Anonymous Connections and Onion Routing

Michael G. Reed, Paul F. Syverson, and David M. Goldschlag *
Naval Research Laboratory

Abstract

Onion Routing is an infrastructure for private communication over a public network. It provides anonymous connections that are strongly resistant to both eavesdropping and traffic analysis. Onion routing's anonymous connections are bidirectional and near real-time, and can be used anywhere a socket connection can be used. Any identifying information must be in the data stream carried over an anonymous connection. An onion is a data structure that is treated as the destination address by onion routers; thus, it is used to establish an anonymous connection. Onions themselves appear differently to each onion router as well as to network observers. The same goes for data carried over the connections they establish. Proxy aware applications, such as web browsing and e-mail, require no modification to use onion routing, and do so through a series of proxies. A prototype onion routing network is running between our lab and other sites. This paper describes anonymous connections and their implementation using onion routing. This paper also describes several application proxies for onion routing, as well as configurations of onion routing networks.

1 Introduction

Is Internet communication private? Most security concerns focus on preventing eavesdropping [18], i.e., outsiders listening in on electronic conversations. But encrypted messages can still be tracked, revealing who is talking to whom. This tracking is called traffic analysis and may reveal sensitive information. For example, the existence of inter-company collaboration may be confidential. Similarly, e-mail users may not wish to reveal who they are communicating with to the rest of the world. In certain cases anonymity may be desirable also: anonymous e-cash is not very anonymous if delivered with a return address. Web based shopping or browsing of public databases should not require revealing one's identity.

This paper describes how a freely available system, *onion routing*, can be used to protect a variety of Internet services against both eavesdropping and traffic analysis attacks, from both the network and outside observers. This paper includes a specification sufficient to guide both re-implementations and new applications of onion routing. We also discuss configurations of onion routing networks and applications of onion routing, including Virtual Private Networks (VPN), Web browsing, e-mail, remote login, and electronic cash.[1]

A purpose of traffic analysis is to reveal who is talking to whom. The *anonymous connections* described here are designed to be resistant to traffic analysis, i.e., to make it difficult for observers to learn identifying information from the connection (e.g., by reading packet headers, tracking encrypted payloads, etc.). Any identifying information must be passed as data through the anonymous connections. Our implementation of anonymous connections, onion routing, provides protection against eavesdropping as a side effect. Onion routing provides bidirectional and near real-time communication similar to TCP/IP socket connections or ATM AAL5 [6]. The anonymous connections can substitute for sockets in a wide variety of unmodified Internet applications by means of proxies. Data may also be passed through a privacy filter before being sent over an anonymous connection. This removes identifying information from the data stream, to make communication anonymous too.

Although onion routing may be used for anonymous communication, it differs from anonymous remailers [7, 16] in two ways: Communication is real-time and bidirectional, and the anonymous connections are application independent. Onion routing's anonymous

*Address: (For Reed and Syverson) Naval Research Laboratory, Center For High Assurance Computer Systems, Washington, D.C. 20375-5337, USA, phone: +1 202.767.2389, fax: +1 202.404.7942, e-mail: {*last_name*}@itd.nrl.navy.mil. (For Goldschlag) Divx, 570 Herndon Parkway, Herndon, VA 20170, USA, phone: +1 703-708-4028. fax: +1 703-708-4088, e-mail: david.goldschlag@divx.com

[1] Preliminary versions of portions of this paper have appeared in [28, 14, 24].

connections can support anonymous mail as well as other applications. For example, onion routing may be used for anonymous Web browsing. A user may wish to browse public Web sites without revealing his identity to those Web sites. That requires removing information that identifies him from his requests to Web servers and removing information from the connection itself that may identify him. Hence, anonymous Web browsing uses anonymized communication over anonymous connections. The Anonymizer [1] only anonymizes the data stream, not the connection itself. So it does not prevent traffic analysis attacks like tracking data as it moves through the network.

This paper is organized in the following way: Section 2 presents an overview of onion routing. Section 3 presents empirical data about our prototype. Section 4 defines our threat model. Section 5 describes onion routing and the application specific proxies in more detail. Section 6 describes the implementation choices that were made for security reasons. Section 7 describes how onion routing may be used in a wide variety of Internet applications. Section 8 contrasts onion routing with related work, and section 9 presents concluding remarks.

2 Onion Routing Overview

In onion routing, instead of making socket connections directly to a responding machine, initiating applications make connections through a sequence of machines called *onion routers*. The *onion routing network* allows the connection between the *initiator* and *responder* to remain anonymous. Anonymous connections hide who is connected to whom, and for what purpose, from both outside eavesdroppers and compromised onion routers. If the initiator also wants to remain anonymous to the responder, then all identifying information must be removed from the data stream before being sent over the anonymous connection.

Onion routers in the network are connected by longstanding (permanent) socket connections. Anonymous connections through the network are multiplexed over the longstanding connections. For any anonymous connection, the sequence of onion routers in a route is strictly defined at connection setup. However, each onion router can only identify the previous and next hops along a route. Data passed along the anonymous connection appears different at each onion router, so data cannot be tracked en route, and compromised onion routers cannot cooperate by correlating the data stream each sees. We will also see that they cannot make use of replayed onions or replayed data.

2.1 Operational Overview

The onion routing network is accessed via a series of *proxies*. An initiating application makes a socket connection to an *application proxy*. This proxy massages connection message format (and later data) to a generic form that can be passed through the onion routing network. It then connects to an *onion proxy*, which defines a route through the onion routing network by constructing a layered data structure called an *onion*. The onion is passed to the *entry funnel*, which occupies one of the longstanding connections to an onion router and multiplexes connections to the onion routing network at that onion router. That onion router will be the one for whom the outermost layer of the onion is intended. Each layer of the onion defines the next hop in a route. An onion router that receives an onion peels off its layer, identifies the next hop, and sends the embedded onion to that onion router. The last onion router forwards data to an *exit funnel*, whose job is to pass data between the onion routing network and the responder.

In addition to carrying next hop information, each onion layer contains key seed material from which keys are generated for crypting[2] data sent forward or backward along the anonymous connection. (We define *forward* to be the direction in which the onion travels and *backward* as the opposite direction.)

Once the anonymous connection is established, it can carry data. Before sending data over an anonymous connection, the onion proxy adds a layer of encryption for each onion router in the route. As data moves through the anonymous connection, each onion router removes one layer of encryption, so it arrives at the responder as plaintext. This layering occurs in the reverse order for data moving back to the initiator. So data that has passed backward through the anonymous connection must be repeatedly post-crypted to obtain the plaintext.

By layering cryptographic operations in this way, we gain an advantage over link encryption. As data moves through the network it appears different to each onion router. Therefore, an anonymous connection is as strong as its strongest link, and even one honest node is enough to maintain the privacy of the route. In link encrypted systems, compromised nodes can cooperate to uncover route information.

Onion routers keep track of received onions until they expire. Replayed or expired onions are not forwarded, so they cannot be used to uncover route information, either by outsiders or compromised onion

[2] We define the verb *crypt* to mean the application of a cryptographic operation, be it encryption or decryption.

routers. Note that clock skew between onion routers can only cause an onion router to reject a fresh onion or to keep track of processed onions longer than necessary. Also, since data is encrypted using stream ciphers, replayed data will look different each time it passes through a properly operating onion router.

Although we call this system onion routing, the routing that occurs here does so at the application layer of the protocol stack and not at the IP layer. More specifically, we rely upon IP routing to route data passed through the longstanding socket connections. An anonymous connection is comprised of portions of several linked longstanding multiplexed socket connections. Therefore, although the series of onion routers in an anonymous connection is fixed for the lifetime of that anonymous connection, the route that data actually travels between individual onion routers is determined by the underlying IP network. Thus, onion routing may be compared to loose source routing.

Onion routing depends upon connection based services that deliver data uncorrupted and in-order. This simplifies the specification of the system. TCP socket connections, which are layered on top of a connectionless service like IP, provide these guarantees. Similarly, onion routing could easily be layered on top of other connection based services, like ATM AAL5.

Our current prototype of onion routing considers the network topology to be static and does not have mechanisms to automatically distribute or update public keys or network topology. These issues, though important, are not the key parts of onion routing and will be addressed in a later prototype.

2.2 Configurations

As mentioned above neighboring onion routers are neighbors in virtue of having longstanding socket connections between them, and the network as a whole is accessed from the outside through a series of proxies. By adjusting where those proxies reside it is possible to vary which elements of the system are trusted by users and in what way. (For some configurations it may be efficient to combine proxies that reside in the same place, thus they may be only conceptually distinct.)

2.2.1 Firewall Configuration

In the *firewall configuration*, an onion router sits on the firewall of a sensitive site. This onion router serves as an interface between machines behind the firewall and the external network. Connections from machines behind the firewall to the onion router are protected by other means (e.g., physical security). To complicate tracking of traffic originating or terminating within the sensitive site, this onion router should also route data between other onion routers. This configuration might represent the system interface from a typical corporate or government site. Here the application proxies (together with any privacy filters), and the onion proxies would typically live at the firewall as well. (Typically, there might only be one onion proxy.)

There are three important features of this basic configuration:

- Connections between machines behind onion routers are protected against both eavesdropping and traffic analysis. Since the data stream never appears in the clear on the public network, this data may carry identifying information, but communication is still private. (This feature is used in section 7.1.)

- The onion router at the originating protected site knows both the source and destination of a connection. This protects the anonymity of connections from observers outside the firewall but also simplifies enforcement of and monitoring for compliance with corporate or governmental usage policy.

- The use of anonymous connections between two sensitive sites that both control onion routers effectively hides their communication from outsiders. However, if the responder is not in a sensitive site (e.g., the responder is some arbitrary Web server) the data stream from the sensitive initiator must also be anonymized. If the connection between the exit funnel and the responding server is unencrypted, the data stream might otherwise identify the initiator. For example, an attacker could simply listen in on the connections to a Web server and identify initiators of any connection to it.

2.2.2 Remote Proxy Configuration

What happens if an initiator does not control an onion router? If the initiator can make encrypted connections to some remote onion router, then he can function as if he is in the firewall configuration just described, except that both observers and the network can tell when he makes connections to the onion router. However, if the initiator trusts the onion router to build onions, his association with the anonymous connection from that onion router to the responder is hidden from observers and the network. In a similar way, an encrypted connection from an exit funnel to a responder hides the association of the responder with the anonymous connection .

Therefore, if an initiator makes an anonymous connection to some responder, and layers end-to-end encryption over that anonymous connection, the initiator and responder can identify themselves to one another, yet hide their communication from the rest of the world. So we can build virtual private networks without protected sites.

Notice, however, that the initiator trusts the remote onion router to conceal that the initiator wants to communicate with the responder, and to build an anonymous connection through other onion routers. The next section describes how to shift some of this trust from the first onion router to the initiator.

2.2.3 The Customer–ISP Configuration

Suppose, for example, an Internet Services Provider (ISP) runs a funnel that accepts connections from onion proxies running on subscribers' machines. In this configuration, users generate onions specifying a path through the ISP to the destination. Although the ISP would know who initiates the connection, the ISP would not know with whom the customer is communicating, nor would it be able to see data content. So the customer need not trust the ISP to maintain her privacy. Furthermore, the ISP becomes a *common carrier*, who carries data for its customers. This may relieve the ISP of responsibility both for whom users are communicating with and the content of those conversations. The ISP may or may not be running an onion router as well. If he is running an onion router, then it is more difficult to identify connections that terminate with his customers; however, he is serving as a routing point for other traffic. On the other hand, if he simply runs a funnel to an onion router elsewhere, it will be possible to identify connections terminating with him, but his overall traffic load will be less. Which of these would be the case for a given ISP would probably depend on a variety of service, cost, and pricing considerations. Note that in this configuration the entry funnel must have an established longstanding connection to an onion router just like any neighboring onion router. (Cf. section 5.6 for a description of how these are established.) But, in most other cases, where the funnel resides on the same machine as the onion router, establishing an encrypted longstanding connection should not be necessary since the funnel can be directly incorporated into the onion router.

3 Empirical Data

We invite readers to experiment with our prototype of onion routing by using it to anonymously surf the Web, send anonymous e-mail, and do remote logins. For instructions please see `http://www.itd.nrl.navy.mil/ITD/5540/projects/onion-routing/`.

One should be aware that accessing a remote onion router does not completely preserve anonymity, because the connection between a remote machine and the first onion router is not protected. If that connection were protected, one would be in the remote proxy configuration, but there would would still be no reason to trust the remote onion router. If one had a secured connection to an onion router one trusted, our onion router could be used as one of several intermediate routers to further complicate traffic analysis.

We have recently set up a thirteen node distributed network of government, academic, and private sites. However, at press time we have not yet gathered performance data for this network. The data we present are for a network running on a single machine. In our experimental onion routing network, five onion routers run on a single Sun Ultra 2 2170. This machine has two 167 MHz processors, and 256MB of memory. Anonymous connections are routed through a random sequence of five onion routers. Connection setup time should be comparable to a more distributed topology. Data latency, however, is more difficult to judge. Clearly, data will travel faster over socket connections between onion routers on the same machine than over socket connections between different machines. However, on a single machine the removal or addition of layers of encryption is not pipelined, so data latency may be worse.

Onion routing's overhead is mainly due to public key cryptography and is incurred while setting up an anonymous connection. On our Ultra 2 running a fast implementation of RSA [2], a single public key decryption of a 1024 bit plaintext block using a 1024 bit private key and a 1024 bit modulus takes 90 milliseconds. Encryption is much faster, because the public keys are only 16 bits long. (This is why RSA signature verification is cheaper than signing). So, the public key cryptographic overhead for routes spanning five onion routers is just under 0.5 seconds. This overhead can be further reduced, either with specialized hardware, or even simply on different hardware (a 200 MHz Pentium would be almost twice as fast).

In practice, our connection setup overhead does not appear to add intolerably to the overhead of typical socket connections. Still, it can be further reduced.

There is no reason that the same anonymous connection could not be used to carry the traffic for several 'real' socket connections, either sequentially or multiplexed. In fact, the specification for HTTP 1.1 defines pipelined connections to amortize the cost of socket setup, and pipelined connections would also transparently amortize the increased cost of anonymous connection setup. We are currently updating our Web proxy to be HTTP 1.1 compliant.

4 Threat Model

This section outlines our threat model. It does not intend to quantify the cost of attacks, but to define possible attacks. Future work will quantify the threat. First some vocabulary. A session is the data carried over a single anonymous connection. Data is carried in fixed length cells. Since these cells are multiply encrypted and change as they move through an anonymous connection, tracking cells is equivalent to tracking markers that indicate when cells begin. In a marker attack, the attacker identifies the set of outbound connections that some distinguished marker may have been forwarded upon. By intersecting these sets for a series of distinguished markers belonging to the same session, an attacker may determine, or at least narrow, the set of possible next hops. In a timing attack, the attacker records a timing signature for a session that correlates data rate over time. A session may have a very similar timing signature wherever it is measured over a route, so cooperating attackers may determine if they carry a particular session.

We assume that the network is subject to both passive and active attacks. Traffic may be monitored and modified by both external observers and internal network elements, including compromised onion routers. Attackers may cooperate and share information and inferences. We assume roving attackers that can monitor part, but not all, of the network at a time.

Our goal is to prevent traffic analysis, not traffic confirmation. If an attacker wants to confirm that two endpoints often communicate, and he observes that they each connect to an anonymous connection at roughly the same time, more often than is statistically expected, it is reasonable to infer that the endpoints are indeed communicating. Notice that this attack is infeasible if endpoints live in protected networks behind trusted onion routers on firewalls.

If the onion routing infrastructure is uniformly busy, then passive external attacks are ineffective. Specifically, neither the marker nor timing attacks are feasible, since external observers cannot assign markers to sessions. Active attacks are possible since reducing the load on the system makes the network easier to analyze (and makes the system not uniformly busy).

Passive internal attacks require at least two compromised onion routers. Since onion routers can assign markers to a session, both the marker and timing attacks are possible. Specifically, timing signatures can be broadcast, and other compromised onion routers can attempt to find connections with matching timing signatures.

Another attack that is only feasible as an internal attack is the volume attack. Compromised onion routers can keep track of the number of cells that have passed over any given anonymous connection. They can then simply broadcast totals to other compromised onion routers. Cell totals that are close to the same amount at the same time at different onion routers are likely to belong to the same anonymous connection.[3]

Active internal attacks amplify these risks, since individual onion routers can selectively limit traffic on particular connections. An onion router could, for example, force a particular timing signature on a connection, and advertise that signature.

5 Onion Routing Specifics

5.1 Onion Routing Proxies

A proxy is a transparent service between two applications that would usually make a direct socket connection to each other but cannot. For example, a firewall might prevent direct socket connections between internal and external machines. A proxy running on the firewall may enable such connections. Proxy aware applications are becoming quite common.

Our goal has been to design an architecture for private communication that would interface with *unmodified* applications, so we chose to use proxies as the interface between applications and onion routing's anonymous connections. For applications that are designed to be proxy aware, (e.g., WWW browsers), we simply design appropriate interface proxies. Surprisingly, for certain applications that are not proxy aware (e.g., RLOGIN), we have also been able to design interface proxies.

Because it is necessary to bridge between applications and the onion routing network, proxies must understand both application protocols and onion routing protocols. Therefore, we modularize the design into components: the application proxy, the onion proxy, and the entry funnel. The application proxy bridges between a socket connection from an application and

[3] Thanks to Gene Tsudik for noting this attack and for helpful discussions.

a socket connection to the onion proxy. It is the obligation of the application proxy to massage the data stream so the onion proxy, the entry funnel and the exit funnel can be application independent. Specifically, the application proxy must prepend to the data stream a *standard structure* that identifies the ultimate destination by either hostname/port or IP address/port. Additionally, it must process a one byte return code from the exit funnel and either continue if no error is reported or report the onion routing error code in some application specific meaningful way. The application proxy may also contain an optional privacy filter for sanitizing the data stream.

Upon receiving a new request, the onion proxy builds an onion defining the route of an anonymous connection. (It may use the destination address in the prepended structure to help define the route.) It then passes the onion to the funnel, and repeatedly precrypts the standard structure. Finally, it passes the precrypted standard structure through the anonymous connection to the exit funnel, thus specifying the ultimate destination. From this point on, the onion proxy blindly relays data back and forth between the application proxy and the onion routing network (and thus the exit funnel at the other end of the anonymous connection). Of course, it must apply the appropriate keystreams to incoming and outgoing data when blindly relaying data.

The entry funnel multiplexes connections from onion proxies to the onion routing network. For the services we have considered to date, a nearly generic exit funnel is adequate. Its function is to demultiplex connections from the last onion router to the outside. When it reads a data stream from the terminating onion router the first datum received will be the standard structure specifying the ultimate destination. The exit funnel makes a socket connection to that IP address/port, reports a one byte status message back to the onion routing network (and thus back to the onion proxy which in turn forwards it back to the application proxy), and subsequently moves data between the onion routing network and the new socket. (For certain services, like RLOGIN, the exit funnel also infers that the new socket must originate from a trusted port.) Entry and exit funnels are not application specific but must understand the onion routing protocol, which defines how multiplexed connections are handled.

As an example, consider the application proxy for HTTP. The user configures his browser to use the onion routing proxy. His browser may send the proxy a request like
`GET http://www.domino.com/showcase/ HTTP/1.0`
followed by optional fields.

The application proxy is listening for new requests. Once it obtains the `GET` request, it creates the standard structure and sends it (along a new socket connection) to the onion proxy, to inform the onion proxy of the service and destination of the anonymous connection. The application proxy then modifies the `GET` request to `GET /showcase/ HTTP/1.0` and sends it directly (through the anonymous connection) to the HTTP server `www.domino.com`, followed by the optional fields. Notice that the server name and `http://` are eliminated from the `GET` request because the connection is made directly to the HTTP server.

The application proxy essentially makes a connection to `www.domino.com`, and issues a request as if it were a client. Once this request is transmitted to the server, all proxies blindly forward data in both directions between the client and the server until the socket is broken by either side.

For the anonymizing onion routing HTTP proxy, the application proxy proceeds as outlined above with one change: it is now necessary to sanitize the optional fields that follow the `GET` command because they may contain identity information. Furthermore, the data stream during a connection must be monitored, to sanitize additional headers that might occur during the connection. For our current anonymizing HTTP proxy, operations that store cookies on the user's browser (to track a user, for example) are removed. This reduces function, so applications that depend upon cookies (like online shopping baskets) may not work properly.

5.2 Implementation

This section presents the interface specification between the components in an onion routing system. To provide some structure to this specification, we will discuss components in the order that data would move from an initiating client to a responding server.

There are four phases in an onion routing system: network setup, which establishes the longstanding connections between onion routers; connection setup, which establishes anonymous connections through the onion router network; data movement over an anonymous connection; and the destruction and cleanup of anonymous connections. We will commingle the discussion of these below.

5.3 Application Proxy

The interface between an application and the application proxy is application specific. The interface between the application proxy and the onion proxy is defined as follows. For each new proxy request, the application proxy first determines if it will handle or deny the request. If rejected, it reports an application specific error message and then closes the socket and waits for the next request. If accepted, it creates a socket to the onion proxy's well known port. The application proxy then sends a standard structure to the onion proxy of the form:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Version    |    Protocol   |  Retry Count  |  Addr Format  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Version is currently defined to be 1. *Protocol* is either 1 for RLOGIN, 2 for HTTP, or 3 for SMTP. *Retry Count* specifies how many times the exit funnel should attempt to retry connecting to the ultimate destination. Finally, the *Addr Format* field specifies the form of the ultimate destination address: 1 for a NULL terminated ASCII string with the hostname or IP address (in ASCII form) immediately followed by another NULL terminated ASCII string with the destination port number, and all others currently undefined. The ultimate destination address is sent after this standard structure, and the application proxy waits for a one byte error code before sending data.

5.4 Onion Proxy

Upon receiving the standard structure, the onion proxy can decide whether to accept or reject the request based on the protocol, destination host, destination port, or the identity of the application proxy. If rejected, it sends an appropriate error code back to the application proxy, closes the socket, and waits for the next request. If accepted, it proceeds to build the onion and connects to the entry funnel of the first onion router, through the network, and to the exit funnel of the last. It next sends the standard structure to the exit funnel over the anonymous connection, and then passes all future data to and from the application proxy and anonymous connection. The repeated pre and post cryptions and packaging of the standard structure and subsequent data is discussed later in section 5.6.

5.5 Onions

To build the anonymous connection to the exit funnel, the onion proxy creates an onion. An onion is a multi-layered data structure that encapsulates the route of the anonymous connection starting from the onion router for that exit funnel and working backward to the onion router at the entry funnel.

Each layer has the following structure:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|0| Version     |Back  F|Forw  F|       Destination  Port       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Destination Address                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Expiration Time (GMT)                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                       Key Seed Material                       +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

As we will see below, the first bit must be zero for RSA public key cryptography to succeed. Following the zero bit is the *Version Number* of the onion routing system, currently defined to be 1.

The *Back F* field denotes the cryptographic function to be applied to data moving in the backward direction (defined as data moving in the direction opposite that which the onion traveled, usually toward the initiator's end of the anonymous socket connection) using key_2 defined below. The *Forw F* field denotes the cryptographic function to be applied to data moving in the forward direction (defined as data moving in the same direction as that which the onion traveled, usually toward the responder's end of the anonymous socket connection) using key_3 defined below. Currently defined cryptographic functions are: 0 for Identity (no encryption), 1 for DES OFB (output feedback mode) (56 bit key), and 2 for RC4 (128 bit key). The *Destination Address* and *Destination Port* indicate the next onion router in network order and are both 0 for the exit funnel. The *Expiration Time* is given in network order in seconds relative to 00:00:00 UTC January 1, 1970 (i.e., standard UNIX time(2) format) and specifies how long the onion router at this hop in the anonymous connection must track the onion against replays before it expires. *Key Seed Material* is 128 bits long and is hashed three times with SHA to produce three cryptographic keys (key_1, key_2, and key_3) of 128-bits each (the first eight bytes of each SHA output are used for DES and the first 16 bytes for RC4 keys).[4]

Since we use RSA public key cryptography with a modulus size of 1024-bits, the plaintext block size is 1024 bits and must be strictly less than the modulus numerically. To avoid problems, we force this relation by putting the most-significant bit first and setting it to 0 (the leading 0 above). Furthermore, the innermost layer of the onion is padded on the end with an additional 100 bytes prior to RSA encryption being

[4] Details on the cryptographic operations used in this paper can be found in [20, 26].

performed.

In version 1, an onion has five layers, but routes can be shorter. An onion is formed iteratively, innermost layer first. At each iteration, the first 128 bytes of the onion are encrypted with the public key of the onion router that is intended to decrypt that layer. The remainder of the onion is encrypted, using DES OFB with an IV (initialization vector) of 0 and key_1 (derived from *Key Seed Material* in that layer as defined above).[5]

Before discussing how onions and data are sent between onion routers, we will define onion router interconnection.

5.6 Onion Router Interconnection

During onion network setup (not to be confused with anonymous connection setup), longstanding connections between neighboring onion routers are established and keyed. The network topology is predefined and each onion router knows its neighbors.

To remain connected to each of its neighbors, onion routers must both listen for connections from neighbors and attempt to initiate connections to neighbors. To avoid deadlock and collision issues between pairs of neighbors, an onion router listens for connections from neighbors with "higher" IP/port addresses and initiates connections to neighbors with "lower" IP/port addresses. "Higher" and "Lower" are defined with respect to network byte ordering. (This was an expedient way to break symmetry. Ultimately we will want a more flexible solution. For example, when an onion router goes down, it should contact its neighbors upon coming back up. Requiring the neighbors to try to contact the down router until it responds is less efficient. This is not difficult to implement and we will do so in the future.)

The protocol has two phases: connection setup and keying. The initiating onion router opens a socket to a well known port of its neighboring onion router, and sends its IP address and well known port (the port is included to allow multiple onion routers to run on a single machine) in network order to identify itself. The keying phase ensues, using STS [9] which will generate two DES 56-bit keys. The link encryption over the longstanding connections is done by DES OFB with IVs of 0 and these two keys (one for data in each direction).

Once keyed, communication between onion routers is packaged into fixed sized *cells*, which allows for the multiplexing of both anonymous connections and control information over the longstanding connections. (Cell size was chosen to be compliant with ATM.) In version 1 of the onion routing system, there are four types of cells: PADDING (0), CREATE (1), DATA (2), and DESTROY (3).

Cells have the following structure:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|              ACI              |    Command    |    Length     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
......................Payload  (44 bytes)......................
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The *ACI* (anonymous connection identifier) and *Command* fields are always encrypted using the link encryption between neighboring nodes. Additionally, the *Length* and *Payload* fields are encrypted using the link encryption between neighboring nodes if the command is either PADDING (0) or DESTROY (3). For CREATE (1) commands, the length is link encrypted, but the payload is already encrypted because it carries the onion. For DATA (2) commands, the length and entire payload are encrypted using the anonymous connection's forward or backward cryptographic operations.

Each anonymous connection is assigned an ACI at each onion router, which labels an anonymous connection when it is multiplexed over the longstanding connection to the next onion router. ACIs must be unique on their longstanding connection but need not be globally unique.

To move an onion through the system, an onion router peels off the outermost layer, identifying the next hop. It checks the freshness (not expired and not replayed) of the onion, computes the necessary cryptographic keys, initializes the forward and backward cryptographic engines, chooses a new ACI for the next hop in the new connection, and then builds a data structure associated with that connection which maps incoming to outgoing ACIs and the cryptographic engines associated with forward and backward data. Since neighboring onion routers choose ACIs for each other on the thick pipe that they share, each is assigned half of the naming space. The neighboring onion router with a "higher address" chooses ACIs in the top half of that space, while its neighbor with the lower address chooses ACIs from the bottom half of that space. After the outermost layer of onion is peeled off, the rest of the onion is padded randomly to its original length, placed into CREATE cells, and then sent out in order to the appropriate neighbor. The payload of the last cell is padded with random bits to fill the cell if necessary (to

[5]We use DES to encrypt the onion, and for link encryption between onion routers, because it has no licensing fees and can be used as a pseudorandom number generator. However, we would be happy to use a stronger pseudorandom number generator.

avoid traceability).

Data moves through an anonymous connection in DATA cells. At each onion router both the length and payload fields of a cell are crypted using the appropriate cryptographic engine. The new cell is sent out to the appropriate neighbor. The onion proxy must repeatedly crypt data to either add the appropriate layers of cryption on outgoing data, or remove layers of cryption from incoming data. When constructing a DATA cell from a plaintext data stream, the cell is (partially) filled, its true length is set, and all 45 bytes of the length and payload fields are repeatedly crypted using the stream ciphers defined by the onion. Therefore, when the cell arrives at the exit funnel, the length field reflects the length of the actual data carried in the payload.

If a connection is broken, a DESTROY command is sent to clean up state information. The ACI field of the DESTROY command carries the ACI of the broken connection. The length and payload must be random. Upon receipt of a DESTROY command, it is the responsibility of an onion router to forward the DESTROY appropriately and to acknowledge receipt by sending another DESTROY command back to the previous sender. After sending a DESTROY command about a particular ACI, an onion router may not send any more cells along that anonymous connection. Once an acknowledgment DESTROY message is received, an onion routing node considers the anonymous connection destroyed and the ACI can be used as a label for a new anonymous connection.

The PADDING command is used to inject data into a longstanding socket to further confuse traffic analysis. PADDING cells are discarded upon receipt.

Each onion router also reorders cells moving through it. All cells that arrive at an onion router within a fixed interval of time on any connection are mixed pseudorandomly, except that the order of cells in each anonymous connection is preserved.

5.7 Exit Funnel

When a routing node receives an onion with *Destination Address* and *Destination Port* of 0, it knows it is the terminal onion router for the connection and passes the connection not to another onion router but to its own exit funnel. The funnel proceeds to read the standard structure that will be the first data across the anonymous socket connection, establishes a connection to the ultimate destination as indicated, and returns the status code. After this, it will blindly forward data between the anonymous connection and the connection to the responder's machine.

6 Implementation Vulnerabilities

An implementation of a secure design can be insecure. In this section, we describe several implementation decisions that were made for security considerations.

Onions are packaged in a sequence of cells that must be processed together. This onion processing involves a public key decryption operation which is relatively expensive. Therefore, it is possible to imagine an implementation that clears outgoing queues while an onion is being processed, and then outputs the onion. Therefore, any period of inactivity on the out-bound queues is likely to be followed by a sequence of onion cells being output on a single queue. Such an implementation makes tracking easier and should be avoided.

After processing at each onion router, onions are padded at the end to compensate for the removed layer. This padding must be random, since onions are not link encrypted between onion routers. Similarly, the length and payload of a DESTROY command must be new random content at each onion router; otherwise, compromised onion routers could track that payload.

In a multi-threaded implementation, there is a significant lure to rely upon apparent randomness in scheduling to reorder events. If reordering is important to the secure operation of the system, deliberate reordering is crucial, since low level system randomness may in fact be predictable.

There are two vulnerabilities for which we do not have good solutions. If part of the onion routing network is taken down, traffic analysis may be simplified. Also, if a longstanding connection between two onion routers is broken, it will result in many DESTROY messages, one for each anonymous connection that was routed through that longstanding connection. Therefore, a compromised onion router may infer from near simultaneous DESTROY messages that the associated anonymous connections had some common route. Delaying DESTROY messages hurts performance, since we require that a DESTROY message propagate to the endpoints to take down the connection that is visible to the user. Carrying the DESTROY message through the anonymous connection and garbage collecting dormant anonymous connections later would be ideal, but we do not know how to efficiently insert control information into a raw data channel, especially considering our layered encryption. One possibility is for the onion router on the initiator side of a break to send some large predetermined number of one bits back to the initiator followed by a message that the connection is destroyed. The onion proxy could then check for such a signal after it strips off each layer of each packet, and notify the application proxy if it receives the signal. The initiator can contact the responder out of band, presumably through another anonymous connection, authenticate itself by some means as the initiator of the broken connection, and notify the responder of the break. Onion routers can either be notified directly by the onion proxy after some random delay or possibly garbage collect least recently used ACIs. We will continue to explore the feasibility of this and other possibilities.[6]

7 Applications

We first describe how to use anonymous connection in VPNs, anonymous chatting services, and anonymous cash. We then describe onion routing proxies for three Internet services: Web browsing, e-mail, and remote logins. These three onion routing proxies have been implemented. Anonymizing versions of these proxies that remove the identifying information that may be present in the headers of these services' data streams have been implemented as well.

7.1 Virtual Private Networks

If two sites wanted to collaborate, they could establish one or more long term tunnels that would multiplex many socket connections, or even raw IP packets, over a single anonymous connection. This would effectively hide who is collaborating with whom and what they are working on, without requiring the construction of an individual anonymous connection for each connection made. Such long term anonymous connections between enclaves provide the analog of a leased line over a public network. Note that the protection provided a VPN by onion routing is broader than that provided by encrypting firewalls. Basic encrypting firewalls encrypt payloads only. Thus, they protect confidentiality, but do nothing to protect against traffic analysis. IPSEC will protect traffic for individual connections by encapsulating packets in encrypted packets from the firewall, but this will not protect against institutional level traffic analysis. Communication between two such firewalls will still indicate a collaboration between the sites behind them. Constant padding may be added, but this is very expensive. And, unless many unrelated sites agree to do it, it still does not hide the existence of the VPN established between those sites that are so padding.

[6]Thanks to Gene Tsudik for some of the fundamental elements of this proposal.

7.2 Anonymous Chatting

Anonymous connections can be used in a service similar to IRC, where many parties meet to *chat* at some central server. The chat server may mate several anonymous connections carrying matching tokens. Each party defines the part of the connection leading back to itself, so no party has to trust the other to maintain its privacy. If the communicating parties layer end-to-end encryption over the mated anonymous connections, they also prevent the central server from listening in on the conversation.

7.3 Anonymous Cash

Certain forms of e-cash are designed to be anonymous and untraceable, unless they are double spent or otherwise misused. However, if a customer cannot contact a vendor without identifying himself, the anonymity of e-cash is undermined. For transactions where both payment and product can be conveyed electronically, anonymous connections can be used to hide the identities of the parties from one another [27].

How can the customer be prevented from taking his purchase without paying for it (e.g., by closing the connection early) or the vendor be prevented from taking the customer's e-cash without completing the transaction? This is a hard problem [12, 4]. In the case of a well known vendor, a practical solution is to require customers to pay first. The vendor is unlikely to deliberately cheat its customers since it may be caught in an audit.

7.4 Remote Login

We proxy remote login requests by taking advantage of the option `-l username` to `rlogin`. The usual `rlogin` command is of the form:

`rlogin -l` *username server*

To use `rlogin` through an onion routing proxy, one would type

`rlogin -l` *username@server proxy*

where *proxy* refers to the onion routing proxy to be used and both *username* and *server* are the same as specified above. A normal rlogin request is transmitted from a privileged port on the client to the well known port for rlogin (513) on the server as:

`\0` *username on client* `\0` *username on server* `\0` *terminal type* `\0`

where *username on client* is the username of the individual invoking the command on the client machine, *username on server* is either the `-l` field (if specified) or the username of the individual invoking the command on the client machine (if no `-l` is specified), and the *terminal type* is a standard termcap/linespeed specification. The server responds with a single zero byte if it will accept the connection or breaks the socket connection if an error has occurred or the connection is rejected. Our normal rlogin proxy therefore receives the initial request:

`\0` *username on client* `\0` *username@server* `\0` *terminal type* `\0`

The proxy creates an anonymous connection to the RLOGIN port on the *server* machine and proceeds to send it a massaged request of the form:

`\0` *username* `\0` *username* `\0` *terminal type* `\0`

Once this request is transmitted to the server, the proxy blindly forwards data in both directions between the client and server until the socket is broken by either side.

Notice that the onion router does not send the *server* the client's username on the client, so communication is anonymous, unless the data-stream subsequently reveals more information.

7.5 Web Browsing

Proxying HTTP requests follows the IETF HTTP V1.0 Specification [3]. An HTTP request from a client through an HTTP proxy is of the form:

```
GET http://www.server.com/file.html HTTP/1.0
```

followed by optional fields. Notice that an HTTP request from a client to a server is of the form:

```
GET file.html HTTP/1.0
```

also followed by optional fields. The server name and protocol scheme are missing, because the connection is made directly to the server.

As an example, a complete request from Netscape Navigator to an onion router HTTP proxy may look like this:

```
GET http://www.server.com/file.html HTTP/1.0
Referer: http://www.server.com/index.html
Proxy-Connection: Keep-Alive
User-Agent: Mozilla/3.0 (X11; I; SunOS 5.4 sun4m)
Host: www.server.com
Accept: image/gif, image/x-xbitmap, image/jpeg
```

The proxy must create an anonymous connection to `www.server.com`, and issue a request as if it were a client. Therefore, the request must be massaged to remove the server name and scheme, and transmitted to `www.server.com` over the anonymous connection. Once this request is transmitted to the server, the proxy blindly forwards data in both directions between the client and server until the socket is broken by either side.

For privacy filtering of HTTP, the proxy proceeds as outlined above with one change. It is now necessary to sanitize the optional fields that follow the `GET` command because they may contain identity information. Furthermore, the data stream during a connection must be monitored, to sanitize additional headers that might occur during the connection.

The *Anonymizer* [1] also provides anonymous Web browsing. Users can connect to servers through the Anonymizer and it strips off identifying headers. This is essentially what our filtering HTTP proxy does. But packets can still be tracked and monitored. The Anonymizer could be used as a front end to the onion routing network to provide effective protection against traffic analysis. We discuss this further in section 8.

7.6 Electronic Mail

Electronic mail is proxied by utilizing the `user%host@proxy` form of e-mail address instead of the normal `user@host` form. This form should work with most current and older mail systems. Under this form, the client contacts the proxy server's well known SMTP port (25). Instead of the normal mail daemon listening to that port, the proxy listens and interprets what it receives following a strict state machine: wait for a valid `HELO` command, wait for a valid `MAIL From:` command, and then wait for a valid `RCPT To:` command. Each command argument is temporarily buffered. Once the `RCPT To:` command has been received, the proxy proceeds to create an anonymous connection to the destination server and relays the `HELO` and `MAIL From:` commands exactly as received. The `RCPT To:` command is massaged and forwarded. Any subsequent `RCPT To:` commands are rejected. Once the `DATA` request is transmitted to the server, the proxy forwards data in both directions from the client and server. An example of e-mail from `joe@sender.com` on the machine `sender.com` to `mary@recipient.com` via the `onion.com` onion router is given below. Joe types `mail mary%recipient.com@onion.com`. First the communications from the client on `sender.com` to the onion router SMTP proxy on `onion.com` is given, followed by the communications from the exit funnel to `recipient.com`:

```
220 onion.com SMTP Onion Routing Network.
```

```
HELO sender.com
250-onion.com -- Connection from
250 sender.com (2.0.0.1).
MAIL From: joe@sender.com
250 Sender is joe@sender.com.
RCPT To: mary%recipient.com@onion.com
```

The proxy massages the RCPT To: line to make the address mary@recipient.com and makes an anonymous connection to recipient.com. It then replays the massaged protocol to recipient.com:

```
220-recipient.com Sendmail 4.1/SMI-4.1 ready
220 at Wed, 28 Aug 96 15:15:00 EDT
HELO Onion.Routing.Network
250-recipient.com Hello Onion.Routing.Network
250 [2.0.0.5], pleased to meet you
MAIL From: joe@sender.com
250 joe@sender.com... Sender ok
RCPT To: mary@recipient.com
250 mary@recipient.com... Recipient ok
DATA
354 Enter mail, end with "." on a line by itself
```

At this point, the proxy forwards data in both directions, until a line containing only a period is sent from the sender to the recipient:

```
This is a note
.
```

The proxy forwards the line containing only a period to the recipient, and forwards the recipient's response to the sender. At that point, the proxy sends QUIT to the recipient, reads the response and closes the connection to the recipient. The proxy then waits for a command from the sender; if that command is QUIT, the proxy sends a response and closes its connection to the sender:

```
250 Mail accepted
QUIT
221 onion.com Service closing transmission channel.
```

If the command is not QUIT, then it is MAIL, and the protocol repeats. Anything else prompts an error response, and the proxy waits for the next correct command.

For the privacy filtered proxying of electronic mail, the proxy proceeds as outlined above with a few changes. It is now necessary to sanitize both the MAIL From: command and the header portion of the actual message body. Sanitization of the MAIL From: command is trivial with a simple substitution of anonymous for joe@sender.com. For the header sanitization, we have taken the conservative approach of deleting all headers, but this may be modified in the future to only remove identifying information and leave the remaining header information intact.

8 Comparisons with Related Work

Chaum [5] defines a layered object that routes data through intermediate nodes, called *mixes*. These intermediate nodes may reorder, delay, and pad traffic to complicate traffic analysis. In mixes, the assumption is that a single perfect mix adequately complicates traffic analysis, but a sequence of multiple mixes is typically used because real mixes are not ideal. Because of this, mix applications can use mixes in fixed order, and often do. Onion routers differ from mixes in at least two ways: onion routers are more limited in the extent to which they delay traffic at each node because of the real-time expectations that the applications demand of socket connections. Also, in a typical onion routing configuration, onion routers are also entry points to the onion routing network, and traffic entering or exiting at those nodes may not be visible. This makes it hard to track packets, because they may drop out of the network at any node, and new packets may be introduced at each node. While onion routing cannot delay traffic to the extent that mixes can, traffic between onion routers is multiplexed over a single channel and is link encrypted with a stream cipher. This makes it hard to parse the stream.

Anonymous remailers like Penet [17] strip headers from received mail and forward it to the intended recipient. They may also replace the sender's address with some alias, permitting replies. These sorts of remailers store sensitive state: the mapping between the alias and the true return address. Also, mail forwarded through a chain of remailers may be tracked because it appears the same to each remailer.

Mix based remailers like [7, 16] use mixes to provide anonymous e-mail services. Essentially, the mail message is carried in the innermost layer of the onion data structure. Another onion type structure, used for a return address, can be contained in the message. This makes the return path self contained, and the remailer essentially stateless. Onion routing shares many structures with Babel [16] but it uses them to build (possibly long lived) application independent connections. This makes anonymous connections accessible to a wide variety of applications. For application to e-mail it has both advantages and disadvantages. Onion routing's service makes an anonymous connection directly to the recipient's sendmail daemon. A disadvantage is that, since the connection is made in real-time, there is less freedom in mixing, which therefore might not be done as well. An advantage is that the anonymous connection is separated from the application, so anonymous e-mail systems are considerably simplified because the application specific part does not have to move data through the network. Furthermore, because the onion routing network can carry many types of data, it has the potential to be more heavily utilized than a network that is devoted only to e-mail. Heavy utilization is the key to anonymity.

In [10], a structure similar to an onion is used to forward individual IP packets through a network. By maintaining tracking information at each router, ICMP error messages can be moved back along the hidden route. Essentially, a connection is built for each packet in a connectionless service. Although a followup paper [11] suggests that performance will be good, especially with hardware based public key cryptography, our experience suggests that both the cryptographic overhead of building onions and the tracking of onions against replay is not efficiently done on a packet-by-packet basis. However, it is easy to imagine an onion routing proxy that collects IP packets and forwards them over some anonymous connection. In this way, communication is anonymous at the IP layer, but connections need not be built for each IP packet. This anonymous IP communication may be more robust than our current architecture: it could survive a broken anonymous connection, since IP does not expect reliable delivery.

In [22], mixes are used to provide untraceable communication in an ISDN network. Here is a summary of that paper. In a phone system, each telephone line is assigned to a particular local switch (i.e., local exchange), and switches are interconnected by a (long distance) network. Anonymous calls in ISDN rely upon an anonymous connection between the caller and the long distance network. These connections are made anonymous by routing calls through a predefined series of mixes within each switch. The long distance endpoints of the connection are then mated to complete the call. (Notice that observers can tell which local switches are connected.) Also, since each phone line has a control circuit connection to the switch, the switch can broadcast messages to each line using these control circuits. So, within a switch a truly anonymous connection can be established: A phone line makes an anonymous connection to some mix. That mix broadcasts a token identifying itself and the connection. A recipient of that token can make another anonymous connection to the specified mix, which mates the two connections to complete the call.

Our goal of anonymous connections over the Internet differs from anonymous remailers and anonymous ISDN. The data is different, with real-time constraints more severe than mail, but somewhat looser than voice. Both HTTP and ISDN connections are bidirectional, but, unlike ISDN, HTTP connections are likely to be small requests followed by short bursts of returned data. Most importantly, the network topology of the Internet is more akin to the network topology of the long distance network between switches, where capacity is a shared resource. In anonymous ISDN, the mixes hide communication within the local switch, but connections between switches are not hidden. This implies that all calls between two businesses, each large enough to use an entire switch, reveal which businesses are communicating. In onion routing, mixing is dispersed throughout the Internet, which improves hiding.

Pipe-net [8] is a proposal similar to onion routing. It has not been implemented, however. Pipe-net's threat model is more paranoid than onion routing's: it attempts to resist active attacks by global observers. For example, Pipe-net's connections carry constant traffic (to resist timing signature attacks) and disruptions to any connection are propagated throughout the network.

The Anonymizer is a Web proxy that filters the HTTP data stream to remove a user's identifying information, essentially as our filtering HTTP proxy does. For example, the Anonymizer will "strip out all references to your e-mail address, computer type, and previous page visited before forwarding your request" [1]. This makes Web browsing private in the absence of any eavesdropping or traffic analysis. The Anonymizer is vulnerable in three ways: First, it must be trusted. Second, traffic between a browser and the Anonymizer is sent in the clear, so that traffic identifies the true destination of a query, and includes the identifying information that the Anonymizer would filter. Third, even if traffic between the browser and the Anonymizer were encrypted, passive external observers could mount the volume attack mentioned in section 4. The Anonymizer, however, is now readily available to everyone on the Web.

NetAngels [21] is similar to the Anonymizer, except that it builds personal profiles of its subscribers and targets advertisements to match the profile. However, the profile is not released to the advertiser and is deleted when a subscription is canceled. Subscribers must trust NetAngels, and connections to the service are subject to the same attacks as the Anonymizer.

LPWA [19, 13] (formerly known as Janus) is a "proxy server that generates consistent untraceable aliases for you that enable you to browse the Web, register at web sites and open accounts, and be 'recognized' upon returning to your accounts, all while still *preserving your privacy.*" Like the previous two, the LPWA proxy is at a server that is remote from the user application. It is thus subject to the same trust and vulnerability limitations.

It is possible, however, to shift trusted elements to the user's machine (or to a machine on the boundary between his trusted LAN and the Internet). Shifting trust in this way can improve the security of other privacy services like the Anonymizer, NetAngels, and LPWA. Currently, those are centralized to provide an intermediary that masks the true source of a connection. If anonymous connections are used to hide the source address instead, the other functions of these services may run as a local proxy on the user's desktop. Security is improved because privacy filtering and other services are done on a trusted machine and because communication is resistant to traffic analysis. Also, there is no central point of failure.

Another approach to anonymous Web connections is Crowds [25]. Crowds is essentially a distributed and chained Anonymizer, with encrypted links between crowd members. Web traffic is forwarded to a crowd member, who flips a weighted coin and, depending on the result, forwards it either to some other crowd member or to the destination. This makes communication resistant to local observers.

9 Conclusion

This paper describes anonymous connections, their realization in onion routing, and some of their applications. Anonymous connections are resistant to both eavesdropping and traffic analysis. They separate the anonymity of the connection from the anonymity of communication over that connection. For example, two parties controlling onion routers can identify themselves to each other without revealing the existence of a connection between them. This paper demonstrates the versatility of anonymous connections by exploring their use in a variety of Internet applications. These applications include standard Internet services like Web browsing, remote login, and electronic mail. Anonymous connections can also be used to support virtual private networks with connections that are resistant to traffic analysis and that can carry connectionless traffic.

Anonymous connections may be used as a new primitive that enables novel applications in addition to facilitating secure versions of existing services [24]. Besides exploring other novel applications, future work includes a system redesign to improve throughput and an implementation of *reply onions* [15, 23]. Reply onions are basically reply addresses that enable connections to be established back to an anonymous party. We will be implementing other mechanisms for responding to anonymous connections as well. We are also beginning a detailed analysis of onion routing to enable a quantitative assessment of resistance to traffic analysis.

The onion routing network supporting anonymous connections can be configured in several ways, including a firewall configuration and a customer-ISP configuration, which moves privacy to the user's computer and may relieve the carrier of responsibility for the user's connections.

Onion routing moves the anonymous communications infrastructure below the application level, properly separating communication and applications. Since the efficacy of mixes depends upon sufficient network traffic, allowing different applications to share the same communications infrastructure increases the ability of the network to resist traffic analysis.

Acknowledgments

We have had helpful comments from and discussion with people too numerous to mention. We note especially the help of Birgit Pfitzmann, Gene Tsudik, and James Washington. We also thank the anonymous referees, the Levien family for hosting the onion dinner, and the Isaac Newton Institute for hosting one of the authors while some of this work was done. The fast UltraSparc implementation of RSA was done by Tolga Acar and Çetin Kaya Koç. This work was supported by ONR and DARPA.

References

[1] The Anonymizer. `http://www.anonymizer.com`

[2] T. Acar, B. S. Kaliski, Jr., and Ç. Koç. "Analyzing and Comparing Montgomery Multiplication Algorithms", *IEEE Micro*, 16(3):26-33, June 1996.

[3] T. Berners-Lee, R. Fielding, and H. Frystyk. *Hypertext Transfer Protocol – HTTP/1.0,* ftp://ds.internic.net/rfc/rfc1945.txt

[4] L. J. Camp, M. Harkavey, B. Yee, J. D. Tygar, "Anonymous Atomic Transactions", *Second USENIX Workshop on Electronic Commerce*, 1996.

[5] D. Chaum. "Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms", *Communications of the ACM*, v. 24, n. 2, Feb. 1981, pp. 84-88.

[6] D. E. Comer. *Internetworking with TCP/IP, Volume 1: Principles, Protocols, and Architecture,* Prentice–Hall, Engelwood Cliffs, New Jersey, 1995.

[7] L. Cottrell. *Mixmaster and Remailer Attacks,* `http://obscura.obscura.com/~loki/remailer/remailer-essay.html`

[8] W. Dai. Pipe-net, February 1995, post to the `cypherpunks` mailing list.

[9] Whitfield Diffie, Paul C. van Oorschot, and Michael J. Wiener. "Authentication and Authenticated Key Exchanges". *Designs, Codes, and Cryptography,* 2:107–125, 1992.

[10] A. Fasbender, D. Kesdogan, O. Kubitz. "Variable and Scalable Security: Protection of Location Information in Mobile IP", 46^{th} *IEEE Vehicular Technology Society Conference*, Atlanta, March 1996.

[11] A. Fasbender, D. Kesdogan, O. Kubitz. "Analysis of Security and Privacy in Mobile IP", 4^{th} *International Conference on Telecommunication Systems Modeling and Analysis*, Nashville, March 1996.

[12] M. Franklin and M. Reiter, "Fair Exchange with a Semi-Trusted Third Party", *Fourth ACM Conference on Computer and Communications Security*, Zurich, April 1997.

[13] E. Gabber, P. Gibbons, Y. Matias, and A. Mayer. "How to Make Personalized Web Browsing Simple, Secure, and Anonymous", *Financial Cryptography '97*, February 1997, final proceedings to appear.

[14] D. Goldschlag, M. Reed, and P. Syverson. "Privacy on the Internet", INET '97, Kuala Lumpur, June 1997.

[15] D. Goldschlag, M. Reed, P. Syverson. "Hiding Routing Information", in *Information Hiding*, R. Anderson, ed., LNCS vol. 1174, Springer-Verlag, 1996, pp. 137–150.

[16] C. Gülcü and G. Tsudik. "Mixing Email with *Babel*", *1996 Symposium on Network and Distributed System Security*, San Diego, February 1996.

[17] J. Helsingius. www.penet.fi.

[18] Internet Engineering Task Force. `http://www.ietf.org/`

[19] http://lpwa.com:8000/

[20] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*, CRC Press, 1997.

[21] http://www.netangels.com

[22] A. Pfitzmann, B. Pfitzmann, and M. Waidner. "ISDN-Mixes: Untraceable Communication with Very Small Bandwidth Overhead", *GI/ITG Conference: Communication in Distributed Systems*, Mannheim Feb, 1991, Informatik-Fachberichte 267, Springer-Verlag, Heidelberg 1991, pp. 451-463.

[23] M. G. Reed, P. F. Syverson, and D. M. Goldschlag. "Proxies for Anonymous Routing", *Proc.* 12^{th} *Annual Computer Security Applications Conference*, San Diego, CA, IEEE CS Press, December, 1996, pp. 95–104.

[24] M. Reed, P. Syverson, and D. Goldschlag. "Protocols using Anonymous Connections: Mobile Applications", *1997 Security Protocols Workshop*, Paris, April 1997, final proceedings to appear.

[25] M. Reiter and A. Rubin. *Crowds: Anonymity for Web Transactions (preliminary announcement),* DIMACS Technical Reports 97-15, April 1997.

[26] B. Schneier. *Applied Cryptography: Protocols, Algorithms and Source Code in C,* John Wiley and Sons, 1994.

[27] D. Simon, "Anonymous Communication and Anonymous Cash", in *Advances in Cryptology–CRYPTO'96*, N. Koblitz, ed., LNCS vol. 1109, Springer-Verlag, 1996, pp. 61–73.

[28] P. Syverson, D. Goldschlag, and M. Reed. "Anonymous Connections and Onion Routing", *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, Oakland, CA, IEEE CS Press, May 1997, pp. 44–54.