site through Onion Routing is hidden both from that web site and from the Onion Routing network itself. On the other hand, Onion Routing separates anonymity of the communication from that of the data stream. That is, a procurement officer can place orders with a vendor and completely authenticate himself to the vendor while still hiding the communication from any observers—including compromised Onion Routing network components. Onion Routing can also be used to provide location hidden servers with better protection and yet less redundancy than standard approaches to distributed denial of service. In this paper we provide a brief overview of the Tor design. More detailed description is given in [10]. As we describe the system design, we will note how Onion Routing can be used to protect military communications in the above described settings.

## 1.1 Related Work

We give here a broad description of prior work; for a more complete list of references and comparisons, see [10].

Modern anonymity systems date to Chaum's **Mix-Net** design [5]. Chaum proposed hiding the correspondence between sender and recipient by wrapping messages in layers of public-key cryptography, and relaying them through a path composed of "mixes." Each mix in turn decrypts, delays, and re-orders messages, before relaying them toward their destinations.

Subsequent relay-based anonymity designs have diverged in two main directions. Some have tried to maximize anonymity at the cost of introducing comparatively large and variable latencies. Because of this decision, these *high-latency* networks resist strong global adversaries, but introduce too much lag for interactive tasks like web browsing, Internet chat, or SSH connections.

Tor belongs to the second category: *low-latency* designs that try to anonymize interactive network traffic. These systems handle a variety of bidirectional protocols. They also provide more convenient mail delivery than the high-latency anonymous email networks, because the remote mail server provides explicit and timely delivery confirmation. But because these designs typically involve many packets that must be delivered quickly, it is difficult for them to prevent an attacker who can eavesdrop both ends of the communication from correlating the timing and volume of traffic entering

the anonymity network with traffic leaving it. These protocols are also vulnerable to active attacks in which an adversary introduces timing patterns into traffic entering the network and looks for correlated patterns among exiting traffic. Although some work has been done to frustrate these attacks, most designs protect primarily against traffic analysis rather than traffic confirmation (cf. Section 2.1).

The simplest low-latency designs are single-hop proxies such as the Anonymizer [2], wherein a single trusted server strips the data's origin before relaying it. More complex are distributed-trust, circuit-based anonymizing systems. In these designs, a user establishes one or more medium-term bidirectional end-to-end circuits, and tunnels data in fixed-size cells. Establishing circuits is computationally expensive and typically requires public-key cryptography, whereas relaying cells is comparatively inexpensive and typically requires only symmetric encryption. Because a circuit crosses several servers, and each server only knows the adjacent servers in the circuit, no single server can link a user to her communication partners.

There are many other circuit-based designs, that make a variety of design choices; we again refer the reader to [10] for more information.

# 2 Design goals and assumptions

## Goals

Like other low-latency anonymity designs, Tor seeks to frustrate attackers from linking communication partners, or from linking multiple communications to or from a single user. Within this main goal, however, several considerations have directed Tor's evolution.

**Diversity:** If all onion routers were operated by the defense department or ministry of a single nation and all users of the network were DoD users, then traffic patterns of individuals, enclaves, and commands can be protected from hostile observers, whether external or internal. However, any traffic emerging from the Onion Routing network to the Internet would still be recognized as coming from the DoD, since the network would only carry DoD traffic. Therefore, it is necessary that the Onion Routing network carry traffic of a broader class of users. Similarly, having onion routers run by diverse entities, including nonmilitary entities and entities from different countries, will help broaden and enlarge the class of users who will trust that system insiders will not monitor their traffic. This will provide both a greater diversity and greater volume of cover traffic. Unlike confidentiality, a single entity cannot achieve anonymity without collaboration, no matter how strong the technology.

**Deployability:** The design must be deployed and used in the real world. Thus it must not be expensive to run (for example, by requiring more bandwidth than onion router operators are willing to provide); must not place a heavy liability burden on operators (for example, by allowing attackers to implicate onion routers in illegal activities); and must not be difficult or expensive to implement (for example, by requiring kernel patches, or separate proxies for every protocol). We also cannot require non-anonymous parties (such as websites) to run our software.

**Usability:** A hard-to-use system has fewer users—and because anonymity systems hide users among users, a system with fewer users provides less anonymity. Usability

is thus not only a convenience: it is a security requirement [1, 3]. Tor should therefore not require modifying applications; should not introduce prohibitive delays; and should require users to make as few configuration decisions as possible. Finally, Tor should be easily implemented on all common platforms; we cannot require users to change their operating system in order to be anonymous. (The current Tor implementation runs on Windows and assorted Unix clones including Linux, FreeBSD, and MacOS X.)

**Flexibility:** The protocol must be flexible and well-specified, so Tor can serve as a test-bed for future research. Many of the open problems in low-latency anonymity networks, such as generating dummy traffic or preventing Sybil attacks (where one entity masquerades as many) [11], may be solvable independently from the issues solved by Tor. Hopefully future systems will not need to reinvent Tor's design. (But note that while a flexible design benefits researchers, there is a danger that differing choices of extensions will make users distinguishable. Experiments should be run on a separate network.)

**Simple design:** The protocol's design and security parameters must be well-understood. Additional features impose implementation and complexity costs; adding unproven techniques to the design threatens deployability, readability, and ease of security analysis. Tor aims to deploy a simple and stable system that integrates the best accepted approaches to protecting anonymity.

## Non-goals

In favoring simple, deployable designs, we have explicitly deferred several possible goals, either because they are solved elsewhere, or because they are not yet solved.

**Not peer-to-peer:** Tarzan and MorphMix aim to scale to completely decentralized peer-to-peer environments with thousands of short-lived servers, many of which may be controlled by an adversary. This approach is appealing, but still has many open problems, such as greater effects of Sybil attacks and of greater network dynamics [12, 17].

**Not secure against end-to-end attacks:** We do not claim that Tor provides a definitive solution to end-to-end attacks, such as correlating the timing of connections opening or correlating when users are on the system with when certain traffic is observed (also known as intersection attacks). Some approaches may help, for example, accessing the network only through your own onion router; see [10] for more discussion.

**No protocol normalization:** Tor does not provide *protocol normalization* like Privoxy [15] or the Anonymizer [2]. In other words, Tor anonymizes the channel, but not the data or applications that pass over it. This means that Tor in itself will not hide, for example, a web surfer from being identified by the data or application protocol information observed at a visited web site. If anonymization from the responder is desired for complex and variable protocols like HTTP, Tor must be layered with a filtering proxy such as Privoxy to hide differences between clients, and expunge protocol features that leak identity. Note that by this separation Tor can also provide services that are anonymous to the network yet authenticated to the responder, like SSH. So, for example, road warriors can make authenticated connections to their home systems without revealing this to anyone including the local network access point. Similarly, Tor does not currently integrate tunneling for non-stream-based protocols like UDP;

this too must be provided by an external service.

**Not steganographic:** Tor does not try to conceal who is connected to the network from someone in a position to observe that connection.

### 2.1 Threat Model

A global passive adversary is the most commonly assumed threat when analyzing theoretical anonymity designs. But like all practical low-latency systems, Tor does not protect against such a strong adversary. Instead, we assume an adversary who can observe some fraction of network traffic; who can generate, modify, delete, or delay traffic; who can operate onion routers of its own; and who can compromise some fraction of the onion routers.

In low-latency anonymity systems that use layered encryption, the adversary's typical goal is to observe both the initiator and the responder. By observing both ends, passive attackers can confirm a suspicion that Alice is talking to Bob if the timing and volume patterns of the traffic on the connection are distinct enough; active attackers can induce timing signatures on the traffic to force distinct patterns. Rather than focusing on these *traffic confirmation* attacks, we aim to prevent *traffic analysis* attacks, where the adversary uses traffic patterns to learn which points in the network he should attack.

Our adversary might try to link an initiator Alice with her communication partners, or try to build a profile of Alice's behavior. He might mount passive attacks by observing the network edges and correlating traffic entering and leaving the network—by relationships in packet timing, volume, or externally visible user-selected options. The adversary can also mount active attacks by compromising routers or keys; by replaying traffic; by selectively denying service to trustworthy routers to move users to compromised routers, or denying service to users to see if traffic elsewhere in the network stops; or by introducing patterns into traffic that can later be detected. The adversary might subvert the directory servers to give users differing views of network state. Additionally, he can try to decrease the network's reliability by attacking nodes or by performing antisocial activities from reliable servers and trying to get them taken down; making the network unreliable flushes users to other less anonymous systems, where they may be easier to attack.

## 3 Highlights of the Tor Design

The Tor network is an overlay network; each onion router (OR) runs as a normal user-level process without any special privileges. Each onion router maintains a long-term TLS [8] connection to every other onion router. Using TLS conceals the data on the connection with perfect forward secrecy (see below), and prevents an attacker from modifying data on the wire or impersonating an OR. Each user runs local software called an onion proxy (OP) to fetch directories, establish circuits across the network, and handle connections from user applications. These onion proxies accept TCP streams and multiplex them across the circuits. The onion router on the other side of the circuit connects to the destinations of the TCP streams and relays data.

Traffic passes along these connections in fixed-size cells. Each cell is 512 bytes, and consists of a header and a payload. The header includes a circuit identifier (circID) that specifies which circuit the cell refers to (many circuits can be multiplexed over each TLS connection), and a command to describe what to do with the cell's payload. (Circuit identifiers are connection-specific: each single circuit has a different circID on each OP/OR or OR/OR connection it traverses.) Based on their command, cells are either *control* cells, which are always interpreted by the node that receives them, or *relay* cells, which carry end-to-end stream data.

Relay cells have an additional header (the relay header) after the cell header, containing a stream identifier (many streams can be multiplexed over a circuit); an end-to-end checksum for integrity checking; the length of the relay payload; and a relay command. The entire contents of the relay header and the relay cell payload are encrypted or decrypted together as the relay cell moves along the circuit, using the 128-bit AES cipher in counter mode to generate a cipher stream.

In Tor, just as each connection can be shared by many circuits, each circuit can be shared by many application-level TCP streams. To avoid delays, users construct circuits preemptively. To limit linkability among their streams, users' OPs build a new circuit periodically if the previous one has been used, and expire old used circuits that no longer have any open streams. OPs consider making a new circuit once a minute: thus even heavy users spend negligible time building circuits, but a limited number of requests can be linked to each other through a given exit node. Also, because circuits are built in the background, OPs can recover from failed circuit creation without delaying streams (which would harm user experience).

The full Tor design paper [10] describes the Onion Routing protocol in detail; we highlight a few of its properties here:

- **Perfect forward secrecy:** Onion Routing was originally vulnerable to a single hostile node recording traffic and later compromising successive nodes in the circuit and forcing them to decrypt it. Rather than using a single multiply encrypted data structure (an *onion*) to lay each circuit, Tor now uses an incremental or *telescoping* path-building design, where the initiator negotiates session keys with each successive hop in the circuit. Once these keys are deleted, subsequently compromised nodes cannot decrypt old traffic. As a side benefit, onion replay detection is no longer necessary, and the process of building circuits is more reliable, since the initiator knows when a hop fails and can then try extending to a new node.

- **Leaky-pipe circuit topology:** Through in-band signaling within the circuit, Tor initiators can direct traffic to nodes partway down the circuit. This novel approach allows traffic to exit the circuit from the middle—possibly frustrating traffic shape and volume attacks based on observing the end of the circuit. (It also allows for long-range padding if future research shows this to be worthwhile.)

- **End-to-end integrity checking:** The original Onion Routing design did no integrity checking on data. Any node on the circuit could change the contents of data cells as they passed by—for example, to alter a connection request so

it would connect to a different webserver, or to 'tag' encrypted traffic and look for corresponding corrupted traffic at the network edges [7]. Tor hampers these attacks by checking data integrity before it leaves the network.

- **Improved robustness to failed nodes:** A failed node in the old design meant that circuit building failed, but thanks to Tor's step-by-step circuit building, users notice failed nodes while building circuits and route around them. Additionally, liveness information from directories allows users to avoid unreliable nodes in the first place.
- **Congestion control:** Even with bandwidth rate limiting, we still need to worry about congestion, either accidental or intentional. If enough users choose the same OR-to-OR connection for their circuits, that connection can become saturated. For example, an attacker could send a large file through the Tor network to a webserver he runs, and then refuse to read any of the bytes at the webserver end of the circuit. Without some congestion control mechanism, these bottlenecks can propagate back through the entire network. We don't need to reimplement full TCP windows (with sequence numbers, the ability to drop cells when we're full and retransmit later, and so on), because TCP already guarantees in-order delivery of each cell. Tor provides both circuit and stream level throttling.

# 4 Other design decisions

## 4.1 Resource management and denial-of-service

Providing Tor as a public service creates many opportunities for denial-of-service attacks against the network. While flow control and rate limiting prevent users from consuming more bandwidth than routers are willing to provide, opportunities remain for users to consume more network resources than their fair share, or to render the network unusable for others. We discuss some of these in [10].

## 4.2 Exit policies and abuse

Exit abuse is a serious barrier to wide-scale Tor deployment. Anonymity presents would-be vandals and abusers with an opportunity to hide the origins of their activities. Attackers can harm the Tor network by implicating exit servers for their abuse. Also, applications that commonly use IP-based authentication (such as institutional mail or webservers) can be fooled by the fact that anonymous connections appear to originate at the exit OR.

We stress that Tor does not enable any new class of abuse. Spammers and other attackers already have access to thousands of misconfigured systems worldwide, and the Tor network is far from the easiest way to launch antisocial or illegal attacks. But because the onion routers can easily be mistaken for the originators of the abuse, and the volunteers who run them may not want to deal with the hassle of repeatedly explaining anonymity networks, we must block or limit the abuse that travels through the Tor network.

To mitigate abuse issues, in Tor, each onion router's *exit policy* describes to which external addresses and ports the router will connect. This is described further in [10].

Finally, we note that exit abuse must not be dismissed as a peripheral issue: when a system's public image suffers, it can reduce the number and diversity of that system's users, and thereby reduce the anonymity of the system itself. Like usability, public perception is a security parameter. Sadly, preventing abuse of open exit nodes is an unsolved problem, and will probably remain an arms race for the foreseeable future. The abuse problems faced by Princeton's CoDeeN project [14] give us a glimpse of likely issues.

### 4.3 Directory Servers

First-generation Onion Routing designs [4, 16] used in-band network status updates: each router flooded a signed statement to its neighbors, which propagated it onward. But anonymizing networks have different security goals than typical link-state routing protocols. For example, delays (accidental or intentional) that can cause different parts of the network to have different views of link-state and topology are not only inconvenient: they give attackers an opportunity to exploit differences in client knowledge, by observing induced differences in client behavior. We also worry about attacks to deceive a client about the router membership list, topology, or current network state. Such *partitioning attacks* on client knowledge help an adversary to efficiently deploy resources against a target [7].

Tor uses a small group of redundant, well-known onion routers to track changes in network topology and node state, including keys and exit policies. Each such *directory server* acts as an HTTP server, so participants can fetch current network state and router lists, and so other ORs can upload state information. Onion routers periodically publish signed statements of their state to each directory server. The directory servers combine this state information with their own views of network liveness, and generate a signed description (a *directory*) of the entire network state. Client software is preloaded with a list of the directory servers and their keys, to bootstrap each client's view of the network. More details are provided in [10].

Using directory servers is simpler and more flexible than flooding. Flooding is expensive, and complicates the analysis when we start experimenting with non-clique network topologies. Signed directories can be cached by other onion routers, so directory servers are not a performance bottleneck when we have many users, and do not aid traffic analysis by forcing clients to periodically announce their existence to any central point.

## 5 Rendezvous points and hidden services

Rendezvous points are a building block for *location-hidden services* (also known as *responder anonymity*) in the Tor network. Location-hidden services allow Bob to offer a TCP service, such as a webserver, without revealing its IP address. This type of anonymity protects against distributed DoS attacks: attackers are forced to attack the onion routing network as a whole rather than just Bob's IP address.

Our design for location-hidden servers has the following goals. **Access-controlled:** Bob needs a way to filter incoming requests, so an attacker cannot flood Bob simply by making many connections to him. **Robust:** Bob should be able to maintain a long-term pseudonymous identity even in the presence of router failure. Bob's service must not be tied to a single OR, and Bob must be able to tie his service to new ORs. **Smear-resistant:** A social attacker who offers an illegal or disreputable location-hidden service should not be able to "frame" a rendezvous router by making observers believe the router created that service. **Application-transparent:** Although we require users to run special software to access location-hidden servers, we must not require them to modify their applications.

We provide location-hiding for Bob by allowing him to advertise several onion routers (his *introduction points*) as contact points. He may do this on any robust efficient key-value lookup system with authenticated updates, such as a distributed hash table (DHT) like CFS [6][1] Alice, the client, chooses an OR as her *rendezvous point*. She connects to one of Bob's introduction points, informs him of her rendezvous point, and then waits for him to connect to the rendezvous point. This extra level of indirection helps Bob's introduction points avoid problems associated with serving unpopular files directly (for example, if Bob serves material that the introduction point's community finds objectionable, or if Bob's service tends to get attacked by network vandals). The extra level of indirection also allows Bob to respond to some requests and ignore others.

### 5.1 Integration with user applications

Bob configures his onion proxy to know the local IP address and port of his service, a strategy for authorizing clients, and a public key. Bob publishes the public key, an expiration time ("not valid after"), and the current introduction points for his service into the DHT, indexed by the hash of the public key. Bob's webserver is unmodified, and doesn't even know that it's hidden behind the Tor network.

Alice's applications also work unchanged—her client interface remains a SOCKS proxy. We encode all of the necessary information into the fully qualified domain name Alice uses when establishing her connection. Location-hidden services use a virtual top level domain called `.onion`: thus hostnames take the form `x.y.onion` where `x` is the authorization cookie, and `y` encodes the hash of the public key. Alice's onion proxy examines addresses; if they're destined for a hidden server, it decodes the key and starts the rendezvous as described above.

## 6 Future Directions

Tor brings together many innovations into a unified deployable system. The next immediate steps include:

*Scalability:* Tor's emphasis on deployability and design simplicity has led us to adopt a clique topology, semi-centralized directories, and a full-network-visibility model

[1]Rather than rely on an external infrastructure, the Onion Routing network can run the DHT itself. At first, we can run a simple lookup system on the directory servers.

for client knowledge. These properties will not scale past a few hundred servers. The Tor design paper [10] describes some promising approaches, but more deployment experience will be helpful in learning the relative importance of these bottlenecks.

*Bandwidth classes:* This paper assumes that all ORs have good bandwidth and latency. We should instead adopt the MorphMix model, where nodes advertise their bandwidth level (DSL, T1, T3), and Alice avoids bottlenecks by choosing nodes that match or exceed her bandwidth. In this way DSL users can usefully join the Tor network.

*Incentives:* Volunteers who run nodes are rewarded with potentially better anonymity, and those who value the notoriety can be rewarded with publicity [1]. More nodes means increased scalability, and more users can mean more anonymity. We need to continue examining the incentive structures for participating in Tor.

*Padding Cover traffic:* Currently Tor omits padding for cover traffic—its costs in performance and bandwidth are clear but its security benefits are not well understood. We must pursue more research on link-level cover traffic and long-range cover traffic to determine whether some simple padding method offers provable protection against our chosen adversary.

*Caching at exit nodes:* Perhaps each exit node should run a caching web proxy, to improve anonymity for cached pages (Alice's request never leaves the Tor network), to improve speed, and to reduce bandwidth cost. On the other hand, forward security is weakened because caches constitute a record of retrieved files. We must find the right balance between usability and security.

*Better directory distribution:* Clients currently download a description of the entire network every 15 minutes. As the state grows larger and clients more numerous, we may need a solution in which clients receive incremental updates to directory state. More generally, we must find more scalable yet practical ways to distribute up-to-date snapshots of network status without introducing new attacks.

*Implement location-hidden services:* The design in Section 5 has not yet been implemented. While doing so we are likely to encounter additional issues that must be resolved, both in terms of usability and anonymity.

*Further specification review:* Although we have a public byte-level specification for the Tor protocols, it needs extensive external review. We hope that as Tor is more widely deployed, more people will examine its specification.

*Multisystem interoperability:* We are currently working with the designer of MorphMix to unify the specification and implementation of the common elements of our two systems. So far, this seems to be relatively straightforward. Interoperability will allow testing and direct comparison of the two designs for trust and scalability.

*Wider-scale deployment:* The original goal of Tor was to gain experience in deploying an anonymizing overlay network, and learn from having actual users. As of writing there is a distributed network of roughly a dozen nodes. We are now at a point in design and development where we can start deploying a wider network. Once we have many actual users, we will doubtlessly be better able to evaluate some of our design decisions, including our robustness/latency tradeoffs, our performance tradeoffs (including cell size), our abuse-prevention mechanisms, and our overall usability.

## References

[1] Alessandro Acquisti, Roger Dingledine, and Paul Syverson. On the economics of anonymity. In Rebecca N. Wright, editor, *Financial Cryptography*. Springer-Verlag, LNCS 2742, 2003.

[2] The Anonymizer. `http://anonymizer.com/`.

[3] Adam Back, Ulf Möller, and Anton Stiglic. Traffic analysis attacks and trade-offs in anonymity providing systems. In Ira S. Moskowitz, editor, *Information Hiding (IH 2001)*, pages 245–257. Springer-Verlag, LNCS 2137, 2001.

[4] Philippe Boucher, Adam Shostack, and Ian Goldberg. Freedom systems 2.0 architecture. White paper, Zero Knowledge Systems, Inc., December 2000.

[5] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 4(2), February 1981.

[6] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with CFS. In *18th ACM Symposium on Operating Systems Principles (SOSP '01)*, Chateau Lake Louise, Banff, Canada, October 2001.

[7] George Danezis, Roger Dingledine, and Nick Mathewson. Mixminion: Design of a type III anonymous remailer protocol. In *2003 IEEE Symposium on Security and Privacy*, pages 2–15. IEEE CS, May 2003.

[8] T. Dierks and C. Allen. The TLS Protocol — Version 1.0. IETF RFC 2246, January 1999. `http://www.rfc-editor.org/rfc/rfc2246.txt`.

[9] Whitfield Diffie and Susan Landau. *Privacy On the Line: The Politics of Wiretapping and Encryption*. MIT Press, 1998.

[10] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. Submitted for publication. Available at `http://freehaven.net/tor/`.

[11] John Douceur. The Sybil Attack. In *Proceedings of the 1st International Peer To Peer Systems Workshop (IPTPS 2002)*, March 2002.

[12] Michael J. Freedman and Robert Morris. Tarzan: A peer-to-peer anonymizing network layer. In *9th ACM Conference on Computer and Communications Security (CCS 2002)*, Washington, DC, November 2002.

[13] David M. Goldschlag, Michael G. Reed, and Paul F. Syverson. Hiding routing information. In R. Anderson, editor, *Information Hiding, First International Workshop*, pages 137–150. Springer-Verlag, LNCS 1174, May 1996.

[14] Vivek S. Pai, Limin Wang, KyoungSoo Park, Ruoming Pang, and Larry Peterson. The Dark Side of the Web: An Open Proxy's View. Submitted to HotNets-II. `http://codeen.cs.princeton.edu/`.

[15] Privoxy. `http://www.privoxy.org/`.

[16] Michael G. Reed, Paul F. Syverson, and David M. Goldschlag. Anonymous connections and onion routing. *IEEE Journal on Selected Areas in Communications*, 16(4):482–494, May 1998.

[17] Marc Rennhard and Bernhard Plattner. Practical anonymity for the masses with morphmix. In Ari Juels, editor, *Financial Cryptography*. Springer-Verlag, LNCS (forthcoming), 2004.

[18] Paul Syverson, Michael Reed, and David Goldschlag. Onion Routing access configurations. In *DARPA Information Survivability Conference and Exposition (DISCEX 2000)*, volume 1, pages 34–40. IEEE CS Press, 2000.

[19] Paul Syverson, Gene Tsudik, Michael Reed, and Carl Landwehr. Towards an Analysis of Onion Routing Security. In H. Federrath, editor, *Designing Privacy Enhancing Technologies: Workshop on Design Issue in Anonymity and Unobservability*, pages 96–114. Springer-Verlag, LNCS 2009, July 2000.