Anonymous Connections and Onion Routing

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

Abstract

Onion Routing provides anonymous connections that are strongly resistant to both eavesdropping and traffic analysis. Unmodified Internet applications can use these anonymous connections by means of proxies. The proxies may also make communication anonymous by removing identifying information from the data stream. Onion routing has been implemented on Sun Solaris 2.X with proxies for Web browsing, remote logins, and e-mail. This paper's contribution is a detailed specification of the implemented onion routing system, a vulnerability analysis based on this specification, and performance results.

1 Introduction

Private electronic communication is becoming an increasingly important public issue. Encryption can effectively hide the content of a conversation from eavesdroppers, and this protection is being integrated into many systems. But, hiding the identities of communicating parties from eavesdroppers, or from each other, is usually not considered.

Who is communicating with whom, however, may be sensitive too. E-mail users may wish to hide their addresses. Anonymous cash is not anonymous if the communications channel identifies the purchaser. The amount of information revealed through Web browsing should be deliberate. Inter-company collaboration may be confidential. Revealing identities in a cellular phone system reveals a user's location, since the cellular phone network must track handsets' locations.

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 [6]. The anonymous connections can substitute for sockets in a wide variety of unmodified Internet applications by means of proxies. The proxies may also remove 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, 11] in two ways: Communication is real-time and bidirectional, and the anonymous connections are application independent. Onion routing's anonymous 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.

A preliminary description of onion routing is found in [10, 13]. Those papers mainly present the goals of onion routing, and some of the basic structure of our solution. However, they do not give enough detail to properly evaluate the security of onion routing. The original content of this paper includes: a detailed specification of the onion routing system; a description of implementation choices that were influenced by considerations not apparent at a more abstract level; a vulnerability analysis based on the specification; and performance results for our prototype. The specifi-

*Address: 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.

cation presented here is sufficient to guide both re-implementations and new applications of onion routing.

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 system's vulnerabilities, and section 7 describes the implementation choices that were made for security reasons. Section 8 presents 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. We call this an *anonymous socket connection* or anonymous connection. Anonymous connections hide who is connected to whom, and for what purpose, from both outside eavesdroppers and compromised onion routers. If anonymity is also desired, then all identifying information must be removed from the data stream before being sent over the anonymous connection.

We call the onion routing network topology that we use in this paper the *basic configuration*. This is illustrated in figure 1.

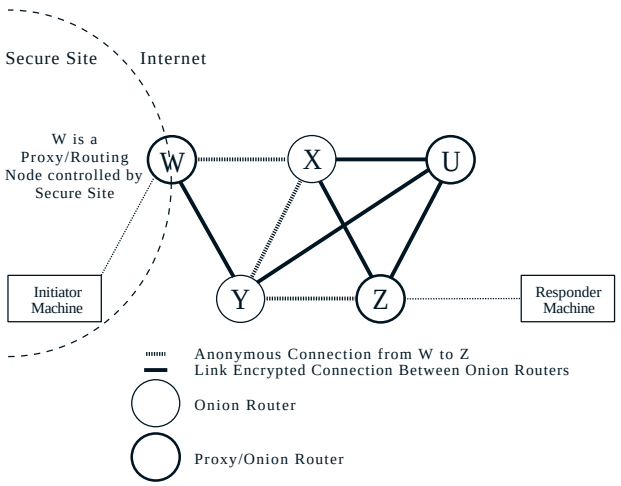


Figure 1. Routing Topology.

In the basic 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 is the basic topology that we will use for the rest of this paper.

The use of anonymous connections by 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. Otherwise, even rudimentary analysis of the unprotected communication between the last onion router in the anonymous connection and the responder may reveal the initiator's identity.

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.

The onion routing network is accessed via *proxies*. An initiating application makes a socket connection to an application specific proxy on some onion router. That proxy defines a route through the onion routing network by constructing a layered data structure called an *onion* and sending that onion through the network. 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. After sending the onion, the initiator's proxy sends data through the anonymous connection.

The last onion router forwards data to another type of proxy on the same machine, called the responder's proxy, whose job is to pass data between the onion network and the responder. An example onion routing network and anonymous socket connection is also illustrated in figure 1.

In addition to carrying next hop information, each onion layer contains key seed material from which keys are generated for crypting[1] data sent forward or back-

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

ward 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 initiator's onion router 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 receiver 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 postcrypted 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.

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 longstanding socket connections. An anonymous connection is comprised of several linked longstanding 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.

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.

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`.

Be aware that accessing a remote onion router does not really preserve anonymity, because the connection between your machine and the first onion router is not protected. Even if that connection were protected, you have no reason to trust the remote onion router. If you had a secured connection to an onion router you trust, it could use our onion router as one of several intermediate routers to further complicate traffic analysis. Remote use of our site provides no greater anonymity than is provided by the Anonymizer [1].

In our experimental onion routing network, five onion routers run on a single Sun UltraSparc 2270. This machine has two processors, and 256MB of memory. Anonymous connections are routed through a random sequence of five onion routers.[2] 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, the removal or addition of layers of encryption is not pipelined, so data latency may be worse on a single machine.

Onion routing's overhead is mainly due to public key cryptography and is incurred while setting up an anonymous connection. On an UltraSparc 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 on PCs (a 200 Mhz Pentium would be twice as fast).

Relatively large connection setup overhead may be tolerable in some applications. For example, socket connection setup may be slow anyway. If a connection is long lived, setup overhead may be reasonable. For example, in WWW requests, a single document may require several requests to the same host to retrieve different components of the same document. Although each individual request and response pair may be short, the combination of all request/response pairs may be lengthy. There is no reason that the same anonymous

[2] Five onion routing hops per connection provides reasonable security at reasonable cost. See section 6.

connection could not be used to carry the traffic for each of the real socket connections, either sequentially or multiplexed. In fact, the preliminary 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. Our Web proxy will be made HTTP 1.1 compliant when HTTP 1.1 is adopted.

4 Threat Model

We assume that the network is subject to both passive and active eavesdropping. That is:

- All traffic is visible.
- All traffic can be modified.
- Onion routers may be compromised.
- Compromised onion routers may cooperate.

In addition, a sophisticated adversary may be able to detect timing coincidences such as the near simultaneous opening of connections. Timing coincidences are very difficult to overcome without wasting network capacity, especially when real-time communication is important.

The initiator's proxy and the first onion router are the most trusted elements of the onion routing system. That is one reason why, in our basic configuration, both the proxy and onion router are placed under the control of the sensitive site.

This threat model directly motivates certain design decisions in onion routing. Because traffic is visible, the headers and payload of all traffic are essentially link encrypted between onion routers so the same data looks different when traveling between onion routers. Because traffic can be modified, stream ciphers [14] are used for encryption. Inserting, deleting, modifying, or replaying traffic anywhere en route will disrupt the stream and will result in persistent unrecognizable changes downstream; thus, data cannot be tracked moving through the system. However, the plaintext will be unreadable by the responder, causing a denial-of-service attack. Because onion routers may be compromised, anonymous connections span several onion routers. Because compromised onion routers may cooperate, data is encrypted in a layered fashion so it appears different to each onion router, not only between onion routers.

In general, our design chooses denial-of-service over the compromise of private information. For example, we assume that data moves through sockets in order and uncorrupted. A compromised onion router can easily violate this assumption; however, the result is unpredictable and unreadable data emerging from the system rather than the direct release of any information. Since replay of an onion will cause the same embedded onions to appear downstream, onion replay may reveal connection information. However, onions themselves cannot be replayed through an honest node. Onion routers remember onions they have passed by storing a hash of previously passed onions. If a replay is detected, the onion is simply dropped. To control storage requirements, onions are equipped with expiration times. Here too, denial-of-service supersedes compromise. If clocks are far enough out of synchronization one way, the only possible result is for a fresh onion to be viewed as expired and ignored. If they are far enough out of synchronization the other way, the only possible result is for a passed onion to be stored beyond its expiration.

5 Onion Routing

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. In this paper, we will focus on the HTTP proxy for Web browsing.

In the basic configuration where a firewall lives between a trusted and untrusted network, the onion router and its proxies live on the firewall. There are two classes of proxies: one that bridges connections from initiating applications into the onion routing network (the application proxy), and another that completes the connection from the onion routing network to responders (the responder proxy).

Because the application proxy bridges between applications and the onion routing network, it must understand both application protocols and onion rout-

ing protocols. Therefore, to simplify the design of application specific proxies, we partition the proxy into two components: the *client proxy* and the *core proxy*. The client proxy bridges between a socket connection from an application and a socket connection to the core proxy. It is the obligation of the client proxy to massage the data stream so both the core proxy and the responder proxy can be application independent. Specifically, the client 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 responder proxy and either continue if no error is reported or report the onion routing error code in some application specific meaningful way.

Upon receiving a new request, the core proxy uses the prepended standard structure as a hint in building an onion defining the route of an anonymous connection to that destination. It then passes the onion to the onion routing network building the anonymous connection to the responder proxy, and then passes the prepended standard structure to the responder proxy specifying the ultimate destination. From this point on, the core proxy blindly relays data back and forth between the client proxy and the onion routing network (and thus the responder proxy at the other end of the anonymous connection).

For the services we have considered to date, a nearly generic responder proxy is adequate. Its function is to read the data stream from the terminating onion router. The first datum received will be the standard structure specifying the ultimate destination. The responder proxy makes a socket connection to that IP/port, reports a one byte status message back to the onion routing network (and thus back to the core proxy which in turn forwards it back to the client proxy), and subsequently moves data between the onion routing network and the new socket. (For certain services, like RLOGIN, the responder proxy also infers that the new socket must originate from a trusted port.)

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

The client 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 core proxy, to inform the core proxy of the service and destination of the anonymous connection. The client proxy then modifies the `GET` request to `GET /showcase/ HTTP/1.0` and sends it directly (through the anonymous connection) to the HTTP server, followed by the optional fields. Notice that the server name and `http://` are eliminated because the connection is made directly to the HTTP server.

The client 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 client 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 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.

The core proxy's function is to pass data between multiple socket connections from client proxies and the first onion router. Therefore, the core proxy is not application specific but must understand the onion routing protocol, which defines how multiplexed connections are handled. The core proxy must repeatedly pre-crypt the data stream before passing it along the onion routing network. The repeated pre-cryptions are the inverses of the cryptographic functions that will be applied by the onion routers as the data moves along the anonymous connection. Similarly, the core proxy must repeatedly post-crypt data from the anonymous connection with the inverses of the cryptographic functions that were applied by the onion routers, before passing the plaintext to the client proxy.

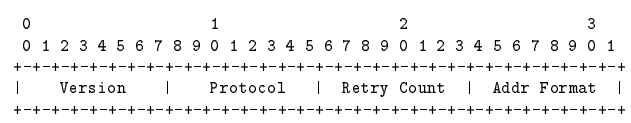
5.2 Implementation

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 a anonymous connection; and the destruction and cleanup of anonymous connections. We will commingle the discussion of these below.

5.3 Client Proxy

The interface between an application and the client proxy is application specific. The interface between the client proxy and the core proxy is defined as follows. For each new proxy request, the client 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 core proxy's well known port. The client proxy then sends a standard structure to the core proxy of the form:



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 responder proxy 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 immediately followed by another NULL terminated ASCII string with the destination port number, or a 2 for *sockaddr_in* data structure specifying both the internet address and the destination port. The ultimate destination address is sent after this standard structure, and the client proxy waits for a one byte error code before sending data.

5.4 Core Proxy

Upon receiving the standard structure, the core proxy can decide whether to accept or reject the request based on the protocol, anonymity, destination host, destination port, or the identity of the client proxy. If rejected, it sends an appropriate error code back to the client proxy, closes the socket, and waits for the next request. If accepted, it proceeds to build the anonymous connection to the responder proxy using the standard structure, sends the standard structure to the responder proxy over the anonymous connection, and then passes all future data to and from the client proxy and anonymous connection. The repeated pre and post cryptions and packaging of the data is discussed later in section 5.6.

5.5 Onions

To build the anonymous connection to the responder proxy, the core proxy creates an onion. An onion is a multi-layered data structure that encapsulates the route of the anonymous connection starting from the responder proxy and working backward to the core proxy.

Each layer has the following structure:

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 opposite direction that 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 that 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 responder proxy. 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).[3]

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

[3]Details on the cryptographic operations used in this paper can be found in [14].

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 performed.

In version 1, an onion has five layers. 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).[4]

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

5.6 Onion Router Interconnection

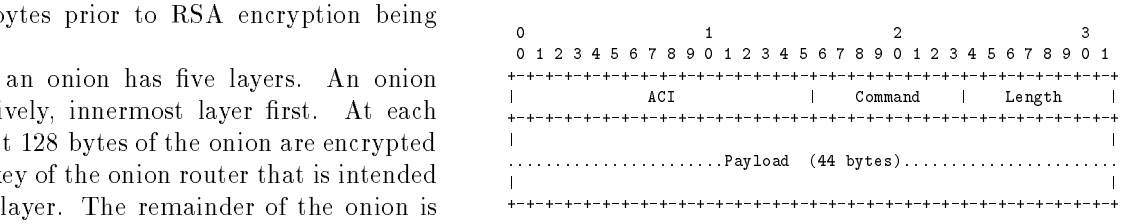
During onion network setup (not to be confused with anonymous connection setup), longstanding connections between neighboring onion routers are established and keyed. The network topology is predefined and each onion router knows its neighbors and the RSA public keys of all nodes in the network.

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.

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 [8] 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. 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:



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, seeds 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.

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 avoid traceability).

Data moves through an anonymous connection in DATA cells. At each onion router, except for the initiator's, 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 initiator's onion router 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

[4] 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. We would be happy to use a stronger pseudorandom number generator, however.

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 responder proxy, 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, according to a scheme that we call layered reordering (see section 6) that both preserves the order of cells in each anonymous connection and caps how long cells may be delayed by an onion router.

5.7 Responder Proxy

When a routing node receives an onion with *Destination Address* and *Destination Port* of 0, it knows it is to act as a responder proxy. It 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 Vulnerabilities

Onion routing is not invulnerable to traffic analysis attacks. We now define an attack approach based on several simplifying assumptions. We characterize the complexity of an attack based on these assumptions, and note that real attacks will be more expensive. We also note that any scheme resistant to traffic analysis cannot increase the space of potential recipients of a message to more than the number of possible recipients on the network. So, expecting the complexity of traffic analysis to be similar to the cost of brute force attack on cryptographic algorithms (e.g., 2^{56}) is unreasonable. Furthermore, the cost of traffic analysis searches may be higher, since a network must be monitored.

Assume that each onion router has connections to eight (2^3) other onion routers, and that each onion router has a secured connection to a sensitive site. Assume that all anonymous connections pass through five onion routers. Assume furthermore that since all cells moving through an onion routing network are of fixed length (48 bytes) it is possible to identify when cells begin and end. So we reduce the problem to tracking markers, where markers indicate the beginning of a cell. Assume further, that an attacker can start monitoring the system when an onion router's incoming queues and outgoing queues are empty, so the attacker can determine the order in which markers arrive at an onion router.

Under these assumptions, tracking markers through the network depends on the reordering done at each onion router. If no reordering is done (i.e., cells move from incoming to outgoing queues using a FIFO strategy), then it appears easy to track markers. The first marker to reach on onion router will be the first to leave, and its route through the network can be followed. But analysis is not that simple. Since each onion router is both an onion routing proxy and an intermediate onion router, a new marker may enter the network at any time, and it is impossible for an observer to determine whether the new marker arrived before or after the first marker on the incoming queues (since a sensitive site's connection to its onion router is protected). If the onion routing proxy is busy, the marker could end up on one of two outgoing queues. (If the onion routing proxy is not busy, only one outgoing queue will be active.) Under the most complicating conditions, a marker could ultimately end up in one of 2^5 outgoing queues.

To further complicate traffic analysis, onion routers reorder incoming markers, so data does not move through the network in a simple FIFO manner. The optimal goal would be to make it equally likely that an incoming marker is output on any of an onion router's 8 outgoing queues. In that case, a single marker could end up at one of $(2^3)^5$ outgoing queues. Notice that we cannot improve on this number, since it defines all possible reachable queues.[5]

Since onion routing is meant for real-time communication, we use a limited amount of reordering. At any

[5] This does not imply that one can only reach 2^{15} sites via onion routing. Since the responder's proxy can make connections to any Internet site, one can anonymously browse any Web site. If the goal is to have anonymous connections between two sensitive sites, then any one site can communicate with at most 2^{12} other sensitive sites.

point in time, markers on several incoming queues may be considered to arrive at the same time. These markers may be moved to outgoing queues in any arbitrary order that both maintains fairness of data movement for every anonymous connection and preserves the order of data on each anonymous connection.

We define n-layered reordering as moving the first n markers on each incoming queue to outgoing queues in any arbitrary order subject to the order preserving restriction just described. If an incoming queue has fewer than n markers, all markers on the queue are moved. (In 1-layer reordering, the order preserving restriction is trivially satisfied.) Notice that although markers may be delayed at any particular onion router, on average data latency is not hurt since markers are equally likely to be forwarded early.

Using layered reordering, traffic analysis becomes more complicated, since a marker could end up on one of several output queues. However, imagine that we know that two markers belong to the same anonymous connection. So, if we know which outgoing queues are possible for each of the markers, the intersection of those sets defines which queues are possible for the next hop in the anonymous connection. The goal, therefore, is to choose a layered reordering depth that makes it very likely that all possible outgoing queues will be present in each set most of the time.

Since data latency is not hurt by layered reordering it is possible to predict the window during which a marker is likely to exit the onion routing network. This invites another kind of traffic analysis.[6] It should be possible to identify the near simultaneous opening of endpoint connections. More specifically, if an attacker wishes to confirm that two parties are communicating frequently, if they happen to have many more simultaneous connection openings than is expected by chance, they are probably communicating. This attack, however, is not possible if the onion routing basic configuration is only used for communication between sites that control onion routing proxies, since the connection between the site and its onion router is assumed to be protected. But, the basic configuration is not appropriate everywhere, so this attack may persist in certain scenarios.

7 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.

[6] Thanks to John Kelsey for helpful comments on this point.

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 the apparent scheduling randomness 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 that we do not yet know how to address. If part of the onion routing network is taken down, traffic analysis is greatly 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.[7]

8 Related Work

Chaum [3] defines a layered object that routes data through intermediate nodes, called *mixes*. These in-

[7] One could imagine sending control information by inserting some random cell into the data stream. The application or its proxy could detect corrupted data, and terminate at the application level, without destroying the anonymous connection. However, this is risky for two reasons: it may not always be possible to detect corrupted data, and a random inserted cell may appear uncorrupted.

termediate nodes may reorder, delay, and pad traffic to complicate traffic analysis. Our onion routers are based on mixes. Some work has been done using mixes in ATM networks [5].

Anonymous Remailers like [7, 11] use mixes to provide anonymous e-mail services. Some invent an address through which mail can be forwarded back to the original sender. Remailers work in a store and forward manner at the mail application layer, by stripping off headers at each mix and forwarding the mail message to the next mix. Some remailers provide confirmation of delivery.

In [9], 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.

In [12], mixes are used to provide untraceable communication in an ISDN network. As described there, in an ISDN system, each ISDN 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 within each switch between the caller and the long distance network, which is obtained by routing calls through a predefined series of mixes. The long distance endpoints of the connection are then mated to complete the call. (Notice that observers can tell which local switches are connected.) This approach relies upon two unique features of ISDN switches as described in [12]. Since each ISDN line has a subset of the switch's total capacity pre-allocated to it, there is no (real) cost associated with keeping an ISDN line active all the time, either by making calls to itself, to other ISDN lines on the same switch, or to the long distance network. Keeping ISDN lines active complicates traffic analysis because an observer cannot track coincidences.

Since each ISDN 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: An ISDN 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 circuit. 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, would reveal which businesses are communicating. In onion routing, mixing is dispersed throughout the Internet, which improves hiding.

9 Conclusion

Anonymous socket connections provide protection against both eavesdropping and traffic analysis. Although our focus is on anonymous connections, and not anonymous communication, anonymous communication is also possible by removing identifying information from the data stream. Onion routing's anonymous connections are application independent and can interface with unmodified Internet applications by means of proxies. Our implementation of onion routing includes proxies for Web browsing, e-mail, and remote login. We have also implemented anonymizing versions of the Web and e-mail proxies.

It is instructive to compare onion routing's anonymous e-mail service with other anonymous remailers. All services remove identifying headers. Most remailers work in a store and forward manner, either between mixes or simply sendmail daemons. Onion routing's service, however, makes an anonymous connection directly to the recipient's sendmail daemon. This has both advantages and disadvantages. The disadvantage is that mixing is not done as well, since the connection is made in real time. The 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.

Anonymous remailers typically provide a mechanism to reply to anonymous e-mail. A remailer may assign pseudonyms through which mail is forwarded. These pseudonyms must be stored at the remailer in order to properly process replies. In onion routing, it is possible for a sender to build a *reply onion* that defines an anonymous connection to him. This reply onion can be included in mail messages. When a response is sent to the appropriate proxy on an onion router, the reply onion is first processed to create the anonymous connection back to the sender. The reply is then sent over that anonymous connection. Notice that the reply onion is equivalent to a pseudonym, except that it is not stored at any onion router. So onion routers are stateless remailers. To identify users of anonymous onion routed e-mail, reply onions must first be obtained and all relevant onion routers must be compromised.

Anonymous connections may be used as a new primitive that enables novel applications in addition to fa-

cilitating secure versions of existing services. For example, in a cellular phone system, the location of handsets must be tracked, even when the phone is waiting for a call (in standby mode). This is because the cellular network must know through which base station to route calls. However, it may be undesirable to let the cellular network know the location of its subscribers. An alternative architecture that protects such location information, may be constructed using anonymous connections. To make a call, the phone constructs an onion which defines a route through the local base station to some billing station. The phone identifies the subscriber to the billing station (for billing purposes) but does not have to reveal its location. The billing station completes the call. To receive a call, the handset is paged over a large area. This paging turns on the handset, which then makes a call to the paged number (through an anonymous connection as described above). As an aside, the paging approach to receiving a call significantly conserves battery use, since the phone is off unless it is involved in a call.

Alternatives to the basic configuration exist which move trust closer to the user. For example, an Internet Services Provider (ISP) could run an onion router that accepts onions from its subscribers. Subscribers would generate these onions on their trusted local machines. The ISP would not know with whom the customer is communicating. And the subscriber need not fully trust the ISP to maintain his privacy.

Acknowledgments

We were helped by discussions with many people including Ran Atkinson, Markus Jakobbsen, John Kelsey, John McLean, Cathy Meadows, Andy Moore, Moni Naor, Holger Peterson, Birgit Pfitzmann, Michael Steiner, and James Washington. We thank the anonymous referees for helpful suggestions. We thank the Isaac Newton Institute, Cambridge. Some of these discussions were conducted while one of the authors was in residence there. The fast UltraSparc implementation of RSA was done by Tolga Acar and Çetin Kaya Koç. This work was supported by ONR.

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

[4] D. Chaum, The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability, *Journal of Cryptology*, 1/1, 1988, pp. 65-75.

[5] S. Chuang. *Security Management of ATM Networks,* Ph.D. thesis, in progress, Cambridge University.

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

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

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

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

[12] 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, Heildelberg 1991, pp. 451-463.

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

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