

On Anonymity in an Electronic Society: A Survey of Anonymous Communication Systems

Matthew Edman
Rensselaer Polytechnic Institute
and
Bülent Yener
Rensselaer Polytechnic Institute

The past two decades have seen a growing interest in methods for anonymous communication on the Internet, both from the academic community and the general public. Several system designs have been proposed in the literature, of which a number have been implemented and are used by diverse groups, such as journalists, human rights workers, the military, and ordinary citizens, to protect their identities on the Internet.

In this work, we survey the previous research done to design, develop, and deploy systems for enabling private and anonymous communication on the Internet. We identify and describe the major concepts and technologies in the field, including mixes and mix networks, onion routing, and Dining Cryptographers networks. We will also review powerful traffic analysis attacks that have motivated improvements and variations on many of these anonymity protocols made since their introduction. Finally, we will summarize some of the major open problems in anonymous communication research and discuss possible directions for future work in the field.

Categories and Subject Descriptors: C.2.0 [**Computer-Communication Networks**]: General—*Security and Protection*; C.2.2 [**Computer-Communication Networks**]: Network Protocols—*Applications*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed Applications*; K.4.1 [**Computers and Society**]: Public Policy Issues—*Privacy*

General Terms: Security

Additional Key Words and Phrases: Anonymity, Anonymous Communication, DC-nets, Mixes, Mix Networks, Onion Routing, Privacy, Traffic Analysis

1. INTRODUCTION

A famous cartoon in *The New Yorker* once carried the caption, “On the Internet, nobody knows you’re a dog” [Steiner 1993]. This phrase symbolizes the general public’s perception that the Internet allows one to communicate in relative obscurity and anonymity. Quite the contrary, today’s modern communications infrastructure is indeed capable of identifying and recording what we do, where we go, and even sometimes what we say on the Internet.

Authors’ address: M. Edman & B. Yener, Rensselaer Polytechnic Institute, Department of Computer Science, 110 8th Street, Troy, NY 12180.

Permission to make digital/hard copy of all or part of this material without fee for personal or classroom use provided that the copies are not made or distributed for profit or commercial advantage, the ACM copyright/server notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or a fee.

© 2008 ACM 0000-0000/2008/0000-0001 \$5.00

Each time we send an email, visit a Web page, or talk to friends and family via instant messaging, we send packets of data across the Internet that contain information regarding where the message is going and who sent it (e.g., IP addresses, SMTP header information, etc.). As packets are transmitted via several hops from the source to their destination, anybody observing a link along that path can roughly identify who is communicating with whom based on information contained in each packet. Even if the packet data is encrypted, the source and destination addresses in the packet's IP header are still visible to an observer.

Research into designing and building anonymity systems seeks to build an infrastructure running on top of the existing Internet protocols that allows people to communicate with each other without necessarily revealing their personal network identifiers, such as IP addresses.

We can imagine many situations in which someone might desire strong communications privacy and anonymity on the Internet. Law enforcement agencies may want to set up an online anonymous "tip line", where people can report information they may have on unlawful activity without fear of retribution or punishment. Government intelligence agencies may need to be able to monitor known extremist websites without revealing their presence to the website operators. Private citizens may want to be able to freely browse the Web, without advertisers collecting statistics on their personal browsing habits and selling that personal information to other companies.

Anonymous communication systems can also provide a strong foundation for censorship resistance for people living under oppressive regimes that try to limit what their citizens can say and do on the Internet. There has been significant and interesting work on designing censorship-resistant systems that allow people to anonymously store, publish, and retrieve data, while making it resilient to attack by an adversary [Waldman and Mazières 2001; Dingedine et al. 2000; Waldman et al. 2000; Anderson 1996; Clarke et al. 2000]. Concepts from the design of anonymous communication systems have also been applied to the fields of cryptographic voting [Kilian and Sako 1995; Jakobsson et al. 2002; Boneh and Golle 2002], electronic currencies [Camenisch et al. 2005], anonymous auctions [Harkavy et al. 1998], and anonymous credentials [Camenisch and Lysyanskaya 2001; Camenisch et al. 2006].

In this work, we survey the previous research done to design, develop and deploy systems for enabling private and anonymous communication on the Internet. We review the major technologies in the field, including mix networks, onion routing and Dining Cryptographers networks, as well as improvements and variations on those primitives made since their introduction. We will also summarize some of the powerful attacks an adversary might conduct in order to identify communication patterns in an anonymity system. Our survey concludes with a discussion of the major open problems in anonymous communication and future research directions.

2. BACKGROUND

In this section, we first define what we mean by "anonymity" and "anonymity systems," as well as discuss some of the most commonly used terms in the literature. We also provide a classification for the various types of adversaries anonymity system designs most often seek to defend against.

2.1 Anonymity & Pseudonymity

In an effort to standardize the terminology used in the literature on anonymous communication systems, Pfitzmann and Hansen have written a consolidated proposal of terminology [Pfitzmann and Hansen 2008]. We largely follow their definitions in this survey. We consider a system to be a collection of actors, such as clients, servers or peers, in a communication network. These actors exchange messages via public communication channels (e.g., the Internet). In borrowing terms from graph theory, we sometimes refer to the actors in a network as *nodes* and the communication channels between them as *links*.

Pfitzmann and Hansen [2008] informally defined anonymity as, “the state of being not identifiable within a set of subjects, the *anonymity set*.” In other words, the anonymity set is the set of all possible actors in a system that could have been the sender or recipient of a particular message. We can further refine the anonymity set based on the particular role subjects have with respect to a specific message. The set of subjects who could have been the sender of a particular message is known as the *sender anonymity set* and, similarly, the set of subjects who could have been the recipient of a particular message sent is the *recipient anonymity set*.

In general, anonymity systems seek to provide *unlinkability* between sent messages and their true recipients (*recipient anonymity*), and between received messages and their true senders (*sender anonymity*). Pfitzmann and Hansen further defined the unlinkability between senders and recipients in an anonymity system as *relationship anonymity*. Put more simply, relationship anonymity means that an adversary observing the senders and recipients in the network is unable to discern who is communicating with whom. We consider in greater detail the types of adversaries that may be observing the network in Section 2.2.

Note that so far we have not considered unlinkability between senders and sent messages, or between recipients and received messages. It seems intuitive that someone observing network traffic sent from or received by a subject in an anonymity system can easily identify when that subject sends or receives a message. However, we will describe in Section 5 anonymity systems that also try to hide the fact that a subject is sending or receiving a message by introducing additional random “cover” traffic, intended to obscure the true communication pattern between senders and recipients. The property that actual messages sent from or received by a subject are indistinguishable from random noise is known as *unobservability*, a stronger security property than unlinkability [Pfitzmann and Hansen 2008].

It is important to distinguish anonymity from *pseudonymity*. Where anonymity implies that actors are not identifiable within the anonymity set, pseudonymity simply means that actions, such as sending or receiving a message, are linked to a particular identifier other than the actor’s true identity. Pfitzmann and Hansen [2008] also noted that a pseudonym need not be associated with only one holder. Instead, multiple distinct subjects can share a single pseudonym, known as a *group pseudonym*. In the case of a group pseudonym, the set of subjects represented by a single pseudonym may indeed form an anonymity set.

While anonymity and pseudonymity are separate but related concepts, Goldberg [2000] illustrated with his “Nymity Slider” concept that it is possible to build pseudonymity on top of anonymity. An actor in an anonymity system can

simply adopt a particular pseudonym used to link a series of transactions over the anonymity system to that identifier, thus providing pseudonymity rather than anonymity. The reverse, however, is not necessarily true.

2.2 Adversarial Models

As in nearly all security-related research, adversaries are most often defined according to their goals and strengths. Raymond [2000] identified a number of properties that can be used in combination to describe the strength of an adversary. We summarize and classify these properties as follows:

- **Capability** An important property of an adversary is whether that adversary is *active* or *passive*. A passive adversary is one who is able to monitor and record the traffic on network links entering and exiting clients or servers in an anonymity network. A passive adversary is also able to record metadata about network traffic flows, such as packet lengths and arrival times. Even simply passively monitoring traffic can enable powerful statistical attacks, which will be discussed later in Section 6.

- An *active* adversary has all the monitoring capabilities of a passive adversary, but is also able to manipulate network traffic perhaps by either controlling one or more network links or by operating a node in the anonymity network. He can modify or drop traffic in parts of the network, as well as insert his own traffic or replay legitimate network traffic that he previously recorded.

- **Visibility** While not specifically enumerated by Raymond [2000], the *visibility* of the adversary is a property often considered in the literature when defining a threat model [O'Connor 2005; Newman et al. 2003; Danezis et al. 2003; Dingleline et al. 2004a; Reiter and Rubin 1998]. The visibility of an adversary determines how much of the network he is able to passively monitor or actively manipulate. A *global* adversary, as the name suggests, is a powerful observer that has access to all network links between clients and servers in an anonymity network.

- In contrast, a *partial* adversary is only able to monitor the links entering and exiting a particular node in the network, or a small, related subset of nodes in the network. An example of a partial adversary might be someone on the same local area network as a node in the anonymity network, or even a common Internet Service Provider (ISP) among multiple network nodes [Feamster and Dingleline 2004; Murdoch and Zieliński 2007].

- **Mobility** Even if an adversary does not have the large scale infrastructure required to monitor all clients and servers in an anonymity network simultaneously, he may be able to select which subsets of the network he wants to monitor based on previously obtained information. We refer to such an adversary as an *adaptive* adversary. For example, a government may be able to subpoena a selection of ISPs under their legal jurisdiction to monitor the network traffic of some servers in an anonymity network. An adaptive adversary could then choose to monitor a different, possibly overlapping, subset of servers based on information learned from the network traces obtained from the first subpoena.

- In contrast, a *static* adversary is able to compromise or monitor some subset of the network, but is not able to change which subset he observes at will. A global adversary is, almost by definition, a static adversary: since he has a global view of

the anonymity network, he does not need to selectively monitor smaller portions of the network.

. **Participation** Raymond [2000] also suggested characterizing adversaries as being either *internal* or *external*. An internal adversary is one who participates in the anonymity network's protocol as a client, or perhaps operates a piece of the network's infrastructure by running a server in the network. Note that an internal adversary is not by definition also an active adversary. Instead, he could simply be monitoring the traffic that passes through his own server without actively modifying it.

An external adversary, on the other hand, does not participate in the anonymity network or its protocol. Instead, he compromises the communication medium used by clients (i.e., their network links) in order to monitor or manipulate their network traffic.

The type of adversary most commonly considered in the literature on anonymous communication systems is the powerful passive global adversary. Since it is a common security practice to try to defend against a worst-case scenario, assuming the presence of a passive global adversary appears prudent.

Some have argued, however, that a passive global adversary is unrealistic in practice for large anonymity networks deployed across the Internet [Newman et al. 2003]. The geographic and jurisdictional diversity of the Internet makes it unlikely that a single entity, or even a small collaboration of entities, can monitor all servers in a widely distributed anonymity network. Further, if an adversary is powerful enough to monitor the entire network, it seems unreasonable to assume that adversary is unable to also actively manipulate network traffic. Instead, a more likely real world adversary may be an active adversary with only a partial view of the network.

3. HIGH-LATENCY ANONYMITY SYSTEMS

Designs for anonymous communication can often be classified into two general categories: high-latency systems and low-latency systems. High-latency anonymity systems are able to provide strong anonymity, but are typically only applicable for non-interactive applications that can tolerate delays of several hours or more, such as email. As expected, low-latency anonymity systems often provide better performance and are intended for real-time applications, particularly Web browsing. Consequently, some have referred to high-latency anonymity systems as *message-based* systems, and low-latency anonymity systems as *connection-based* systems [Serjantov and Sewell 2003].

3.1 Overview

To illustrate a basic approach to anonymous mail, we begin with a hypothetical situation. Imagine Alice is a corporate whistleblower who wants to mail a letter to *The New York Times* revealing some misconduct, but still wants to remain anonymous for fear of losing her job. Similar to how IP addresses can reveal who sent a particular packet, simply writing a letter and dropping it in the local post office box would leak information to the recipient about where the message originated because of the postal markings on the envelope.

Instead, Alice places the envelope addressed to *The New York Times* inside another envelope addressed to a third party named Charlie. This outer envelope is then placed inside yet another envelope addressed to another person, Bob. Alice then places the multiply-enveloped letter in her local post office box. When Bob receives the envelope, he opens it and finds another envelope addressed to Charlie. Recognizing the envelope is not addressed to him, Bob then drops it in his local post office box. Similarly, when Charlie receives and opens the envelope addressed to him, he finds another addressed to *The New York Times*, which he then also forwards on.

Since the envelope received at *The New York Times* came from Charlie’s post office, it does not give away any information about Alice’s location. The envelope received by Bob can be used to identify Alice, but Bob only knows the next hop in the path and not the letter’s content or its intended destination. Only if all people in the letter’s path collaborate can they identify Alice as the sender of the letter received by the newspaper.

Clearly there are still a number of side-channel approaches to identifying the sender of the original letter, such as DNA material left on the letter or envelope, handwriting analysis, or a linguistic analysis of the letter’s content — an interesting field of study known as *stylometry*. Any recipient along the letter’s path could also simply open all of the envelopes and read the message contents.

Still, the basic idea of *remailing* a message through several intermediaries underlies many modern anonymous communication systems, and cryptography provides us the tools necessary for preventing others along the path from “opening” the letter. It is worth noting that current systems for anonymity and pseudonymity do not necessarily protect against linguistic or stylometric linking of plaintext messages written by the same author [Rao and Rohatgi 2000].

3.2 Mixes & Mix Networks

3.2.1 A Simple Mix. The basic building block of nearly all modern high-latency anonymous communication systems is a *mix*, a concept introduced by Chaum [1981]. At a high level, a mix is a process that accepts encrypted messages as input, groups several messages together into a *batch*, and then decrypts and forwards some or all of the messages in the batch.

More precisely, a mix first generates a public and private key pair and makes the public component known to clients who wish to relay messages through the mix. We let $C = E_x(M)$ denote encryption of a message M with mix x ’s public key, and also let $M = D_x(C)$ denote decryption of a ciphertext C with mix x ’s corresponding private key. Further, we let \mathcal{A}_x represent the identity or address of mix x (e.g., x ’s IP address).

Consider a sender, Alice, who wants to anonymously send a message M to a recipient, Bob, via a single mix x . In Chaum’s original design, Alice would compute $E_x(R_x, M, \mathcal{A}_B)$, where R_x is a string of random bits and \mathcal{A}_B is Bob’s address. She then sends the resulting ciphertext to the mix, who can use its private key to retrieve M and \mathcal{A}_B . R_x is simply discarded, but is included in the encryption to help prevent an adversary from identifying two identical messages encrypted under the same asymmetric key. More sophisticated approaches to probabilistic encryption, like RSA-OAEP, were not invented until later [Goldwasser and Micali

1984; Bellare and Rogaway 1994].

The mix collects messages into a *batch* until it has received “enough” (we elaborate on this in the next section), and then forwards each to the destination address extracted from the decrypted input message. Chaum’s original design was to output messages in lexicographic order, although later schemes output messages in a random order [Möller et al. 2003; Danezis et al. 2003].

If Alice knows Bob’s public key (perhaps learned out-of-band or from a public key server), she could also first encrypt the original message M with Bob’s public key in order to prevent mix x from being able to read the plaintext message. If Alice and Bob want to authenticate each other over the anonymous channel, then they must both have exchanged and verified each other’s public keys prior to communicating over the channel.

We can more concisely represent the above situation using the following notation, where the left-hand side of the arrow represents the input to a mix, \xrightarrow{x} represents processing done by mix x , and the right-hand side of the arrow indicates the mix’s output:

$$E_x(R_x, M, \mathcal{A}_B) \xrightarrow{x} M$$

An observer who can only monitor the messages entering and exiting the mix is unable to correlate the inputs to outputs based on the message bits, since the mix cryptographically transforms each message. Additionally, the messages have been delayed, batched, and reordered, so the observer cannot link messages based on their message arrival and departure times. Only the mix itself knows which input messages correspond to which output messages; however, senders must trust the mix not to reveal to others how the messages were reordered.

At the time of Chaum’s seminal paper, public key cryptosystems like RSA were still relatively new in academic research [Rivest et al. 1983]. Consequently, Chaum’s design used RSA directly, which was later shown to be susceptible to a “tagging” attack that allows an adversary to modify an encrypted input message and then identify the corresponding output message [Pfitzmann and Pfitzmann 1990]. Later schemes employ a hybrid encryption scheme in which RSA is only used for encrypting a symmetric key, and the symmetric key is used for encrypting the remainder of the message [Möller et al. 2003; Danezis et al. 2003].

3.2.2 Mix Networks. Clearly using a single mix suggests not only a single point of failure, but also a single point of trust since a dishonest mix can reveal the true input-output correlations. Instead of sending messages through a single mix, senders can choose an ordered sequence of mixes through which to send their messages. We refer to the sequence of mixes through which a message is routed as that message’s *path*.

If Alice wants to anonymously send a message M to Bob via a path $\mathcal{P} = \{x, y, z\}$, she would iteratively create a layer of encryption, in the same manner as above, for each mix starting with the last mix in the path and working back towards the first. Alice then sends the resulting multiply-encrypted ciphertext to the first mix in the path. Each mix can remove a layer of encryption to extract the address of the next mix in the path and a ciphertext message to forward to that mix.

As the message traverses the path \mathcal{P} , it results in the following cryptographic transformations:

$$\begin{aligned} E_x(R_x, E_y(R_y, E_z(R_z, M, \mathcal{A}_B), \mathcal{A}_z), \mathcal{A}_y) &\xrightarrow{x} E_y(R_y, E_z(R_z, M, \mathcal{A}_B), \mathcal{A}_z) \\ &\xrightarrow{y} E_z(R_z, M, \mathcal{A}_B) \\ &\xrightarrow{z} M \end{aligned}$$

As before, in the case of a single mix, an observer watching the inputs and outputs of any mix in the path is unable to link input messages of a mix to its outputs. Additionally, each mix along a message’s path knows only the hop before and after itself in the path. Thus, no single mix operator is able to compromise the anonymity of messages sent through his mix.

To prevent the lengths of messages from leaking information about how many mixes a message has already passed through, mixes can pad the messages to a fixed length before forwarding them. After removing the message header, a mix simply generates a block of uniformly random bytes indistinguishable from the encrypted payload and appends it to the message. The last hop in the path can then simply separate the randomly generated padding from the resulting plaintext message before sending the original message to its final recipient.

How best to choose the sequence of mixes to use for a message’s path has been the subject of some debate in the literature [Berthold et al. 2000; Dingleline et al. 2004]. There are two general path selection strategies typically considered: *free routes* and *mix cascades*. A free route topology, as the name suggests, means clients are able to choose any ordered sequence of mixes in the network for their message’s path. In a mix cascade topology, there are one or more predefined routes through which all client traffic is relayed.

We can also imagine several variations on these general approaches. For example, Gülcü and Tsudik [1996] designed a mix system called Babel that used a free route topology, but also allowed messages to take inter-mix “detours” along the path specified by the client. Any mix on a message’s path could re-encrypt that message and send it through a series of mixes not originally specified by the client before returning to its original path. Danezis [2003a] also suggested another mix network topology based on expander graphs.

Berthold et al. [2000] argued that mix cascades provide stronger anonymity than free routes when the adversary compromises a large number of mixes in the network. They presented several attacks that showed an adversary who controls all but one mix in a free route network can exploit his knowledge of how many mixes a message has passed through in order to partition messages into smaller anonymity sets, thus decreasing their anonymity.

Dingleline et al. [2004] countered that the security of mix cascades is not actually a result of that particular topology, but rather because messages enter the network and proceed through the network in lockstep (i.e., *synchronous batching*). The authors then showed that free route networks can actually provide greater anonymity than cascades when synchronous batching is used. Additionally, free route networks can provide better resilience in the event of node failure, since a

single failed mix in a cascade breaks *all* communication through that cascade. In a free route network, clients can simply route their traffic around the failed node.

3.3 Flushing Algorithms

In the introduction to mixes and mix networks above, we intentionally left undefined how mixes decide when to send out (or “flush”) a batch of messages they have collected. The algorithm a mix uses to determine which messages to forward to their next destination and when to do so is often referred to as a *flushing algorithm* [Serjantov et al. 2002]. Flushing algorithms are sometimes equivalently called *batching strategies* [Díaz and Serjantov 2003].

There have been numerous flushing algorithms discussed in the literature, as well as simple but effective active attacks against them, which we review below.

3.3.1 Threshold Mixes. The classic mixes originally defined by Chaum are known as *threshold mixes* [Chaum 1981]. A threshold mix simply collects incoming encrypted messages until it has received n messages, where n is a parameter defined by the system or the mix operator. After receiving n messages, the mix then decrypts all of them and forwards them to their next destination in a random order.

A threshold mix is a simple flushing algorithm, but it is vulnerable to an active attack called the $(n - 1)$ attack, or “flooding” attack [Serjantov et al. 2002]. In the $(n - 1)$ attack, an attacker delays a legitimate message M from entering the mix. He then generates and sends his own “dummy messages” into the mix until it flushes. The dummy messages are generated such that they are indistinguishable from legitimate messages to anyone but the attacker. Finally, the attacker then allows M into the mix and sends more of his dummy messages until the mix flushes again. Since the attacker generated the dummy messages himself, he can easily identify which of the decrypted mix outputs are his messages. The only mix output that is not one of his own dummy messages thus corresponds to input M .

3.3.2 Timed Mixes. Rather than collecting a fixed number of messages like a threshold mix, a *timed mix* instead collects messages for a fixed length of time t [Serjantov et al. 2002]. After t seconds have elapsed, the mix decrypts all messages it has received and then forwards them to their next destination in a random order.

Timed mixes are also vulnerable to an active attack similar to the $(n - 1)$ -attack against threshold mixes. In a *trickle attack*, an active attacker delays all messages from entering the mix until t seconds elapse and the mix flushes. The attacker then lets a particular message of interest M enter the mix, but continues to prevent all other messages from entering the mix. When the mix flushes again, it will have only a single output which trivially corresponds to M .

The $(n - 1)$ attack and the trickle attack are often collectively referred to as *blending attacks* [O’Connor 2005; Serjantov et al. 2002; Diaz and Preneel 2004].

3.3.3 Threshold and/or Timed Mixes. We can also consider a combination of the simple threshold and timed flushing algorithms. A *threshold-or-timed* mix collects messages until either it has received n messages *or* until t seconds have elapsed since the last time the mix was flushed, whichever occurs first. Similarly, a *threshold-and-timed mix* collects messages until t seconds have elapsed *and* it has collected at least n messages. With both algorithms, all collected messages will be flushed

at the end of each round [Serjantov et al. 2002].

Combining the threshold and timed flushing algorithms does not mitigate the threat of $(n - 1)$ or trickle attacks. Instead, the adversary simply combines the two attacks, flooding the mix with dummy messages and then waiting t seconds for the mix to flush.

3.3.4 Pool Mixes. Instead of flushing all messages collected during the previous round, *pool mixes* select a random subset of the collected messages to flush and then retain the rest in the mix for the next round [Serjantov et al. 2002; Díaz and Serjantov 2003; Serjantov and Newman 2003]. The set of messages retained in the mix is referred to as the *pool*. When the mix process is first started, the pool is initialized with dummy messages that are indistinguishable from actual messages.

The simplest type of pool mix simply retains a constant number of messages in the pool each round, which we denote as N_p . In the case of a *threshold pool mix*, the mix waits until it has received n messages in addition to the N_p messages in the pool. It then randomly selects n messages from the $n + N_p$ total messages in the mix to forward.

Timed pool mixes operate in a similar manner, collecting messages for t seconds and adding them to the pool. Let n_t denote the number of messages received by the mix in the previous t seconds. The mix then randomly selects n_t messages from the pool of $n_t + N_p$ total messages to forward.

Instead of retaining a constant number of messages in the pool and flushing the rest, *dynamic pool mixes* send a fraction f ($0 < f \leq 1$) of the total messages in the pool at the end of a round. If there are $n + N_p$ messages in the mix at the end of the round, a dynamic pool mix randomly selects $\max\{1, \lfloor n \cdot f \rfloor\}$ messages to forward [Serjantov et al. 2002]. The rest are retained in the pool for the next round. In some sense, the simple threshold and timed mixes above could also be seen as instances of a dynamic pool mix, but with $N_p = 0$ and $f = 1$.

While pool mixes do not eliminate the threat of $(n - 1)$ and trickle attacks, they do increase the effort required on the part of the adversary. Instead of positively identifying a message of interest after attacking a single mix round, the attack becomes probabilistic since the message may be retained in the pool for an unpredictable number of rounds.

Of course, the probability that a message will remain in the pool for r rounds decreases as r increases. O'Connor [2005] presented a full analysis of the number of rounds required for an adversary to perform an $(n - 1)$ attack against threshold, timed, and dynamic pool mixes.

3.3.5 Stop-and-Go Mixes. Kesdogan et al. [1998] proposed a mixing algorithm they termed a *Stop-and-Go mix* (SG-mix), sometimes also categorized as a *continuous mix* [Danezis 2004]. Instead of batching messages together, SG-mixes individually delay messages as they pass through the mix.

A client sending a message through a series of n SG-mixes calculates a time window $[t_{min}, t_{max}]_i$ and chooses a random delay t_i from an exponential distribution, for each mix $i = 1 \dots n$. The time windows are computed such that $(t_{min} + t_i) \in [t_{min}, t_{max}]_{i+1}$ and also $(t_{max} + t_i) \in [t_{min}, t_{max}]_{i+1}$. The time window and random delay for mix i are included in the message encrypted with mix

i 's public key.

After decrypting a received message, mix i can extract the time window $[t_{min}, t_{max}]_i$ and the random delay t_i chosen by the sender. If the current time t_{now} is outside the given time window, the mix discards the message. Otherwise, the mix will forward the message to the next hop at time $t_{now} + t_i$.

3.3.6 Binomial Mixes. All of the flushing algorithms presented thus far output a deterministic number of messages given the internal state of the mix and its parameters. The *binomial mix* instead flushes a random fraction of messages it has collected over previous rounds [Díaz and Serjantov 2003].

At the end of a mix round lasting t seconds, a binomial mix flips a biased coin with a forwarding probability $p_f = P(n)$ for each message currently contained in the mix, where n is the total number of messages in the mix and $P(n)$ is the coin's bias when the mix contains n messages. If the result of the coin flip is heads, then the message is forwarded to its next hop. Otherwise, the message is retained in the mix for the next round. The result of this approach is that the actual number of messages output each round is selected from a binomial distribution with mean np_f and variance $np_f(1 - p_f)$.

Again, this algorithm does not eliminate the possibility of active blending attacks. Díaz and Serjantov [2003] showed, however, that it can increase the amount of time required for a successful attack, as well as require the adversary to spend more resources sending dummy messages into the mix. [O'Connor 2005] also derived the expected number of mix rounds required for an adversary to perform an $(n - 1)$ attack against binomial mixes.

3.3.7 RGB Mixes. While none of the flushing algorithms proposed to date have entirely eliminated the threat of the simple but effective $(n - 1)$ attack, Danezis and Sassaman [2003] proposed a technique for detecting when an attacker is conducting the attack, and minimizing its negative impact on the anonymity of legitimate messages. The authors called their approach Red-Green-Black (RGB) mixes.

In RGB mixes, the genuine client messages received during a mix round are considered *black* messages. Since messages are encrypted, it is impossible for a mix to tell what fraction of the received messages are black messages and which are messages generated by an adversary conducting an $(n - 1)$ attack. In order to estimate the number of legitimate messages received during a round, mixes send out "heartbeat" traffic, called *red* messages, with each output batch. Red traffic is indistinguishable from black traffic to an outside observer, and is "anonymously" addressed back to the mix itself. Since the red traffic is addressed to itself, the mix can monitor the fraction of red messages it receives compared to black traffic. If it detects a statistically significant drop in its received red traffic, the mix can deduce that it is currently under attack.

When the mix detects an attack, it generates additional dummy messages, called *green* traffic, and sends them out with each batch. The intuition behind the green messages is that they increase the anonymity of the legitimate, black messages output during each round, even though the mix is under attack. Additionally, an adversary is unable to distinguish green traffic from legitimate black traffic, which makes identifying the message targeted by an $(n - 1)$ attack more difficult.

3.4 Deployed Systems

While some of the work on mixes and mix networks has been purely academic and theoretical, there have been a number of high-latency remailer systems publicly available.

3.4.1 *anon.penet.fi*. One of the early deployed remailers was the rather popular Finnish `anon.penet.fi` pseudonymity service started by Johan Helsingius in 1993 [Helmert 1997]. The technical details of Helsingius's service are quite simple. A client simply sends a message to the service, which then generates an alias for the sender, if one does not already exist, and stores a mapping of that alias to the sender's real email address in a database. The service then strips any identifying information from the message and forwards it to the intended recipient. If the remailer receives a message addressed to a previously-generated alias, the remailer looks up the alias's true address in the database and the message can be delivered to the pseudonymous user.

At one time, `anon.penet.fi` processed over 10,000 messages daily and the database of pseudonyms contained over 200,000 entries [Helmert 1997]. Clearly the pseudonym database represents a centralization of trust and a likely target for attack. Indeed, Helsingius was eventually subjected to legal attacks from the Church of Scientology claiming a user posted a message to a newsgroup that infringed on their copyright, and they wanted Helsingius to reveal that user's identity [Newman 1996]. The service was subsequently shut down in 1996.

3.4.2 *Cypherpunk Remailers*. Cypherpunk remailers, sometimes referred to as Type I remailers, were first developed by Hughes and Finney and are loosely based on Chaum's mix design [Parekh 1996]. Unlike the `anon.penet.fi` system, the cypherpunk remailer system consists of multiple, distributed servers. Clients pick a sequence of servers and include a header for each hop that specifies the next server in the path to which the payload should be forwarded. Each hop uses its private key to decrypt the message it receives, removes the header information, and forwards the payload to the next server.

Unlike Chaum's design, however, cypherpunk remailers do not add random padding to messages after removing the control commands contained in the headers encrypted for each hop. A passive adversary observing the network can thus potentially correlate messages using the knowledge that the messages decrease in length as they pass through the network. Additionally, cypherpunk remailers do not do any explicit batching and delaying, so a passive adversary can also correlate messages passing through a remailer based on entry and exit times.

Cypherpunk remailers supported anonymous replies through a construction called *reply blocks*. A reply block is constructed in a layered manner similar to a normal message. A sender can then include the reply block in a message anonymously sent to a recipient, who would attach it to their reply. The reply is routed through the remailer network according to the reply block and back to the original sender.

Matt Ghio later implemented a *nymserver* that simplified the use of reply blocks [Parekh 1996]. A user can anonymously send their reply block to the nymserver through a chain of remailers. The nymserver then allocates an alias for the received reply block, and stores the pair in a database. When the nymserver receives a message

addressed to an alias, it forwards the message through the remailer network using the stored reply block for that alias. We see the nymserver operates similar to `anon.penet.fi`, but, since the reply blocks are sent to the nymserver anonymously, the nymserver does not map an alias to an actual identity.

3.4.3 *Mixmaster*. The Mixmaster, or Type II remailer design, made several design improvements over the cypherpunk remailers, including the addition of message padding and batching [Möller et al. 2003]. Mixmaster uses a timed dynamic pool flushing algorithm. Once every t seconds, a mix randomly chooses

$$\min\{N - N_p, N * f\}$$

messages to forward, where N is the total number of messages in the mix, N_p is the minimum number of messages to retain in the pool, and f is the fraction to forward each round.

Mixmaster additionally tries to defeat replay attacks by recording packet IDs included in a message header, encrypted for each hop. If a mix receives a message with an ID of a packet it has already processed, the duplicate is simply discarded. Unlike cypherpunk remailers, however, Mixmaster does not include support for anonymous replies.

As of December 2007, the public Mixmaster network has 24 running servers.

3.4.4 *Mixminion*. Mixminion, or Type III remailer, is the current state-of-the-art in remailer design [Danezis et al. 2003].

Mixmaster's approach to replay detection required the mix to keep a list of recently processed message IDs. Since mixes have a limited amount of memory, old message IDs are necessarily purged after a period of time. An adversary can easily replay a message again after its message ID has been purged from the mix's history. Mixminion also keeps a history of recently processed messages, but mixes periodically rotate the key clients use to encrypt messages. After the key is rotated, messages encrypted with the old key are no longer accepted and cannot be replayed. Thus, Mixminion servers must only retain hashes of previously processed messages for a relatively shorter amount of time.

Mixminion's design is also an improvement over Mixmaster in that it includes a specification for how clients can learn about remailers in the Mixminion network, as opposed to Mixmaster's informal, *ad hoc* approach. A distributed set of redundant *directory servers* provide clients with information about the current mixes in the network (e.g., their IP addresses, public key information, etc.). When clients join the network, they can fetch a summary of this information from the directory servers.

Like the cypherpunk remailers, Mixminion also supports anonymous reply blocks. In the cypherpunk remailer and Babel designs, a reply block can be used multiple times which can be exploited by an active attacker to trace the intended recipient back through the network. The adversary can send a message with the same reply block into a mix over multiple mix rounds and observe the outputs. The next hop for that particular reply block must be in the intersection of all observed outgoing batches. Repeating this process through several hops can eventually lead the attacker to identify the original owner of the reply block.

Rather than allowing the same reply block to be replayed multiple times, Mixmin-

ion’s reply blocks instead are *single-use* reply blocks (SURBs). The anonymous recipient must therefore create a new SURB for each reply he wants to receive. Unlike Type I remailers, the designers of Mixminion were careful to ensure replies are indistinguishable from normal forward messages. Since replies are likely to be less frequent than forward messages, it is beneficial for the former to share an anonymity set with the latter message type.

While Mixminion’s SURBs resist replay attacks, they are still susceptible to long-term intersection attacks (we will describe such attacks in Section 6). Rather than rely on reply blocks, designs like the Pynchon Gate [Sassaman et al. 2005] have suggested instead storing mail sent to a pseudonym on a nymserver. The pseudonym holder can then use a private information retrieval protocol [Chor et al. 1995] to fetch its mail from the nymserver.

As of December 2007, the public Mixminion network has 29 running servers.

4. LOW-LATENCY ANONYMITY SYSTEMS

The flushing algorithms used by the mix-based systems above can introduce large delays on the order of several hours or even days between the time a message is sent and when it arrives at the intended recipient. For applications like email, such delays are tolerable and often even expected; however, real-time, interactive applications, like Web browsing and SSH, require significantly lower latency.

The improved performance of low-latency anonymity systems, however, can increase a system’s susceptibility to certain traffic analysis attacks [Shmatikov and Wang 2006; Serjantov and Sewell 2003; Liberatore and Levine 2006]. We will examine such attacks in greater detail in Section 6.

4.1 Anonymizer.com

We saw in the previous section that the basic building block for many high-latency anonymity systems is typically a mix. Low-latency anonymity systems, however, are often based on the notion of a *proxy*. While mixes explicitly batch and reorder incoming messages, proxies simply forward all incoming traffic (e.g., a TCP connection) immediately without any packet reordering.

`anonymizer.com`, for example, is a company that offers a commercial proxy service. Clients pay a subscription fee and then are allowed to relay their Web traffic through proxy servers operated by the company. Client Web requests then appear to originate from `anonymizer.com`’s servers, rather than from the client’s own IP address. Such an approach is simple and efficient, but it requires clients to trust the company to not monitor or log their traffic.

In the remainder of this section, we will consider other system designs that seek to provide strong anonymity to clients while maintaining a level of performance acceptable for interactive applications.

4.2 Onion Routing

Arguably the most prevalent design for low-latency anonymous communication in the literature is an approach called *onion routing*, initially developed by Goldschlag et al. [1996]. Unlike Crowds, onion routing clients do not necessarily relay traffic for other clients. Instead, there is a set of servers called *onion routers* that relay traffic for clients. We let $\mathcal{R} = \{R_1, R_2, \dots, R_n\}$ represent the set of n onion routers

in the network. As in mix networks, each onion router also maintains a private and public key pair, the public component of which should be made known to clients. The original onion routing prototype, however, did not include a formal mechanism for distributing such information to clients or other onion routers [Syverson et al. 2000].

The first step to establishing an anonymous application-level connection over an onion routing network is for the initiator to construct a multiply-encrypted tunnel, or *circuit*, through the network. The initiator first selects an ordered sequence of servers in the network to use as the circuit's path, much like in a mix network.

In mix networks, however, the initiator has to perform separate asymmetric cryptographic operations for each mix along the path for each message it wants to send. Additionally, each mix also has to do an asymmetric cryptographic operation for every message it processes. Such an approach is acceptable for email messages that are likely fairly infrequent; however, loading a single Web page, for example, may require the initiator to also fetch several large images or many other Web objects. Using asymmetric cryptography for every object requested would place a large computational cost on both initiators and servers in the network. To minimize the computational costs imposed, senders in an onion routing network only use public key cryptography to establish the encrypted circuit and then use faster symmetric key cryptography to transfer the actual data.

The initiator randomly generates two symmetric secret keys—a forward key KF_i and a backward key KB_i —for each router R_i along its path. It also chooses forward and backward cryptographic functions, f_i and f_i^{-1} , respectively, corresponding to the generated symmetric keys. The pair (KF_i, f_i) is used to crypt data as it traverses the path from the initiator towards the responder (e.g., a website), and the pair (KB_i, f_i^{-1}) is applied to data from the responder to the circuit initiator.

As an example, if the initiator chose the path $\mathcal{P} = \{R_x, R_y, R_z\}$, the initiator would construct an encrypted “onion” as follows:

$$E_x(t_x, f_x, KF_x, f_x^{-1}, KB_x, R_y, \\ E_y(t_y, f_y, KF_y, f_y^{-1}, KB_y, R_z, \\ E_z(t_z, f_z, KF_z, f_z^{-1}, KB_z, \emptyset)))$$

The values t_i in the onion indicate the expiration time of the onion, and the symbol \emptyset is used to indicate to R_z that it is the last router in the path. The initiator sends the onion to the first router in the path, R_x , who removes the outermost layer of encryption using its private key, and learns the symmetric keys KF_x and KB_x generated by the initiator, as well as the next server in the path. R_x then pads the remaining encrypted payload with random bytes, so the onion maintains a constant length, and sends the result to R_y . Each server along the path repeats this process, until the onion reaches the end of the path.

Once the circuit is constructed, the initiator can relay its application traffic over the circuit using the symmetric keys generated for each hop. Similar to how the onion was constructed, the initiator encrypts a message (e.g., a Web request) once for each router in the circuit using the generated forward keys KF_i and then sends it to the first router in the circuit. Each router in the circuit can remove a layer of encryption and forward the result to the next server in the circuit, until it reaches the last server who can then forward the original message to the destination (e.g.,

a Web server).

When a response is sent along the reverse path of a circuit, a layer of encryption is *added* by each router in the circuit using the backward key KB_i and cryptographic function f_i^{-1} given during circuit setup. The initiator can then remove all layers of encryption to recover the original response, since he has the secret keys he previously generated for each router.

4.3 PipeNet

PipeNet was a low-latency anonymity system independently and informally proposed around the same time as onion routing [Dai 1998]. Users in PipeNet first select a random sequence of servers in the network, similar to path selection in onion routing. Clients then set up a multiply encrypted tunnel by establishing a symmetric key with the first hop, tunneling through that encrypted connection and establishing another key with the second hop, and so on. PipeNet's design was, however, impractical for real networks like the Internet, and was never publicly deployed.

All communication in PipeNet occurs in discrete time units. The design assumed there is exactly one packet sent over each link between network nodes during a time unit. If a packet is not received by a node over any one of its links, it does not forward *any* packets for that time unit. As a result, a single misbehaving (or malfunctioning) node can force all communication in the entire network to cease by simply not sending a message for one or more time units [Back et al. 2001].

4.4 Crowds

The Crowds system was specifically designed for anonymous Web browsing [Reiter and Rubin 1998]. Users in the system are represented by processes called *jondos* (*a la* “John Doe”). Jondos are assigned to a *crowd* of other jondos by an administrative process called a *blender*. The blender is also responsible for informing new jondos of other members of the crowd.

When the user's browser first makes a Web request, his jondo establishes a random path through the network by first randomly picking another jondo (perhaps even itself) from the crowd and forwarding the request to it. That jondo then flips a coin biased with a forwarding probability $p_f > \frac{1}{2}$. Depending on the outcome of the coin flip, the jondo either randomly selects another member of the crowd to which the request will be forwarded, or it forwards the request to the intended Web server. Each jondo also remembers the hop before it along the forwarding path, so that when a reply is received from the Web server the reply is relayed via the reverse of the established path. The pairwise connections between jondos are encrypted using shared keys assigned by the blender when a new jondo joins the crowd.

Since each member of the crowd relays requests for any other member, when a jondo receives a request it does not know if that jondo was the original requester or if it simply forwarded the request for another jondo; however, one or more malicious jondos on the path may collaborate to try to identify the true initiator of a connection. Consider a path that includes at least one malicious, collaborating jondo. The Crowds designers proved the first collaborating node's predecessor is no more likely to be the true initiator of that connection than to *not* be the initiator

(a property they defined as *probable innocence*) if

$$n \geq \frac{p_f}{p_f - 1/2}(c + 1),$$

where n is the total number of jondos, of which c are collaborating malicious jondos.

Clearly there is also trade-off between the forwarding probability p_f and the length of the established path and, consequently, performance for the user. Note that there is no explicit batching, delaying, or reordering of messages as they are relayed through the crowd. This is true of nearly all low-latency anonymity system designs.

4.5 Java Anonymous Proxy (JAP)

Web Mixes [Berthold et al. 2000], as the name suggests, is another system intended to provide anonymity for interactive, real-time Internet applications, like Web browsing. Web Mixes, however, is quite different from the onion routing and Crowds designs.

The system consists of the Java Anonymous Proxy (JAP) software that runs on a client's computer and forwards client traffic into one of several available mix cascades. Each mix in a cascade cryptographically transforms received packets and reorders packets among data streams before forwarding them to the next mix. The last mix in the cascade forwards traffic to a “cache-proxy,” which forwards the anonymized Web request to the intended destination. The cache-proxy also scans the Web response for embedded objects and fetches those as well, before sending the entire response back through the cascade to the client.

In an effort to hinder traffic analysis, the Web Mixes design paper calls for the JAP software to maintain a constant traffic rate between the client and the first mix in the cascade. It will insert dummy traffic if the client does not have any actual data traffic to transmit. Constant-rate traffic padding is also suggested between the cache-proxy and the last mix of each cascade. The deployed implementation, however, does not use any cover traffic due to the excessive load it would place on the network.

The JAP software has been publicly available for several years and is second in popularity only to Tor (discussed in Section 4.8). It has recently evolved into a commercial operation under the name JonDonym [JonDos GmbH 2008].

4.6 Tarzan

Tarzan is a low-latency anonymity system [Freedman and Morris 2002], and is loosely related to the original onion routing design. Like onion routing, initiators in the Tarzan network construct circuits through the network by generating symmetric keys for each hop and iteratively encrypting them with public keys of the servers in the circuit. The symmetric keys are then used to relay data over the circuit. Tarzan, however, uses UDP for a transport protocol, as opposed to Tor that uses TCP.

Like Crowds, all participants in the network relay traffic for other users. The novelty in the Tarzan design is in how clients learn about other servers in the network. Rather than rely on a small set of directory servers with a complete view

of the network, Tarzan uses a peer-to-peer “gossip” protocol to share information about other servers. When a node initializes, it discovers other servers by selecting a random neighboring node that it already knows about (say, from a set of bootstrap nodes) and asking that neighbor for all the other servers the neighbor knows about. The requesting node can then select another random node from the newly-learned set of servers and repeat the process.

Tarzan also introduces cover traffic in the network, intended to help obscure actual data traffic patterns and prevent a global passive adversary from linking traffic flows to their initiators. When a node joins the network, it selects k other nodes called *mimics*, and asks them to exchange “mimic traffic” with it. Each node selects its mimics based on the result of repeated application of a cryptographic hash function incorporating the node’s IP address and the current date. As a result, mimic selections are universally verifiable, preventing an adversary from selecting more than k mimics. Mimic selections are also symmetric, otherwise nodes would only send data on their outgoing links.

The initiator of an anonymous tunnel creates a circuit through the Tarzan network by choosing the first hop randomly from his set of mimics. The second hop is selected from the set of mimics chosen by the first hop, and so on. Application traffic is relayed over the circuit using a layered encryption approach similar to onion routing.

While a node is online, it exchanges mimic traffic with its neighboring mimics that is indistinguishable to an outside observer from application traffic. The rate at which a node sends outgoing traffic is intended to approximately match the rate at which it is receiving traffic. Since the initiator of a circuit is also exchanging mimic traffic with other nodes, someone watching the node has a greater difficulty identifying it as the source of a particular circuit. Additionally, the first hop in a circuit does not know whether the traffic it is receiving is cover traffic or application traffic.

We note that, due to the cover traffic included in the Tarzan design, we could perhaps have classified it as a system for unobservability in Section 5 and still adhere to Pfitzmann and Hansen’s definition of unobservability [Pfitzmann and Hansen 2008]; however, due to its close relation to onion routing and MorphMix, we have chosen to include it in this section for clarity and coherence.

4.7 MorphMix

Similar in spirit to Tarzan, MorphMix [Rennhard and Plattner 2002] is a peer-to-peer low-latency anonymity system in which all nodes in the network relay traffic for other nodes. Like Tarzan and onion routing, MorphMix uses multiple nested symmetric encryptions to transfer data along circuits. MorphMix’s approach to peer-discovery and circuit construction, however, is quite different from previous approaches and is considerably more complex.

To construct a circuit, the circuit initiator a chooses a node b from the set of nodes it currently knows about. a then requests from b a signed list of other nodes that it knows about, from which a can pick the next hop of its circuit. Let x be the next hop chosen by a from the set of nodes supplied by b . Further, let w be another node previously known to a , which will be used as a “witness” node.

Node a establishes a symmetric key with x by sending to b half of a Diffie-Hellman

(DH) handshake encrypted with the witness w 's public key. b relays the ciphertext to w who removes the layer of encryption and forwards the result to x . x generates its half of the DH handshake and sends it to a via b . Both sides are then able to generate their shared symmetric key. Note that no nodes other than a and x ever see both halves of the DH handshake, thus there is no opportunity for a man-in-the-middle attack (assuming x is honest). The initiator repeats this process for each hop in its circuit.

If a new node queries a malicious node collaborating with other nodes, however, the malicious node could send the witness only nodes with which it is collaborating. Thus, he can induce a to choose a path consisting only of malicious nodes, who can then collude to identify the initiator and his destination. To try to minimize this attack, the MorphMix designers attempted to add a collusion detection mechanism that tracks which nodes appear in a tunnel together. If certain nodes are colluding, then they will appear together in tunnels with higher probability than non-colluding nodes. Unfortunately, this approach was shown to be easily defeated by an adversary by offering sets of colluding nodes to the initiator that intersect as little as possible [Tabriz and Borisov 2006].

4.8 Tor

Tor [Dingledine et al. 2004a] is the most recent evolution of onion routing and represents the current state-of-the-art in low-latency anonymity systems. The Tor design makes several modifications and improvements over the original onion routing design in terms of security, efficiency, and deployability. The Tor network is currently the largest deployed anonymity network ever, consisting of over 1,000 servers and more than an estimated 250,000 users as of December 2007.

4.8.1 Directory Servers. The original onion routing prototype did not include any formal specification for how clients learn about routers in the network. The Tor design includes a specification for a small set of trusted *authoritative directory servers*, similar to those in Mixminion. The directory servers are responsible for aggregating and distributing signed information about known routers in the network. The signed directory information can also be mirrored by other routers in the network in order to reduce load on the authoritative directory servers.

4.8.2 Circuit Establishment. Tor clients use a *telescopic* approach to circuit construction, a concept first introduced by Cebolla [Brown 2002]. The initiator of a circuit iteratively negotiates ephemeral session keys with each router in the circuit's path using Diffie-Hellman (DH) key negotiation and one-way RSA authentication. When a key is established with the first hop, the initiator can tunnel through that encrypted connection, establish an ephemeral session key with the second hop, and so on. When a circuit is no longer used, the session keys are discarded.

This telescopic approach to circuit establishment results in two key benefits. First, discarding the session keys when the circuit is closed means that subsequently compromising a router does not allow an adversary to recover the session key and decrypt traffic encrypted under that key, giving Tor clients *perfect forward secrecy*. Second, onion routers no longer need to store hashes of previously processed onions in order to prevent replay attacks. Replaying one side of a DH handshake results in a different session key, yielding no information to an attacker.

There have recently been two separate proposals for modifying Tor’s circuit construction algorithm. Kate et al. [2007] suggested a protocol using pairing-based cryptography to establish circuits. A benefit of their approach is that session keys can be established with in “one pass” rather than interactively negotiating keys with each successive hop in the circuit, resulting in lower computation and communication overhead. Their design, however, relies on one or more trusted nodes to generate and distribute private keys for all routers in the network.

Another alternative circuit construction algorithm presented by Øverlier and Syverson [2007] relies on precomputed DH parameters, the public components of which are published by each router in Tor’s directory service and updated regularly. Clients can then generate their own DH parameters and, using the routers’ public DH information, compute the session keys to be shared with routers in their circuit. The approach of Øverlier and Syverson [2007] effectively eliminates three RSA encryption operations for clients and one RSA decryption operation for each node in a circuit. It also has the benefit of allowing clients to do DH computations during their idle CPU cycles.

4.8.3 Location-hidden Services. The Tor design also added an interesting feature called *location-hidden services*, or simply *hidden services* for short. Hidden services allow users to provide services, such as a Web server, accessible to anyone with a Tor client, but without revealing the IP address or location of that server [Dingledine et al. 2004a].

Hidden services maintain anonymous connections to a subset of nodes in the network known as *introduction points*. Clients can then connect to the hidden service’s introduction points and specify another node known as a *rendezvous point*, along with half of a DH handshake. Both the client and the hidden service connect to the rendezvous point and establish an encrypted tunnel through it. This somewhat roundabout approach is intended to prevent the introduction points from being perceived as responsible for the content offered by a disreputable hidden service.

Øverlier and Syverson [2006] described an attack that can allow an adversary to identify the true host of a hidden service. The adversary simply runs a node in the network and makes many requests to the hidden service. Over time, the hidden service will repeatedly choose the adversary’s server as the first hop in his circuit to the rendezvous point, since circuit paths are chosen randomly. The adversary can then statistically determine the most likely host of the hidden service.

The authors described a countermeasure to their attack called *entry guards*, which are similar in concept to *helper nodes* introduced by Wright et al. [2003]. The hidden service chooses a small set of nodes (its “entry guards”) from which it will always select the first hop in its circuits. Of course, if some or all of the chosen entry guards are controlled by the adversary then the attack can still succeed; however, if none of the selected entry guards are compromised then the attack described above can *never* succeed since the adversary’s node will never be adjacent to the hidden service on the hidden service’s circuit to the rendezvous point. Thus, entry guards give the hidden service some possibility of defending against this attack, whereas without them the attack can always succeed.

5. SYSTEMS FOR UNOBSERVABILITY

The anonymity systems described in the previous section aim to prevent an attacker from determining who is communicating with whom. Even though the adversary is unable to identify the intended recipient of a particular message seen entering the network, or the original sender of a message observed exiting the network, he can still identify which senders or recipients were active during a period of observation. We can also imagine scenarios in which users want to hide even the fact that they sent or received a message.

In this section, we describe anonymity systems that provide *unobservability*—a stronger property than unlinkability. In a system providing unobservability, an adversary monitoring the users of the system is unable to distinguish messages carrying actual content from random noise. Consequently, not only does the system conceal who is communicating with whom, but it also hides which users sent or received a message during a period of observation.

As we will see in this section, the current designs for systems that provide unobservability also introduce significant latency and overhead that often make them impractical for real-time anonymous communication over the Internet.

5.1 A Naïve Solution

To illustrate a naïve implementation of a system for unobservability, consider a system consisting of n users connected via a broadcast channel. During an initial setup phase, each user generates a public and private key pair under an asymmetric cryptosystem that provides key privacy [Bellare et al. 2001]. The public component is distributed to all other users via an authenticated—but not necessarily private—channel. The system also has a parameter t that specifies the length of a time interval, and all users have approximately synchronized clocks.

At the expiration of the time interval t , all n users in the system simultaneously broadcast a fixed-length message of l bytes. If one user wants to send a message to another user during that time interval, the sender encrypts the message m with the public key of the recipient and broadcasts the resulting ciphertext to all other users. If necessary, m is either truncated to l bytes or padded with $l - |m|$ randomly generated bytes before encrypting the message. Users who do not want to send an actual message to another user during that time interval simply broadcast l random bytes.

Upon receiving a broadcast message c , each recipient attempts to decrypt c using his private key. Under many standard encryption systems, if the result is a plaintext message that “makes sense”, then the recipient knows that another user sent it a message. If c is simply a message of random bytes, or is the ciphertext of a message encrypted with another user’s public key, then the result of a “decryption” will also appear to be random bytes. Thus, the other users in the system who are not the intended recipients do not learn whether the broadcasted message is a ciphertext intended for another user, or if it is simply a randomly generated block of bytes.

Since the transmitted messages consisting of randomly generated bytes are bit-wise indistinguishable from an encrypted ciphertext, and all n users broadcast to all other users during each broadcast round, even a global adversary observing the network is unable to determine which users transmitted an actual message to an-

other user during the round. If we assume users have a way of masking or rewriting the source address of a message sent on the broadcast channel, then the system also provides anonymity for the sender.

This naïve system as described is conceptually simple and easy to implement, but is clearly inefficient in terms of network overhead. The time interval parameter t also implies a performance bound on the system. That is, users cannot communicate at more than l/t bytes per second. As t approaches zero, the system starts to behave as a constant-rate broadcast channel which is impractical for a diverse, wide-area network such as the Internet.

In the rest of this section, we consider more sophisticated designs that provide unobservability with improved efficiency, scalability, robustness, or some combination of these properties. Yet, as we will see, many of them still have a similar broadcast-based component that limits their general deployability.

5.2 Dining Cryptographer Networks

In addition to inventing the concept of a mix, David Chaum also devised a system called Dining Cryptographer networks (DC-nets) [Chaum 1988]. In his original work, he proposed the following problem that gives rise to the protocol's name. A group of cryptographers are seated around a table having dinner at a restaurant. At the conclusion of their meal, they are informed by the waiter that their bill has already been paid for anonymously. They speculate that their bill was either paid by one of the cryptographers seated at the table, or by the United States National Security Agency (NSA). The cryptographers respect each other's right to have anonymously paid for the meal, but they are still curious if NSA paid for it instead. If one of the cryptographers did, in fact, pay the bill, can he let the others know while remaining anonymous?

It turns out that indeed such a protocol exists. Assume the cryptographers are seated in a circle. Each cryptographer secretly flips a coin and records a 0 if the result of the flip is *tails* and 1 if it is *heads*. Each then privately shares the result of his own flip with the cryptographer on his left. Given these two values, the cryptographers can each compute $z = x \oplus y$, where x is the result of his own flip, y is the value learned from the cryptographer to his right, and \oplus represents addition modulo 2 (i.e., XOR). He then announces the result to the group. If one of the cryptographers paid for the meal he instead announces the value $z \oplus 1$. Let z_1, z_2, \dots, z_n be the values announced by the n cryptographers seated at the table. They can all jointly compute $Z = z_1 \oplus z_2 \oplus \dots \oplus z_n$.

If none of the cryptographers paid for the meal, then the result Z will be 0. If $Z = 1$, then one of the cryptographers at the table paid the bill. The reasoning is easy to see when we consider the n cryptographers as nodes in an undirected, circular graph, with links between them representing the shared values between neighboring cryptographers. We call this graph a *key graph*. Let $x_{i,j}$ be the bit shared between neighboring cryptographers i and j . Further, let s_i be cryptographer i 's secret bit indicating whether or not he paid for the meal. Thus, each cryptographer i is announcing the result of $z_i = x_{(i-1),i} \oplus x_{i,(i+1)} \oplus s_i$. It follows that

$$\begin{aligned}
Z &= z_1 \oplus z_2 \oplus \cdots \oplus z_n \\
&= (x_{n,1} \oplus x_{1,2} \oplus s_1) \oplus (x_{1,2} \oplus x_{2,3} \oplus s_2) \oplus \cdots \oplus (x_{(n-1),n} \oplus x_{n,1} \oplus s_n) \\
&= x_{n,1} \oplus x_{n,1} \oplus s_1 \oplus x_{1,2} \oplus x_{1,2} \oplus s_2 \oplus \cdots \oplus x_{(n-1),n} \oplus x_{(n-1),n} \oplus s_n \\
&= s_1 \oplus s_2 \oplus \cdots \oplus s_n,
\end{aligned}$$

where the third step follows from a simple reordering of terms. If no cryptographer paid for the meal, $s_i = 0$ for all i . Otherwise, there is precisely one s_i that is nonzero and the result $Z = 1$. This example demonstrates the basic principle behind DC-nets, showing how a member of the network can transmit a single bit without any other communicant knowing who transmitted a message. It is easy to see how this approach can be generalized to arbitrary message lengths and shared secret sizes.

So far we have assumed all nodes in the network are honest, which is certainly not always (if ever) true. Assume cryptographer i did, in fact, pay for the dinner. Now, consider if cryptographers $(i - 1)$ and $(i + 1)$ suspect i as having paid the bill and collude to try to confirm their suspicions. After i announces z_i , the collaborating nodes can combine their shared secrets $x_{(i-1),i}$ and $x_{i,(i+1)}$ to compute

$$\begin{aligned}
z_i \oplus x_{(i-1),i} \oplus x_{i,(i+1)} &= (x_{(i-1),i} \oplus x_{i,(i+1)} \oplus s_i) \oplus x_{(i-1),i} \oplus x_{i,(i+1)} \\
&= x_{(i-1),i} \oplus x_{(i-1),i} \oplus x_{i,(i+1)} \oplus x_{i,(i+1)} \oplus s_i \\
&= s_i.
\end{aligned}$$

Thus, two collaborating nodes can collude to reveal the message sent by a node between them. In order to prevent this, nodes can share a secret key (coin flip) with more than just their neighbors, depending on the expected number of colluding adversaries in the network. Whereas before we had a circular key graph with n nodes and n links, every node can instead exchange a secret key with every other node in the network. The resulting key graph is thus a complete graph. Only if all nodes collude to compromise the anonymity of another can they determine the sender of a particular message.

Clearly the scheme presented thus far is also inefficient in terms of bandwidth overhead. Each message bit anonymously transmitted requires all users to send the result of their local computation to every other user. Thus, a total of $n(n-1)$ bits are transmitted in order to anonymously send a single message bit. Instead, users can arrange themselves in a ring where some “initiating” node i sends $z_i = x_{i,(i-1)} \oplus s_i$ to node $i + 1$. Node $i + 1$ computes $z_{i+1} = z_i \oplus x_{(i+1),i} \oplus s_{i+1}$ and sends z_{i+1} to the next node $i + 2$, and so on. This process repeats around the ring until the last node in the ring broadcasts the final result to all other nodes. This approach only requires $2n$ transmissions, rather than $n(n - 1)$. As n increases to support a large number of users, however, the bandwidth overhead and delay induced by this approach can still become burdensome.

Another pernicious problem with DC-nets is that of message collisions. If two senders try to transmit messages at the same time, their messages will be superimposed yielding an unintended output. Consider again the cryptographers at the table. If two cryptographers claim to have paid for the dinner (i.e., $s_i = s_j = 1$ for

some i, j), the resulting value of Z will be 0. Indeed a single user wanting to disrupt the system for everyone else can simply always broadcast random bits, performing an effective denial of service attack. Detecting and preventing these attacks has been the focus of some research [Waidner and Pfitzmann 1990; Golle and Juels 2004], but the bandwidth costs and inefficiencies of the protocol remain high.

DC-nets are unique in that they offer information-theoretically secure anonymity, rather than relying on computational assumptions. That is, modern cryptosystems, such as RSA, are based on the computational unfeasibility of solving certain problems. The security of DC-nets, however, can not be broken given an unlimited amount of computing power and time [Chaum 1988]. Unfortunately, as we mentioned above, their bandwidth overhead and susceptibility to collisions makes them quite inefficient and infeasible in practice.

5.3 Herbivore

The Herbivore anonymity system (a play on the name of the FBI's Carnivore monitoring system [Electronic Privacy Information Center 2005]) attempted to adapt DC-nets into a design that would make them more efficient and suitable for use in low latency, real-time Internet applications [Goel et al. 2003].

In the simple DC-net example above, all users of the system must combine their privately computed bits to transmit a single anonymous message bit. Clearly this approach does not scale to beyond a few users communicating on a lightly loaded network. Herbivore instead takes a hierarchical approach. When a new user joins the network, it is assigned to one of many smaller groups of users called *cliques*. Every clique has between k and $3k$ users, where k is a parameter of the system. If a clique becomes too small, its users are merged into other cliques. Similarly, once a clique grows too large, it can be separated into smaller cliques. Users within each clique are logically arranged in a star topology, with one user at the center, and all other nodes communicate via this center node. Each user in the clique still has a shared key with every other member of the clique. At a higher level, cliques are arranged in a ring topology, allowing inter-clique communication.

A novel contribution of the Herbivore system design is an anonymous slot-reservation protocol for collision avoidance in a DC-net. Communication in Herbivore takes place over a series of rounds, during which nodes reserve bandwidth on the channel and then transmit fixed blocks of data.

To reserve a slot, each node that would like to transmit during a given round picks a random number $i \in [1, m]$ and then generates an m -bit *reservation vector* r with $r_i = 1$ and the rest filled with zeros. If a node does not want to transmit during a round, it simply generates an all-zero reservation vector. All nodes then simultaneously and anonymously broadcast their reservation vector to the group. Consequently, the reservation vectors will be XORed, with the resulting m -bit vector having a 1 in each position in which some node wants to transmit. Nodes then transmit their message in order during the transmission phase of the round, according to their randomly chosen slot in this reservation vector. The length of the vector is a system parameter chosen according to how many users on average are expected to transmit during a single round.

During each transmission slot in a given round, each node locally computes the XOR of the message it wants to send (if it reserved the current slot) and the pairwise

keys it shares with all other clique members. The center node then computes the XOR of its own local value and those sent by the other clique members, and then broadcasts the result. This star topology approach requires $2(k-1)$ bits to transmit one anonymous message bit.

5.4 \mathcal{P}^5

The \mathcal{P}^5 [Sherwood et al. 2002] system design is not based on DC-nets, but instead relies on users to generate and broadcast cover traffic to hide the real traffic. \mathcal{P}^5 groups users in a logical hierarchy of *broadcast groups*, arranged as a binary tree. When a message is sent to a broadcast group g , the message is also sent to all groups that are descendants of g as well as all groups that have g as a descendant. To send a message in \mathcal{P}^5 , a user first encrypts the message with the intended recipient's public key and then broadcasts the ciphertext to one of the broadcast groups the sender has joined. If the recipient is not in one of the sender's broadcast groups, the message can be anonymously broadcast across other groups in the binary tree.

A unique feature of the \mathcal{P}^5 design is that it allows users to trade off anonymity for performance by letting users choose whether to join groups higher or lower in the broadcast hierarchy depending on whether they desire more anonymity or better performance, respectively. For example, joining the broadcast group at the root of the tree means that all messages sent in that group are also broadcast to every other group in the tree.

Users in \mathcal{P}^5 also generate *noise* traffic at a constant rate, such that the noise packets are indistinguishable from an authentic message packet. As a result, an adversary observing the network is unable to discern when a user is sending an actual message. The noise traffic is sent to a group chosen uniformly at random. Due to the constant rate of traffic from each user and broadcast nature of the system, \mathcal{P}^5 users may have to drop traffic they do not have the resources to handle. Since groups higher up in the broadcast hierarchy receive the most traffic, those channels may experience the highest loss rates.

5.5 Nonesuch

While the previous systems prevent an adversary from distinguishing an actual message transmission from random noise, the adversary is still able to discern which users are participating in the anonymity network. In contrast, the Nonesuch system tries to prevent an adversary from determining that a sender has participated in an anonymity protocol by steganographically hiding messages for a high-latency mix network inside an image [Heydt-Benjamin et al. 2006]. The design assumes the existence of a suitable steganographic algorithm, rather than designing a new one.

In Nonesuch, users use steganography to hide a message in an image file, and then post that image file to a Usenet newsgroup, for example one about photography. Since a simple inspection of an image should not reveal whether it contains a hidden stegotext, the other images in the newsgroup serve as cover traffic. Nonesuch nodes periodically retrieve all news images that were posted to the group and try to extract a stegotext from each.

If a plaintext message were embedded as the stegotext, it would be trivial for an adversary to simply retrieve all images in the group, run the stegotext extraction algorithm on each to identify those carrying a hidden message, and then identify

the senders based on who posted the image. To prevent this simple attack, the hidden messages are in fact multiply-encrypted messages to be routed through a high-latency mix network. Consequently, the stegotext is indistinguishable from random bits.

Since Nonesuch nodes cannot initially distinguish actual messages from cover traffic, all stegotext is treated as if it were a legitimate message. After Nonesuch nodes attempt to extract a stegotext from an image, whether that image carries content or is cover traffic, the nodes use a hash of a portion of the stegotext as an index to a global routing table. With some probability, depending on the size of the hash and density of the routing table, the header of a cover traffic message will hash to a position in the global routing table that indicates the next hop in the mix network. If the hash does not match an index in the routing table, the message is simply dropped. Thus, cover traffic is attenuated as nodes attempt to propagate it through the network.

The process of cryptographically transforming a message, looking up the next hop in a routing table, and forwarding the message repeats until all layers of encryption have been removed and the message is delivered to the recipient, or the message is dropped as cover traffic.

5.6 Anonymous Buses

Beimel and Dolev [2003] presented a rather novel approach to unobservability not based on DC-nets. Consider a public transportation system with buses that pick up passengers at predefined bus stations and drop them off at another station according to some fixed schedule. Senders and recipients in a communication network can be similarly modeled as “stations” and the bus can be thought of as carrying messages between them.

In the simplest approach, the path the bus travels is formed by an Euler tour of the senders and recipients in the network. The bus has n^2 “seats”, with seat $s_{i,j}$ reserved for sending a message from user i to user j . If i has a message to send to some node j , then it encrypts its message with j ’s public key and places it in seat $s_{i,j}$ when the bus stops at i . Otherwise, i simply fills $s_{i,j}$ with some random bytes and then sends the bus to the next scheduled stop. When the bus arrives at j , j can try to decrypt the contents of all $s_{*,j}$ to see if it received any messages. Due to the semantic security of the encryption, an observer is unable to discern whether a seat on the bus contains an actual message or simply some random bytes.

The naïve approach above suggests an $O(n)$ time complexity and $O(n^2)$ storage complexity for the bus. Additionally, an active adversary could simply overwrite arbitrary seats on the bus with random data to cause messages to become corrupted. The authors also analyzed more complex variations that attempt to optimize time and storage complexity, as well as Byzantine fault tolerance [Beimel and Dolev 2003].

6. TRAFFIC ANALYSIS ATTACKS

Just as resistance to cryptanalysis attacks is a yardstick by which cryptographic algorithm designers evaluate the security of their algorithms, traffic analysis is often an effective tool that designers of anonymity systems use to determine the resilience of their system against an attacker. Traffic analysis ignores the content of messages

and instead tries to derive as much information as possible from only network traffic metadata, such as packet arrival times and message lengths.

The field of traffic analysis research is indeed broad enough to merit its own survey [Danezis and Clayton 2007]. Instead, here we only focus on some of the most significant traffic analysis techniques that have been applied to anonymity systems. These attacks, broadly, seek to correlate senders and recipients using only metadata passively collected about the messages or traffic flows in the system.

6.1 Website Fingerprinting

If the adversary has a list of guesses as to which site the target user is visiting, the adversary can simply fetch those sites as well. He observes the quantity and length of data packets received in response. From this metadata, the adversary can build a fingerprint of what the website's response traffic looks like when fetched via an encrypted connection [Hintz 2002; Sun et al. 2002].

For example, if you were to visit `cnn.com`, you would fetch not just the page's text, but also several embedded objects such as images. Even if the request is encrypted with SSL, the size and rate of packets returned is not well obscured. This approach, however, does require the adversary to have a set of guesses as to which site a target user is visiting so he can build up a database of website fingerprints to match against.

To mitigate this attack, most modern anonymity systems split messages into fixed-length cells. Messages shorter than the length of a cell are typically padded with random data. Ensuring that all transmitted data packets are of the same length helps to obscure the true nature of the response traffic, potentially frustrating a simple fingerprinting attack. Liberatore and Levine [2006] examined the effectiveness of various approaches to padding data packets. The authors conjectured that they would be able to still approximate the underlying packet size (potentially broken across multiple cells) for low-latency anonymity systems like Tor by also taking into account packet inter-arrival times.

6.2 Timing Attacks

As discussed in Section 4, low-latency anonymity systems like onion routing do not introduce any delays or significantly alter the timing characteristics of an anonymous connection. As a result, a passive global adversary who is able to observe connections entering and exiting the anonymity network may be able to link inputs and outputs based on their patterns of packet inter-arrival times.

The effectiveness of such attacks has been extensively analyzed in the literature [Levine et al. 2004; Zhu et al. 2004; Murdoch and Zieliński 2007; Shmatikov and Wang 2006]. Levine et al. [2004] proposed an approach to mitigating timing attacks called *defensive dropping* in which nodes introduce random packets called *dummy traffic* at the start of the circuit. The dummy traffic is then probabilistically dropped as it passes through the network. Shmatikov and Wang [2006] also proposed a traffic padding scheme where nodes in a circuit inject dummy traffic between each other according to a probability distribution over packet inter-arrival times.

The defenses proposed by Levine et al. [2004] and Shmatikov and Wang [2006] assume a passive adversary. Zhu et al. [2004] showed that an active attacker can ac-

tually induce timing delays into the client’s traffic that allows him to easily correlate input and output flows in an anonymity network.

Timing attacks can also be performed by an active attacker who is unable to observe both ends of a connection. Murdoch and Danezis [2005] demonstrated such an attack against the Tor network. If an adversary can induce a client to connect to a website he controls, the malicious website can send back data in an identifiable traffic pattern. The adversary then probes routers in the Tor network with constant rate traffic. When the adversary probes a router used in the client’s path, the identifiable traffic pattern sent by the malicious website causes the probe response traffic to exhibit a similar timing pattern. The adversary can repeat this attack to trace the client’s connection back to the first router in the client’s circuit.

While the attack proposed by Murdoch and Danezis [2005] only enables the attacker to determine the routers used in a client’s path, Hopper et al. [2007] leveraged the Murdoch-Danezis attack and extended it to try to approximate the client’s network location from among a set of candidates. After applying the Murdoch-Danezis attack, the adversary can estimate the round-trip time between the client and the first node in the inferred path in order to narrow down the set of candidate client locations. After repeating this attack against the client several times over different circuits, the adversary can approximate the likely network location of the client with increasing precision.

6.3 Predecessor Attacks

In an analysis of their Crowds design, Reiter and Rubin [1998] considered an attack in which a number of attackers join a crowd and attempt to identify the initiator of a particular persistent connection. When the paths are periodically reformed in a crowd, the persistent connection will have to be reestablished. Over time, the true initiator of the connection is far more likely to be an adversarial node’s immediate predecessor in a path than any other node. The attack assumes the connection in question can be identified after each path reformation.

Syverson et al. [2000] analyzed a related attack against an onion routing network. If an adversary can observe the first and last hop in a client’s circuit, he can perform a timing analysis attack to try to link the two. The circuit’s initiator is thus the immediate predecessor of the adversary observing the client’s entry node. The authors derived a $\frac{c^2}{n^2}$ bound on the probability that an adversary who has compromised c of out n nodes succeeds in being selected as the first and last nodes in the initiator’s circuit. As the initiator repeatedly establishes a circuit, the probability that one of the paths chosen is compromised increases.

Wright et al. [2004] formalized a general model of these attacks, called *predecessor attacks*. The authors analyzed the expected number of rounds (path reformations) it takes an adversary to be able to identify the initiator of a connection with high confidence for Crowds, onion routing, mix networks, and DC-nets. They showed that a fully-connected DC-net offers the best resilience against the predecessor attack. Such a network topology suffers from quite limited scalability, though, for the reasons discussed in Section 5.2.

6.4 Disclosure Attacks

Kesdogan et al. [2002] introduced an attack they termed the *disclosure attack*. Consider a user, Alice, who repeatedly sends messages to one of m different communication partners in each mix round. A passive adversary observes the messages entering and exiting the mix and wants to identify with whom Alice is corresponding. The adversary first waits until he has observed m mutually disjoint set of recipients $\langle R_1, \dots, R_m \rangle$. Each of these sets contains precisely one of Alice's m communication partners.

The adversary then refines these sets by waiting to observe a new recipient set R that intersects with only one previously observed R_i . R_i is then reduced to $R_i \cap R$. The adversary can continue this attack until he has reduce all R_i to only a single element, thus identifying each of Alice's m communication partners. For this reason, the disclosure attack has also been referred to as an *intersection attack* in the literature [Danezis and Serjantov 2004; Berthold et al. 2000].

The basic disclosure attack is computationally difficult to perform due to the need to find m mutually disjoint sets. Indeed, the first part of the attack is equivalent to the Constraint Satisfaction Problem, which is NP-complete [Kesdogan et al. 2002]. Kesdogan and Pimenidis [2004] proposed more efficient variants called the *hitting set attack* and the *HS** attack that yield “exact” results with fewer observations and lower computational complexity.

Rather than mounting an exact attack that precisely identifies Alice's communication partners, Danezis [2003b] defined a *statistical* disclosure attack that reveals Alice's *probable* recipients. The attack is based on compiling a large number of observation vectors whose elements represent the probability that Alice sent a message to each of the N recipients in that round. From this, the attacker can approximate Alice's behavior.

Disclosure and statistical disclosure attacks rely on several assumptions that may not hold in practice. In particular, they assume that Alice sends to only one of her m communication partners in each round, and that these recipients are chosen with uniform probability. Mathewson and Dingleline [2004] extended the attack to include Alices who send to multiple recipients in each round with non-uniform probability. The model used in the original disclosure attack also assumes a simple threshold mix, though Danezis and Serjantov [2004] showed statistical disclosure can be effective against pool mixes as well. Danezis et al. [2007] later described a variation called the *two-sided statistical disclosure attack* which also takes into account anonymous replies in order to refine the attack.

No efficient method for absolutely preventing intersection attacks has been found; however, inserting dummy traffic is a measure often proposed to reduce the effectiveness of such attacks. Berthold and Langos [2002] observed that intersection attacks are possible if not all users of the system are active all the time. They proposed a scheme where Alice pre-generates dummy messages while she is online, which are subsequently sent by other nodes on her behalf while she is offline. Mathewson and Dingleline [2004] considered Alices who send precisely M messages every round (either real or dummy messages) and showed that the attack can still succeed even if Alice appears online only 99% of the time.

Unfortunately, dummy traffic padding schemes effective against intersection at-

tacks often induce significant bandwidth overhead. Diaz and Preneel [2004] present an overview of the issues related to dummy traffic in mix networks.

7. FUTURE DIRECTIONS

Research in the last 30 years has made significant progress towards enabling private and anonymous communication on the Internet. We have seen how a careful application of simple cryptographic primitives can allow people to communicate freely and openly, without fear of retribution, persecution, or profiling. With an increasing level of public awareness about threats to personal privacy, such as identity theft or online advertisers tracking user behavior, academic and public interest in anonymous communication systems is likely to continue to increase in the near future. Much work remains, however, before anonymous communication becomes practical, reliable, and accessible for all Internet users.

7.1 Blocking Resistance

With recent attention on government-sponsored Internet censorship, such as the Great Firewall of China [Clayton et al. 2006; Crandall et al. 2007], the ease with which anonymizing technologies can be blocked will likely become more problematic. In the case of Tor [Dingledine et al. 2004a], a list of all servers in the network is necessarily publicly available—clients need to be able to learn the IP addresses of servers in the network. A government operating a national firewall can simply retrieve this list of server IP addresses and block connections to them from clients inside the firewall. Letting legitimate clients obtain a list of server IP addresses while preventing an adversary trying to block the network from doing the same is a challenging problem.

The Psiphon circumvention system [Citizen Lab 2008] takes a social network approach to server discovery. The system requires a user on the uncensored side of the firewall to run a proxy server and then communicate its address to a trusted user on the censored side through an out-of-band mechanism. Since the proxy’s address is not publicly listed, the censor no longer has an easy method of identifying and blocking connections to proxy servers outside the firewall. Unfortunately, censored clients who do not know anyone on the outside also have no way of learning about uncensored proxy servers.

Anonymity systems can also potentially be blocked by network censors based on characteristics unique to their network protocols. If a firewall also inspects the payload of packets, as opposed to simply looking at IP headers, it can detect patterns unique to the protocols used by some anonymity networks. For example, early versions of the Tor protocol include the string “Tor” in the certificates used during TLS handshakes [Dingledine and Mathewson 2007], which made the protocol easily identifiable. A firewall could conceivably then terminate connections identified as matching the anonymity network protocol by injecting TCP RST packets [Clayton et al. 2006]. Designing efficient anonymity network protocols with steganographic properties such that they can not be easily fingerprinted and censored while still remaining usable is an open and challenging problem.

Köpsell and Hilling [2004] suggested a blocking-resistant system should follow a “many access points” principle, such that the client only needs to find a single unblocked access point to connect to the rest of the anonymity system. The intuition

is that the adversary can block some, but likely not all of the access points. The question then becomes how do clients learn about access points to the network that are not currently blocked.

Tor has recently adopted this approach with its *bridge relay* design. Volunteers in uncensored countries operate relays called “bridges” that are not listed by the directory servers. Since there is no complete public listing of all bridges, it becomes more difficult for an adversary to find and block all of them. Blocked clients can learn about a small subset of available bridges via a website or automated email responder, and then use the bridges to connect to the Tor network. As Köpsell and Hilling [2004] suggested, however, the problem of building a blocking-resistant anonymity system is likely to continue to be a race between the blockers trying to learn about available access points (bridges) and the anonymity system designers trying to come up with alternate ways of distributing bridge relay addresses.

7.2 Usability

An often overlooked aspect of anonymity is the usability of deployed anonymity systems. Yet, as Back et al. [2001] noted, usability should be a *security* objective in any anonymity system.

With the increasing amount of active Web 2.0 content, such as Flash, Java, and JavaScript, the number of ways for attackers to subvert a user’s anonymity through misconfigured applications or poorly-written software continues to grow. These threats were recognized as early as the original onion routing [Goldschlag et al. 1996] and Crowds [Reiter and Rubin 1998] designs. Clayton et al. [2001] also identified a number of application-level attacks that can compromise a user’s anonymity.

Yet, after over 10 years, the same threats present perhaps the easiest method of attacking a user’s anonymity. Developing better ways for users to identify and avoid potentially malicious Web content is a challenging open problem, not just for anonymity systems, but for Web security in general.

Improving user interfaces for anonymity systems is also a valuable research direction [Dingledine et al. 2007]. Installing and using many of the past deployed anonymity systems has been somewhat cumbersome, limiting their adoption by the general public. Tor has made some progress in this direction by bundling the Tor software with a user interface and other commonly-used tools [Clark et al. 2007], which may have contributed to its popularity. Making it even easier for users to install and configure anonymity software securely can increase a system’s userbase and, thus, increase anonymity for all users [Dingledine and Mathewson 2006].

7.3 Incentives

Any successful anonymity network requires a diverse user base, as well as server operators all over the world. But how can we encourage people to participate in an anonymity system? Acquisti et al. [2003] presented a framework for analyzing potential incentives for actors participating in an anonymity system. The framework incorporates several cost-benefit factors of participating in an anonymity system, such as the financial or time expenses of sending an anonymous message, and benefits of acting as either an honest or dishonest node in the network.

The Freedom Network took a commercial approach, by paying people to operate

servers and charging clients a subscription fee for the service [Boucher et al. 2000]; however, the commercial model was unsustainable and the service quickly went bankrupt. In 2007, JonDonym began offering a similar paid anonymization service based on the JAP system [JonDos GmbH 2008]. The software and a basic service is available to clients at no cost. Optionally, those who desire improved performance can purchase a “premium” account that allows clients to use cascades with improved bandwidth. Operators of mixes in premium cascades can receive payment based on the volume of traffic relayed through their mix.

Tor’s entirely volunteer-based network has thus far proven more successful, as it is the largest deployed anonymity network to date. Still, the demand from clients can exceed the capacity of the relatively limited number of servers. We must devise incentives for volunteers to operate servers in the network, for example by providing higher Quality-of-Service guarantees for the traffic of those who do. Such an approach could have unexplored anonymity implications, though, since identifying a network traffic flow as belonging to a server operator necessarily leaks information about the initiator.

While an anonymity system almost invariably benefits from having more servers in the network, encouraging more users to be clients in the network also helps improve security [Dingledine and Mathewson 2006]. One of the common criticisms of Tor is that Web browsing over Tor is slow, leading some users to simply stop using the system. Given the wide variety of people who use the Internet, it seems only natural that they would also have a wide variety of security and anonymity requirements. That is, users in China discussing Falun Gong may have greater anonymity requirements than someone simply reading the latest news stories, and would be willing to tolerate higher latencies for a greater degree of anonymity.

Consequently, we may be able to encourage more users to participate in an anonymity system by giving them the ability to control their own performance and anonymity trade-offs [Snader and Borisov 2008]. It is important, however, for all user traffic to still blend together despite their self-selected performance differences [Dingledine et al. 2006]. To accurately consider trade-offs between anonymity and performance, users must also have a way of reasoning about the level of anonymity provided to them by the system under their currently chosen performance parameters.

7.4 Scalability

The majority of deployed anonymity systems have not had to be concerned with scalability, since the number of users participating in the network has been relatively limited. As the general public becomes increasingly concerned about threats to personal privacy on the Internet, the number of users of anonymity systems is likely to grow. For an anonymous communication system to be able to support truly ubiquitous use, designs must be able to support not just more users, but also more servers in the anonymity network.

Many designs like Tor rely on a small subset of servers to publish lists of all servers currently available in the network through which clients can route their traffic. It is easy to see that the size of the server directory grows linearly with the number of servers in the network. As a result, more network bandwidth is consumed by sending server directories to clients, instead of actually relaying application traffic.

An assumption inherent in many centralized, directory-based anonymity systems is that every node in the network can connect directly to every other node. As the number and diversity of network nodes increase, it is unlikely that this assumption will hold. Network routes can fail, potentially leaving some nodes unreachable from certain parts of the Internet. Additionally, servers will not have a sufficient number of sockets available to maintain connections to all other servers in a network consisting of hundreds of thousands of servers [Dingledine et al. 2005].

One approach to supporting improved scalability with weaker connectivity assumptions may be to simply split an anonymity network into several smaller networks [Dingledine et al. 2005]. Rather than partitioning the network, Danezis [2003a] examined an alternate mix network topology based on sparse expander graphs. He argued that such network topologies offer improved scalability over fully connected and cascade topologies, while still providing desirable resistance against traffic analysis.

Peer-to-peer designs like MorphMix also present an interesting direction for research into building massively scalable anonymity systems. More recently, Salsa was designed with scalability in mind and is based on a distributed hash table [Nambiar and Wright 2006]. Unfortunately, the peer-to-peer designs presented thus far also have challenging anonymity problems [Tabriz and Borisov 2006; Borisov et al. 2007].

8. CONCLUSIONS

We live in a digital society. Every day many of us read the news, manage our banking, conduct business, and chat with friends and family online. As we place more and more of our lives on the Internet, it seems the threats to our personal privacy are constantly increasing. Data storage is cheap, so many of our activities can be recorded. Advertisers are coming up with more and more interesting ways to harvest and mine personal information from this data for purposes such as targeted advertising.

Fortunately, academic researchers and cypherpunks alike have helped advance the state of the art in anonymous communication system design and implementation in the past thirty years. Publicly available high-latency mix networks, like Mixminion, can provide strong anonymity for email. Low-latency anonymity systems, like Tor, enable anonymous Web browsing, IRC, etc.; however, the rate at which new and interesting attacks against these systems are discovered seems to be outpacing our ability to design effective defenses against them. Some attacks, such as application-level Java- and Flash-based proxy-bypass attacks, remain unaddressed over a decade after they were initially noted.

We feel it is unlikely that there will be a “silver bullet” solution that provides provably secure, efficient, and practical anonymity for all users over the current Internet infrastructure. Though as the threats to personal privacy online continue to grow, public and academic interest in improving existing systems for anonymity is only going to increase. Lessons learned from current anonymity research may also help enable and encourage future global network designs to include privacy and anonymity as a fundamental property.

Finally, we refer interested readers to the Anonymity Bibliography¹ to find many of the references cited in this survey, as well as additional works not covered herein.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their comments and suggestions. We also thank Paul Syverson and Aaron Johnson for helpful discussions while preparing this survey.

REFERENCES

- ACQUISTI, A., DINGLEDINE, R., AND SYVERSON, P. 2003. On the economics of anonymity. In *Proceedings of Financial Cryptography (FC '03)*, R. N. Wright, Ed. Springer-Verlag, LNCS 2742.
- ANDERSON, R. 1996. The eternity service. In *Proceedings of Pragocrypt '96*.
- BACK, A., MÖLLER, U., AND STIGLIC, A. 2001. Traffic analysis attacks and trade-offs in anonymity providing systems. In *Proceedings of Information Hiding Workshop (IH 2001)*, I. S. Moskowitz, Ed. Springer-Verlag, LNCS 2137, 245–257.
- BEIMEL, A. AND DOLEV, S. 2003. Buses for anonymous message delivery. *Journal of Cryptology* 16, 1, 25–39.
- BELLARE, M., BOLDYREVA, A., DESAI, A., AND POINTCHEVAL, D. 2001. Key-privacy in public-key encryption. In *ASIACRYPT '01: Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security*. Springer-Verlag, London, UK, 566–582.
- BELLARE, M. AND ROGAWAY, P. 1994. Optimal asymmetric encryption – how to encrypt with rsa. In *Proceedings of EUROCRYPT 1994*.
- BERTHOLD, O., FEDERRATH, H., AND KÖPSELL, S. 2000. Web MIXes: A system for anonymous and unobservable Internet access. In *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, H. Federrath, Ed. Springer-Verlag, LNCS 2009, 115–129.
- BERTHOLD, O. AND LANGOS, H. 2002. Dummy traffic against long term intersection attacks. In *Proceedings of Privacy Enhancing Technologies Workshop (PET 2002)*, R. Dingledine and P. Syverson, Eds. Springer-Verlag, LNCS 2482.
- BERTHOLD, O., PFITZMANN, A., AND STANDTKE, R. 2000. The disadvantages of free MIX routes and how to overcome them. In *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, H. Federrath, Ed. Springer-Verlag, LNCS 2009, 30–45.
- BONEH, D. AND GOLLE, P. 2002. Almost entirely correct mixing with application to voting. In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS 2002)*, V. Atluri, Ed. Washington, DC, 68–77.
- BORISOV, N., DANEZIS, G., MITTAL, P., AND TABRIZ, P. 2007. Denial of service or denial of security? How attacks on reliability can compromise anonymity. In *Proceedings of CCS 2007*.
- BOUCHER, P., SHOSTACK, A., AND GOLDBERG, I. 2000. Freedom systems 2.0 architecture. White paper, Zero Knowledge Systems, Inc. December.
- BROWN, Z. 2002. Cebolla: Pragmatic IP anonymity. In *Proceedings of the 2002 Ottawa Linux Symposium*.
- CAMENISCH, J., HOHENBERGER, S., KOHLWEISS, M., LYSYANSKAYA, A., AND MEYEROVICH, M. 2006. How to win the clonewars: efficient periodic n-times anonymous authentication. In *Proceedings of the 13th ACM conference on Computer and communications security (CCS 2006)*. ACM Press, New York, NY, USA, 201–210.

¹<http://freehaven.net/anonbib/>

- CAMENISCH, J., HOHENBERGER, S., AND LYSYANSKAYA, A. 2005. Compact e-cash. In *Proceedings of EUROCRYPT 2005*, R. Cramer, Ed. Lecture Notes in Computer Science, vol. 3494. Springer, 302–321.
- CAMENISCH, J. AND LYSYANSKAYA, A. 2001. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT '01)*. Springer-Verlag, London, UK, 93–118.
- CHAUM, D. 1981. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM* 4, 2 (February).
- CHAUM, D. 1988. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology* 1, 65–75.
- CHOR, B., GOLDREICH, O., KUSHILEVITZ, E., AND SUDAN, M. 1995. Private information retrieval. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*. 41–50.
- CITIZEN LAB. 2008. Psiphon. <http://psiphon.civisec.org>.
- CLARK, J., VAN OORSCHOT, P. C., AND ADAMS, C. 2007. Usability of anonymous web browsing: an examination of Tor interfaces and deployability. In *Proceedings of the 3rd Symposium on Usable Privacy and Security (SOUPS 2007)*. ACM, New York, NY, USA, 41–51.
- CLARKE, I., SANDBERG, O., WILEY, B., AND HONG, T. W. 2000. Freenet: A distributed anonymous information storage and retrieval system. In *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*. 46–66.
- CLAYTON, R., DANEZIS, G., AND KUHN, M. G. 2001. Real world patterns of failure in anonymity systems. In *Proceedings of Information Hiding Workshop (IH 2001)*, I. S. Moskowitz, Ed. Springer-Verlag, LNCS 2137, 230–244.
- CLAYTON, R., MURDOCH, S. J., AND WATSON, R. N. M. 2006. Ignoring the Great Firewall of China. In *Proceedings of the Sixth Workshop on Privacy Enhancing Technologies (PET 2006)*, G. Danezis and P. Golle, Eds. Springer, Cambridge, UK, 20–35.
- CRANDALL, J. R., ZINN, D., BYRD, M., BARR, E., AND EAST, R. 2007. ConceptDoppler: a weather tracker for internet censorship. In *CCS '07: Proceedings of the 14th ACM conference on Computer and communications security*. ACM, New York, NY, USA, 352–365.
- DAI, W. 1998. Popenet 1.0. Post to Cypherpunks mailing list.
- DANEZIS, G. 2003a. Mix-networks with restricted routes. In *Proceedings of Privacy Enhancing Technologies workshop (PET 2003)*, R. Dingleline, Ed. Springer-Verlag, LNCS 2760, 1–17.
- DANEZIS, G. 2003b. Statistical disclosure attacks: Traffic confirmation in open environments. In *Proceedings of Security and Privacy in the Age of Uncertainty, (SEC2003)*, Gritzalis, Vimercati, Samarati, and Katsikas, Eds. IFIP TC11, Kluwer, Athens, 421–426.
- DANEZIS, G. 2004. The traffic analysis of continuous-time mixes. In *Proceedings of Privacy Enhancing Technologies workshop (PET 2004)*. LNCS, vol. 3424. 35–50.
- DANEZIS, G. AND CLAYTON, R. 2007. Introducing traffic analysis. In *Digital Privacy: Theory, Technologies, and Practices*, A. Acquisti, S. Gritzalis, C. Lambrinouidakis, and S. di Vimercati, Eds. Chapter 5, 95–117.
- DANEZIS, G., DIAZ, C., AND TRONCOSO, C. 2007. Two-sided statistical disclosure attack. In *Proceedings of the Seventh Workshop on Privacy Enhancing Technologies (PET 2007)*, N. Borisov and P. Golle, Eds. Springer, Ottawa, Canada.
- DANEZIS, G., DINGLELINE, R., AND MATHEWSON, N. 2003. Mixminion: Design of a type III anonymous remailer protocol. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*.
- DANEZIS, G. AND SASSAMAN, L. 2003. Heartbeat traffic to counter (n-1) attacks. In *Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2003)*. Washington, DC, USA.
- DANEZIS, G. AND SERJANTOV, A. 2004. Statistical disclosure or intersection attacks on anonymity systems. In *Proceedings of 6th Information Hiding Workshop (IH 2004)*. LNCS. Toronto.
- DIAZ, C. AND PRENEEL, B. 2004. Taxonomy of mixes and dummy traffic. In *Proceedings of I-NetSec04: 3rd Working Conference on Privacy and Anonymity in Networked and Distributed Systems*. Toulouse, France.

- DÍAZ, C. AND SERJANTOV, A. 2003. Generalising mixes. In *Proceedings of Privacy Enhancing Technologies workshop (PET 2003)*, R. Dingledine, Ed. Springer-Verlag, LNCS 2760, 18–31.
- DINGLEDINE, R., FREEDMAN, M. J., AND MOLNAR, D. 2000. The Free Haven project: Distributed anonymous storage service. In *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, H. Federrath, Ed. Springer-Verlag, LNCS 2009.
- DINGLEDINE, R. AND MATHEWSON, N. 2006. Anonymity loves company: Usability and the network effect. In *Proceedings of the Fifth Workshop on the Economics of Information Security (WEIS 2006)*, R. Anderson, Ed. Cambridge, UK.
- DINGLEDINE, R. AND MATHEWSON, N. 2007. Tor protocol specification. <https://svn.torproject.org/svn/tor/trunk/doc/spec/tor-spec.txt>.
- DINGLEDINE, R., MATHEWSON, N., AND SYVERSON, P. 2004a. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*.
- DINGLEDINE, R., MATHEWSON, N., AND SYVERSON, P. 2005. Challenges in deploying low-latency anonymity. NRL CHACS Report 5540-625, U.S. Naval Research Laboratory.
- DINGLEDINE, R., MATHEWSON, N., AND SYVERSON, P. 2007. Deploying low-latency anonymity: Design challenges and social factors. *IEEE Security and Privacy* 5, 5, 83–87.
- DINGLEDINE, R., SERJANTOV, A., AND SYVERSON, P. 2006. Blending different latency traffic with alpha-mixing. In *Proceedings of the Sixth Workshop on Privacy Enhancing Technologies (PET 2006)*, G. Danezis and P. Golle, Eds. Springer, Cambridge, UK, 245–257.
- DINGLEDINE, R., SHMATIKOV, V., AND SYVERSON, P. 2004. Synchronous batching: From cascades to free routes. In *Proceedings of the Privacy Enhancing Technologies workshop (PET 2004)*. LNCS, vol. 3424. 186–206.
- ELECTRONIC PRIVACY INFORMATION CENTER. 2005. Carnivore. <http://epic.org/privacy/carnivore/>.
- FEAMSTER, N. AND DINGLEDINE, R. 2004. Location diversity in anonymity networks. In *Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2004)*. Washington, DC, USA.
- FREEDMAN, M. J. AND MORRIS, R. 2002. Tarzan: A peer-to-peer anonymizing network layer. In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS 2002)*. Washington, DC.
- GOEL, S., ROBSON, M., POLTE, M., AND SIRER, E. G. 2003. Herbivore: A Scalable and Efficient Protocol for Anonymous Communication. Tech. Rep. 2003-1890, Cornell University, Ithaca, NY. February.
- GOLDBERG, I. 2000. A pseudonymous communications infrastructure for the internet. Ph.D. thesis, UC Berkeley.
- GOLDSCHLAG, D. M., REED, M. G., AND SYVERSON, P. F. 1996. Hiding Routing Information. In *Proceedings of Information Hiding: First International Workshop*, R. Anderson, Ed. Springer-Verlag, LNCS 1174, 137–150.
- GOLDWASSER, S. AND MICALI, S. 1984. Probabilistic encryption. *Journal of Computer and System and Science* 28, 2, 270–299.
- GOLLE, P. AND JUELS, A. 2004. Dining cryptographers revisited. In *Proceedings of Eurocrypt 2004*.
- GÜLCÜ, C. AND TSUDIK, G. 1996. Mixing E-mail with Babel. In *Proceedings of the Network and Distributed Security Symposium - NDSS '96*. IEEE, 2–16.
- HARKAVY, M., TYGAR, J., AND KIKUCHI, H. 1998. Electronic auctions with private bids. In *Proceedings of the 3rd conference on USENIX Workshop on Electronic Commerce (WOEC 1998)*. USENIX Association, Berkeley, CA, USA, 61–74.
- HELMERS, S. 1997. A brief history of anon.penet.fi - the legendary anonymous remailer. <http://www.december.com/cmc/mag/1997/sep/helmerts.html>.
- HEYDT-BENJAMIN, T. S., SERJANTOV, A., AND DEFEND, B. 2006. Nonesuch: a mix network with sender unobservability. In *Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2006)*. ACM Press, New York, NY, USA, 1–8.

- HINTZ, A. 2002. Fingerprinting websites using traffic analysis. In *Proceedings of Privacy Enhancing Technologies workshop (PET 2002)*, R. Dingledine and P. Syverson, Eds. Springer-Verlag, LNCS 2482.
- HOPPER, N., VASSERMAN, E. Y., AND CHAN-TIN, E. 2007. How much anonymity does network latency leak? In *Proceedings of CCS 2007*.
- JAKOBSSON, M., JUELS, A., AND RIVEST, R. L. 2002. Making mix nets robust for electronic voting by randomized partial checking. In *Proceedings of the 11th USENIX Security Symposium*.
- JONDOS GMBH. 2008. Jondonym. <https://www.jondos.de/en/>.
- KATE, A., ZAVERUCHA, G., AND GOLDBERG, I. 2007. Pairing-based onion routing. In *Proceedings of the Seventh Workshop on Privacy Enhancing Technologies (PET 2007)*, N. Borisov and P. Golle, Eds. Springer, Ottawa, Canada.
- KESDOGAN, D., AGRAWAL, D., AND PENZ, S. 2002. Limits of anonymity in open environments. In *Proceedings of Information Hiding Workshop (IH 2002)*, F. Petitcolas, Ed. Springer-Verlag, LNCS 2578.
- KESDOGAN, D., EGNER, J., AND BÜSCHKES, R. 1998. Stop-and-go MIXes: Providing probabilistic anonymity in an open system. In *Proceedings of Information Hiding Workshop (IH 1998)*. Springer-Verlag, LNCS 1525.
- KESDOGAN, D. AND PIMENIDIS, L. 2004. The hitting set attack on anonymity protocols. In *Proceedings of 6th Information Hiding Workshop (IH 2004)*. LNCS. Toronto.
- KILIAN, J. AND SAKO, K. 1995. Receipt-free MIX-type voting scheme - a practical solution to the implementation of a voting booth. In *Proceedings of EUROCRYPT 1995*. Springer-Verlag.
- KÖPSELL, S. AND HILLING, U. 2004. How to achieve blocking resistance for existing systems enabling anonymous web surfing. In *Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2004)*. Washington, DC, USA.
- LEVINE, B. N., REITER, M. K., WANG, C., AND WRIGHT, M. K. 2004. Timing attacks in low-latency mix-based systems. In *Proceedings of Financial Cryptography (FC '04)*, A. Juels, Ed. Springer-Verlag, LNCS 3110, 251–265.
- LIBERATORE, M. AND LEVINE, B. N. 2006. Inferring the source of encrypted HTTP connections. In *Proceedings of the 13th ACM conference on Computer and Communications Security (CCS 2006)*. 255–263.
- MATHEWSON, N. AND DINGLEDINE, R. 2004. Practical traffic analysis: Extending and resisting statistical disclosure. In *Proceedings of Privacy Enhancing Technologies workshop (PET 2004)*. LNCS, vol. 3424. 17–34.
- MÖLLER, U., COTTRELL, L., PALFRADER, P., AND SASSAMAN, L. 2003. Mixmaster Protocol — Version 2. IETF Internet Draft.
- MURDOCH, S. J. AND DANEZIS, G. 2005. Low-cost traffic analysis of Tor. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy*. IEEE CS.
- MURDOCH, S. J. AND ZIELIŃSKI, P. 2007. Sampled traffic analysis by internet-exchange-level adversaries. In *Proceedings of the Seventh Workshop on Privacy Enhancing Technologies (PET 2007)*, N. Borisov and P. Golle, Eds.
- NAMBIAR, A. AND WRIGHT, M. 2006. Salsa: A structured approach to large-scale anonymity. In *Proceedings of CCS 2006*.
- NEWMAN, R. 1996. The church of scientology vs. anon.penet.fi. <http://www.xs4all.nl/~kspaink/cos/rnewman/anon/penet.html>.
- NEWMAN, R. E., MOSKOWITZ, I. S., SYVERSON, P., AND SERJANTOV, A. 2003. Metrics for traffic analysis prevention. In *Proceedings of Privacy Enhancing Technologies workshop (PET 2003)*, R. Dingledine, Ed. Springer-Verlag, LNCS 2760.
- O’CONNOR, L. 2005. On blending attacks for mixes with memory. In *Proceedings of Information Hiding Workshop (IH 2005)*. 39–52.
- ØVERLIER, L. AND SYVERSON, P. 2006. Locating hidden servers. In *Proceedings of the 2006 IEEE Symposium on Security and Privacy*. IEEE CS.
- ØVERLIER, L. AND SYVERSON, P. 2007. Improving efficiency and simplicity of Tor circuit establishment and hidden services. In *Proceedings of the Seventh Workshop on Privacy Enhancing Technologies (PET 2007)*, N. Borisov and P. Golle, Eds. Springer, Ottawa, Canada.

- PAREKH, S. 1996. Prospects for remailers. *First Monday* 1, 2 (August).
- PFITZMANN, A. AND HANSEN, M. 2008. Anonymity, unobservability, and pseudonymity: A consolidated proposal for terminology. Draft.
- PFITZMANN, B. AND PFITZMANN, A. 1990. How to break the direct RSA-implementation of mixes. In *EUROCRYPT '89: Proceedings of the workshop on the theory and application of cryptographic techniques on Advances in cryptology*. Springer-Verlag, New York, NY, USA, 373–381.
- RAO, J. R. AND ROHATGI, P. 2000. Can pseudonymity really guarantee privacy? In *Proceedings of the 9th USENIX Security Symposium*. USENIX, 85–96.
- RAYMOND, J.-F. 2000. Traffic analysis: Protocols, attacks, design issues, and open problems. In *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, H. Federrath, Ed. Springer-Verlag, LNCS 2009, 10–29.
- REITER, M. AND RUBIN, A. 1998. Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security* 1, 1 (June).
- RENNHARD, M. AND PLATTNER, B. 2002. Introducing MorphMix: Peer-to-peer based anonymous internet usage with collusion detection. In *Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2002)*. Washington, DC, USA.
- RIVEST, R. L., SHAMIR, A., AND ADLEMAN, L. 1983. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM* 26, 1, 96–99.
- SASSAMAN, L., COHEN, B., AND MATHEWSON, N. 2005. The pynchon gate: A secure method of pseudonymous mail retrieval. In *Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2005)*. Arlington, VA, USA.
- SERJANTOV, A., DINGLEDINE, R., AND SYVERSON, P. 2002. From a trickle to a flood: Active attacks on several mix types. In *Proceedings of Information Hiding Workshop (IH 2002)*, F. Petitcolas, Ed. Springer-Verlag, LNCS 2578.
- SERJANTOV, A. AND NEWMAN, R. E. 2003. On the anonymity of timed pool mixes. In *Proceedings of the Workshop on Privacy and Anonymity Issues in Networked and Distributed Systems*. Kluwer, Athens, Greece, 427–434.
- SERJANTOV, A. AND SEWELL, P. 2003. Passive attack analysis for connection-based anonymity systems. In *Proceedings of ESORICS 2003*.
- SHERWOOD, R., BHATTACHARJEE, B., AND SRINIVASAN, A. 2002. P5: A protocol for scalable anonymous communication. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*.
- SHMATIKOV, V. AND WANG, M.-H. 2006. Timing analysis in low-latency mix networks: Attacks and defenses. In *Proceedings of ESORICS 2006*.
- SNADER, R. AND BORISOV, N. 2008. A tune-up for Tor: Improving security and performance in the Tor network. In *Proceedings of the Network and Distributed Security Symposium - NDSS '08*. Internet Society.
- STEINER, P. 1993. Cartoon. *The New Yorker*, 61.
- SUN, Q., SIMON, D. R., WANG, Y.-M., RUSSELL, W., PADMANABHAN, V. N., AND QIU, L. 2002. Statistical identification of encrypted web browsing traffic. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*. Berkeley, California.
- SYVERSON, P., TSUDIK, G., REED, M., AND LANDWEHR, C. 2000. Towards an analysis of onion routing security. In *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, H. Federrath, Ed. Springer-Verlag, LNCS 2009, 96–114.
- TABRIZ, P. AND BORISOV, N. 2006. Breaking the collusion detection mechanism of morphmix. In *Proceedings of the Sixth Workshop on Privacy Enhancing Technologies (PET 2006)*, G. Danezis and P. Golle, Eds. Springer, Cambridge, UK, 368–384.
- W Aidner, M. AND PFITZMANN, B. 1990. The dining cryptographers in the disco: Unconditional sender and recipient untraceability. In *Proceedings of EUROCRYPT 1989*. Springer-Verlag, LNCS 434.

- WALDMAN, M. AND MAZIÈRES, D. 2001. Tangler: a censorship-resistant publishing system based on document entanglements. In *Proceedings of the 8th ACM Conference on Computer and Communications Security (CCS 2001)*. 126–135.
- WALDMAN, M., RUBIN, A., AND CRANOR, L. 2000. Publius: A robust, tamper-evident, censorship-resistant and source-anonymous web publishing system. In *Proceedings of the 9th USENIX Security Symposium*. 59–72.
- WRIGHT, M., ADLER, M., LEVINE, B. N., AND SHIELDS, C. 2003. Defending anonymous communication against passive logging attacks. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*. 28–43.
- WRIGHT, M., ADLER, M., LEVINE, B. N., AND SHIELDS, C. 2004. The predecessor attack: An analysis of a threat to anonymous communications systems. *ACM Transactions on Information and System Security (TISSEC)* 4, 7 (November), 489–522.
- ZHU, Y., FU, X., GRAHAM, B., BETTATI, R., AND ZHAO, W. 2004. On flow correlation attacks and countermeasures in mix networks. In *Proceedings of Privacy Enhancing Technologies workshop (PET 2004)*. LNCS, vol. 3424. 207–225.

Received January 2008; revised June 2008; revised September 2008; accepted September 2008.