Sabbia: a low-latency design for anonymous networks

Claudio Agosti[1] and Stefano Zanero[2]

[1] Sikurezza.org research group
http://www.sikurezza.org
vecna@sikurezza.org
[2] DEI-Politecnico di Milano,
via Ponzio 34/5 - 20133 Milano Italy
zanero@elet.polimi.it

Abstract. We present Sabbia, a novel approach for building a low latency anonymous network. Sabbia is characterized by a simple design which offers a perfect forward anonimity to internet traffic, using normal, user-space software. Sabbia is based on natural network concepts to optimize routing, a steganographic approach for data hiding which does not heavily impact performance, and uses expert algorithms and protocol dissection at application layer in order to solve automatically some well-known security problems.

1 Introduction

The Internet is growing up, from its early stages of experimentation to widespread and ubiquitous usage. New problems are thus arising due to the different needs of its different user communities. Since the Internet is becoming an area where significant amounts of money can be made and lost, information exchange and free speech, which were the true innovation behind the first age of the net, are now threatened by information censorship, sometimes openly declared, sometimes more sneaky. Protecting these fundamental rights is the real motivation behind research into privacy, secrecy and anonymity preserving technologies.

Anonymous networks have been widely studied in the past, and can be divided in a rough taxonomy as follows:

1. High latency anonymous networks, e.g. MixMinion [1] and MixMaster [2], used for non interactive data block (such as e-mail transfer)
2. Closed anonymous networks, e.g. FreeNet [3]
3. Low latency anonymous networks, used for anonymous interactive sessions like web navigation, remote administration, and instant messagging

Our study focuses on low latency anonymous networks. The state of art for this type of networks is described in TOR [4], but some questions and problems

[1] This work was partially supported by the Italian FIRB-Perf project. The authors wish to thank Michele Mazzucchi for his help in proofreading the text.

are left open. In this paper, we examine these open issues and propose a new architecture, named Sabbia ("sand" in Italian), which aims to solve them.

In anonymous networks, each of the peers must offer its own address as a temporary "shield" to other participants. One of the unique characteristics of Sabbia is the ability, for each peer in the network, to choose exactly how much bandwidth it is willing to donate for this purpose. In addition, Sabbia's design has been heavily influenced by social network studies [5], trying to emulate some charateristics of the social network which can help in optimizing the anonymous network. Usually, the number of hops is proportional to the security of the network, and thus also proportional to the loss of perfomance: Sabbia uses social networks concepts in order to minimize the length of the anonymous path to reach a specific destination.

The paper is organized as follows: in Section 2 we analyze related works, in particula TOR. We describe both the benefits that this architecture brings, and the questions left open in TOR design. In Section 3 we analyze end user and developer requirements, in order to design a protocol which is both easy to implement and to use. We analyze usability, required features, underlying design assumptions, security and considerations about the extensibility of our design. In Section 4 we describe the network architecture and show how it has been inspired by natural networks. In Section 5 we describe the architecture of the Sabbia protocol, providing multilayer anonymity and safe forwarding with optimized path discovery. In Section 6 we analyze known active and passive attacks, and show how the Sabbia protocol protects the user against them. Finally, in Section 7 we draw some conclusions and outline our future work on the subject.

2 Related Work and open questions

The concept of data anonimization through cryptography and forwarding was originally introduced by Chaum [6]. His work laid out the foundation of anonymous remailer technology, the most stable anonymization technoloy deployed on the Internet: they hide the relationship between sender and recipient by wrapping messages in multiple layers of public-key cryptography, and delivering them through a path of relays (mixes) which, in turn, decrypt, delay, and re-order messages before relaying them. After his original work, two newer generations of remailers ensued: Mixmaster [2] and later Mixminion [1], the latter being the state of the art technology in use nowadays.

The first pseudo-anonymous remailer (known by its hostname as "anon.penet.fi") simply stripped headers and user informations from e-mails, remailing them and waiting for an answer to send back to the user. In a similar way, the first low-latency anonymous network available was a single hop anonymizing proxy: Anonymizer. These systems shared the same defect: an attacker able to see incoming and outgoing traffic would be able to correlate requests with the sources. This is called a passive attack because the attacker just needs to observe the traffic, as opposed to active attacks, where the attacker needs to be able to mangle traffic inside the network.

As we said, e-mail anonymizers use multiple layers of public key encription to avoid traffing interception and the insertion of potentially untrusted mixes into the path, and introduce large, variable forwarding delays to avoid traffic correlation attacks. These techniques make high-latency system solidly safe. However, low-latency systems can not deploy such techniques, because they require a bidirectional, interactive exchange of data. They are consequently vulnerable to various types of attacks.

JAP, the Java Anon Proxy, which uses an encrypted chain forwarding hop and fake payload adding, is vulnerable to passive and active attacks exposed in [7]. The same paper finds out PipeNet [8] to be vulnerabile to a catastrophic Denial of Service attack. The authors propose then a modification of the forwarding techniques in order to twarth the attacks they detected.

Distributed-trust, circuit-based anonymizing systems are more complex. In these designs, a user establishes one or more bidirectional circuits, and tunnels data in fixed-size cells. Circuits are tipically established through public-key cryptography, whereas cells are transmitted using symmetric encryption. Because a circuit crosses several servers, and each server only knows the adjacent servers in the circuit, no single server can link a user to her communication partners.

Tarzan [9] and MorphMix [10] are peer-to-peer networks, where participants both generate and relay traffic for others, in order to hide who actually generated traffic as opposed to simply relayed it. This is a good solution, even if not perfect, however it requires peers to offer their bandwidth to others, and if not limited in some way, this could quickly saturate the link of a peer with unwanted traffic.

Cebolla [11] and Rennhard's Anonymity Network [12] use instead a layered "onion" of public-key encrypted messages, each layer of which provides session keys and the address of the next server in the circuit. But as it was shown in [13] both peer-to-peer and onion networks share the vulnerable concept of a "chain" of servers, and a passive attacks on an anonymous chain may reveal the original sender of an anonymous connection, even if it is rebuilt and reencryptet each time in random ways inside the network.

Low-latency anonymous networks have thus been shown to be vulnerable to passive and active attacks. Attacks of the former kind can be carried out by correlating the startpoint of the network and the target anonymous session. Traffic correlation could seem difficult to perform on compressed, splitted and padded traffic. However, correlation between bandwidth peaks and interarrival times between peaks would still be possible. For example an attacker could easily correlate timings and volumes of the traffic entering the anonymous network with those leaving it, whenever it can eavesdrop on both the ends of the communication [14].

On the other hand, active attacks work by simply disrupting packets and tracking which session gets delayed, or even by inserting specific patterns in an answer and trying to detect which anonymized session gets disrupted or altered, thus effectively backtracking the traffic to its source. Similarly, an adversary could inject timing patterns (or traffic with other discernible features) into the traffic entering the network and looks for corresponding patterns among exiting

traffic. This does not always require enormous power on the attacker side. For instance, on IIP (Invisibile IRC Protocol) an attacker can perform such an attack by simply using a client connected to the IRC network and communicating, with a fixed time/size pattern, with the anonymous user he wants to trace.

While solutions to these problems have been under development for some time, we notice that a large part of the current systems address better the first class of problems rather than the second.

The state of the art in low-latency network design is TOR [4]. While it solves a number of problems, it leaves four issues open:

- It lacks the added reliability of a peer-to-peer design, which protects against server instability
- It does not completely protect against end-to-end attacks based on traffic timing, size and pattern correlation
- It lacks protocol normalization, thus allowing an attacker to detect differences in protocol handling between various clients
- It lacks a steganographic approach, to hide any difference in the anonymous channel usage, wether or not an incoming or outgoing anonymous session is present

3 Sabbia: end goals and requirements analysis

While designing Sabbia, we assumed an adversary who is able to read and to write arbitrary traffic on each point of the Internet, who can install modified Sabbia nodes and offer them as forwarding peers, and memorize unlimited traffic with access to huge computational power to analyze it.

The main requirements we set for Sabbia are:

Deployability: the Sabbia protocol does not require administrative access on the host in order to be deployed

Stability of bandwidth requirement: Sabbia requires equires the user to set the maximum bidirectional bandwidth he desires to volunteer to the anonymous link (for connections both initiated or received by the user). Safeguards are built into the protocol to avoid that this amount of "offered" bandwidth is shaped or otherwise delayed. This limitation also avoids traffic peaks and bursts due to flood or abuse of bandwidth due to large downloads. Overall, a low-medium bandwidth offer (2kb/sec - 8kb/sec) is enough for low latency applications such as instant messaging, web browsing, email sending, while it avoids typical abuses such as the download of vast amounts of copyright protected material

Protocol normalization: HTTP, IRC, E-mail transfer and other application protocols contain information related with the client and its operating system, allowing fingerprinting. On "specialized" anonymous relays such as an anonymous remailer, sensitive information relative to the original sender can be stripped because the protocol is known a priori; on this type of network, we need an application layer dissector, able to find sensitive communication fields, and to change them with the endpoint fields.

Simulated Steganography: steganography [15] usually is meant as concealing information into other information; in Sabbia, we use a concept we call "simulated steganography", by which we mean we force the network to present always the same communication pattern, thus avoiding observers to detect the presence or absence of anonymous traffic

Secure against end-to-end attack: Sabbia is designed to be resistant against correlation even if an attacker creates a number of malicious nodes inside the network, since its security is not based on a simple forwarding chain

Secure against passive attacks: Sabbia is studied to be proof against a correlation attack brought by the all-powerful attacker we assumed, because all peers produce continuously the same traffic patterns, eluding most passive attacks

Secure against active attacks: Sabbia is also studied to be secure under active attacks brought by the same attacker; an emulation engine at the application layer allows to automatically manage sessions in case the anonymous peer is delayed or otherwise disturbed by the attacker in order to cause an interruption which can be correlated

Exit policy and traffic selection for endpoint user: the user can set an "exit policy" for traffic outgoing from its node, which is trasmitted along with the key and the network information to the other peers, in order to allow them to route fastly traffic to a peer which is near to the connection destination and which allows the required service in outgoing traffic

Best of both worlds: Sabbia tries to achieve the best features of both chain-based networks and peer-to-peer networks; the former are strongly secure against infiltration of malicious peers, while the latter supports dynamic, short-lived peers in the network. The drawback is that in a peer-to-peer structure an adversary could be running a large part of the peers for analysis and tracking pourposes. On the other hand, a safe and long chain causes a great loss of performance. Sabbia mixes a trusted forwarder chain with random peers, in order to keep the chain short but still safe, and introducing a trusting mechanism between peers via public key signatures

4 Sabbia network structure

4.1 Trust mechanism

Sabbia bases its network and its trust mechanisms on concepts drawn from the study of natural networks [5]. Figure 1 shows how a generic network of Sabbia nodes could be interconnected.

Sabbia does not require a strong trust mechanism in order to work, but being able to reach a trusted entrypoint increses performance and security. The trust mechanism we chose is a simple mechanism based on key signing, in a very similar way to the PGP/GPG "web-of-trust" concept for certifying keys [16].

Sabbia uses the PGP keyservers to store the node keys, which enable clients to upload public keys, add signatures to keys and retrieve the signature list. The owner of the node can, if he wants to, sign with his own key the node public

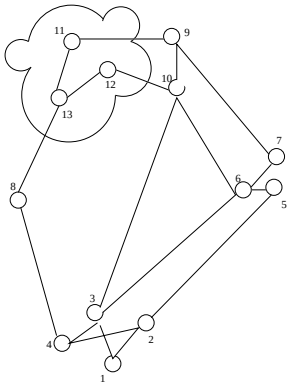

Fig. 1. A Typical Sabbia Network

key, thus linking the trust network of Sabbia with the personal trust network of PGP. All the keys share a common keyword, thus making search simple and enabling the client to download all of them in a bulk (hiding which is the key we are really looking for). They also present a unique ID not directly linked to the user who generated them.

Once the keys have been obtained, it is straightforward to build a trust network, where the "distance" between nodes (the number of trusted introductions needed to recognize a key) is an inverse index of trustability (i.e. a node distant 2 is inherently less trustworthy than a node distant 1, which has been directly recognized by the user). The user himself can fixate how much he will trust other nodes, depending on this distance. This process can become totally automated and transparent, or the user can choose to decide case by case wether or not he trusts a particular key.

4.2 Small world effect

The anonymous network created by Sabbia is not uniformly spread over the network. Local or national interest groups on privacy technology diffusion, close communities of friends interested in this technology, and collaborative peers in privacy-sensitive work environments cause anonymous network to show a "small world" effect, which makes a totally random choice of the chain of forwarders suboptimal for performance purposes. However, since none of these effects is linked to network access choices, it is likely that the distribution of such nodes between different autonomous systems and ISPs will be more or less random, which means that the larger an ISP is, the most likely it is to find a trusted node already operating inside its domain.

Sabbia tries to leverage the small world effect, connecting nearby pears together through a central point called Central Peer (CP). Usually, this will be the peer with the best bandwidth offer and uptime, and can be auto-elected or chosen by others as a trusted peer. Endpoint peers (EP) therefore can connect

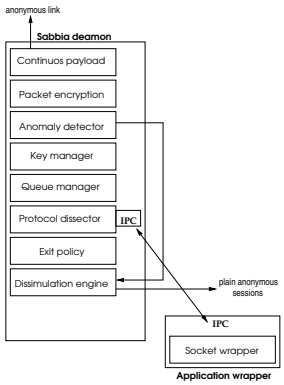

Fig. 2. Design of Sabbia: Daemon Layers and Application Wrapper

to the network using a stable, trusted peer which they also have a good connection to. Each CP stores and forwards to other CPs informations on the EPs connected to it (network information, public key and exit policy), hiding their IP addresses and assigning them an anonymous ID for reference. Each node can become a CP, and in fact a node could be contemporaneously in both roles for different neighborhoods. Except for network organization purposes, CP and EP nodes can perform the same operations.

Routing operations are also mediated by CPs. In order to route efficiently, besides the information on trust transmitted through key exchange nodes must take into account throughput and latency values. It is plain that in a virtual network such as Sabbia a multiple-hop path could be much more convenient than a single-hop, high-latency one, providing a better latency-anonimity trade-off. Thus, CPs must also store and forward transmission statistics (bandwidth, latency, packet loss) of nodes connected besides them.

5 Application Architecture

Sabbia has a multilayer application architecture, as shown in Figure 5. Basically, the daemon is a software component which must always run while connected to the Sabbia network, and manages user anonymization and handles sessions from anonymous users received while working as a relay. The application wrapper component instead is activated whenever the user wants to anonymize a session.

5.1 The Application Wrapper

The Sabbia Application Wrapper (SAW) wraps all the system calls related to network communications (which is easy to implement and port in most common operating systems, through the concept of shared libraries and redirection). In

this way, it makes transparent for the applications the use of the Sabbia anonymous network, while it forwards to the daemon any connection request and/or application data, and passes back to the application any answer.

This also solves a simple but common problem, the fact that usually anonymous network do not conceal DNS requests. Wrapping the name resolution requests to the operating system makes SAW able to use the anonymous network for name resolution, hiding the anonymous session better.

Wrapping sessions directly is important for the Sabbia protocol, which thus interacts poorly with other technologies that already incapsulate sessions (for instance, VPN tunnels) under both the performance and the security profile. Performance decreases because re-encapsulating an encapsulated TCP connection causes an higher packet loss and possibly fragmentation due to MSS restriction. Security is impacted because Sabbia cannot anonymize the upper layer protocols in the encrypted link.

5.2 The Sabbia Daemon

The Sabbia deamon (SaD) is the core application which handles incoming and outgoing anonymous sessions. As shown in Figure 5 the daemon is structured in a multilayer fashion and sits between applications (requesting network communication through the wrapper), and the operating system, which will ultimately send data over the network.

Let us consider the various components of SaD, beginning from the one closer to the network stack: the **Continuous Payload Generator**. As we said, we want that any peer always generates the same amount of traffic ("simulated steganography"). Once the bandwidth has been fixated by the user, it can be divided into equally sized, equally spaced packets; e.g if the bandwidth is 4 kb/sec, SaD can send three packets each second, each with 1365 bytes of data. This will constantly happen, bidirectionally, no matter what happens on the anonymous network. If the link is unused, packets contain just padding. If the link is used, part of the padding is substituted with encrypted data. There is no way, for an external observer, to infer the switch from one state to the other, no signalling happens between the peers.

We use UDP for this fixed-size transmissions. Inside the payload, a Sabbia Header (SH) is present, containing the following transmission control fields:

Random Value: a nonce used for freshness in encryption, due to the use of an ECB mode as opposed to classical CBC mode (see below)

Checksum: an encrypted parity check, used to understand if the packet has been tampered with

Incremental Counter: each packet is numbered to avoid replications and to detect packet loss

Timestamp: timing the RTT between hops is needed both for routing purposes and for intrusion detection purposes

Message length: used to discriminate between real data and padding. If 0, the packet is fake

Since timing is so crucial, the safe-side decision is that when a packet cannot be transmitted or encrypted safely at the precise time, SaD opts to send a meaningless padding packet instead; also, precise safeguards are adopted to avoid the packet sending to be delayed due to interactions with other software components on the system.

Encryption is performed by the **Packet Encryption Layer**. Data compression is applied before encrypting, and the Sabbia Header is added. The random nonce and the timestamp add entropy to the plaintext, which makes it safe to use an ECB-mode algorithm. ECB is more resilient to packet loss, but requires freshness in order to avoid well-known attacks (since identical plaintexts encrypt to identical cyphertexts). We use the AES algorithm [17]. Obviously, the arbitrary division of the bandwidth in fixed payload sizes has to be adjusted in order to be divisible with the encryption block size of AES.

A shared session key is exchanged between the peers using public key encryption, and is changed after a random number of packets, chosen by the peers. In order to explain how the key exchange system works, let us consider how an EP would encrypt a connection. Obviously, to enter the Sabbia network, it has connected with a CP and has established a bidirectional link with it, exchanging a shared encryption key using the CP public key. The protocol, however, behaves differently depending on the level of trust EP has in CP. If EP trusts CP, it can send packets encrypted with the shared key to CP, and CP will receive the packet, decrypt it and retrieve the real length of the message. If the lenght is 0, the packet contains only random data and can be discarded, else the packet contains some data and it is thus passed on for processing or forwarding.

If EP does not trust CP, however, we must take additional steps to protect the communication. We need to find a trusted remote peer among the ones connected to CP, let us call this trusted peer EP2. Obviously, we do not directly know its address, but we know the ID which CP has assigned to it, and we have routing information to choose a path (in the case we are collected to multiple CPs announcing a route to that host). What we need to do, henceforth, is to exchange a shared key with EP2, on the anonymous link relayed by CP. We send the key exchange packet on the link, encrypting it with the key we share with CP. The CP decrypts the external layer, but it only learns who is the intended recipient and nothing more. It will then relay the key exchange session to the recipient. Once a shared key has been negotiated between EP and EP2, CP will act as a simple message passer, since it cannot decrypt packets anymore. This has the drawback that CP is also forced to forward random padding packets, which could saturate its links and activate the policy manager, making our session lose priority.

The **Intrusion Detection Layer** makes use of the timestamps in order to verify routing problems, congestions and possible tampering on the network. This helps prevent active attacks where an adversary tries to slow down the link between two anonymous peer for correlating and backtracing the sessions being anonymized on that link. This is not just impossible to avoid, but also really difficult to detect. But Sabbia, thanks to this layer, can detect these malfunc-

tionings. It then tries to understand wether they are natural or induced by an attacker. If an attack is detected, the "dissimulation engine" is invoked and the user is notified that someone is tampering with his link.

The **Key Manager** is a utility layer which supports key selection, parsing and signature checks in order to manage received public keys. It builds the trust network and manages key exchange, rotation and recomputation, handing out to the Packet Encryption Layer all the information they require in order to perform cryptography operations.

The **Queue Manager** performs the strong traffic shaping required by Sabbia in order to preserve the exact timing of packet sending. It collects traffic incoming from the anonymous link which must be forwarded (either encrypted or in plain text, depending on wether the remote host trusted us or not), and outbound data coming from the local machine. The Queue Manager assign priorities to the traffic trying to optimize performance. A simple policy is to give top priority to highly interactive protocols like SSH or Telnet, and to penalize Web browsing and other bulk transfer protocols. On the top of this we can grant higher priority to sessions initiated by the user as opposed to forwarded sessions, and further penalize continuous encrypted sessions resulting from an EP not trusting the CP.

The **Protocol Dissector** is an intermediate layer which receives the incoming sessions, and acts as an application layer proxy, i.e. it receives commands and transmits them, and then relays back the answers. This is useful because in this way the exit point can relay the commands to the final destination and then send back the answers without caring about the application protocol, atomicity is preserved and performance on the anonymous link is optimized. Also, by caching the transactions for a fixed amount of time the peers can retransmit data lost on the anonymous link without making a new request to the external server. In addition, Protocol Dissector strips fingerprinting information, e.g. Mail Tranport Agent headers, HTTP User Agent, and such, substituting them with anonymous alternatives. Since the Protocol Dissector must be easily extendible, it has been implemented as a series of external plug-in modules which can be loaded on request.

The **Exit Policy Checker** verifies the constraints set by the user on the allowable outgoing protocols from its node. In this way a user can avoid to become the unwilling relay of attacks, or to allow unlawful or morally debatable content flowing through his system.

The **Dissimulation Engine** is a defensive segment which can be evoked by the Intrusion Detection Layer. The engine tries to emulate the presence of a user behind an anonymous session which is being maliciously slowed down or blocked, in order to avoid active attacks which try to correlate the link disruption with an interruption in the anonymous session.

In order to work, the Engine needs access to application layer data, thus it can work only over protocols that are supported by the Protocol Dissector. The actual evasive action depends heavily on the application: for instance, let us consider an IRC session. While it is perfectly normal for a user to be idle, it

is anomalous for him not to answer PING requests from the server. Thus, the Dissimulation Engine will in this case answer to PING on behalf of the client. An HTTP session could be emulated requesting some random link using the last URL as the HTTP Referral. These dissimulative actions could be refined further, but they are in fact enough to make an active attack not automatically executable. It would still be difficult to fool an human observer.

6 Sabbia against active and passive attacks

In this section we recapitulate how Sabbia protects the user agains known attacks against low latency anonymous networks. The first layer of protection is provided by the joint use of the Application Wrapper and the Protocol Dissector: this allows to get rid of application-dependent fields which could allow an attacker to use passive identification. Since Sabbia avoids to use the OS TCP/IP stack during encapsulation, this side channel is also avoided.

The Continuous Padding Generator, along with the Packet Encryption Layer, protects Sabbia against passive correlation attacks, making an attacker unable to distinguish between the real presence of traffic. In addition, only "trusted" hops (verified with sound public-key signatures) can see the real passage of traffic, the others will always see the same apparently random traffic. The presence of the nonce, sequence number and timestamp avoids active replay attacks from inside or outside the network. The encrypted checksum protects against active bit-flipping attacks. Freshness grants that traditional attacks against ECB mode cannot work.

The Intrusion Detection Layer can detect active attacks such as the two outlined above, and also those based on traffic shaping and blocking, since the exact timing of packet transmission causes the detection of any network malfunctioning, whether incidental or hostile. The Dissimulation Engine can then take over and provide an emulation of user activity over a disrupted link. Obviously, we still cannot prevent an adversary running a Sabbia peer to maliciously disrupt communications inside the network, but we can avoid him to use this disruption to correlate anonymous sessions.

7 Conclusions

This paper offers an overview of the design of Sabbia. Sabbia is intended as a proof-of-concept implementation of a set of guidelines and improvements over state-of-the-art protocol, rather then a full fledge production protocol. We believe that these guidelines define a winning approach to the problem of low-latency anonymous networks. The Sabbia network aims to develop trust-based relationships and optimized paths. The security of the Sabbia protocol against passive and active attacks has been discussed. So far, Sabbia turned out to resist to all of the attacks we outlined in this paper.

Future extensions of this work will analyze the prototype in order to optimize the performance of networks based on continuous padding and to study how

to optimally determine size, frequency and randomic variation of their padding packets. Further optimizations to the bandwith usage could be achieved by diminishing the usage of random traffic, transmitting useful data along, or using a randomly variable bandwidth instead of a fixed bandwidth. A possible future extension could allow for an Intrusion Detection System to be plugged into the Exit Policy Checker, in order to allow the user more granularity in defining unacceptable requests and attacks to be dropped.

References

1. George Danezis, Roger Dingledine, and Nick Mathewson. Mixminion: Design of a type III anonymous remailer protocol. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, May 2003.
2. Ulf Möller, Lance Cottrell, Peter Palfrader, and Len Sassaman. Mixmaster Protocol — Version 2. Draft, July 2003.
3. Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: a distributed anonymous information storage and retrieval system. In *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, pages 46–66, July 2000.
4. Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, August 2004.
5. M. Buchanan. *Small World: Uncovering Nature s hidden net- works*. Weidenfeld & Micolson, London, 2002.
6. David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 4(2), February 1981.
7. Adam Back, Ulf Möller, and Anton Stiglic. Traffic analysis attacks and trade-offs in anonymity providing systems. In Ira S. Moskowitz, editor, *Proceedings of Information Hiding Workshop (IH 2001)*, pages 245–257. Springer-Verlag, LNCS 2137, April 2001.
8. Wei Dai. Pipenet 1.1. Usenet post, August 1996.
9. Michael J. Freedman and Robert Morris. Tarzan: A peer-to-peer anonymizing network layer. In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS 2002)*, Washington, DC, November 2002.
10. Marc Rennhard and Bernhard Plattner. Practical anonymity for the masses with morphmix. In Ari Juels, editor, *Proceedings of Financial Cryptography (FC '04)*. Springer-Verlag, LNCS 3110, February 2004.
11. Zach Brown. Cebolla: Pragmatic IP Anonymity. In *Proceedings of the 2002 Ottawa Linux Symposium*, June 2002.
12. M. Rennhard, S. Rafaeli, L. Mathy, B. Plattner, and D. Hutchison. Analysis of an anonymity network for web browsing. In *IEEE 7th Intl. Workshop on Enterprise Security (WET ICE 2002)*, June 2002.
13. Andrei Serjantov, Roger Dingledine, and Paul Syverson. From a trickle to a flood: Active attacks on several mix types. In Fabien Petitcolas, editor, *Proceedings of Information Hiding Workshop (IH 2002)*. Springer-Verlag, LNCS 2578, October 2002.
14. A. Serjantov and P. Sewell. Passive attack analysis for connection-based anonymity systems. In *Computer Security - ESORICS 2003, LNCS 2808*. Springer-Verlag, October 2003.

15. Fabien A. P. Petitcolas, Ross J. Anderson, and Markus G. Kuhn. Information hiding — A survey. *Proceedings of the IEEE*, 87(7):1062–1078, 1999.
16. S. Garfinkel. *PGP: Pretty Good Privacy.* O'Reilly & Associates, Inc., 1995.
17. William Stallings. The advanced encryption standard. *Cryptologia*, XXVI(3):165–188, 2002.