Freie Universität Berlin
Department of Mathematics and Computer Science
Institute of Computer Science

Bachelor Thesis

A Defense Against Replay Attacks on Chaumian Mixes

Georg Wittenburg – georg.wittenburg@gmx.net

August 15, 2003

Examiner:	Prof. Dr. Hannes Federrath
Tutor:	Dipl.-Inf. Heinrich Langos

Abstract

This paper describes the replay attack against traditional and channel-based Chaumian mixes and puts it into context with other attacks on anonymizing services. It then evaluates different possible approaches to a defense against said attack, concentrating on the question of efficiently recognizing potentially compromised messages within a channel-based cascade of mixes. Finally, an exemplary implementation of a solution is presented for the anonymizing cascade of mixes of the AN.ON project.

Acknowledgements

I would like to thank Prof. Dr. Hannes Federrath for introducing me to the field of IT security and giving me the opportunity to write my thesis on this subject, even though he is teaching at a different university by now. To Heinrich Langos I am indebted for sharing his time, his qualified insight and his good sense of humor and generally making the writing of this paper a much more enjoyable experience than it could have been otherwise. I am also very grateful to Prof. Dr. Elfriede Fehr who made my change into the newly created bachelor degree program as painless as possible and thus gave me the chance to write the first bachelor thesis at this faculty.

My very special thanks go to Jessica Schlegel who bore with me while I was staying up late, made sure that I didn't starve, and made me turn off the computer every once in a while. Finally, I would also like to thank my parents Jürgen Wittenburg and Christina Wittenburg-Benens for supporting me all the time during my studies here in Berlin for these past three years.

Contents

Chapter 1

Introduction to Mixes

The original Internet was designed to provide information services in case of a nuclear catastrophe. Fortunately, this scenario never became a reality and the Internet had a chance to gain wide-spread adoption in civil society. Its core technology however, still remains centered around the goal of reliability and best-effort service, while typical civil use cases have much more differentiated requirements. If the Internet is to become the communication backbone of modern society and assuming that changes in the core technology are unlikely to occur, these additional requirements need to be provided for by building new services based on the existing core.

1.1 Anonymity and Unobservability

Among the three major goals of information security – confidentiality, integrity, and availability – anonymity and unobservability are subgoals of confidentiality. Rather than concealing the content of a given message, it is the goal to hide the identity of the communicating parties from each other and from any outsider. Given a sufficiently strong attacker, it also becomes necessary to conceal the existence of the communication relationship along with its defining parameters such as time, duration and volume of the transmitted messages, thus rendering the communication relationship unobservable. The questions that should remain unanswered are *"Who"*, *"When"* and *"If."*

1.1.1 Applications of Anonymity and Unobservability

If a technology aims to provide comprehensive communication services, a way to anonymously exchange information needs to be among them, as many key activities of individuals or the society as a whole could not be carried out without it. These activities include:

Elections: At the core of the democratic process is the secret vote [GG, Art. 38 (1)] that conceals the identity of the voter from the institution that is counting the votes. This is possible because there is no way to deduce the voter based on a given ballot.

Business: The existence of a communication relationship between business partners taken together with other secondary information such as market position, product line or past communication habits may reveal information about internal strategies or decisions. In the hands of a competitor, this information might endanger the economic success of a company.

Medical Counsel: Counsel related to health or other issues of a very personal nature is commonly offered over the telephone network. In this case, the individual trusts the operator of the telephone network not to reveal his[1]contact information to the other party.

Freedom of Press / Speech: While anonymous statements can safely be regarded as less trustworthy than those of which the author is known, cases are still plausible, in which an author deems it beneficial for his personal safety to remain unknown. [Re03] has a recent example.

Personal Privacy: Not a cornerstone of society as the other items, but still worth mentioning because of the volume of this kind of communication, is the possibility of individuals to exchange messages anonymously mainly out of personal comfort. For instance, in the case of a letter or advertisement published anonymously in a newspaper, the individual trusts the publisher not to reveal his identity to a third party.

The intention to allow for these activities to take place over the Internet is reflected in current policy and legislation: They recognize the importance of anonymity [CE99, II. 3.], confirm the existence of a right of the individual to remain anonymous [Be92], and specifically require a provider of information services on the Internet to allow for his services to be used anonymously or with a pseudonym whenever this is technically feasible. [TDDSG, §4 (6)]

1.1.2 A Formal Definition

True anonymity – the complete lack of any kind of information on who sent a particular message – is not possible as an individual can only be anonymous within a set of subjects, the *"anonymity set"*. Ideally, within this set only a probabilistic measure remains on whom a message belongs to. The set needs to be as heterogeneous as possible, because any feature visibly shared between all members would lead to the attacker gaining information about the individual member in relation to said feature. For instance, if the Alcoholics Anonymous were to set up a anonymous message service for their members, this would not conceal the information that each individual sender of a message has a personal problem related to alcohol, as this feature is visibly shared between all members.

For a single message m one would ideally expect the probability of it belonging to a specific individual i to be equal to the reciprocal of the size of the anonymity set to which the individual belongs:

$$\bigwedge i \in S \qquad \bigwedge m \in \bigcup_{s \in S} M_s \quad : \qquad Pr(m \in M_i) = \frac{1}{|S|}$$

with S being the anonymity set in question and M_i the set of messages sent by an individual i.

Based on this assumption, a simple metric for anonymity $A(i)$ of an individual i comes to mind:

$$\bigwedge i \in S \quad : \qquad A(i) = 1 - \frac{1}{|S|}$$

[1]For the sake of readability only, the male form will be used throughout this text.

with $A(i) = 1$ standing for absolutely anonymous communication and $A(i) = 0$ for no anonymity at all.

This simple scheme however does not take into account the circumstances of the transmission, the differences in communication habits of the individuals, or changes in the composition of the anonymity set. [SeDa02] offers a more in-depth evaluation of possible metrics of anonymity.

1.2 The Concept of a Mix

A technical solution to make anonymous communication over an untrusted medium with a potentially strong attacker possible was first proposed by David L. Chaum in 1981 [Ch81]. His idea is centered around a special entity in the communication network called a *“mix”*. This mix would serve as a relay for messages while taking special steps to render the communication relationship anonymous. For reasons described later in this text, a mix will typically not work alone, but be part of a *“cascade of mixes”* – a chain of interconnected mixes with a distinct first and last mix.

1.2.1 Protection Goals

By being relayed via a cascade of mixes a communication relationship will gain the following qualities:

- The sender of a message remains concealed from its receiver.[2]
- The sender of a received message and respectively the receiver of a sent message remain concealed from third parties.
- The existence of the communication relationship between sender and receiver remains concealed.

The distinction between *“received message”* – a message in the shape in which it reaches the receiver – and *“sent message”* – a message in the shape in which it leaves the sender – is necessary as later on they will become synonymous with messages before and respectively after being processed by a mix.

Figure 1.1 illustrates how these goals create different points of view on a messages: The entire message is only known to the sender; everything except the identity of the sender is visible to the receiver; and the attacker ideally is not even aware that the message exists. As a side note, there are cases in which the receiver is not directly known to the sender, but he is rather sending his message to an untraceable return address as described in [Ch81].

[2]Of course, if either a verification of the sender’s identity or a reply is required, the necessary information may be added to the message, respectively in the form of a digital pseudonym or an untraceable return address as described in [Ch81].

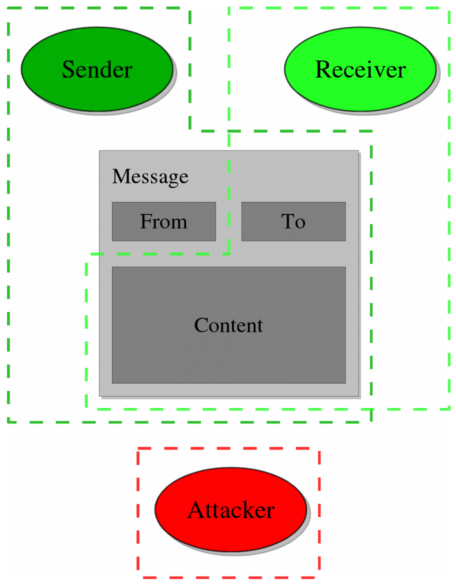

Figure 1.1: Protection Goals

1.2.2 The Attacker Model

The single most important parameter when designing a security related system is the set of assumptions about the capabilities of whoever might try to break into the system – the *"attacker model"*. If the attacker described in these assumptions is weaker than an attacker in the real world, the system will be insecure. On the other hand, if the assumptions are too strong, the system will either be inefficient or might even be completely infeasible.

The assumptions about the attacker under which the goals for a cascade of mixes are to be reached are as follows:

1. The attacker may have both passive and active control over the network connection, i.e. he can intercept any message, modify or delete it, and he can send any freely constructed message to any node in the system.
2. With n being the total number of mixes in the cascade, the attacker may have control over $n-1$ of these mixes and use them both actively and passively for insider attacks.
3. The attacker has no significant control over the members of the anonymity set.
4. The cryptographic algorithms used to built the system are not broken, that is they allow for no correspondence to be found between encrypted and unencrypted messages, nor to create forged messages without knowing the secret key of the corresponding user.

The last of these assumptions relates to the underlying technology used to built a mix. The difficult part is of course that the security of some cryptographic algorithms – or rather the mathematical assumptions they are based on – has yet to be proven. These algorithms are deemed hard to break not because of mathematical proof, but rather because no method is currently known that could compromise them.

On the other side, the first three assumptions are more related to the specific scenario at hand: Control over the network components and the ability to manipulate network traffic are quite realistic for any sufficiently big Internet carrier; at least one mix in the cascade must not be compromised, as otherwise it would be impossible for a mix-based system to achieve its goals; and finally, if the attacker had control over a sufficiently large part of the anonymity set, the corrupt members could adjust their communication patterns to successively isolate their target within the set.

What remains is a fairly powerful attacker with the only major limitation specific to this scenario being that he must not control all mixes.

1.2.3 Notation

While a mix per se is not a very complex entity, the fact that we are dealing with a dynamic system with various participants makes the matter complicated enough for an uniform notation to make sense. Unfortunately, two of the mayor papers ([Ch81] and [PfPfWa89]) use different notations.

The following compromise between the two will be used in this paper:

Notation	**Semantics**
M_i	A mix with $i \in [1, n]$ being the position of the mix in the cascade and n being the total number of mixes in the cascade.
K_i, K_i^{-1}	Public and private key of the mix M_i.
A, B	Communicating parties or – depending on the context – also their addresses as used by the anonymizing system.
r_i	Random data used to pad messages while being encrypted.
m, x, y, z	Messages that are sent through the system.
$K_i(x, r) = y$	Message x is padded with random data r and encrypted with the public key of mix M_i, thus yielding message y.
$K_i^{-1}(y) = (x, r)$	Message y is decrypted with the private key of mix M_i, thus yielding message x and the random data r used for padding.
$M_i(x) = y$	Message x is being processed by mix M_i into message y.
$x \xrightarrow{M_i} y$	A short form of the above for better readability.

Table 1.1: Notational Conventions

The key idea of this notation is to differentiate between the static and the dynamic entities of a mix, such that for example the mix itself and its public and private keys are denoted by an upper-case character, while messages that pass through the mix are given lower-case characters. Additions to this notation further on in this text will also adhere to this idea.

1.2.4 Theory of Operation

A mix is a special machine in a network that relays messages from many senders to their respective receivers. Apart from the obvious technical components, a mix M_i consists of a pair of keys,[3] the public key K_i and the private key K_i^{-1}:

$$M_i = \{K_i,\, K_i^{-1}\}$$

The effect of anonymous communication is reached by the following steps:

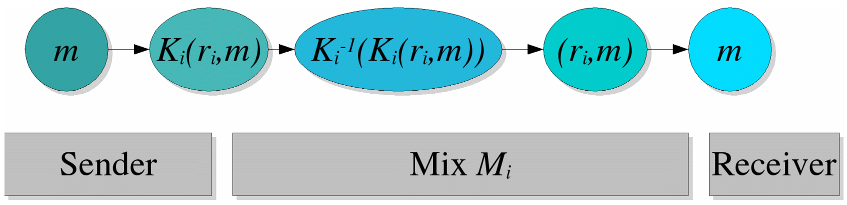

Figure 1.2: Path of a Message Through a Mix

1. The sender encrypts his message m padded with random data r_i with the public key K_i of the mix M_i and sends it to the mix.

2. M_i receives the encrypted message $K_i(r_i, m)$ and checks whether this message has been received in the past.

3. It then decrypts the message with his private key $K_i^{-1}(K_i(r_i, m)) = (r_i, m)$, discards r_i and retains m.

4. Simultaneously with other messages and in an order unrelated to the input sequence, M_i sends the message m to the receiver.

5. Except the information necessary to perform step 2, the mix does not retain any information about these operations after the process in completed.

In the first step, the additional random data is necessary to increase the size of the message to a previously defined message size. This makes it impossible for the attacker to assess the real size of the message and furthermore ensures that encrypting two identical messages produces different crypted messages.[4] Furthermore, the possibility for the attacker to capture the outbound messages, encrypt them with the public key of the mix and easily match them to the inbound messages is eliminated. In case of the message m being

[3]With this pair of keys the mix is a participant of a public key cryptosystem: The public key K_i is distributed among other users, while the private key K_i^{-1} is only known to the mix. In the cryptosystem the two keys are inverses of each other, such that a message m can be encrypted by anyone for the mix as $K_i(m)$ and only the mix can decrypt it with $K_i^{-1}(K_i(m)) = m$. See [RiShAd77] for a thorough description.

encrypted itself, the mix also needs to be informed separately about the addressee by adding the unencrypted address A to the message $K_i(r_i, m, A)$ before sending it to the mix.

The second step makes it impossible for the same message to travel more than once through the mix. This prevents the so-called *"replay attack"* and is the topic of this paper. In order to make timing attacks impossible, a message is relayed together with a set of other messages – a *"batch"* – in step four. Finally, for the above steps to make sense, the mix must not retain more information about this process than absolutely necessary to guarantee proper operation.

If any single one of these steps is carried out incorrectly, the anonymity of sender and receiver is at stake – either intentionally or due programming or configuration error. In order to reduce the risk of trusting a compromised mix, a set of mixes – each of which preferably is run by an independent institution – is chained together to form a cascade. Only one mix in this cascade has to operate correctly to ensure anonymity.

The chain-like configuration of a set of mixes into a cascade is one of many possible topologies. The opposite would be a completely random routing through a set of mixes – a *"mix-net"* – following a path laid out by the sender for each of his messages. Combinations of these two are also possible. While [Ra01] claims that mix-nets are even more secure against traffic analysis than cascades, the chain-like configuration has several compelling advantages: Superior performance can be achieved as network traffic within the cascade is well known and predictable and can be taken into account when allocating network resources; sticking to a default route for all messages keeps the overhead of the protocol small and gives more room for the payload of the messages; and finally, it helps to isolate the end-user from the internal structure of the mix network. Due to these reasons, this paper will concentrate on the cascade of mixes rather than the alternatives.

A message that is sent through a cascade of n mixes is encased for each hop by the sender. The first mix receives $K_1(r_1,\ K_2(r_2,\ \ldots K_{n-1}(r_{n-1},\ K_n(r_n,\ m))\ \ldots\ ))$, performs the steps two through five from above and sends the result to the second mix. For the i-th mix in the cascade the transformation is

$$\begin{aligned} & K_i(r_i,\ K_{i+1}(r_{i+1},\ K_{i+2}(r_{i+2},\ \ldots\ K_{n-1}(r_{n-1},\ K_n(r_n,\ m))\ \ldots\ ))) \\ \xrightarrow{M_i}\ & K_{i+1}(r_{i+1},\ K_{i+2}(r_{i+2},\ \ldots\ K_{n-1}(r_{n-1},\ K_n(r_n,\ m))\ \ldots\ )) \end{aligned}$$

or in a more functional notation, with $snd(x, y) := y$

$$M_i(x) := snd\,.\,K_i^{-1}(x)$$

This procedure is repeated for each mix in the cascade until the final mix receives $K_n(r_n, m)$ and sends the real message m to the receiver.

The major drawback of the traditional mix system is the fact that it was not conceived for a high load of potentially time-critical traffic. It is well suited for asynchronous, latency-independent traffic, such as electronic mail which also was its original field of application.

[4] If the random data was not added before the message m is encrypted, the attacker could try to guess the content of the message m_{guess} and verify his guess by checking whether $K_i(m) = K_i(m_{guess})$.

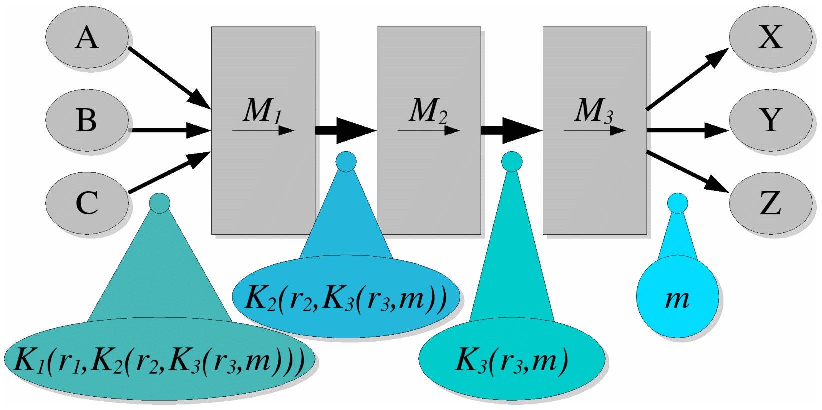

Figure 1.3: Flow of Messages Through a Cascade of Mixes

1.2.5 Channel-Based Mixes

The performance problem of the traditional Chaumian mixes is mainly caused by the computational complexity of the asymmetric cryptographic algorithms involved, which compared with a symmetric algorithm is more computationally intensive roughly by a factor of one thousand. It makes thus sense to replace this mechanism with hybrid cryptography: Asymmetric cryptography is only used to distribute a common secret between the communicating parties. The actual data is then transfered using symmetric cryptography based on the previously shared common secret.

This extension of the concept of a mix was first conceived by Andreas Pfitzmann *et al.* in 1989 [PfPfWa89]. It expands the mix M_i to hold additional state about the currently open channels C_i – most notably about their symmetric keys $k_{i\,j}$, $1 < j < |C_i|$ – and allows the user to establish a persistent high-throughput connection through a mix:

$$M_i = \{K_i,\, K_i^{-1},\, C_i\}, \quad C_i = \{k_{i\,1}, \ldots, k_{i\,n}\}$$

The messages sent to a mix are divided into two categories: normal data messages consisting of layers of symmetrically encrypted messages and special control messages. Noteworthy among these control messages is the one used to open a new channel through the cascade of mixes, because it is the only one that is asymmetrically encrypted and carries the shared secret for the following symmetrically encrypted communication. This leads to a significant gain in performance when compared to the traditional mix concept which required all messages to be encrypted asymmetrically.

While originally proposed for anonymous communication over an ISDN telephone network, the same concept is also applicable to TCP/IP based networks: TCP as a connection based protocol maps naturally to connections routed through a cascade of mixes.[5]

[5]In an actual implementation however, this direct mapping must be avoided as otherwise it would create a point of attack for an attacker to do traffic analysis on.

1.3 Limitations of Mix-Based System

A mix-based system cannot always guarantee full anonymity even if all requirements are met and the system is operating properly. This is due to some aspects of communication simply being beyond the scope of this approach. The two most prominent ones are:

- The anonymity set is weakened over the time by the changing of its composition and thus its behavior. For an observer with sufficient time, patience and the possibility to isolate sessions of related messages patterns will become visible. Based on these patterns and his knownledge about which subset of the anonymity set may have caused them, he can successively reduce the size of the anonymity set until isolating an individual. This attack is known as *"intersection attack"* and as it strikes at the non-technical core of anonymity, a technical defense may be impossible [BeFeKö00] or is at least extremely difficult to implement [La01].
- It is impossible to prevent the sender from unknowingly compromising the anonymity gained by sending his messages through a cascade of mixes by including sensitive information into the messages themselves. This matter is relevant as the vast majority of the messages will not be composed by the sender directly, but rather by an application which he is using. Unfortunately, some popular applications only offer limited support for controlling which information they include in their messages.[6]

[6] Partial solutions to this problem exist for specific applications. For instance, closely related to the AN.ON project a program called *"CookieCooker"* has been developed that helps the user not to reveal his identity on websites that make use of cookies. See `http://cookie.inf.tu-dresden.de/`.

Chapter 2

The AN.ON Project

The *“AN.ON: Anonymity.Online”* project is an implementation of the channel-based mix architecture as described in [PfPfWa89] tailored to the requirements of communicating over a TCP/IP based network – the Internet – rather than the ISDN telephone network. The development started in the year 2000 as part of the *“Projekt Anonymität im Internet”* at the Dresden University of Technology and is sponsored by the German Research Foundation and the Federal Ministry of Economics and Technology. The goal of the project is to evaluate the feasibility and costs of anonymity on the Internet against a strong attacker that may have control about large parts of the system and is capable of performing traffic analysis while providing a secure and anonymous technical infrastructure being able to cope with said attacker. In contrast to other projects that tend to reduce the strength of the attacker in order to reduce complexity and improve performance, AN.ON aims to be a real-time service and provide true anonymity and unobservability at the same time.

The system has been open to the public since the year 2000 and routes 4 TB of traffic per month through its four cascades of mixes in peak times. Current work is centered around implementing batch-based communication between the mixes, remote mix configuration and a system to charge the users for the service offered. The homepage of the project is located at http://anon.inf.tu-dresden.de/.

2.1 System Overview

While it would be technically possible to route all kinds of proxy-based network traffic through the cascade of mixes, the current implementation only supports HTTP traffic. This decision has been made to reduce the risk of illegitimate uses of the system.

AN.ON consists of the following logical components:

JAP: JAP (short for JAP Anon Proxy) is the client-side connector to the cascade of mixes. Written in Java to ensure platform independence, it runs as local proxy and forwards all incoming requests to the first mix of the cascade after they have been properly encrypted. It also queries the InfoService in order to retrieve information about the currently active cascades and to give the user feedback about the current level of his anonymity. The only step necessary to integrate JAP into the user’s setup is to the change the browser configuration to use the port provided by JAP as its HTTP, HTTPS and FTP proxy.

Cascade of Mixes: At the heart of AN.ON is a cascade of mixes that widely follows the specification laid out above: The first mix in a cascade receives its input message from JAP, decrypts them and forwards them to the next mix. That last mix sends the messages to the cache-proxy.

Cache-Proxy: At the end of the cascade of mixes is the cache-proxy. It receives the decrypted messages from the last mix and finally sends them to the webservers to which they are addressed. As the replies from the servers are cached in the proxy, future request can be handled more quickly and network traffic is avoided.

Furthermore, this setup can easily be expanded to scan the replies received from the webservers for items that a user is likely to request in near future and retrieve these items automatically from the corresponding server – this behavior is referred to as *"prefetching"* – thereby further reducing response time. Sending these items through the cascade of mixes before they are requested by JAP would yield additional performance.

InfoService: The InfoService provides the user with data about the status of currently available cascades of mixes. Most importantly, it distributes the addresses and public keys of the mixes, but also statistical data such as system load and information about the level of anonymity.

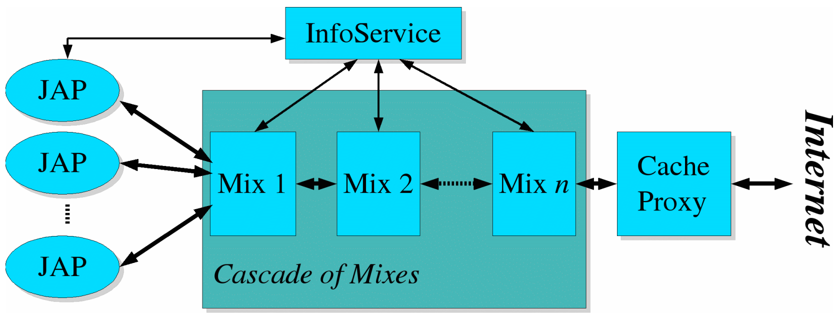

Figure 2.1: Components of the AN.ON System

While the InfoService is not immediately concerned with the security of individual messages, it still represents a single point of failure in the system: Without the InfoService it would be impossible for JAP to retrieve the information about the currently active cascades of mixes and the service would effectively be unavailable to the user. A more redundant solution that replicates the InfoService over several machines is planned for the future.

The AN.ON system and its components are described in more detail in [BeFeKö01].

2.2 Enhancements Over Channel-Based Mixes

The original channel-based mix makes several assumptions about the environment it operates in: The average traffic volume and thus the bandwidth and performance requirements for a mix are assessable based on limitations and statistical information of the ISDN telephone network. As these design parameters change in a TCP/IP environment, the implementation of a channel-based mix needs to be adapted. Additionally, several other enhancements were or will be introduced as part of the AN.ON project:

Adaptive Chop-and-Slice Algorithm: While channels in the ISDN telephone network have a constant bandwidth of 64 kb/s, the bandwidth used by traffic on a TCP/IP network varies greatly depending on the service or application used. Frequently messages will be bigger than a mix packet and hence it is necessary to chop these message into chunks of constant length, called *"slices"*. Depending on the traffic situation it will also be possible to modify throughput and duration of a channel.

Dummy Traffic: Dummy messages are constantly sent into the cascade of mixes by all users thus concealing the existence of a real communication. These dummy messages may consist of an encrypted constant value or encrypted random data. They need to reach either the receiver or at least the last mix of the cascade as otherwise an attacker controlling the first or a middle mix could identify the dummy traffic.[1]

Ticket-Based Authentication System: As the amount of message being processed by a mix at any given time is limited, it is possible for an attacker to gain information about a single message by having access to all other messages currently in the mix. In order to bring about this situation he must *"flood"* the internal message store of the mix with his own messages. He can then identify his own messages in the output of the mix and the remaining one has to be the one sent by his victim.

In order to make the flooding of a mix impossible, precautions have to be taken to make sure a user can only have an amount of message in the mix that is considerably smaller than the total capacity of the message store of the mix. This is achieved by issuing a limited number of *"tickets"* to each user which are valid for sending a message through the mix.[2]

Feedback System: In order to enhance the user's awareness about the quality of the service provided by the system – or lack thereof in case it is impossible to guarantee anonymity – feedback needs to be provided to the user about the current state of the system. This feedback visualizes the level of anonymity depending on the number of active users within the system and thus the magnitude of the anonymity set.

2.3 Technical Details of Interest

The AN.ON system has several peculiar features which will become relevant when examining the possibilities of carrying out a replay attack against it and when designing the countermeasures. Altough a bit technical, they are mentioned at this point to allow for a better evaluation of the the concepts presented later on.

- The size of a packet traveling through the AN.ON cascade is fixed to 998 bytes. This includes a small overhead for signaling and meta information.

[1]It turns out that on a TCP/IP network these messages are also necessary in order to allow regular traffic to pass through NAT gateways which would otherwise regard a TCP connection as inactive and purge it from their routing tables.

[2]Furthermore, it was initially planned to use these tickets as a means of accounting and charge the users for the anonymity service based on them.

- AN.ON uses RSA as asymmetric and AES-128/128 as symmetric crypto algorithm. AES is running in output feedback mode OFB-128. See [RiShAd77] and [AES] for the complete specification of these algorithms.
- The communication between user and the first mix as well as the communication between mixes is encrypted and authenticated with SSL.
- Between the mixes, all channels are multiplexed over one TCP connection and demultiplexed at the receiving end based on their channel ID, which is a 32-bit random value.

Further details can be found in the technical documentation of the AN.ON system [ANONdoc]. Additionally, a short overview of the classes and their functions is given in appendix A.

Chapter 3

The Replay Attack

A correctly implemented mix-based anonymity service guarantees protection against attackers that could easily compromise systems based on other concepts such as anonymizing proxies.[1] While these systems are only secure against an attacker with very limited capabilities, a mix-based system can even guarantee anonymity when more powerful attacks are performed. One of these is the so-called *"replay attack"* which [Ch81] briefly describes as follows:

> *"[...] if just one item is repeated in the input [of a mix] and is allowed to be repeated in the output [of said mix], then the correspondence [between sender and receiver] is revealed for that item."*

This chapter will put the replay attack into perspective with other attacks, examine possible interactions, and investigate how it could be implemented by an attacker.

3.1 Modeling the Attacker

Of the many conceivable attacks on an anonymizing service most attacks are of generic nature and are feasible against all kinds of systems. They can be put into categories based on the capabilities an attacker requires to perform them. The more sophisticated an anonymizing system is, the more capabilities may the attacker have against whom it provides protection. For all systems the most challenging attacks are the ones whose attacker has a set of capabilities that stretches the design parameters of the system, as it can be assumed that a system, that for example protects against a strong passive attacker will also trivially protect against a weak passive attacker.

The attacker can thus be classified by the capabilities that are required to perform the attack. In this paper the following set of capabilities is used for classification:

Weak Passive (P_w): Being able to intercept relevant network traffic from a limited part of the network for a limited period of time is the most basic of all capabilities. Any Internet service provider has this capability over his users. Obviously, it is required for all types of attacks.

Strong Passive (P_s): This attacker can intercept network traffic from any part of the network over a period of time of his choice and has the resources to store and process it.

[1] A typical example for an anonymizing proxy is Anonymizer.com (`http://www.anonymizer.com/`).

Note that the requirements on these resources are proportional to the size and average traffic of the anonymity set to which the target of the attacker belongs and can thus be of considerable size. Major network carriers with a strategically connected backbone and possibly government agencies fall into this category.

Weak Active (A_w): The attacker has the capability to send carefully constructed messages to any part of the network. While basically all machines connected to the network are capable of doing this, it depends on the quality – or rather the lack thereof – of their Internet service provider which of the messages pass the filtering procedure that may be in place and reach their destination.

Strong Active (A_s): Rather than sending new messages to hosts on the network, this attacker can intercept messages sent by others and remove them from the network thus causing them never to reach their destination. The entities capable of performing this attack are similar to the ones mentioned in the description of the strong passive attacker, except for the fact that it should be rather hard for them to hide their existence for a longer period of time provided that the network topology does not undergo major changes.

Insider / Client (I_c): The attacker is or rather controls one or more users of the system. Assuming that the anonymizing service is open to the general public this could literally be anyone.

Insider / Server (I_s): The attacker is or rather controls one or more components of the system, e.g. the mixes in the cascade. Note that the assumptions from section 1.2.2 still need to hold true and only $n-1$ mixes may be controlled by the attacker.

Finding a way to sort these capabilities in a meaningful way is difficult, because depending on the circumstances a capability may have a radically increased importance that is not given in most other cases. Furthermore, the set of capabilities that an attacker possesses may either be regarded as strong or weak depending on the target of his attack.

In order to allow for a comparison to be made between the capabilities, the following assumption is made: An attacker has to acquire his set of capabilities in the real world. If he has managed to acquire a capability to which access is highly restricted, it can be assumed that he has the resources to acquire all capabilities to which access can be considered easier. Therefore, the capabilities may be sorted by the difficulty an attacker has while getting access to them, the *"strongest"* capability being the one to which access is most difficult. The resulting half-order is depicted in figure 3.1.

The rationale for this classification is as follows: Anyone can become user of an open anonymizing service, most people can send relatively freely constructed messages, and there is a huge abundance of Internet service providers. Quantifying whether it is more difficult to gain access to a network carrier or become part of a cascade of mixes is not obvious, yet it seems as if carriers are more likely to run a mix than the other way round. Finally, the level of access required to remove messages exceeds the one required to merely capture them.

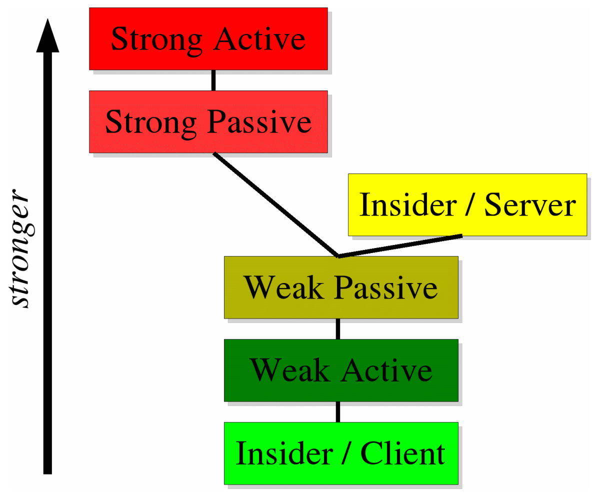

Figure 3.1: Capabilities of Attacker Sorted by Strength

3.2 Attacks on Anonymizing Services

A summary of relevant attacks on anonymizing services is given in table 3.1. Additional attacks mentioned in [BeFeKö00] and [Ra01] such as the collusion attack, active attacks exploiting user reactions, the "sting" attack, the "send 'n seek" attack and message delaying are omitted as they are rather concerned with interaction between users or the setup of the user's system rather than the implementation of the anonymity service itself.

These attacks can be compared with each other based on the half-order established in figure 3.1. In figure 3.2 the attacks are sorted by the strongest capability they require. The *"Insider / Server"* classification of the replay attack depends on the implementation of the cascade of mixes, i.e. whether the communication between the mixes is encrypted or not.

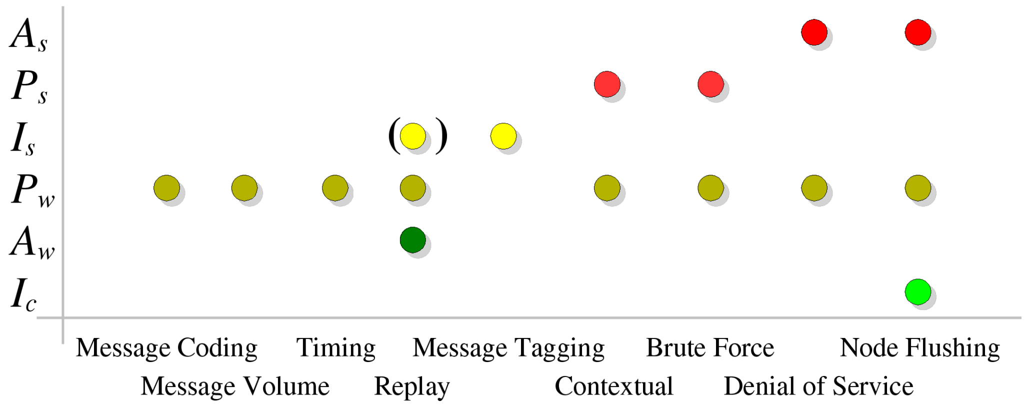

Figure 3.2: Capabilities Required to Carry Out Attacks

A more detailed description of these attacks can be found in [BeFeKö00] and [Ra01].

[2]As we will see in section 3.5, for a replay attack to be successful on one of the AN.ON mixes, the attacker needs to control one of the mixes.

Name	**Description**	**Required Capabilities**					
		P_w	P_s	A_w	A_s	I_c	I_s
Brute Force Attack	Following all possible paths that a message may have taken will yield a set of potential receivers.	●	●				
Contextual Attacks (including Communication Pattern and Intersection Attack)	Gaining information about the context of a communication (usage patterns, online / offline periods) may help to single out an user within the anonymity set.	●	●				
Denial of Service / Routing Attack	In the case of a mix-net causing one or more mixes to be unreachable may reveal how an user's messages are routed.	●			●		
Message Coding Attack, Attacks on Distinguishing Features	The lack of or only weak changes in the coding of messages being transmitted between mixes makes correlating messages feasible.	●					
Message Tagging Attack	Similar to the message coding attack, but first and last mix can agree on a non-intrusive form to tag messages.						●
Message Volume and Packet Counting Attack	Observing the size of messages either directly or split into several packets can yield matching parties among senders and receivers.	●					
Node Flushing Attack (a.k.a Spam Attack, Flooding Attack, $n-1$ Attack)	The attacker *"floods"* the system with messages filling up the internal messages store of a mix. For any additional message being processed by a mix in this situation it is possible to find its corresponding output message as all other outbound messages are known to the attacker.	●			●	●	
Replay Attack	By resending a previously captured message a mix is tricked into processing it again, thereby making it distinguishable in the output.	●		●			●[2]
Timing Attack	Similar to the message volume attack, but instead of message size the time and duration of the communication is observed.	●					

Table 3.1: Attacks on Anonymizing Services

3.3 Motivation for the Replay Attack

As shown in table 3.2, a mix-based system as proposed in [Ch81] eliminates the prerequisites for each of the attacks that a weak passive attacker could carry out. It is thus impossible for an attacker with only weak passive capabilities to correlate sender and receiver of a message.

Attack	Countermeasure
Message Coding Attack	Strong cryptography layered around the message
Message Volume Attack	Random data used to fill up messages to constant length; long messages split into multiple shorter ones using the chop-and-slice algorithm
Timing Attack	Delayed batches of messages

Table 3.2: Countermeasures Against Passive Attacks

As mere observation is no longer sufficient to extract the desired information, the attacker requires an additional capability in order to try the next potentially successful attack: By acquiring the resources to actively send constructed messages to a machine of the anonymizing system, he can try to trick a mix into revealing this information. The replay attack is thus to be regarded as the closest possible replacement for the various passive attacks that are no longer feasible against a mix-based system.

3.4 Theory of Operation

As mentioned briefly in table 3.1, the general idea behind the replay attack is to resent a previously captured message through a mix, hoping that the mix will process it again, and that the intersection of the original outbound batch with the current one will contain exactly one message. This message corresponds to the replayed inbound message and in- and output of the mix has thus been successfully correlated. The attack exploits the fact that a mix needs to operate deterministically and identical input always must yield identical output. It is not possible to move away from this mode of operation as otherwise the concept of the user adding layers of cryptography for each mix in the cascade would be infeasible.

The replay attack works differently for channel-based mixes than for traditional mixes and they have to be considered separately.

3.4.1 Traditional Mixes

In order to carry out a replay attack against a traditional mix, the following steps need to be performed:

1. Capture a message from the set of messages being sent to the mix by the user whom to attack. This only is possible for the first mix of a cascade as otherwise the sender of a message would already be concealed. However, when assuming that the attack

can successfully be applied to all mixes in the cascade, it is possible to break the anonymity hop by hop starting at the first mix.

2. Capture the batch of messages being transmitted by the mix after it has received the captured message. Note that depending on the application of the traffic being routed through the mix, it may not be necessary for the message to be part of the next outbound batch. In this case it will be necessary to capture multiple batches, possible increasing the amount of batches captured over several tries.

3. Resend the captured message to the mix.

4. Capture the next batch as it leaves the mix. If the mix processed the replayed message again, the batch will contain an exact match of a message contained in one of the previously captured batches. The correlation between input and output of the mix has been successful.

Alternatively, a variation of the attack is conceivable in which the intercepted message is sent to the mix repeatedly in quick succession. After it being processed by the mix, the outbound batch of messages is searched for multiple occurrences of the same message, thereby establishing a corelation between input and output message. [Pf00]

This process may be repeated for each mix in the cascade until reaching the last mix. Successfully attacking this mix will yield the receiver of the message from the user being attacked.

3.4.2 Channel-Based Mixes

Channel-based mixes are less vulnerable to the replay attack than traditional mixes. The distinction between control and data messages, and among the control messages the channel-open message, makes it possible to add a countermeasure to the system: As all messages except for the channel-open message are transmitted using symmetric encryption, an intelligent choice of which symmetric crypto algorithm to use will protect these messages.

Contrary to the asymmetric encryption algorithms, which in order to perform their task properly either have to remain stateless or make their state publically visible (e.g. dependent on the system time), the symmetric encryption algorithms can already rely on the state – the common secret – previously shared with the communicating party in the channel-open message. If this state is fed back into the encryption algorithm to successively affect all messages traveling through the channel, an attacker without knowledge of this state is unable to inject or replay messages into the channel. It would not be possible to encrypt newly created messages for injection at all, and replayed messages would be out of sync with the current state of the symmetric encryption algorithm. This can either cause the channel to break down, if the symmetric algorithm is unable to compensate for this unexpected input, or otherwise cause a *"hiccup"* that will correct itself after another few blocks of data have passed the decryption function. In either case it is possible to detect the intrusion into the existing channel.

Replay attacks against messages belonging to a channel are thus not possible, except for the case in which the attacker captures the entire set of messages belonging to the channel in question including the channel-open message, and is able to replay the complete flow of messages. However, this is only possible if the channel-open message is accepted twice by the mix. In the end, what remains is indeed only the channel-open message as the single one requiring additional protection.

3.5 Existing Protective Measures

Several protective measures are already in place that make it harder although not impossible to perform a replay attack against an AN.ON mix. Note that their original intention was not to protect against this attack, it is just to be regarded as a side effect. Hence, the protection cannot be regarded as complete.

TCP Sequence Numbers: The connection between users and mixes and among mixes is based on TCP connections. Packets belonging to a TCP connection have a sequence number to identify them. In order to hijack a connection, it is necessary to predict this sequence number, which depending on the implementation of the TCP/IP stack may be very easy or very hard.

Secure Socket Layer / SSL: The communication between users and mixes and among mixes is encrypted and authenticated using SSL. This is a major problem for the attacker as it makes it virtually impossible for him to send a forged packet to a mix. It is thus not possible for an external attacker to perform a replay attack. Once again, he needs an additional capability, this time *"Insider / Server"*. When controlling the mix he can trivially send forged packets through the SSL connection.

These measures add up to several layers of protection. Only after breaking or circumventing them by being in control of a mix, the attacker will gain access to the asymmetrically encrypted message which is his immediate target. Thus, hijacking a connection is hard, but as we will see insider attacks are still possible.

3.6 Implementation

An external attacker would have to circumvent the protective measures described above, which is a rather daunting task. Rather than hijacking a TCP connection and breaking into a SSL connection, it is assumed at this point that the attacker will be an insider, i.e. he directly controls at least two mixes, one for replayed messages and a second one for capturing the output of the attacked mix. Note that this scenario is valid for any cascade with more than two mixes, as one mix still remains trustworthy.

In the implementation, the attacker starts with creating a storage for captured messages. The capturing and later on the replay of packets can be easily triggered on a running system, by installing signal handlers for the signals `SIGUSR1` and `SIGUSR2`. When receiving `SIGUSR1` the mix will wait for the next channel-open message, capture it, and then go on to successively capture all packets belonging to the channel in question. This process is terminated either if the storage is full, or when the channel is closed. With this data stored in the mix, the attacker can now trigger a `SIGUSR2` and replay these captured messages over the existing SSL connection. The decrypted replayed packets will now appear among the outbound packets of the attacked mix.

Chapter 4

Countermeasures

Prevention of the simple version of the replay attack, which correlates messages by searching for multiple occurrences in the outbound batch, can trivially be achieved by eliminating duplicate messages while assembling the batch. However, in order to prevent all forms of replay attacks from succeeding, a more general solution is required and two approaches come to mind: On one hand, the risk of a mix being compromised by a replay attack can be reduced by only allowing messages to be processed during a certain period of time. On the other hand, a mix simply has to keep a record of which messages it has already processed and ignore these messages in case it receives them again.

These two approaches may be combined to meet the requirements as closely as possible. Except for the already mentioned fact that in the case of channel-based mixes only the channel-open messages need to be considered, the countermeasures presented in this chapter equally apply to both traditional and channel-based mixes. However, the main focus remains on channel-based mixes as this concept is more applicable to the AN.ON project.

4.1 Requirements

The requirements for an implementation of countermeasures against a replay attack are as follows, sorted by their relative importance, most important first:

1. First and foremost, the implementation obviously has to **prevent a replay attack** from being carried out successfully, i.e. it must make it impossible for duplicates to occur in the output of a mix. More functionally speaking, the processing function of a mix has to be changed to

   $$M_i(x, P_i) := \begin{cases} \textit{``Detected Replay Attack!''} & \text{if } x \in P_i \\ (snd\,.\,K_i^{-1}(x),\ P_i \cup \{x\}) & \text{if } x \notin P_i \end{cases}$$

   which expands the concept of a mix to

   $$M_i = \{K_i,\, K_i^{-1},\, C_i,\, P_i\}$$

   with P_i being the set of messages that the mix M_i has already processed in the past. Note that apart from processing the message x, the mix also changes it internal state by adding x to P_i.

2. It has to **integrate into the existing system** without requiring relevant changes in the way the system is currently running. This is particularly true for the AN.ON system, which – as pointed out in section 2 – already has a well established user base.

3. The implementation has to be **run-time efficient**, especially in the common case of a message not being part of a replay attack. The AN.ON project aims to provide real-time service, which considering the expectancy of the users accustomed to an ISDN connection would be around 7.5 KB/s with a latency of 150 ms.[1]

4. It should only cause a **minimal network overhead**, if any. As the size of a packet transmitted through the AN.ON cascade is fixed to 998 bytes, any additional overhead will have a direct negative impact on the payload of each packet.

5. Finally, the implementation has to be **scalable** in both processing time and memory usage to allow for the cascade of mixes to transmit more traffic in the future. It seems reasonable to allow for a growth in traffic volume by a factor of ten as this would exceed the bandwith that currently connects the AN.ON cascades to the Internet.

The solution has to be implemented for all mixes in the cascade. Otherwise, if a single mix is left unprotected, the attacker model would allow for all other mixes to be compromised and the attacker would thus be able to circumvent the replay protection of the entire cascade.

4.2 Limiting the Validity of Messages

A first step to reduce the vulnerability of mixes against replay attacks is to limit which messages may be processed by a mix at a given time, i.e. to introduce a clearly bounded period of time during which a message is valid. This concept helps to protect against a replay attack, but cannot prevent it from happening. As usually there are no guarantees on how fast a messages will be relayed over the network, the duration of the time interval – the window of validity – needs to be long enough to compensate for these delays and also allow for the clocks of the user and the clocks of the mixes to be slightly out of sync. Hence, there is still an opportunity for attack, although it is considerably smaller as compared to a system with unlimited validity of messages.

4.2.1 Timestamps

Including a timestamp in the message is the obvious way to limit its validity and has already been suggested in [Ch81] and [PfPfWa89]. The timestamp needs to contain either implicitly or explicitly two separate pieces of information: the start time of the interval and its duration (or rather start and end time, which is equivalent).

Several alternatives on how to represent this information are conceivable:

Store start and end time explicitly: Assuming that the system time of a computer is an unsigned long integer, which is a 32-bit value on most common architectures, describing the seconds passed between current time and the Epoch of the respective operating system, this would cause $2 \times 32 = 64$ bits of overhead in each packet.

[1]Currently, traffic relayed through an AN.ON cascade reaches 22.36 KB/s with a latency of 1604 ms on average. The overhead of mixing the messages causes the latency to increase by a factor of ten, yet it is still within the limits of real-time web browsing.

Store start time and duration explicitly: If the duration is represented with a granularity of seconds, then eight bits of space would already allow for the interval to range between 0 and $255 \div 60 = 4.25$ minutes, which is plenty considering that an user would certainly abort an interactive session if it takes just a quarter of this delay. This option leaves us with 40 bits of overhead per packet.

Only store start time explicitly: The duration needs to be implicitly known as a constant in the anonymizing system. As network conditions are not expected to vary greatly between users and as otherwise there is no need for them to control the duration of the time interval, it may well be set to a constant value within the system. This reduces the overhead to mere 32 bits.

Reduce the granularity by which the time is resolved: Taking seconds as the unit of time by which the time intervals are identified is not necessary, as for the scenario at hand longer units will do just as fine. Assuming a resolution of time units of 4.25 minutes, this would allow for the less important eight bits of the time to be discarded, leaving us with 24 bits of overhead.

Adapt the Epoch of the system: The anonymizing system can use a different Epoch than the operating system it is running on. If this Epoch is kept variable and would start over every new year, then the year could be divided into time intervals of a duration of $60 \times 24 \times 365 \div 2^{16} = 8.02$ minutes which would only require a 16-bit value to identify them. The drawback is of course that provisions have to be taken to match an interval to the correct Epoch as otherwise it would be possible to replay a message once every year.

Introducing timestamps into the messages in order to control the validity of the messages would naturally require for the mixes within the cascade to diverge only slightly in their system time on the internal side of the cascade. On the external side, the user would be required to synchronize his time with the time of the cascade.

What remains is the question of where to store the timestamp within the message. It could of course have its own field in the message header, but alternatively it could be contained in the random part r_i sent with each message as suggested by [Ch81]. At least for the channel-open messages of channel-based mixes it could also be part of the symmetric key contained in this message, however this approach would reduce the randomness of the common secret shared between sender and mix and open the door to attacks on the symmetrical crypto algorithm.

4.2.2 Changing the Asymmetric Key Pair

Another possibility to make messages only valid during a limited period of time is to change the key pair K_i and K_i^{-1} used for the asymmetric encryption algorithm.

A message encrypted with the old public key would cease to be valid once the new public key has been available to the users and a predetermined time has passed. This extra time would allow for the users to reinitialize their encryption functions and for the remaining old messages to be transmitted normally through the cascade of mixes. During this period of transition, both the old and the new asymmetric key pair have to be valid. An invalid – and thus potentially replayed – message would be recognized simply by the fact that the mix is unable to decrypt it properly as $K_{i\,new}^{-1}(K_{i\,old}(m))$ would not yield any meaningful data.

The drawback of this approach is that it requires an infrastructure to automatically make the new public keys known to the users. For traditional systems, these keys are usually regarded as quite static and are only retrieved when a user first starts to use a cascade of mixes. Therefore, it is not necessary to check these keys again whenever a new session is started, let alone allow for them to change during a session. What is worse, if these keys are allowed to change, an opportunity for an attack is opened in which the attacker foists a corrupted key on a user. Even more than for a system in which the asymmetric key pair does not change, in this case extra precautions would have to be taken, e.g. every new public key needs to be signed by a trusted authority.

Based on this observation, changing the asymmetric key pair may be a viable option as long as it is not required to undergo this procedure on a regular, short term basis. It may however provide just the fitting solution for the problem of identifying the Epoch of the cascade of mixes described above.

4.3 Recognizing Previously Sent Messages

There are essentially two places in which the information needed to recognize a previously sent message can be stored: either in the mix or in the message itself. Keeping it in the message however requires the mix to be able to authenticate this information and it thus does not eliminate the need for the mix to store data but merely reduces the memory needed to do so. In fact, any reduction in memory usage is highly desirable as the size of the required storage is the crucial aspect of this approach.

Apart from the two possibilities examined below, [No00] proposes a third alternative: For each time period a so-called *"time vector"* is created, which is essentially a bit field addressed by the fingerprint of a message. The idea is to set the bit corresponding to a message once it has been received by the mix and use this bit as indication for a replay attack for the rest of the time period. Even when only using a 32-bit fingerprint (rather than the 64-bit value suggested in [No00]), the memory requirements of $2^{32} \div 8 = 512$ MB per time period still render this solution unrealistic.

4.3.1 Hash Table of Message Fingerprints

Storing all messages that are potential targets of a replay attacks directly in the mix is not an option as this would violate the requirement of low memory consumption. Hence, as a first step the amount of data to be stored is reduced by only looking at a b-bit fingerprint of each message. Assuming that m relevant messages are processed by the mix while this store is active, there only remains a probability of $\frac{m-1}{2^b}$ for two messages to have the same fingerprint. Setting $b = 64$ bits and $m = 50,000$ messages,[2] this leaves us with a probability of 2.7×10^{-15} for a fingerprint collision to occur, which is smaller than that of a hardware failure of the machine running the mix. However, setting $b = 32$ bits results in a probability of 1.1×10^{-5}, which is in the same order of magnitude as the reliability of the machine.

Therefore, the fingerprint of the messages should be at least 64 bits long, a value which is also suggested by [No00].

There are several possibilities for generating a fingerprint of the message: The message could be split into blocks of the length chosen for the fingerprint, padding the last block with a fixed pattern in case it is too short, and then bitwise XORing these blocks. The resulting fingerprint will obviously be dependent on the entire input message, but the probability distribution of the values it can hold may be less than optimal depending on content and structure of the input message. This fact may cause collisions of the fingerprint to occur more frequently than anticipated and might even allow for an attacker to construct messages to yield a specific fingerprint.

Fortunately, there is no apparent way in which the attacker could use this possibility to his advantage. He could intercept a message, construct another one that has the same fingerprint, and send that one to the mix prior to the original. This would make it seem as if the legitimate user was trying to perform a replay attack and his message would be ignored. However, if the attacker can intercept a message and retain it, there would be no need for him to compromise the mix in order to have the the message dropped. Still, these speculations do not eliminate the fundamental flaw from the system. Hence, the wiser choice is to resort to a well established message digest algorithm such as MD5 or SHA-1 as described in [Ri92] and [SHA1]. These two condense a message into a 128-bit and respectively a 160-bit value and as they are specifically designed to be collision resistant, finding another message that yields the same value is impossible in adequate time.

Yet another option available for channel-based mixes is to store the symmetric key of a channel instead of a fingerprint of the channel-open message. However, this has the drawbacks that all messages need to be decrypted using expensive asymmetrical algorithms in order to retrieve the key and that the symmetrical key is likely to be larger than 64 bits.[3]

As a second step, the fingerprints of the messages need to be accessed efficiently. This can be achieved by storing them in a hash table with collision resolution by "chaining" as described in [Kn98]. Depending on the traffic situation of the mix, the size of the hash table may be adapted in order to avoid collisions as much as possible while still trying to use as little memory as possible.

4.3.2 Ticket-Based Approach

While not eliminating the need for the mix to store data, the ticket-based approach still makes the volume of data independent of the traffic situation. It merely requires a constant amount of memory for each user currently connected to the mix. The general idea behind this approach is for the user and each mix to share a secret – the set of *"tickets"* – which are then sent along with each message. Tickets may only be used once and hence replaying a message without knowing one of the valid tickets will fail.

The critical operation of this solution is that the set of tickets needs to be shared in a secure manner which, of course, must not be subject to a replay attack itself. It has to to take place before the first message is sent to the mix or a channel is opened. As users and mixes are already participants of a public key cryptosystem, the following procedure

[2]The cascades of mixes of the AN.ON project handle roughly 4,000 channel-open messages per minute.
[3]AN.ON uses AES for symmetrical encryption with a 128-bit wide key.

seems adequate: The user sends his public key to the first mix, thereby requesting a set of tickets to be assigned to him. The first mix then generates a set of tickets, encrypts it with the user's public key and sends it back to him. The connections between the mixes are protected in a similar manner: Each mix agrees with it successor on a set of tickets to be used between the two for authentication of messages. Note that if the user generated the tickets and sent them to the mix, the procedure would be vulnerable to a replay attack, as the attacker could request the same tickets for his messages. Furthermore, if the user was to request tickets directly from each mix, then even the last mix of the cascade could still identify the message belonging to this user, which would violate the requirement that the cascade should work properly even with $n-1$ mixes in the hands of the attacker.

For each message to be sent the user can now take one ticket from the set, include it in the headers of the message, and send the message to the first mix. If the mix can confirm that the ticket has indeed been assigned to this user, then the message is valid. On the other hand, if the ticket is invalid or has already been used – and was thus removed from the set of tickets of the user – then the message might be part of a replay attack and must be discarded.

Obviously, the size of the set of valid tickets will decrease over time and precautions have to be taken to ensure that there are always enough tickets available for authenticating messages. One possibility is to request new tickets in the manner describes above. It would however cause a sharp increase in the latency of every message that is processed when the set of tickets is empty. This can be avoided, if with every message that consumes a ticket the user suggest a new ticket to the mix. If the message is processed normally, the user will know that the newly suggested ticket has been added to the set of valid tickets and he can use it in the future to authenticate his messages. The weak point in this scheme is that it assumes that the attacker cannot inject invalid tickets into the communication relationship between user and mix. This assumption holds, as new tickets are only regarded as trustworthy if sent together with a valid ticket, which can only be known to the user. Also, as these two tickets are are asymmetrically encrypted with the public key of the mix, the attacker would need to break this encryption scheme in order to selectively manipulate the new ticket.

The tickets themselves consists of a random value whose size depends on the size of the set of tickets to be used. The bigger the set, the larger the random value, because otherwise the probability of the attacker being able to guess a valid ticket would increase. Preferably, the tickets should be as small as possible as at least one (or two depending on which solution is used to refresh the set of tickets) has to be included in each message sent to the mix, thereby causing the payload size to be reduced and the throughput to drop.

4.4 Evaluation

The ticket-based approach causes additional network overhead and would also require for major adaptions of the existing system. Therefore, the storage of message fingerprints seems to be a better solution, provided that the memory requirements are within acceptable limits. This is achieved by combining the concepts of message fingerprints and timestamps, the latter of which in turn depends on the change of the public keys of the mixes in order to keep the size of the timestamps to a minimum.

The total memory requirements of this solution consists of a 64-bit fingerprint of each channel-open message, of which about 32,000 are expected to be processed by the mix during a time interval. Considering that the requirements demand for a compromise between avoiding collisions in the hash table and still being memory efficient to be found, the load factor of the hash table should not increase above 75%. Hence, there has to be room in the hash table for roughly 43,000 entries. Each entry has a size of 8 bytes and thus the size of the hash table is $8 * 43,000 = 344,000$ bytes. Two of these hash tables have to be kept in memory at the same in order to allow for old packets of the previous time interval to be processed correctly when the switch to the new time interval has just been made.

It now also becomes apparent that the solution also meets the requirement of scalability, as the necessary storage in case the traffic would increase by a factor of ten, still would be below 64MB. Unfortunately, due to the 16 bits required for the timestamp in every packet, even this carefully constructed solution requires for the protocol between JAP and the mixes to be changed. Additionally, the persons in charge of running the mixes have to keep in mind that unless they change the public and private key pair of the mix, their system will be slightly vulnerable.

4.5 Implementation

The implementation of a defense against the replay attack has to integrate directly into the AN.ON mix and hence, just like the AN.ON mix, is written in C++. It generally follows the solution suggested in the evaluation above, however there is a difference that results from peculiarities of the AN.ON system.

The mix packets of the AN.ON system contain signaling information in their headers which is not part of the theoretical mix concept. An attacker could introduce a slight change into these headers and thereby cause the packet to have a different fingerprint when it is calculated upon arrival. The solution to this problem is to calculate the fingerprint of outgoing instead of incoming packets. For the channel-open message, this incurs the additional overhead of having to perform an asymmetrical cryptographic operation.

However, once this is done, the cost of calculating the fingerprint can be saved, because the channel-open message already contains a perfect identifier for itself and the channel: the symmetrical key. Instead of the 64-bit value upon which the evaluation above is based, the symmetrical key is 128 bit wide, thereby doubling the amount of memory needed for the hash table. Still, this option seems adequate, as on the expense of memory consumption the security of the entire system is increased.

Chapter 5

Conclusion

In the first chapter, this paper gave a short introduction to the concepts of anonymity and unobservability along with examples for the importance of these concepts in our society. It then went on to establish the concept of Chaumian mixes, their protection goals and the attacker against whom they are meant to provide protection. It ended with explaining the differences between traditional and channel-based mixes and the limitations of mix-based systems.

The second chapter presented the AN.ON – Anonymity.Online – project, as an example of a real-world implementation of a channel-based mix architecture, which of course is heavily adapted to achieve the goal of providing a solution for real-time anonymous web access.

Now that the background was established, the third chapter gave an overview of possible attacks against anonymizing services and ways to classify them. In this context it introduced the replay attack, explained the motivation for it and detailed how it can be used to attack both traditional and channel-based mixes. Finally, an implementation of the replay attack was briefly discussed.

The fourth chapter first listed the requirements for a successful defense against a replay attack and then went on to evaluate several approaches: Both timestamps and changing the public key of the mix were discussed as possibilities to limit the validity of messages on one side. On the other side, in order to recognize previously sent messages, both the possibilities to efficiently store fingerprints of messages in the mix and to tag the messages with tickets were explored. These approaches were then evaluated according to the requirements specified before and an optimal solution was suggested. Finally, a brief overview over the details of the implementation of the proposed solution was given.

Appendix A

Classes of an AN.ON Mix

CAASymCipher: Methods for de- and encryption of asymmetrically encrypted data using RSA. Also generates and handles the key pair.

CABase64: De- and encodes data using the Base64 algorithm.

CACacheLoadBalancing: Performs load balancing of the cache proxies after the last mix.

CACertificate: Represents the certificate used by the mixes to identify each other.

CACertStore: Stores multiple of these certificates.

CACmdLnOptions: Parses and stores the command line options and data from the XML configuration file. Serves as central store for configuration information.

CAConditionVariable: A synchronization mechanism using pthreads that extends the functionality of CAMutex.

CADatabase: A database which is currently not used by the mix.

CADatagramSocket: A socket implementation.

CAFirstMix: Extends CAMix to act as the first mix in a cascade. Establishes a connection to the second mix and allows for JAP to connect.

CAFirstMixChannelList: Keeps track of the active channels of the first mix.

CAInfoService: Communicates statistical information to the InfoService.

CAIPList: Keeps track of connections to the first mix preventing DoS attacks.

CALastMix: Extends CAMix to act as the last mix in a cascade.

CALastMixChannelList: Keeps track of the active channels of the last mix.

CALocalProxy: Additional mode of operation that behaves like a local proxy (just like JAP).

CAMiddleMix: Extends CAMix to act as the middle mix in a cascade. Connects to the previous middle mix and to the cache proxy.

CAMiddleMixChannelList: Keeps track of the active channels of the middle mix.

CAMix: Common base class of first, middle and last mix and the local proxy.

CAMsg: A singleton class that is used for debugging or informational output of the program.

CAMutex: A synchronization mechanism using pthreads.

CAMuxSocket: The multiplexing / demultiplexing socket that allows for packets from different channels to travel over one TCP connection.

CAPayment: Access to a payment database.

CAQueue: A FIFO queue that allows blocking access.

CASemaphore: Another synchronization mechanism.

CASignature: DSA signature with methods to sign XML data.

CASingleSocketGroup: Allows for polling of a single socket.

CASocket: The socket implementation that is actually used to transmit packets between the mixes.

CASocketAddr: Interface for socket addresses.

CASocketAddrINet: Implements CASockerAddr for a dotted-decimal Internet address.

CASocketAddrUnix: Implements CASockerAddr for a Unix socket.

CASocketGroup: A collection of sockets that allows to wait for input.

CASocketList: Stores the sockets and their respective channels for CAMuxSocket.

CASymCipher: Methods for de- and encryption of symmetrically encrypted data using AES.

CAThread: A thread implementation using pthreads.

CAUtil: Utility functions.

StdAfx: Definitions and includes for the entire project.

proxytest: The main program.

Bibliography

[AES] Announcing the Advanced Encryption Standard (AES) *(Federal Information Processing Standards Publication 197, 2001)*

[ANONdoc] Mixe for Privacy and Anonymity in the Internet Documentation *(*http://anon.inf.tu-dresden.de/develop/doc/mix/*, 2003)*

[BeFeKö00] Oliver Berthold, Hannes Federrath, Marit Köhntopp – Project "Anonymity and Unobservability in the Internet" *(Workshop on Freedom and Privacy by Design / CFP2000, 2000)*

[BeFeKö01] Oliver Berthold, Hannes Federath, Stefan Köpsell – Web MIXes: A System for Anonymous and Unobservable Internet Access *(Designing Privacy Enhancing Technologies: Proceedings of International Workshop on Design Issues in Anonymity and Unobservability, 2001)*

[CE99] Council of Europe, Committee of Ministers – Recommendation No R (99) 5: Guidelines for the Protection of Individuals with Regard to the Collection and Processing of Personal Data on Information Highways *(1999)*

[Ch81] David L. Chaum – Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms *(Communications of the ACM 24/2, 1981)*

[GG] Grundgesetz für die Bundesrepublik Deutschland in der Fassung vom 26.07.2002 *(Bundesgesetzblatt Teil I 2002, page 2863)*

[Kn98] Donald E. Knuth – The Art of Computer Programming: Sorting and Searching *(Addison-Wesley, 1998)*

[La01] Heinrich Langos – Schutz der Anonymität gegen Schnittmengenangriffe durch Mix-Kaskaden *(Technische Universität Dresden, 2001)*

[No00] Inge Margaretha van der Nol – Datenbank in Mix-Systemen *(Technische Universität Dresden, 2000)*

[Be92] Perrin Beatty, Ministry of Department of Communications, Canada – Privacy Protection in Telecommunications *(Department of Communications, 1992)*

[Pf00] Andreas Pfitzmann – Datensicherheit und Kryptographie *(Technische Universität Dresden, 2000)*

[PfPfWa89] Andreas Pfitzmann, Birgit Pfitzmann, Michael Waidner – Telefon-MIXe *(Datenschutz und Datensicherung, 1989)*

[Ra01] Jean-Françoise Raymond – Traffic Analysis: Protocols, Attacks, Design Issues and Open Problems *(Designing Privacy Enhancing Technologies: Proceedings of International Workshop on Design Issues in Anonymity and Unobservability, 2001)*

[Re03] Reporters Sans Frontières – Germany: Annual Report 2003 *(*http://www.rsf.org/article.php3?id_article=6517*, 2003)*

[Ri92] R. L. Rivest – The MD5 Message-Digest Algorithm *(Network Working Group: RFC 1321, 1992)*

[RiShAd77] R. L. Rivest, A. Shamir, L. Adlemann – A Method for Obtaining Digital Signatures and Public-Key Cryptosystems *(Communications of the ACM 21/2, 1977)*

[SeDa02] Andrei Serjantov, George Danezis – Towards an Information Theoretic Metric of Anonymity *(Workshop on Privacy Enhancing Technologies, 2002)*

[SHA1] Secure Hash Algorithm, SHA-1 *(Federal Information Processing Standards Publication 180-1, 1995)*

[TDDSG] Gesetz über den Datenschutz bei Telediensten (Teledienstedatenschutzgesetz - TDDSG) vom 22.07.1997 *(Bundesgesetzblaztt Teil I 1997, page 1871 ff.)*

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig angefertigt und ohne fremde Hilfe verfasst habe, keine außer den von mir angegebenen Hilfsmitteln und Quellen dazu verwendet habe und die den benutzten Werken inhaltlich oder wörtlich entnommenen Stellen als solche kenntlich gemacht habe.

Berlin, den 15. August 2003

Georg Wittenburg