An Architecture for an Anonymity Network

Marc Rennhard*, Sandro Rafaeli†, Laurent Mathy†, Bernhard Plattner* and David Hutchison†
* Swiss Federal Institute of Technology, Computer Engineering and Networks Laboratory; Zurich, Switzerland
† Lancaster University, Faculty of Applied Sciences; Lancaster, UK
{rennhard,plattner}@tik.ee.ethz.ch, {rafaeli,laurent,dh}@comp.lancs.ac.uk

Abstract

It is difficult to design a system that provides anonymity for delay-sensitive services such as Web browsing. Existing systems are either not resistant against sophisticated attacks or they achieve their level of anonymity at the cost of a high bandwidth overhead. In addition, these systems do not meet all of our requirements. In this paper, we present the architecture of our prototype implementation of an anonymity network. Our system is trustworthy, fair, stable, modular, and bases on the well studied and accepted secure sockets layer protocol. With this system, we want to derive quantitative results about the tradeoff between anonymity and performance penalty. We will then extend our anonymity network such that it provides high resistance against various attacks while minimizing its bandwidth overhead.

Keywords: *anonymity, pseudonymity, anonymous Web browsing, mix-networks*

1. Introduction

During the past years, people became more sensitive regarding privacy issues in the Internet. They realized that they leave all kinds of traces when surfing the Web or exchanging e-mails. Encryption provides the means to protect the privacy with respect to third parties in the sense that eavesdropping becomes very difficult, and the popularity of tools such as Pretty Good Privacy (PGP) [14] underlines its need. However, there are situations where the protection of privacy should go even further, and that is when anonymity comes into play. The requirement for anonymity in the Internet is justifiable very much since many situations in the real life are anonymous, e.g. traditional stores offer a certain degree of anonymity for their customers if they pay with cash. Similarly, a spontaneous conversation between two persons is anonymous.

Using the Internet is not anonymous at all. The protocols do nothing to hide the path the data takes through the network, and the packets themselves contain information such as IP-addresses identifying the endpoints of a communication. Bringing anonymity to the Internet would enable several interesting applications, such as anonymous Web browsing or pseudonymous e-commerce [8]. However, all of the higher-level anonymous or pseudonymous applications require hiding the path the data takes from the sender to the destination and vice versa. If it is easy for a shop operator, the administrator of a Web server, or an eavesdropper to derive this end-to-end connection, all additional effort to provide anonymity is pointless.

Our goal is to design and implement an *anonymity network* that can serve as the basis for various anonymous or pseudonymous Internet activities. This anonymity network should fulfill the following requirements:

- It should provide *reasonable resistance against attacks* (section 3). This should be achieved with minimal overhead and without the necessity that the users have to carry out a synchronized protocol in any way.
- The whole system should be *trustworthy for its users*. Since no centralized service can be trustworthy enough to provide anonymity, the system must base on a distributed set of components operated by different providers. The user should be able to use any number of these components and the ones she trusts. In addition, she should be sure to be using indeed the chosen components.
- The system should be based on a *studied and well accepted security protocol*. We do not want to sacrifice anonymity by using vulnerable security protocols.
- The system must be *fair*; no user should be able to block other users by generating lots of traffic.
- The implementation should be *modular* enough to allow for adding and testing new variations of the system rapidly.
- The system has to be *stable*. If a component in the system goes down, then only the connections of users that use that component should be torn down. Similarly, it should be possible to easily integrate components into the system, independent whether they are newly added or just restarted.

This paper is organized as follows: in section 2, we describe the most important techniques for anonymizing network traffic, section 3 describes attacks an anonymity network should protect from. In section 4, we give the basic

0-7695-1269-0/01 /$10.00 © 2001 IEEE

idea of our system, whereas section 5 explains its implementation in more detail. Sections 6 and 7 describe the message format and the protocol used in the anonymity network. We talk briefly about related work in section 8 and conclude our work with an outlook in section 9.

2 Techniques for Anonymizing Data Traffic

We distinguish between sender anonymity, where the sender's identity should remain anonymous, and receiver anonymity, where the receiver's identity should be hidden. Although there are applications for receiver anonymity, i.e. anonymous Web publishing [12], most Internet activities where anonymity is desired require sender anonymity.

Achieving anonymity in a network is very difficult. Although it usually makes use of encryption techniques, it is a fundamentally different problem than achieving data confidentiality. When speaking about encryption, we care only about protecting the data. In contrast, anonymity means protecting the communication itself in the sense that it should not be possible for an attacker to follow the path data packets take through the network. Since it is not possible to get rid of the data itself, we have to try to confuse attackers such that they cannot trace the data from the source to the destination.

Chaum described the concept of a Mix network [3]. It is based on the idea that traffic sent from sender to destination should pass one or more Mixes. A Mix relays data from different end-to-end connections, and the Mixes' task is to reorder and re-encrypt the data such that incoming and outgoing data cannot be related. This heavily complicates traffic analysis and should thwart attempts of an attacker to follow an end-to-end connection.

Another fundamental approach to provide anonymity is based on Chaum's solution to the Dining Cryptographers (DC) problem [4]: DC networks. The main drawback of this approach is that it requires the participants to carry out a synchronized protocol. In this paper, we will follow the idea of a Mix network.

3 Threat Model

In this section, we define a threat model against which the system should be resistant. The following list shows possible attacks against an anonymity network. The order in which they are listed represents the difficulty to prevent them, in ascending order.

1. **Message coding attack:** In this attack, messages that do not change their coding can by traced through the network by pattern matching.

2. **Message length attack:** This attack examines the length of a message as it travels through the network.

3. **Replay attack:** An attacker replays data packets and waits that the Mix processes the same packet repeatedly, therefore enabling the attacker to correlate incoming and outgoing packets.

4. **Collusion attack:** This happens if a certain number of involved parties collude to break the anonymity of connections.

5. **Flooding attack:** Anonymity is usually achieved with respect to a certain group. In this attack, an adversary floods the system to separate certain messages from the group.

6. **Message volume attack:** In this attack, it is tried to detect an end-to-end connection by observing the message volume at the endpoints.

7. **Timing attack:** A timing attack tries to observe the duration of a connection by correlating its establishment or release at the possible endpoints.

8. **Profiling attack:** A profiling attack tracks users over long-term periods. It is basically a combination of the timing and message volume attack over a long time.

It is well known how to prevent the first two attacks: by re-encrypting each message between a pair of Mixes and by making all messages in the anonymity network to have the same length. The replay attack can be prevented if each Mix maintains a database of the packets it has already processed. Incoming packets are compared with the entries in the database and replayed packets are simply discarded. A Mix-network also has the property that if less than all of the involved Mixes of an end-to-end connection collude, then this coalition cannot break the connection anonymity. A flooding attack can be prevented if all users of a Mix are treated fairly in the sense that no user can block others.

It gets more complicated when trying to prevent attacks 6–8. All of them are basically traffic analysis attacks in the network-wide scale. The main problem is that the delay that can be added to messages by the Mixes can only be relatively small for delay-sensitive applications such as Web browsing, and a certain correlation of the messages at the endpoints seems unavoidable. This is where dummy-messages can be added to fool an attacker. If a user gives the impression to be sending and receiving data all the time although only a small amount of this traffic is useful data, then it becomes more difficult to correlate events at the endpoints of a connection.

A system to anonymize telephony [7] is based on the idea that the subscribers connected to the same end-office build a group, and the system provides anonymity within that group. The approach makes heavily use of the synchronized telephony system in the sense that all subscribers are always sending data to the end-office so that real phone calls cannot be distinguished from the dummy data. Another project [1] tries to transfer this idea to the Internet, where the subscribers of an Internet Service Provider (ISP) form the group. However, it is questionable how well this can work in an asynchronous environment where users are connected with different data rates. In addition, this scheme gives anonymity only among the users of the same ISP, and not among all users of the anonymity network.

In general, one can say that the more dummy traffic is used and the more synchronized the system is, the better prevention against attacks 6–8 can be achieved. There is a tradeoff between the overhead and the level of anonymity a system offers, and we argue that this overhead should be kept minimal to achieve a reasonable level of anonymity. It is doubtful that perfect anonymity can be achieved. A very powerful attacker that is able to observe and analyze the traffic at several places in the network over a long time is probably able to derive information about end-to-end connections. Only a highly synchronized network where users are connected all the time and generate vast amounts of dummy traffic might be able to prevent such a powerful attack. On the other hand, it is questionable if such an attack could be carried out by any organization.

With our system, we want to study how well the attacks described above can be prevented by adding the least possible amount of overhead. We also require that users are able to come and go whenever they like, which means that they are not required to be connected to the anonymity network all the time. In addition, no synchronization among the users should be needed, since this interferes with the different speeds the users connect to the Internet. Furthermore, at any given time, all current users of the anonymity network should build the group within each user is anonymous.

4 The Anonymity Network Model

In this section, we describe the basic design of the anonymity network using the Web browsing example illustrated in figure 1.

Alice sits at her computer, having two browser windows (B_1 and B_2) that issue requests to the Web servers (WS_1 and WS_2, respectively). There is a *local proxy* (LP) running on Alice's computer which gives her access to the anonymity network. The LP is used by the Web browser in exactly the same way as a normal Web proxy. One task of the LP is to sanitize the data sent by the application and remove information that could identify Alice. The core of the network are the *anonymity proxies* (APs). Their main task is to hide the path messages take from the LP through the anonymity network by providing the functionality of Mixes.

When Alice wants to browse anonymously, she starts her LP. The LP establishes an *anonymous tunnel* through the anonymity network, using one or more intermediate APs. The proxies to be used are specified by Alice and the specification is stored in a local configuration file. In figure 1, the anonymous tunnel is established from LP via AP_1, AP_2, AP_4, and AP_6. LP is the entry and AP_6 is the exit of Alice's anonymous tunnel. Alice can now use the tunnel to set up *anonymous connections* (ACs). For each of her end-to-end connections from the browser to a Web server, an AC is established. All her ACs are transported within the anonymous tunnel between LP and AP_6.

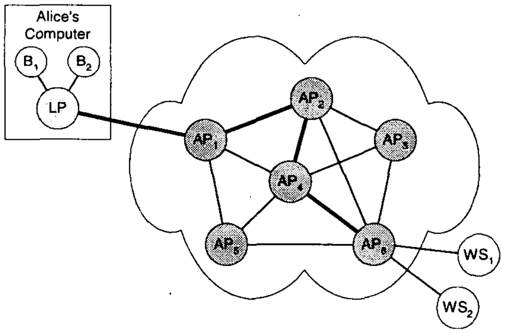

Figure 1. Anonymity Network Overview.

For the Web servers, the requests seem to be coming from AP_6, but they do not know that it is Alice who initiated them. Note also that there is only a single connection between each pair of APs, and this connection is encrypted. The anonymous tunnels active in the anonymity network are not visible for an outsider, because the data are transported within those unique pairwise connections.

5 Implementation

In this section, we describe some implementation issues. There is a technical report available that describes the system in more detail [10]. The first question to answer is how many encryptions are required to protect the data against external eavesdropper and colluding APs. We use an example where an LP has established an anonymous tunnel via three APs. Figure 2 illustrates the required encryptions to protect the tunnel.

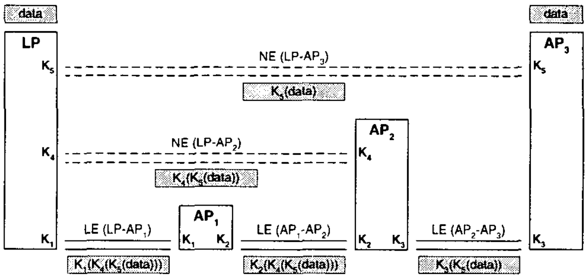

Figure 2. Layers of Encryption.

The data is sent from LP via AP_1, AP_2, and AP_3. Since we want to avoid that external attackers can correlate incoming and outgoing data at each AP, we encrypt the data differently on each link between a pair of proxies. We name these encryptions *link encryptions* (LEs). Before LP sends data to AP_1, it encrypts it with a key K_1 known only to LP and AP_1. When AP_1 receives the data, it decrypts it with K_1 and encrypts it with K_2, known only to AP_1 and AP_2 and so on. As a result, it is not possible for an external attacker to correlate incoming and outgoing data at an AP.

However, this is not enough to protect the anonymous tunnel from internal attackers. If AP_1 and AP_3 collude, then they know the end-to-end connection since after removing the link encryption, they see the same data. To protect the path from internal attackers, we have to add *nested encryptions* (NEs), as illustrated in figure 2. In addition to the link encryption LE_{LP-AP_1} to AP_1, LP has two nested encryptions, NE_{LP-AP_2} to AP_2, and NE_{LP-AP_3} to AP_3. Note that although these NEs allow LP to exchange encrypted data with AP_2 or AP_3, the data is never directly sent to the peer, but is always relayed by AP_1 (and AP_2). This works as follows: before LP sends data over the encrypted link to AP_1, it first encrypts the data with the key K_5 it shares with AP_3, and then with the key K_4 it shares with AP_2. When the data travels through the anonymity network, the APs remove the nested encryptions they are involved in. When AP_2 receives data from AP_1, it first removes $LE_{AP_1-AP_2}$ with K_2, and then NE_{LP-AP_2} with K_4. Then it encrypts the data with K_3 and sends it to AP_3. On the way back, it works similar, but this time AP_2 encrypts the data with K_4 after removing $LE_{AP_2-AP_3}$ with K_3. Then the data is encrypted with K_2 and sent to AP_1. Note that since we have already a link encryption between LP and AP_1, no nested encryption between LP and the first AP is needed.

The anonymous tunnel is now secure against internal attackers. Assume again that AP_1 and AP_3 collude. After removing LE_{LP-AP_1}, AP_1 knows $K_4(K_5(data))$. AP_3 knows $K_5(data)$ after removing $LE_{AP_1-AP_2}$ and $data$ after removing NE_{LP-AP_2}. However, they have no idea if $K_4(K_5(data))$ and $K_5(data)$ are the same data since they do not know K_4. In general, no coalition of less than all of the involved APs in an anonymous tunnel can break that tunnel, since this coalition knows only a subset of the keys that are used for the nested encryptions.

The number of encryptions needed in an anonymous tunnel can easily be computed: if n is the number of APs involved, then we need n link encryptions and $n-1$ nested encryptions, which results in $2n-1$ encryptions. However, it should be noted that the link encryption between a pair of APs are shared by all anonymous tunnels that use that link. Therefore, there is only one unique link encryption per anonymous tunnel (from LP to the first AP), which results in n unique encryptions per anonymous tunnel.

In the requirements, we mentioned that the user should be able to authenticate an AP. Therefore, we integrate authentication with X.509 certificates [6] into the system. There is a certificate authority (CA) that issues certificates for the anonymity proxies. If an AP wants to offer its service, it has to request a certificate from the CA. The CA can be the root of a small public key infrastructure (PKI), but it can also be integrated in a larger scale PKI.

The next question is what protocol to use to implement the LEs and NEs, and how to do the key-management. We decided to use the secure sockets layer (SSL) protocol [5], since it allows for integration of the certificates, it is widely accepted, and it seems to be secure [11]. However, SSL has other advantages that we can make use of. One thing that serves us well is that SSL has its own sequence numbers, which means that replayed messages are recognized and discarded. This prevents attack 3 we described in section 3. Since replayed packets are discarded by SSL, we do not have to check for duplicate packets after the data has been processed by SSL. SSL also handles the key-management: when an SSL-connection is set up, a key-exchange protocol is carried out and a session key is established. This is true for the LEs and the NEs. Note that the SSL-connection setup for a NE is relayed by the intermediate APs, which means that the key is established between the LP and an AP without that AP being able to learn anything about the LP.

The keys are valid as long as the corresponding SSL-connection is not torn down. When a user decides to tear down her anonymous tunnel, she simply terminates the LP. The LE to the first AP and all NEs are torn down and the allocated objects and data structures in the APs are removed. When the user restarts her LP later (possibly using different APs), the necessary LEs and NEs are set up again and fresh keys are generated when establishing the SSL-connections.

6 Message Format

The main complexity of the anonymity network stems from the fact that multiple anonymous tunnels (established from different local proxies) are multiplexed on a single link between a pair of proxies. To do so, we use an identifier that is unique for each anonymous tunnel on a link. This means that we must introduce a message format, since we have to associate the data with an *id* identifying the anonymous tunnel. The message format is depicted in figure 3.

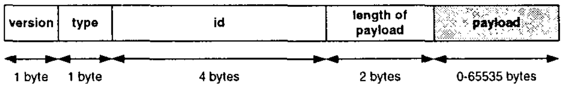

Figure 3. Message Format.

The *version* and *type* fields are defined as unsigned 8-bit integers. The current version is 1.0, and the various types are needed to interpret the messages correctly (section 7). The *id* is an unsigned 32-bit integer and is needed to multiplex multiple data streams on a link. The unsigned 16-bit integer field *length of payload* identifies the number of bytes in the *payload* field. The payload can have any length from 0 to 65535 bytes. Note that the message header only introduces an overhead of 8 bytes. In addition, it is worth mentioning that messages exchanged between objects that do not perform multiplexing do not need the *id* field, reducing the header to 4 bytes.

7 Protocol

In this section, we show how anonymous tunnels and anonymous connections are set up. We assume that a user chooses to use an anonymous tunnel from her local proxy via two anonymity proxies, AP_1 and AP_2. The user then makes a HTTP-request from her browser (B) to a Web server (WS). Figure 4 illustrates which messages are sent in what sequence to serve the user's request.

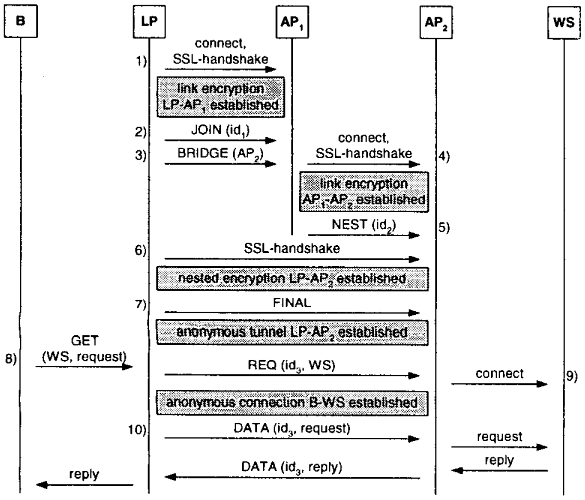

Figure 4. Data Flow.

1. When LP is started, it reads the list of APs to use from its local configuration file and connects to AP_1. They perform an SSL-handshake, which results in the link encryption LE_{LP-AP_1}.
2. Over LE_{LP-AP_1}, the LP sends a *JOIN*-message which carries the id id_1 to be used to multiplex the anonymous tunnel on the link LP–AP_1. The *JOIN*-message tells AP_1 that it is the first AP in the chain. AP_1 creates the required objects and data structures.
3. From the configuration file, the LP gets the second AP to use and sends a *BRIDGE*-message to AP_1, telling it to use AP_2 as the next AP in the anonymous tunnel.
4. AP_1 connects to AP_2, performs the SSL-handshake, and the link encryption $LE_{AP_1-AP_2}$ is established.
5. AP_1 sends a *NEST*-message to AP_2 which carries the id id_2 to be used to multiplex the anonymous tunnel on the link AP_1–AP_2. The *NEST*-message tells AP_2 that it is part of the anonymous tunnel, but not as the first AP. This also instructs AP_2 to get ready for an SSL-handshake to establish the nested encryption. For AP_1, the setup is now complete. Whenever it gets a message with id_1 arriving on the link to LP, it forwards it as a message with id_2 to AP_2 and vice versa.
6. LP performs an SSL-handshake with AP_2, and the nested encryption NE_{LP-AP_2} is established. Steps 3–6 are repeated for each additional AP that is used in the anonymous tunnel.
7. Since AP_2 is the last AP in the chain, LP sends a *FINAL*-message to AP_2, and the anonymous tunnel LP–AP_2 is complete.
8. When the browser issues a HTTP-request, the LP sends a *REQ*-message through the anonymity tunnel to AP_2. The message carries the server to contact and is marked with id_3. This id is needed to multiplex anonymous connections within the anonymous tunnel.
9. AP_2 connects to WS, and the anonymous connection B–WS is established.
10. LP sends the HTTP-request to AP_2. This request is sent as a *DATA*-message carrying id_3 to identify the anonymous connection. AP_2 gets the request, relays it to WS, and gets the reply. The reply is sent back to LP as a *DATA*-message, carrying again id_3 to identify the anonymous connection. LP gets the reply and relays it to the browser.

Note that step 4 to establish the link encryption between APs is only needed if the SSL-connection is not established yet. Since these connections are longstanding, their establishment is usually only required after an AP is started. Similarly, once the anonymous tunnel is established, this tunnel is used by the user for all her anonymous connections until the tunnel is torn down.

It is important to realize that there are two levels of multiplexing: one is when anonymous connections are multiplexed within an anonymous tunnel, and the other is when different anonymous tunnels are multiplexed on the unique link between a pair of proxies. This means that when the LP sends a *DATA*-message with id_3 (identifying the anonymous connection) to AP_2 in step 10, this message is put into another *DATA*-message with id_1 (to identify the anonymous tunnel on the link LP–AP_1) and sent to AP_1. AP_1 gets the data, changes id_1 to id_2 (to identify the anonymous tunnel on the link AP_1–AP_2), and sends the message to AP_2.

Although we demonstrated the functionality of the anonymity network using the Web browsing example, it is not limited to a particular application. It is the LP's task to interpret the data from a user correctly, since the APs are not aware of the higher-level protocol data they carry in their anonymous tunnels and connections.

Our prototype is implemented in Java 1.3 and currently provides support for HTTP and HTTPS. Using the Just-In-

Time (JIT) compiler option, the system provides adequate performance to experiment with and to find out how to make it as resistant as possible against attacks. It is now in a state where it runs very stable and is resistant against failures of nodes. The system also makes sure that each anonymous tunnel gets its fair share of the available bandwidth and computing power by slowing down greedy users if an AP gets more traffic than it can handle.

8 Related Work

Anonymizing Internet traffic has been tried before [13]. Here, we only briefly mention the two best known approaches, Onion Routing and the Freedom System.

Onion Routing [9] uses a Mix network with uniform message length and the possibility to add dummy traffic. The system delivered valuable information on anonymizing delay-sensitive Internet traffic, but went off-line in January 2000. One important result of onion routing is that adding dummy traffic seems to be necessary to thwart sophisticated traffic analysis attacks.

The Freedom System [2] provides a commercial service to browse the Web anonymously. Its approach is very similar to that of onion routing, and the mixing components are the *Anonymous Internet Proxies* (AIPs). From version 1.0 to 2.0, the system underwent some interesting changes, one of them is that packets have no longer a fixed size. The system's designers argument is that the bandwidth overhead is not worth the increased resistance against traffic analysis. One flaw in the system is that an active attacker incorporating two AIPs can trace users using these two AIPs as the first and last AIPs in their chain.

9 Conclusions and Future Work

In its current state, our system fulfills most of the requirements stated in section1. It is trustworthy since a user can select and authenticate the APs she trusts. With SSL, we use a well accepted and studied protocol that even relieves us from having to check for replayed messages. Our system is fair and also very stable since if an AP goes down, only the connections that use that AP are affected. New or restarted APs can easily be integrated since the link encryptions are established during the setup of an anonymous tunnel if needed. Although a single AP is limited in the number of anonymous tunnels and connections it can handle, adding more APs allows a greater number of users of the system.

The system provides a solid basis to experiment with various parameters. Interesting questions include what length to use for the fixed-length messages, where and how much dummy traffic has to be added to make the system resistant against the attacks described in section 3 without adding too much overhead and sacrificing the performance, and how the APs should shuffle messages to maximize complexity of traffic analysis while minimizing the end-to-end delay. We will investigate these questions and hope to be able to present the answers and quantitative results that fulfill the first requirement stated in section 1 soon.

Acknowledgement

The work presented here was done within ShopAware – a research project funded by the European Union in the Framework V IST Programme (project 12361). Marc Rennhard would like to thank also the Swiss Federal Office for Education and Science for his sponsorship.

References

[1] O. Berthold, H. Federrath, and M. Koehntopp. Project "Anonymity and Unobservability in the Internet". Workshop on Freedom and Privacy by Design, CFP2000, York, UK, March 2001, April 4–7 2000.

[2] P. Boucher, A. Shostack, and I. Goldberg. Freedom Systems 2.0 Architecture. White Paper, `http://www.freedom.net/info/whitepapers/index.html`, Dec 2000.

[3] D. L. Chaum. Untraceable Electronic Mail, Return Adresses, and Digital Pseudonyms. *Communications of the ACM*, 24(2), February 1981.

[4] D. L. Chaum. The Dining Cryptographers Problem: Unconditional Sender and Receiver Untraceability. *Journal of Cryptology*, 1(1):66–75, 1988.

[5] T. Dierks and C. Allen. The TLS Protocol, Version 1.0. RFC2246, 1999.

[6] R. Housely and W. Polk. Internet X.509 Public Key Infrastructure. RFC2528, 1999.

[7] A. Pfitzmann, B. Pfitzmann, and M. Waidner. ISDN-MIXes: Untraceable Communication with Very Small Bandwith Overhead. In *Kommunikation in verteilten Systemen, 267*, pages 451–463, 1991.

[8] S. Rafaeli, M. Rennhard, L. Mathy, B. Plattner, and D. Hutchison. An Architecture for Pseudonymous e-Commerce. In *Proceedings of the AISB'01 Symposium on Information Agents for Electronic Commerce*, pages 33–42, York, UK, March 21–24 2001.

[9] M. G. Reed, P. F. Syverson, and D. M. Goldschlag. Anonymous Connections and Onion Routing. *Journal on Selected Areas in Communications*, 16(4), May 1998.

[10] M. Rennhard, S. Rafaeli, and L. Mathy. The Pseudonymity Proxy Architecture. Technical Report MPG-01-02, Computing Department, Lancaster University, Lancaster, UK, February 2001.

[11] D. Wagner and B. Schneier. Analysis of the SSL 3.0 Protocol. Netscape Communications, 1996.

[12] M. Waldmann, A. D. Rubin, and L. F. Cranor. Publius: A Robust, Tamper-Evident, Censorship-Resistant Web Publishing System. In *Proceedings of the 9th USENIX Security Symposium*, August 2000.

[13] N. Weiler and B. Plattner. Secure and Anonymous Multicast Framework. In *Communications and Multimedia Security Issues of the New Century (CMS'2001)*, May 2001.

[14] P. R. Zimmermann. *The Official PGP User's Guide*. Boston: MIT Press, 1995.