in: Hannes Federrath (Ed.): Designing Privacy Enhancing Technologies. Proc. Workshop on Design Issues in Anonymity and Unobservability, LNCS 2009, Springer-Verlag, Heidelberg 2001, 115-129.

Web MIXes: A system for anonymous and unobservable Internet access

Oliver Berthold[1], Hannes Federrath[2], and Stefan Köpsell[1]

[1] Dresden University of Technology, Fakultät Informatik
{ob2, sk13}@inf.tu-dresden.de
[2] International Computer Science Institute, Berkeley
hannes@icsi.berkeley.edu

Abstract. We present the architecture, design issues and functions of a MIX-based system for anonymous and unobservable real-time Internet access. This system prevents traffic analysis as well as flooding attacks. The core technologies include an adaptive, anonymous, time/volume-sliced channel mechanism and a ticket-based authentication mechanism. The system also provides an interface to inform anonymous users about their level of anonymity and unobservability.

1 Introduction

Using Internet services nowadays means leaving digital traces. Anonymity and unobservability on the Internet is a sheer illusion. On the other hand, most people agree that there is a substantial need for anonymous communication as a fundamental building block of the information society. The availability of anonymous communication is considered a constitutional right in many countries, for example for use in voting or counseling.

We are doing research on anonymity and unobservability in the Internet to evaluate the feasibility and costs of such systems and to explore several deployment opportunities within the Internet. Our goal is to explore the foundations and to provide a secure and anonymous technical infrastructure for the Internet.

Systems that provide **unobservability** ensure that nobody, not even the transport network, is able to find out who communicates with whom. However, the communicating parties may know and usually authenticate each other. Example: Paying users browsing a patent data base.

Systems that provide **anonymity** ensure that client or server (or both) can communicate without revealing identity. Example: Users browsing the World Wide Web.

During the last three years we developed several MIX-based and proxy-based anonymity services (for web surfing and similar real-time services). Our academic interest is to show that anonymity and unobservability can be efficiently realized. The special objective is to develop a theoretical background for the efficient implementation of anonymity services in the Internet. We are building an anonymous transport system based on a specific IP format. The goal is to

provide asynchronous (like SMTP) as well as nearly synchronous modes of communication (like HTTP). The system should also be able to handle various kinds of packets.

The web site of our project is http://www.inf.tu-dresden.de/~hf2/anon/.

Anonymity and unobservability are not new security goals. The following systems that provide anonymity services are known both in literature as well as in the Internet: (selection)

- Anonymizer [1],
- Crowds [2],
- Onion Routing [3],
- Freedom [4].

The attacker models for these systems are different, i.e., these systems provide different levels of anonymity. A comparison of these systems is given in [5].

This paper is organized as follows. In section 2 we give an overview of the architecture of our system. Section 3 deals with several attacks and their prevention. In section 4, we explain additional design issues of our practical system and their implementation.

2 Components and Functionality

Basically, we use

- a modified Mix concept with
- an adaptive chop-and-slice algorithm (see below),
- sending of dummy messages whenever an active client has nothing to send,
- a ticket-based authentication system that makes flooding attacks impossible or very expensive and
- a feedback system that gives the user information on his current level of protection.

The MIX concept [6] as well as the adaptive chop-and-slice algorithm will be described in this section. The ticket-based authentication procedure is explained in section 3.1 and the feedback mechanism in section 3.1.

The complete system consists of three logical parts: the JAP (Java Anon Proxy) on the client-side, the MIXes and the cache-proxy on the server-side. These parts are concatenated into a chain, building the anonymous tunnel. All network traffic to be anonymized is sent through this tunnel. In principle, a single user remains aonymous since the tunnel has many entrances (users), but only one exit. Every user can possibly cause the traffic observed at the tunnel-exit.

JAP. JAP is connected to the first MIX via Internet. Ideally, the MIXes should be connected via separate high-speed connections due to performance reasons. However, this would be very complex and expensive, hence our MIXes use only the Internet.

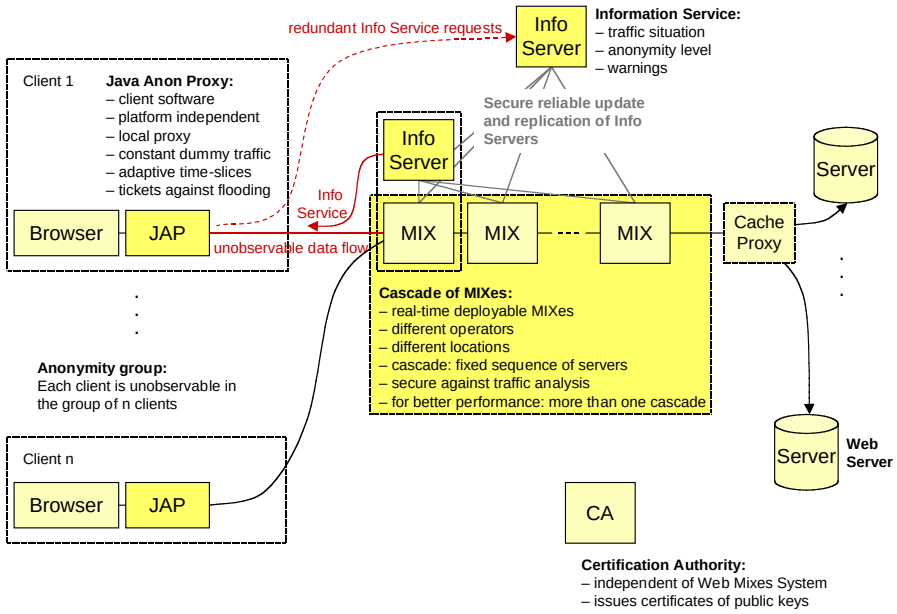

Fig. 1. Architecture of our service

The JAP is a program which is installed locally on each user's computer. All network traffic to be anonymized goes through this software. The JAP transforms the data so that it can be anonymized by the MIXes.

Following functions are provided by JAP:

- Registration of the user to the MIXes,
- Periodical set-up of time-slice-channels (generating and sending of asymmetrically encrypted channel-building messages),
- Sending and receiving data via the channels. Dummy messages (for instance, random bits or encrypted zero bits) are generated if there is nothing to send.
- Listening for requests coming from the browser or other programs of the client that like to communicate in an anonymous way,
- Filtering of content that would be dangerous for the anonymity, e.g., cookies and active contents like JavaScript, ActiveX and other embedded objects,
- Transforming data into the MIX-format and sending through an anonymous channel,
- Receiving data from the active MIX-channel and forwarding it to the originating application,
- Periodical utilization of an info-service, so that the user gets feedback about his current level of anonymity.

MIXes. Our basic concepts are very similar to other systems based on the idea of MIXes [6]. A MIX scrambles the order of data streams and changes their coding using cryptography to make traffic correlation attacks difficult.

The MIXes are simple computers connected via the Internet. They form a logical chain, called "MIX-cascade". The first MIX receives data sent by the JAPs. A MIX makes some cryptographic transformations (strips a layer of encryption, prevents replay attacks, reorders messages and creates a batch that consists of all messages) and sends the data to the next MIX.

The last MIX sends the data to the cache-proxy.

By means of constant dummy traffic, all senders send messages at any time to create the same anonymity group. If necessary, random data is generated which cannot be distinguished from genuine encrypted traffic. Dummy traffic has to be sent between the endpoints of a communication relation. Dummy traffic between MIXes only is not sufficient to prevent traffic analysis.

Our attacker may

- control up to $n-1$ MIXes if n is the total number of MIXes in the MIX-cascade,
- control the cache-proxy and knows the receiver and content of all messages because all traffic goes (mostly unencrypted) through the cache-proxy to the Internet,
- block every message, generate his own messages and modify all messages. These active attacks will be recognized and consequently prevented by a ticket-based authentication system explained in section 3.1.

Cache-proxy. The cache-proxy sends the data to the Internet and receives the answers from the servers (e.g., web servers). The answers will be sent back to the user via the MIXes (in reverse order). As JAP does, the cache-proxy has to send dummy traffic as well.

Once a time-slice-channel is established, it can transport a certain number of bytes. Thereafter, the channel is released automatically. If more data needs to be transmitted sequential time-slice connections must be established. See section 3.1 for more information.

Another functionality of both JAP and cache-proxy affects the HTTP protocol. As far as security and performance are concerned, it makes sense that the cache-proxy automatically loads every object embedded into a HTML-page, e.g., links to embedded images. The cache-proxy can then send the whole page (including the embedded objects) at the same time through the anonymous channel. This idea was proposed the first time by the Crowds project [2].

In order to realize this idea, both cache-proxy and JAP must provide the following functions:

- Cache-proxy scans the data received from a web server for embedded objects.
- Cache-proxy automatically requests these objects and sends them through the anonymous channel to the user's JAP. In addition, cache-proxy should provide traditional caching functionality in order to reduce the number of requests sent to the Internet.

– JAP replies to the requests for embedded objects by sending data already received from the cache-proxy. The number of requests sent from JAP through the anonymous channel is thereby dramatically reduced.

Info-service. Info-service provides data for maintenance and operation of the anonymous network. It provides

– addresses and public keys of the MIXes,
– information on the traffic situation,
– the availability of MIXes,
– data about the achievable level of anonymity, i.e., the number of active users in the anonymous network.

A screenshot of the graphical user-interface of the info-service is given in section 4.2.

3 Attacks and Solutions

For real-time communication, we additionally developed the following concepts both to make traffic analysis harder and to increase the efficiency.

Traffic analysis means that an attacker who is able to control the cache-proxy, can link a particular request to the same user. Every message was possibly sent by a different group of users, so that the attacker can use this information to intersect the anonymity group.

Several attacks exist against the basic concept of MIXes. The attacker's goal is to observe users or to stop the service (Denial-of-Service attack, DoS-attack).

We describe concepts which prevent these attacks or make them very difficult and eventually identify the attacker.

3.1 Solutions against observation

There are two sorts of attacks: passive and active attacks.

A passive attacker can only eavesdrop on communication links, but cannot modify any network traffic.

It is impossible to detect passive attacks. The only solution is to prevent them.

Dummy messages Dummy messages are sent from the starting point (i.e., client) of a communication into the MIX network to make traffic analysis harder.

Sending dummy messages guarantees that all users send the same amount of data during each time slice. Since all traffic (including the dummies) is encrypted, no one, even an attacker, who observes all network cables can know which user sends dummies and which one sends real information.

If the group of users does not change (especially when nobody leaves the group), a passive attacker cannot split the group.

Thus it is necessary that each user operates at least one channel at all times. He has to:

- periodically send channel-building messages,
- send real data or dummy traffic on his channels,
- receive data sent by the cache-proxy.

Each MIX has to send dummy messages back to the user if the user does not receive real data. This ensures that each user receives the same amount of data during each time slice. Since at least the trustworthy (i.e., "unattacked") MIX will send these dummies, it successfully avoids these attacks, supposing that any other MIX (or the cache proxy) does not send dummies.

Adaptive chop-and-slice algorithm Large messages (and streaming data) are chopped into short pieces of a specific constant length, called "slice". Each "slices" is transmitted through an anonymous MIX channel. In addition, active users without an active communication request send dummy messages. Thus nobody knows the starting time and duration of a communication, because all active users start and end their communication at the same time. Otherwise, an observer could determine where and when the anonymous channel starts and ends and could find out who is communicating with whom. Depending on the traffic situation, we modify the throughput and duration of the anonymous channel. The concept of chopping long communications into slices was introduced the first time in [8].

We use a modified version with an adaptive duration or throughput. Once a time-slice-channel is established, it can transport a certain number of bytes and is afterwards released automatically. If an anonymous connection takes longer than one time-slice, it will be composed of a number of time-slices. In this case, JAP provides ID numbers, which cache-proxy uses to refer to an anonymous connection (Slice number SI, see Fig. 2).

In comparison to sending each MIX message separately, a MIX channel is more efficient, because it's not necessary to decrypt all these messages using a slow asymmetric algorithm. Since a MIX must collect all messages of the users before sending them to the next one, the delay time in every MIX is proportional to the length of messages.

To establish a new slice, a user sends (together with all other users) a conventional MIX message through the MIXes. This message contains a symmetric key for each MIX. This key will be used to decrypt or encrypt data, which will be sent later through the channel. The time when the channel starts is defined by the state of the whole system. Normally, it starts when the last slice ends. The slice ends when a committed number of bytes have been transferred. In case of an error, especially if an attacker has manipulated some data, the channel is supposed to stop immediately. Otherwise, the attacker can possibly observe which channel is damaged and is thereby able to correlate sender and receiver.

Ticket-based authentication system A very difficult problem occurs when an active attacker floods the anonymity service with messages in order to uncover a certain message.

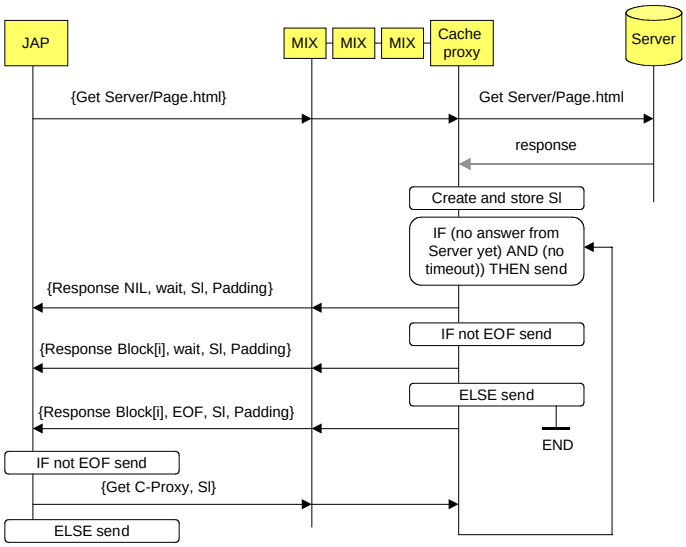

Fig. 2. Time Slice Protocol

We believe that we have found a new concept to suppress flooding of messages both from outsiders (normal users) and insiders (MIXes).

Firstly we limit either the available bandwidth or the number of concurrently used time slices for each user.

Secondly every user has to show that he is allowed to use the system at the respective time "slice" by providing a ticket only valid for a certain "slice". To protect the identity of the user, the ticket is a blinded signature [7] issued by the anonymous communication system. More precisely, each MIX issues a limited number of tickets for each channel and user.

Detailed description of procedure:

Step 1: The user established a connection to a MIX. This connection guarantees confidentiality and integrity and authenticates both MIX and user (i.e., it is possible to use SSL). The user owns a digital certificate and authenticates himself to the MIX. The MIX checks that he gets this certificate for the first time (so it is impossible to get more than one ticket by reconnecting to the MIX). The certificate authority guarantees that each user gets one and only one certificate or it must be recognizable that different certificates belong to the same user (i.e., by including the user's identity).

Step 2: Now the user sends a blinded message to the MIX, which the MIX should sign. This message consists of a key for a symmetric cipher and some bits forming a redundancy.

Step 3: The MIX signs the message using a special key (pair), which is only valid for a certain time slice. We are using RSA and for each new key pair (for each time slice) we use the same modulus n, but we change the public and private exponents.

Step 4: The user unblinds the message and verifies the signature. Now he owns a valid ticket, which is not linkable to him.

Step 5: The user repeats steps 1-4 for each MIX.

Step 6: The user generates the message (channel-building message or data message). Assuming that there are k MIXes. The user concatenates the ticket that he gets from MIX k with the data he wants to send. Then he encrypts the message. He uses the public key of MIX k in order to encrypt the first part of the message. For the rest of the message he uses the symmetric key included in the ticket. Next, he concatenates the ticket issued by MIX $k-1$ with the generated message for MIX k. He encrypts the message in the same way using the public key of MIX $k-1$ and so forth until he encrypts the ticket issued by the first MIX.

Step 7: The user sends the message (generated in step 6) through the MIX-cascade.

If the MIX uses the same prime numbers p and q (and therefore the same modulus n) for the asymmetric encryption/decryption of the message and for signing/checking the tickets, there will be no additional overhead for verifying the tickets.

The ticket exactly fits in the first (asymmetric encrypted) part of the message (step 6). The MIX decrypts the message and verifies the ticket in one step by decrypting the message using the product of it is secret decryption key and it is public signature test key. Furthermore, the MIX extracts the symmetric key, which will be used for the channel and verifies the ticket by checking the redundancy.

As we already explained, a ticket only consists of a symmetric key and a redundancy. For an acceptable level of security, about 200 bits are needed to store a ticket. Since we use RSA, the size of the ticket would increase at least up to 1024 bits.

Each ticket has unused space of about 800 bits. It is possible to store other data in this free space, but it must be known when the ticket is requested. This is actually not a disadvantage, since the ticket is used for the channel-building message. The free space of each ticket could be used to store a part of the message, which is addressed to the next MIX. Thus the channel-building message would become smaller, because we only need 200 bits per Mix instead of 1024 (except for the last MIX).

In order to use this optimization, we have to change the procedure as follows:

Steps 1-4 are modified so that the tickets will be requested sequentially starting at the last MIX. The user generates the next ticket (step 2). He chooses the

symmetric key, computes the redundancy and fills the free space with the first bytes of the already generated message.

Since we know the whole remaining message at the time we generate a ticket, we can use a fingerprint of this message as redundancy. The MIX can calculate the fingerprint in step 7 and thus verify the integrity of the whole received message.

Step 1-4 (and perhaps step 6) can be done parallel, so that we can only send one large message (requesting many tickets) instead of many short ones. This will increase the efficiency, especially the authentication (step 1) has to be done only ones.

If the duration of one time slice is long enough, the overhead for the ticket method won't be very high, since we'll need only one ticket per MIX and channel. The most expensive factor is that the user must directly get the tickets from each mix through an encrypted channel.

Using tickets is useful in order to add the dimension of a prepaid payment system for the anonymity system, too.

However, this has not yet been implemented.

Measurement of anonymity level If the attacker observes all network traffic, he is able to reduce the number of possible senders (i.e., the number of members within the anonymity group) of a message. At the extreme, he may be able to identify the user who sends the message. This is called "intersection attack". The intersection attack can only be prevented if the anonymity group remains constant.

If the group of active users had changed, the linked messages must have been sent by a user, who is member of the intersection of all groups of users.

Dummy traffic makes intersection attacks more difficult, but does not completely prevent them. If all active users permanently send dummy messages, each received message could possibly come from any user.

However, up to now, it is not clear how to prevent such an attack, especially if we consider a global observer.

The anonymity level depends on the number of active users within the system. We need a mechanism or a heuristic that informs the user of his level of protection when he requests contents from the Internet.

In our design we inform each user of his current level of anonymity. The user can decide to recede his communication relation if the anonymity level becomes less than a user-defined threshold, i.e., the number of active users in the system. We believe that it is important for the user to be aware of his degree of privacy. This makes the system more reliable and trustworthy for the user.

Each MIX has to publish periodically

1. the number of active users and
2. the logout time of each user who leaves the group.

The client (JAP) receives this information via the info-service and computes the time when the first linkable message of the current session was sent. JAP

computes how many users were active from this time on, using the published logout times. These users represent the anonymity group, because only they are possible senders of all messages.

When a new connection is established, JAP stores the time and number of active users. Every time the MIXes publishes the logout times, JAP decreases the size of the anonymity group for each detected relevant logout. A logout is relevant if the corresponding logout was earlier than the stored starting time. The user will be alerted if the anonymity group becomes too small.

The information about the number of active users and logout times are digitally signed by each MIX. So it is impossible for the attacker to manipulate them in order to fake a bigger anonymity group.

The client should always assume that the lowest published number of active users and the highest number of logouts is true in order to prevent attacks from a MIX itself.

Since it is impossible to prevent the intersection attack, we can only give advice to users on how to mitigate the effect of intersection attacks: The success of an attack can be further reduced by avoiding linkable events such as Cookies, request of personal web pages or usage of pseudonyms in chat services more than once.

If an attacker also uses active attacks, he can increase his chance to identify a sender. In order to do so, he tries to exclude particular users from the group of possible senders by

- blocking some messages or destroying a network cable,
- sending messages, which appear to be sent by other users, and
- manipulating messages.

However, it is possible to detect active attacks.

3.2 Protection against DoS-Attacks

In section 3.1 we described a procedure for detecting active attacks, but we did not discuss what to do if we detected such one. If an attack is ignored or affected message is simply deleted, the attacker has partly reached his goal.

One solution could be to delete all affected messages or to close all open channels. As a consequence, it would be very easy to start a DoS-attack: The attacker only has to send one broken message in order to impair the whole MIX-cascade. If the attacker is unable to break the anonymity, he may simply prevent the usage of the MIX-cascade.

A better solution is to give each honest user or MIX the chance to prove that he/it has sent correct messages only.

If an error occurs, each MIX will have to prove that it has worked correctly. An attacking MIX cannot do so and is consequently identified. If all MIXes worked correctly, the user, who has sent the broken message is the attacker.

In order to prove the correctness of each participant (including the MIXes) and to detect network errors (or attacks), all output should be digitally signed.

Detailed description of procedure:

a. If a MIX cannot verify the signature, it requests the output of the previous MIX again. If it does not get correct data after a certain period of time (timeout), it will signal a signature-error. In this case, it is not clear who the attacker is. Possible candidates are:
   - the MIX itself,
   - the previous MIX,
   - the network between these MIXes.

b. If MIX i detects an error, it publishes the whole decrypted message (including the error). Now everybody is able to encrypt that message by using the public key of the MIX. If the result is identical with the received message, the MIX has proved that it has decrypted the received message correctly, but that message nevertheless contains an error. The error must have occurred at or caused by the previous MIX.

c. All preceeding MIXes $i - x$ $(x = 1 \ldots i - 1)$ are obliged to contribute to the following "uncovering-procedure" until the error has been found:
   1. MIX $i - x$ has to provide the input-output correlation of the certain message and everyone can verify that the MIX has decrypted the message correctly (see b.).
   2. If the error is found, the uncovering-procedure has to stop immediately and the results have to be published. That means that MIXes $i - x + 1 \ldots i - 1$ are attackers.
   3. If the check was successful, the MIX has to publish the decrypted message (like MIX i).

d. If all MIXes prove that they have worked correctly, only the sender of the broken message can be the attacker. He is the only one who was able to generate a message, which contains an error that would be detected by MIX i.

Hence faulty MIXes can be excluded unless they have been proven that working correctly. A user, who produces errors can also be excluded from the group of users.

A user can only carry through a DoS-attack for a long time if he periodically changes his identity. This is possible if he works together with a corrupt certificate authority. After a period of time, no one will trust this certificate authority any longer.

Nevertheless, there still remains a disadvantage: The trustworthy MIX has to uncover his decryption. If all other MIXes are attackers and work together, they could determine the sender and receiver of a message. However, the attacker "loses" one MIX each time the uncovering-procedure will be performed.

If at least two MIXes are trustworthy, the uncovering-procedure won't break the anonymity:

1. **A user has cheated:** There is no need to protect this user, because the only reason for his message was to confuse the system.

2. **A MIX has cheated:** The first trustworthy MIX of the cascade, which gets a faulty message, starts the uncovering-procedure. This procedure continues until the attacking MIX is reached. If the second trustworthy MIX is before the attacking MIX, the attacker will not be able to detect the sender of a message, because the trustworthy MIX will not perform an invalid uncovering-procedure.
   If the second trustworthy MIX is behind, the receiver of the message is not detectable. The attacker may get the correct message through the uncovering-procedure, but the second trustworthy MIX protects the correlation between sender an receiver.

The described uncovering-procedure is useful for the asymmetric encrypted MIX messages as well as to verify the transmissions of the channels. On each channel, a redundancy for each mix (i.e., a message digest of the data already sent) is sent.

Additionally, each MIX has to store all transmitted data and the channel-building-messages. If an error occurs, it will be able to prove that it has decrypted the data of the channel correctly by using the symmetric key included in the corresponding channel-building message.

One specific feature of the channel is that the data will reach the receiver in very small pieces. If the verification of the channel is done later, the attack may have already been successful. This is possible, even if all other MIXes are trustworthy. But in this case the attacker will lose at least one attacking MIX every time he attacks a channel. This is not acceptable even for a very strong attacker, so that the anonymity of a normal user will not be reduced, if the uncovering-procedure is performed.

4 Other Design Issues

4.1 General

The main focus of our development process is the usability of the whole system. This includes two aspects. Firstly we try to make the handling of the Client (JAP) as easy as possible, so that many people are able to use it. This includes the installation and configuration as well as the design of the user interface.

The second aspect of usability covers the performance, especially the throughput and the delay time of the anonymous network access.

4.2 The Client (JAP)

The development is based on the following requirements:

- The client must be executable on different hardware/operating systems, including at least Windows (95, 98, NT, 2000), Linux, Apple Macintosh.
- The client (JAP) must be easy to install by the user. It should be possible to do this via the Internet or CD-ROM without special knowledge.

- The configuration must be as easy as possible. Even users without extensive knowledge in security and cryptography should be able to configure the system.
- Users must be protected from thinking to be anonymous if they use a misconfigured client.
- A anonymous network access should be possible, even if the user is behind a firewall or proxy.
- The user interface must show and explain the user's current level of anonymity.
- It must be easy to switch between anonymous or non-anonymous network access.

At the moment, JAVA is used as the programming language. That's because of the JAVA-capability: "Write once, run anywhere". Later it should be possible to develop special versions for each operating system. These versions will provide a better performance and lesser resources will be needed. A suitable integration into the look & feel of the target operating system will increase the user-friendliness.

For the execution of the JAP it is necessary to have Java Runtime Environment Version 1.1.8 or equivalent, the GUI-library "Swing" (Version 1.1.1), and a cryptographic library, presently "logi-crypt", installed. These libraries are completely written in JAVA so that we can easily distribute them with our client. All other files needed for the JAP (the JAVA byte code, pictures, help files and so on) are included in one single archive (.jar-file). This file must be copied to the target system and can directly be executed.

In order to make the configuration process easy, it should be possible to download the whole configuration data via Internet. Of course, we have to develop a secure infrastructure for this.

Furthermore, many users gain access to the Internet via an ISP. The provider can force the surfer to use a special web-proxy that does not allow direct Internet connections. In order to give such people the chance to surf anonymously, it must be possible to connect to the first MIX via the web-proxy of the ISP. Therefore, all data must be embedded into the HTTP-Protocol.

Fig. 3 shows the "Anonym-O-Meter", our first attempt to give the user feedback about his current level of protection. This is an area of our future research as well.

4.3 The MIX-Servers

A main point concerning the development of the MIXes is the performance, since this has a great influence on the usability of the whole system. Nevertheless, we also focus on easy administration and maintenance. In contrast to the JAP-user, administrators have a deeper knowledge in computers, and possibly in security.

Our MIXes are implemented using C++ as programming language. We do not use JAVA due to performance reasons. C++ has become a standard and

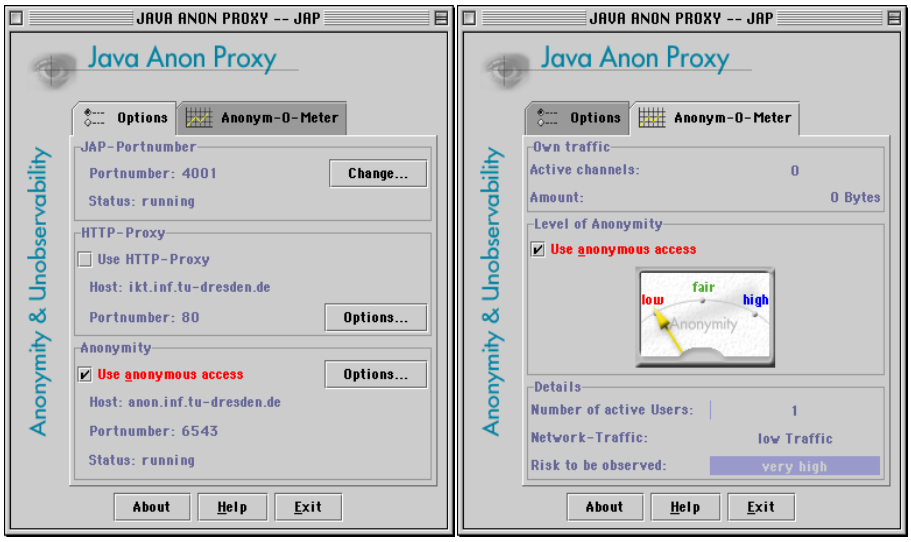

Fig. 3. Screenshot

is available on many operating systems. Differences result from using the system functions like network input/output or from programming the graphical user interface. Since MIXes are console applications without any GUI, this is unproblematic. We only need a few system calls. The main tasks of MIXes are cryptographic and algorithmic operations. In order to port the MIXes to a new target operating system, only a few modules/functions must be adapted. Our goal is to support a few important and capable operating systems (for instance Linux, Solaris, AIX and Windows NT).

The administration is realized via a special network interface implemented by the MIXes. Thus, it is possible to separate the MIXes and the administration tools physically and logically. Presently, we are thinking about a GUI for administration purposes written in JAVA.

Acknowledgements

We would like to thank Prashant Agarwal, Christoph Federrath, Mary Weiss and the anonymous referees for their very useful hints and suggestions.

References

1. The Anonymizer. http://www.anonymizer.com
2. Michael K. Reiter, Aviel D. Rubin: Crowds: Anonymity for Web Transactions. ACM Transactions on Information and System Security 1/1, November 1998, 66-92.
3. Onion Routing. http://www.onion-routing.net
4. The Freedom Network. http://www.freedom.net
5. Oliver Berthold, Hannes Federrath, Marit Köhntopp: Project "Anonymity and Unobservability in the Internet". Workshop on Freedom and Privacy by Design / Conference on Freedom and Privacy 2000, Toronto/Canada, April 4-7, 2000, 57-65.
6. David Chaum: Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. Communication of the ACM 24/2 (1981) 84-88.
7. David Chaum: Blind Signature System. Crypto '83, Plenum Press, New York 1984, 153.
8. Andreas Pfitzmann, Birgit Pfitzmann, Michael Waidner: ISDN-MIXes – Untraceable Communication with Very Small Bandwidth Overhead. 7th IFIP International Conference on Information Security (IFIP/Sec '91), Elsevier, Amsterdam 1991, 245-258.