Dummy Traffic against Long Term Intersection Attacks

Oliver Berthold and Heinrich Langos

Dresden University of Technology, Germany
{ob2,hl6}@inf.tu-dresden.de

Abstract. In this paper we propose a method to prevent so called "intersection attacks" on anonymity services. Intersection attacks are possible if not all users of such a service are active all the time and part of the transfered messages are linkable. Especially in real systems, the group of users (anonymity set) will change over time due to online and off-line periods.
Our proposed solution is to send pregenerated dummy messages to the communication partner (e.g. the web server), during the user's off-line periods.
For a detailed description of our method we assume a cascade of Chaumian MIXes as anonymity service and respect and fulfill the MIX attacker model.

1 Introduction

1.1 Overview

Starting with a description of the attacker model and the nature of intersection attacks in section 1, we present the problem and motivation.

Section 2 lists the basic approaches to prevent these attacks and from section 2.2 on we concentrate on the dummy traffic approach and different ways to deliver dummy messages. A summary of the requirements for a system that resists a global attacker concludes this section.

In section 3 we present a system that fulfills this requirements but does so with a high demand on resources and only for a limited set of applications.

Section 4 summarizes the results and points out areas where further research is needed.

1.2 Motivation

Anonymity on the Internet has been an issue ever since its focus shifted from scientific to social and commercial uses. Without special precautions most users and their actions on the Internet can be observed. To address this threat for privacy, various anonymizing services have been built, but especially real time services are vulnerable to traffic analysis attacks[8].

In this paper we present methods of sending and storing dummy traffic to prevent such attacks.

© Springer-Verlag Berlin Heidelberg 2003

1.3 MIXes

For the most part we will consider the anonymizing service a black box. A set of input messages is received by that black box and transformed to a set of output messages. Each incoming message will be anonymous among the set of outgoing messages.

For particular properties, like a determined delay of messages, we will assume a cascade of Chaumian Batch MIXes [4] or Web MIXes [2].

1.4 Terms and Definitions

A sender or receiver is said to be *active* if he sent or received a message during a period of time.

For each observed message an attacker may construct sets of possible senders and/or recipients, excluding users who at that time where off-line or idle or otherwise unable to send or receive that message. This sets will be called *anonymity sets*.

In lack of a better word, *session* will be used to describe a series of messages that belong to a communication relation. Contrary to the usage in network literature, this relation may continue over several online/off-line periods.

Sessions are considered active if at least one message, that will be linked to that session was transmitted during a period of time.

A Session, for instance could describe the messages that somebody sends to post messages to a political forum under a pseudonym.

A different example would be the requests that a user makes to check a stock market service. Provided that the requests reflect that users portfolio in a sufficiently unique way, there doesn't need to be a pseudonym to recognize him.

1.5 Attacker Model for Intersection Attacks

When talking about the security of MIXes, often the global attacker or adversary is presumed. To start an intersection attack, a somewhat weaker attacker is sufficient, but there are certain other conditions that have to be fulfilled for an intersection attack.

Figure 1 shows his essential areas of control. The darker areas are the minimum that an attacker needs to watch, while the lighter areas show the area that a global attacker can watch and manipulate.

- The attacker is able to watch all user lines.
  If the attacker could access information that allow a mapping of IP-addresses to users, he just needed to watch the entry point of the cascade instead of all user lines[1].
- He is also able to watch all server lines.
  In a MIX cascade there is only one exit line. Watching that line is sufficient to see all requests that are sent to a server through the cascade. Especially when no end-to-end encryption is used.

[1] Presuming no direct Peer-to-Peer connections are made. Like in Crowds[9] networks.

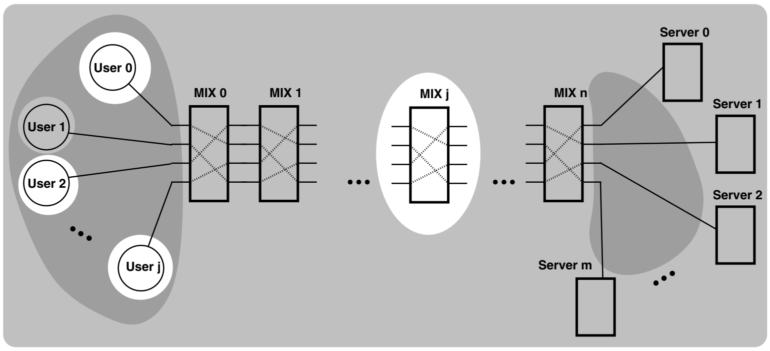

Fig. 1. Attacker model for intersection attacks

- He controls up to $m-1$ MIXes and a small number of users.
  Though the control over MIXes is not technically needed for an intersection attack, it is important to keep this in mind because the counter measures should withstand the global attacker model.
- The attacker knows how long it takes for a message to go through all the MIXes, so he can infer when a message that leaves the cascade has been sent.
  This is certainly true for batch MIXes and Web MIXes. These parameters should even be known to every user. At the very least the attacker could send messages through the system himself, to measure the current delay.
- Messages belonging to a certain session have to have some property that allows an attacker to link them to each other.
  This is of course the most fundamental precondition. Otherwise each message would have an independent anonymity set, consisting of the messages that where processes by the MIX cascade in the same time period.

1.6 Intersection Attack

Intersection attacks can be launched on two different scales. They differ in the time an attacker may need to link a series of messages to a user and the criterion used to decide if a user is possibly the sender of a message.

By watching the lines between each user and the cascade's first MIX, an attacker can see when the user is active and when he is idle or off-line. The attacker also watches all messages M that leave the cascade. Some of these can be linked to previous messages and recognized as belonging to a session s. This reveals when a session is active or inactive.

1.7 Short Term Attacks

Consulting the information he gathered from watching the activity on the users' lines, he can create a set of possible senders U_M for each message. Taking all messages that belong to a session s and intersecting the sets of possible senders will remove users who where not active during the transmission of all messages. This shrinks the set of suspects S possibly until there is only one user left[2].

$$S = \bigcap_{i=0}^{n} U_{M_i^s}$$

If users are not active all of the time, the probability for any non-involved user to be sending messages whenever a message for s is sent, rapidly decreases over time. Even if each user is active 90% of the time, the anonymity set will be down to roughly a third after just 10 messages of s.

While the mathematical concept of intersections is nice for explaining this attack, it does not reflect the reality. An attacker who used plain intersection on the sets of possible senders could be fooled into removing the real user by a single message that he erroneously linked to a session while the session's real user was inactive.

1.8 Long Term Attacks

Long term attacks are possible if messages can be linked across more than one online period. In long term attacks the criterion for including a user in the anonymity set of a message isn't user activity anymore, but session activity. All online users are added to the set of possible senders. In this case an attacker doesn't need to accurately measure latencies and he doesn't need to watch activity closely.

The user sets of those periods will probably be very different and therefore the intersection attack will become more likely to succeed.

2 Counter Measures

In this section we present the two basic approaches to preventing intersection attacks.

2.1 Preventing Linkability

The most effective way to prevent intersection attacks would be, to make messages unlinkable. Unfortunately this is not that easy.

Linking by traffic data, can be prevented to a certain degree. Message sizes can be padded to a common multiple and larger messages can be split into smaller fragments but already at this point the number of transfered packets that enter and leave the anonymizing service can be correlated by an attacker

[2] Recognizing a group instead of a single user may be of interest, too. Especially from the viewpoint of industrial espionage. But for the purpose of perspicuity, we will stick to the one-user-scenario.

who watches all input and output lines. Send and receive times are equalized by collecting messages to form a batch or by delaying each message for a user selected time. This works for email, but for low latency services like web browsing there are rigorous limits to the delay that can be added. Longer delays will not be accepted by most users and therefore the anonymity group will shrink [1].

At the first glance, content based linking seems easier to prevent. Simple End-to-End encryption would protect messages from eavesdroppers. If however the communication partner is not to be trusted or doesn't even provide means of encrypted communication, the situation changes dramatically.

An example of this is again web browsing. There are countless ways by which requests to web servers can be linked. And even if all traffic to and from the web server was encrypted, this does not protect from the server's operator. Appendix C lists some of the methods that can be used to track users on the WWW.

2.2 Simulating Activity

If there is no way to make messages unlinkable, the obvious solution is to make all users potential senders or receivers. This can be done by sending dummy traffic.

Short Term: Against short term attacks it is sufficient to transmit dummy messages between the users and the anonymity service. The dummy traffic conceals periods of inactivity by contributing messages when no real traffic is to be sent. This ensures that each user is in the anonymity set of each message for an active session[3].

If only one MIX is trustworthy, we have to assume the worst case, that it is the last one. Hence the dummy traffic has to be sent through the whole cascade and can safely be dropped by the last MIX. It does no harm if the dummy traffic passes through the decent MIX, and is later discovered by the attacker to be just a dummy, because the trustworthy MIX will stop any attempt to trace the dummy back to its origin.

Long Term: Long term attacks can't be overcome by dummy traffic to the MIX cascade. The attacker can afford to include all online users of the anonymity service and it is very hard to hide the fact of being online and using an anonymity service.

Therefore, instead of simulating constant user activity, the session will have to appear active all the time. Even while the session's real user is off-line.

The dummy traffic mentioned from here on, will have to look like normal user-generated traffic and consequently will have to be sent, like normal traffic, all the way to the server.

[3] But even if the number of users stays more or less constant, the anonymity set for the whole session gradually shrinks. As the session continues, some users will leave the system and others will join it. These users can't be responsible for an ongoing session.

The generation of such dummies will be discussed later. For now is is assumed that dummies are generated by the real user to be sent later by somebody else, while the real user is not online.

2.3 Building a Global Anonymity Set

Making all decent users appear equally likely, to be owners of a session is the ultimate goal in terms of the anonymity set. For real-time services like HTTP, this requires constant activity of all users, and therefore constant connectivity. Otherwise the first message that belongs to a session s, also specifies the starting anonymity set M_0^s, consisting of all users who are currently online and active. This will give the attacker a substantially smaller set of suspects to start with.

If some restrictions on the real-time property can be accepted, there is a way to expand the limits of that initial anonymity set M_0^s. Once, this is achieved, other measures can be used to keep the anonymity set at that size.

To overcome the initial limit for the anonymity set, the traffic for a session would have to start *before* the real user even came online and used the session for the first time. Generating a series of plausible dummy requests without having visited a web site before, may look difficult, but HTTP-to-Email services like www4mail[4] or agora[5] could easily be adapted to use non-real-time channels like Mixmasters[7]. Using them to acquire a document from the WWW would be the first step to generate a set of dummy requests. These requests could be sent like normal dummies (as described below) and would serve as a fore-run to mask the real beginning of usage. The average fore-run period should be long enough for all users to be online at least once, so that each of them could have initiated the session.

2.4 Maintaining the Anonymity Set

An almost completely self reliant solutions could be, to send long running MIX messages, not unlike emails in Stop-and-Go MIXes[7]. But having the anonymizing service simply extended to also handle "slow" traffic however, would not be enough. The dummy would have to be indistinguishable from normal traffic, at least after it passed the one trustworthy MIX. In this situation, the only position where a MIX could offer protection to the nature of a dummy, would be, as the very last MIX in the cascade.

Also for Web MIXes and HTTP access, there needs to be a bidirectional connection, not just a one way delivery. It seems necessary to delegate the actual sending of the dummy to some other entity. Dummies therefore have to be transported from the real user to whomever is intended to send them later. If an attacker observes a dummy somewhere during that transfer, he will be able to recognize it when it is sent through the MIX cascade later. If the transfered dummy was already encrypted for sending, an attacker wouldn't learn anything

[4] `http://www4mail.org/`

[5] `http://www.eng.dmu.ac.uk/Agora/Help.txt`

about its plain text but he could recognize it when it is sent to the MIX cascade and by whom. The attacker could deduce that this user was currently idle and remove him from the anonymity set of each message that was sent at the same time. Therefore dummies should not be transported in plain text and even encrypted dummies need to be hidden from an attacker if possible.

The attacker model puts users in a grey area. A few of them may be in collusion with the attacker while most of them are trustworthy. In respect to this, it will be considered save, if a single user, or a very small group of users, learns about the the nature or even the content of a dummy message. It will be considered insecure however, if a larger group, e.g. the whole anonymity set of a message, could find out that a message was just a dummy.

Dummy Sending Service. The first simple approach is a dummy sending service, that is informed by a user who wants to protect his session s. That dummy service will have to be provided with enough information to produce, or just forward, dummy messages for a session.

To avoid flooding of that service, without identification of its users, blindly signed tickets[5] for dummy traffic can be issued by that service. Also, dummy sending could be paid for in advance with an anonymous payment system.

Any communication with the dummy service, will have to be asynchronous. Otherwise the anonymity service itself will know, when the real user of s is online, and will therefore be able to start an intersection attack by itself.

For services with a small number of randomly distributed requests per day, this service can protect the anonymity. But only as long as the number of requests that the real user adds to a session during his online period is not statistically significant.

For services with more traffic, there has to be a way to regulate the flow of dummies when the real user wants to use the session. Otherwise the real user's messages that belong to s will just add to the dummy traffic and clearly mark the online period of its sender.

Even a widely variable amount of dummy traffic will not help much. Given enough traffic/time pairs, the attacker will always be able to find out the real user. Comparing the average traffic of s during each users online and off-line periods will reveal the source of that additional traffic. The user with the highest difference in those averages is the probably the real user of s.

Telling the service, to pause sending dummies for s is highly dangerous. Even if communication to the service is totally anonymous, it will deliver half the data that is necessary for an intersection attack directly to that service.

Sliding Storage. To avoid trusting a single server with the storage of dummies and to improve the chances to revoke a dummy, a slow message forwarding system was designed. In this system a message doesn't stay at one server but is repeatedly handed around.

It consists of several storage nodes S that roughly resemble a network of Stop-and-Go MIXes. In contrast to MIXes, who push messages to the next MIX, storage nodes show a pull-behavior. Messages are requested by the next storage node instead of being sent to it automatically after a specified time. Those requests are only answered if the asking party can prove some knowledge of the message, but without revealing the identity of the requesting party. To achieve this, communication between the nodes could be run over a normal MIX cascade.

The way of each message through the set of storage nodes $S_{0\ldots n}$ is chosen by its creator and the message is encoded with the public key of each node along the way. Like in a normal MIX network the message is encrypted for the last node first.

Every node S_i along the planed route gets a short note with the information that it needs to request the message from its predecessor S_{i-1}. This includes the time at which to request the message as well as the message's hash value, to proof a request's validity, and of course the address of the predecessor. The delivery of these notes could be left to some asynchronous transport system to ensure that no node gets the request information too far ahead of its time.

While a message is stored at S_j, the creator of that message has the chance to intercept it before it reaches its goal. Knowing the current position and hash value for each message that he sent, he could act like he was node S_{j+1} and simply request it. This empowers the creator to revoke his own dummies without revealing his identity.

To avoid suspicion at S_j and S_{j+1}, repeated requests should be common as well as constructing request chains that are longer than the actual message path.

Though this system seems to be fairly save against attacks from outsiders and single nodes, there are problems when nodes collaborate.

Adjacent nodes along the path could inform each other about requests that they sent/received and by this, could detect the revoking of messages.

A malicious node could also refuse to delete an already requested message and hand it out if it was requested more than once. Though this could be detected by the user if he sent requests to nodes that his message should already have passed. Trustworthy nodes could also do this check by requesting a message more than once. But if such behavior goes unnoticed, the message will reach its goal. This will have an effect on the average number of messages as shown before.

If S_j and S_{j+1} were both corrupt, they could infer from their combined knowledge that the user intercepted a message. Hence, no adjacent nodes may be corrupt. This is of course a much stronger assumption than the original MIX model where only one trustworthy node is needed in the whole message path.

2.5 Requirements

Summarizing the results from examining these approaches here are the general requirements for a system that stores and sends dummy messages.

- Distributed trust.
  Assuming the global attacker model, the adversary controls most (in the extreme case all but one) server nodes and a small part of the client nodes. To build a trustworthy system with these components we have to distribute

our trust. This could be done using protocols that guarantee the security by involving at least one trustworthy node in all security-critical computations. A different approach is to limit the number of involved users and estimate the risk.

- Limited DoS capabilities.
  Denial of Service attacks can not be prevented when the cooperation of possibly corrupt nodes is needed to achieve a result. The best thing we can hope for is to expose the uncooperative nodes. On the other hand a node that intentionally doesn't answer is indistinguishable from a node that fell victim to an active attack. If cooperation is not an issue, the effect of DoS attacks on the system should be confined. Especially flooding attacks should be prevented, where the attacker tries to maximize his share of resources in order to simulate a larger anonymity group or to isolate other users.
- Traffic shaping.
  The traffic that an observer sees for a certain session should be independent of the users online periods. This does not only mean that dummy traffic should be sent during the users off-line period but also that the amount of traffic should not be higher during his online periods. A user therefore should be able to revoke his dummies or replace dummy messages with real messages without an attacker noticing it.

3 Limited Solution

We propose a system for transporting dummies that should fulfill the previously listed requirements.

It is limited to simple non-interactive information retrieval. Access to web pages via basic HTTP authentication is possible but any challenge-response based access control will fail[6].

The bandwidth demand is rather high as is the number of asymmetric cryptographic operation that have to be performed[7].

3.1 Passive Dummies

To handle the full variety of tracking methods, dummy requests will have to be more than just plain HTTP requests. For some tracking methods however, simple, even repeated, requests will be adequate. These passive dummies just contain the requested URL, and all HTTP headers, including cookies, web browser identification and every other header that the real user would have sent himself. If a dynamic page is requested via the HTTP “POST” method, the data that the real user would have submitted, also has to be part of the dummy.

Passive dummies could be automatically generated by the local proxy with very little help from the user by recording repeated requests and asking the user to approve their usage as dummies.

[6] This is no limitation of the transport system though, but a limitation of the trust that one is willing to put into an randomly selected user.

[7] Both can be reduced to a certain degree as shown in appendix B.

The dummy could consist of several prepared requests, meant to be transmitted one after another to emulate the automated loading of embedded images and objects. This is of course only possible if those responses are known in advance. Intermediate changes to the requested page may result in requests for images that are not embedded anymore but this is not that dangerous. Some browsers already use a cached version when re-requesting a document to also re-request embedded images even before the updated version of the document is returned.

Prefabricated Dummies. are dummies that are build and prepared for sending by the real user. This means they are already encrypted. Like the long running dummy messages they are appealing since they don't seem to need much faith in systems outside the users trusted region[8]. The contents of those dummies is only revealed after they have passed the cascade and thereby at least one trustworthy MIX.

The considerable advantage is that since the dummy sender doesn't need to react to the servers response, he doesn't need to understand it. So the returned data doesn't need to be intelligible to him and he doesn't learn what he requested at all.

3.2 One Way Dummy Database

The dummies are stored in a distributed database. It works similar to the message service from [6], but unlike there, the user requests only data from a single address and the data in each cell is scrambled. Only the XOR combined result of each cell's contents at the address l from all n database nodes will reveal the stored data. Instead of hiding the address that is accessed from the database servers, the data itself is hidden.

To make sure that no dummy is given to a decent user *and* a rogue, the database nodes are obligated to issue each cell's content only to one user. Due to the XOR scrambling it only takes one trustworthy database to enforce that rule.

Building data fragments for a system with n-nodes is done by generating $n-1$ blocks of random data and XOR-combining all those blocks and the real data, to form the last block.

Presuming the attacker's presence in most parts of the system it seems best to give the dummy to just one randomly selected user. If flooding attacks can be prevented the chances of giving it to a corrupt user then would be minimal.

Having the user request dummies, instead of sending them, makes more sense in terms of performance, though. The user (or his local proxy) knows more about his current bandwidth demands and wouldn't send dummies if he needed all his bandwidth for himself, anyway. Hence, instead of selecting the user randomly, the dummies are encoded so that they all look like random patterns until requested.

[8] Except for sending them at an appropriate time, which could be tested by submitting some dummies whose sending can be checked by the user.

Traffic Shaping. Each database node publishes a list of cell addresses and hash values computed from that cells contents. Normal users randomly select one of these addresses from that list and request the contents of that address from each database node. The real user can recognize his own dummy by the hash values of each fragment and request that dummy. This allows for "revoking" of one's own dummies and solves the problem that dummy traffic and real traffic add up during the real user's online period.

Preventing Flooding. Accessing the database is done through an anonymous channel, like a MIX cascade, to hide the depositing and withdrawing user from the database nodes and other observers.

Since all database nodes should give the fragments of one dummy to the same user, there has to be some pseudonym, by which the databases can recognize him. Using blindly signed tickets, that are only valid for one deposition or withdrawal, makes the actions unlinkable and prevents correlation between depositing and withdrawing users.

A ticket contains a public key of an asymmetric cryptographic system that will be used to verify the ticket holders signature in a later message[9].

To create such a ticket the user first generates a normal asymmetric key pair. The public key is extended by some redundancy, blinded and sent to each node. The signed replies will be unblinded and the public key with the redundancy and all signatures are combined to form the ticket.

If this ticket is used, to sign a request later, the database node tests its own signature of that ticket and the tickets signature of the whole request.

Leaving it to the user to request dummies, might open an opportunity for corrupt users, to drain or flood the dummy database. But if each user is only given a limited amount of tickets, the few corrupt users are not able to do significant harm. Concentrating on dummies by a certain user or dummies addressed to a certain service is not possible.

Malicious users could try to render up to n dummies useless with one request, by requesting the content of different cells for different nodes. To avoid that, the database nodes will have to inform each other of requested addresses.

Authentication. All communication to the user has to be signed by the database nodes in order to verify the message's integrity and to empower the user to prove misbehavior of database nodes.

Furthermore all database nodes should continuously publish their log files so that their work can be reviewed by a third party in case of doubt. These log files will not contain any actual dummy data, but its hash values.

All deposit and withdrawal messages that are sent to the database nodes contain a ticket and will be signed by the secret key that belongs to the used ticket.

[9] If an symmetric cryptographic system was used, each database node could use the key to impersonate the user and deceive the other nodes.

When depositing a dummy for example, the message to the server is encrypted with the server's public key and signed by the key included in the ticket. Upon receiving a message, the server tests his own signature of the ticket and then the signature of the whole message, made with the ticket's secret key. Then the server decrypts the message's content.

The server could sign its answer and send it back to the client, but as with any other handshake, there is no finite protocol that will totally satisfy the needs of both parties. Publishing the (signed) dummy list, however can be seen as a public statement of commitment. Therefore, at this point the server will include the new dummy's cell address and hash value in its list. The client can check this, by requesting the servers dummy list (without any reference to that certain dummy or ticket).

If a node refuses to accept a ticket, it has to show that this ticket was already used. To prove that, the node will have to keep all user messages as long as the used ticket would have been valid.

Delayed Dummy Release. Assuming that only the real user of s deposits new dummies for s while he is online and new dummies are made available right away. As soon as he goes off-line and stops depositing dummies, the chances of randomly picking a dummy for s will decrease over time. This is because the number of available dummies will shrink with every one that is already sent, while other users deposit their dummies in the newly available cells.

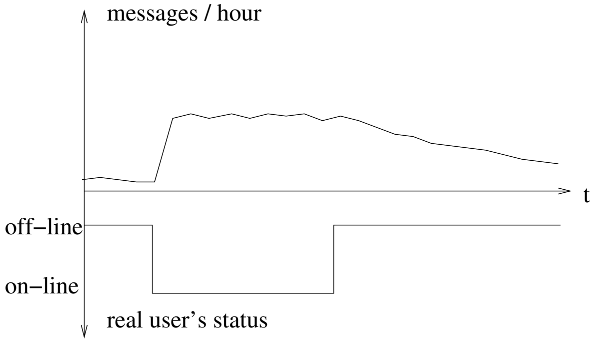

Fig. 2. Changes in dummy traffic for s.

As Fig. 2 shows, re-stocking the dummy supply for s will show up as a sharp rise in the number of requests for s and could be used to select the online users of such periods for an intersection attack.

There are several possible solutions. The deposition of dummies could be made asynchronously with dummies that take more or less time to get to the database nodes. But this would disable the direct control mechanisms that the public dummy list provides. A server could deny having received a message and the user would have no proof against it. The usage of receipts (sent to an anonymous return address) instead of public lists could also mend this, but would add even more delay.

The database nodes themselves could take care of dispersing the dummy release over time. But they would have to be instructed about the time frame for each dummy by its creator, since they themself's don't know which session a certain dummy belongs to.

Instead of allowing an individual time frame for each dummy, it seems more efficient to globally define time frames and to only allow deposition of dummies for future time frames.

Evaluation.

– *Performance:*

Table 1 shows the cost in messages and additional cryptographic operations. Direct messages are assumed to already be authenticated and indirect messages are assumed to be encrypted with the receivers public key and sent through a MIX cascade or some other form of anonymizing device. Sending and receiving are counted likewise, even though the cost of processing can be very different, especially for indirect messages.
The last two columns contain the number of additional asymmetric cryptographic operations (key generation, signing, testing, encryption, decryption) that are needed, apart from the ones for handling of messages. Less demanding operations, like splitting dummies, blinding tickets and computing/comparing hashes, are not included.

- Requests for depositing and withdrawing are sent to each server separately. To stop users from requesting different cells on different servers, the servers inform each other of requests. The n^2 messages for that can easily be reduced to $2n - 1$.
- The requests for signing a ticket are made directly to each server, as is the request for the signed lists, to check deposition.

Table 1. Resource demands

	direct		indirect		crypto. operations	
action	user	servers	user	servers	user	servers
request ticket	$2n$	$2n$	-	-	$1+n$	n
deposit dummy	-	-	n	n	n	$2n+n^2$
check deposition	$2n$	$2n$	-	-	n	n
withdraw dummy	-	-	$2n$	$2n$	$2n$	$3n+n^2$

Depending on the average frequency of dummy messages, the number of messages for storing and retrieving 24h of dummy traffic for a session is shown in Fig. 3. The number of database nodes is assumed to be 3.

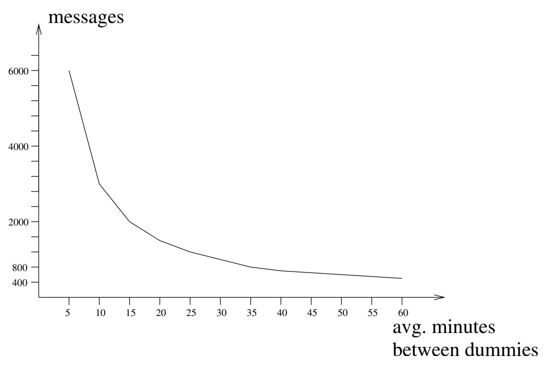

Fig. 3. User messages to protect s for 24 hours.

– *Security:*

The system seems fairly save against passive attacks by nodes and small numbers of users. Active attacks by nodes are limited to denial of service. Delivering false data without risking detection is not possible unless a hash collision can be found.
The usage of asymmetric cryptography in most parts of the system allows for delegation of messages in order to expose uncooperative nodes. If for example, a node refuses to deliver the content of a cell, the user could send his request to a different node with an appeal to act as a proxy. The proxy node could test the validity of the request by checking the signature and consulting its own log files for a previous request for that cell. After establishing the validity, the proxy would forward the request and, if the other node remained uncooperative, ask it for some proof. A proof could be either the request by some other user for the same cell, or a request by the same ticket holder for a different cell.

4 Conclusions

In this paper first the presuppositions of intersection attacks were presented and an specialized attacker model was introduced. Several methods to prevent the result of this attack have been examined.

Among these methods “Sending dummy traffic to the server” was examined in detail and the results of these analysis have been incorporated in the design of a limited solution. This solution provides strong anonymity even in the presence of a global attacker. A distributed database is used to store pregenerated dummies, where a single trustworthy server is able to uphold anonymity.

Another single server could stop the system from working, but the failing server can be identified by trustworthy nodes and a user without revealing the user’s identity.

Even though no general solution was found, the discovered design principles, like “involving every node in decisions”, and “extensive public log files” seem to be important guide lines for further work.

Unfortunately the solution is rather costly in terms of bandwidth and computational resources. But other examined methods don't provide enough security, given the supposed attacker model or are too costly in terms of bandwidth.

Relevant restrictions of other examined methods are:

– Transferring dummies directly to each user who is supposed to form the anonymity set for one's session, is too costly or even impossible if a large anonymity set is needed.
– Delivering dummies through an additional "slow" MIX channel is not possible for bidirectional traffic.
– Leaving the sending or distribution of dummies to a central service is too risky since it introduces a single point of thrust.
– The "sliding storage" network leads in the right direction by distributing trust to several nodes. But the involvement of each node is limited and so is the protection a trustworthy node can provide.

All but the last method suffer from the lack of revocability of dummies.

For weaker adversaries the solutions examined along the way, certainly provide security. But switching to a weaker attacker model makes the usage of MIXes almost obsolete.

Further research should examine the possibilities to reduce the computational and I/O load on the server nodes as well as different kinds of dummy messages (like the one mentioned in Appendix D). Especially an extension to services that use challenge response protocols for authentication would be interesting.

References

1. Adam Back, Ulf Moller, and Anton Stiglic. Traffic analysis attacks and trade-offs in anonymity providing systems. In *Information Hiding*, pages 245–257, 2001.
2. Oliver Berthold, Hannes Federrath, and Stefan Köpsell. Web MIXes: A system for anonymous and unobservable Internet access. In Hannes Federrath, editor, *Designing Privacy Enhancing Technologies*, volume 2009 of *Lecture Notes in Computer Science*, pages 115–129, Berkeley, CA, USA, July 2000. Springer-Verlag, Berlin Germany.
3. O. Berthold, S. Clauß, S. Köpsell, and A. Pfitzmann. Efficiency improvements of the private message service. In Information Hiding. 4th International Workshop, IHW 2001, Pittsburgh, PA, USA, April 25-27, 2001. Proceedings, ISBN: 3540427333, LNCS 2137, Springer, Berlin, 2001.
4. David L. Chaum. Untraceable Electronic Mail, Return addresses, and Digital Pseudonyms. *Communications of the ACM*, 24(2), February 1981. Also availabel at `http://world.std.com/~franl/crypto/chaum-acm-1981.html`.
5. David L. Chaum. Blind signature systems. Advances in Cryptology – CRYPTO '83, International Association for Cryptologic Research, North-Holland/Elsevier, 1983.
6. David A. Cooper and Kenneth Birman. Preserving privacy in a network of mobile computers. In *IEEE Symposium on Security and Privacy*, pages 26–38, May 1995. Also available at `http://cs-tr.cs.cornell.edu/Dienst/UI/1.0/Display/ncstrl.cornell/TR95-1\%490`.

7. D. Kesdogan, J. Egner, and R. Bueschkes. Stop-and-Go-MIXes providing probabilistic anonymity in an open system. *Lecture Notes in Computer Science*, 1525:83–98, 1998.
8. Jean-Francois Raymond. Traffic analysis: Protocols, attacks, design issues and open problems.
9. Michael K. Reiter and Aviel D. Rubin. Crowds: anonymity for Web transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, 1998.

A Example Dummy Database

To illustrate the process, we walk through a small example with just one user and two servers A and B.

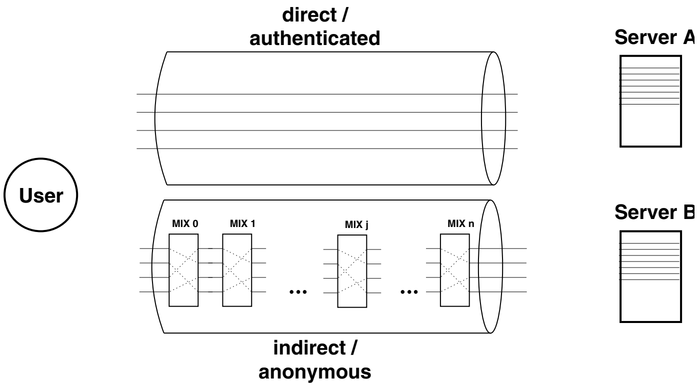

Fig. 4. Small example system.

Request Ticket. After generating a key pair s_t, t_t the user adds some redundancy to the public key, blinds it with b and sends this signed ticket request message for a certain time frame to each server.

$$\boxed{b(t_t, r), \text{frame}, s_{\text{user}}(b(t_t, r), \text{frame})}$$

Each server checks if the user is allowed any more tickets for that certain time frame, and if so, he signs the blinded request with the key for that time frame and returns it to the user.

$$\boxed{b(t_t, r), s_{A_{\text{frame}}}(b(t_t, r)), \text{frame}, s(b(t_t, r), s_{A_{\text{frame}}}(b(t_t, r)), \text{frame})}$$
$$\boxed{b(t_t, r), s_{B_{\text{frame}}}(b(t_t, r)), \text{frame}, s(b(t_t, r), s_{B_{\text{frame}}}(b(t_t, r)), \text{frame})}$$

The user unblinds the key with b^{-1} and is now in possession of a valid ticket.

$$\boxed{s_t, (t_t, r), s_{A_{frame}}(t_t, r), s_{B_{frame}}(t_t, r)}$$

Deposit Dummy. The user generates the fragments F_A, F_B as described above and additionally computes hash values of them. The fragment is encrypted with the corresponding servers public key and is concatenated with the address a, and the ticket. This whole message is signed with the ticket's secret key and sent to the server through an anonymous channel.

$$\boxed{\begin{array}{l} c_A(F_A), a, (t_t, r), s_{A_{\text{frame}}}(t_t, r), s_t(c_A(F_A), a, (t_t, r), s_{A_{\text{frame}}}(t_t, r)) \\ \hline c_B(F_B), a, (t_t, r), s_{B_{\text{frame}}}(t_t, r), s_t(c_B(F_B), a, (t_t, r), s_{B_{\text{frame}}}(t_t, r)) \end{array}}$$

The servers check if the cell at the given address is available, test their own signature of the ticket, test the ticket's signature over the whole message, and finally send each other confirmation messages to make sure that the same address was selected by the same ticket holder at both servers.

If all went well, they decrypt the fragment, store it at the given address and compute the fragment's hash value to update their public list.

Request List. The user simply sends a request for the list to each server and gets a signed reply, that contains the list of addresses and hash values. Comparing the hash values to the ones that he computed himself, he could now make sure that his own dummies have been. Of course the user would test the servers signature of the list as well.

Withdraw Dummy. With another key pair c_t, d_t the user requests a ticket and eventually receives it.

$$\boxed{(c_t, r), s_{A_{\text{frame}'}}(c_t, r), s_{B_{\text{frame}'}}(c_t, r)}$$

After reading the public list, the user decides to to withdraw the dummy at the address a'. He builds withdrawal messages that contain the address and the ticket and are signed by the tickets secret key d_t. These messages are sent to the servers through bidirectional anonymous channels.

$$\boxed{\begin{array}{l} a', (c_t, r), s_{A_{\text{frame}'}}(c_t, r), d_t(a', (c_t, r), s_{A_{\text{frame}'}}(c_t, r)) \\ \hline a', (c_t, r), s_{B_{\text{frame}'}}(c_t, r), d_t(a', (c_t, r), s_{B_{\text{frame}'}}(c_t, r)) \end{array}}$$

The servers check if there is data available at the given address, test their own signature of the ticket, test the ticket's signature over the whole message, and again send each other confirmation messages to make sure that the same address was selected by the same ticket holder at both servers.

If all went well, they take the fragment, encrypt it with the key c_t, sign it by their own public key and return it through the anonymous channel to the ticket holder.

$$\boxed{c_t(F'_A), a', s_A(c_t(F'_A), a')}$$

For each returned message, the user tests the servers signature and decrypts the fragment. After making sure that the hash value for address a' from the server's list, matches the actual data, he combines the fragments and obtains the real data.

B Optimizations

Servers could be trusted to forward messages on behalf of the user when possible. If errors occurred in such a system, the user will simply fall back to connecting to each server individually to find out what went wrong.

For some special cases, this slightly increases the number of messages for some servers but, it reduces the number that a user needs to send and receive.

- The request for signing a ticket can be forwarded by one server to the next and sent back to the user by the last one with the collected signatures. Since the request is rather small and the signatures don't grow the message that much, it is considered better to forward the whole request instead of contacting each server separately.
- When withdrawing a dummy, one server could be asked to collect all fragments and return the XOR combined result. To stop that server from seeing the actual dummy data, each node would not encrypt the fragment with the ticket's public key, but combine the fragment with a pseudo-random stream of the same length and return that masked fragment and the initialization parameters for the random number generator in an encrypted block.
  The collecting server would XOR combine all fragments and append the encrypted blocks. Instead of returning n times the fragment size, only one fragment is returned and n blocks of the asymmetric crypto system.[3]
  Upon receiving this message the user decrypts the n asymmetric blocks and extracts the initialization values for the random number generators. By overlaying all n pseudo random streams the real data is restored.
  This optimization breaks the provability of misbehaving of nodes but if errors occur, it is possible to fall back, to requesting fragments from each server individually
- A request to deposit a dummy could be sent to one randomly selected server, containing all fragments. Each of those encrypted with the respective server's public key. The selected server would split this large message into smaller parts, and forward those parts to the other servers.
  These smaller parts would have arrived from an unknown source at the servers anyway, and are verified by the layers of cryptography that encloses their content, not by their apparent sender.

C Tracking Methods on the WWW

Under normal circumstances the task of an anonymizing service is to prohibit the linking of requests by removing as much "dangerous" information from them as possible. For some applications, however it is essential that messages can be linked for the duration of a session or even longer. Here are some examples of tracking methods and sources of information that can be used to link requests over a long time.

- The requested **URL** may contain an ID in several places. The host name, path, filename and as an CGI parameter behind the filename. The ID is carried over from request to request, either by being included in every link in the HTML document, or by being added to the URL by the browser when requesting some relative URL. Since these IDs are hardly ever human readable they only tend to stick to a user if he puts a URL with an embedded ID in his bookmarks.

- A **Cookie** is an HTTP header that contains a name/value pair. It is initially sent by the server to the browser and is sent back to the server with every further request. It can contain anything from a session-ID to user name and password for that server.
- The HTTP **authentication header** is of course designed to authenticate, and in most cases identify a user. As a tracking method it is very reliable but hardly used since, unlike cookies these name and password are not saved to disk. Therefore it requires the user to enter a user name and a password again and again.
- **Repeated requests** of the same static URL can also be linked. Even if no special tracking methods are used. Just the fact that this URL is rarely requested by anybody else, is sufficient. The same is true for repeated requests to dynamic pages, but dynamic pages generally tend to be more frequently requested. A news site for example will often have a few dynamic entry pages with the headlines, that are very often requested, and many static pages that contain the actual articles and are requested comparatively rarely.

D Active Dummies

A different approach to dummy traffic is, not giving out complete requests, but instructions on how to build dummy requests, and, even more important, what to do with the responses.

These instruction could be quite simple, like a script that loads an HTML document and follows three randomly selected links from that document. They could include cookies that should be sent along with requests and instructions on how to handle received cookies and other received data.

Even HTTP authentication data could be part of these instructions. Although, giving this kind of data to some unknown user is only advisable if no damage can be caused by that user.

Giving access to web based email services like Hotmail[10] or GMX[11] in an active dummy could result in loss of data because the sender could extract the authentication data from the active dummy's instructions and use it to delete all incoming email or even to send out emails himself. Accessing an information service that requires user to register, may be a more suitable application.

Links to follow from the received document, could be selected with a regular expression or some other pattern description. This would allow for embedded session IDs while narrowing down the set of possible links. On a news web site for example, this could be used to select links to political or scientific articles if these categories are part of the link's URL or if they are always in a certain part of the HTML document.

The examples show that active dummies are not that easy to produce and will have to be tailored to each web site to produce some reasonable results. A change of layout or structure on a web site could make them unusable instantly.

[10] `http://www.hotmail.com/`

[11] `http://www.gmx.net/`