An Analysis of GNUnet and the Implications for Anonymous, Censorship-Resistant Networks

Dennis Kügler

Federal Office for Information Security
Godesberger Allee 185-189
53133 Bonn, Germany
Dennis.Kuegler@bsi.bund.de

Abstract Peer-to-peer networks are a popular platform for file sharing. However, only few of them offer strong anonymity to their users. GNUnet is a new peer-to-peer network that claims to provide *practical* anonymous and censorship-resistant file sharing.
In this paper, we show that one of GNUnet's features, originally designed to improve performance, can be exploited to degrade anonymity. We also present an efficient filter mechanism for GNUnet. Assuming that content filtering is legally enforced, GNUnet can be censored very efficiently. Finally, we discuss some possible improvements to GNUnet.

1 Introduction

Peer-to-peer networks are widely used to share all kinds of digital content with other users. It was first demonstrated by Gnutella [FP00] that such peer-to-peer networks can be organized in a decentralized manner, so that there is no practical way to shut down the network. Therefore, Gnutella and related networks have become a popular platform for sharing legal and illegal content (i.e. copyrighted or subject to censorship). While searching for content on Gnutella is relatively anonymous, responses to search queries are non-anonymous as the IP address of the offerer is always exposed. Thus, it is possible to prosecute users who break the law. As this clearly discourages users to share illegal content, censorship is indirectly applied.

The idea of constructing a censorship-resistant network goes back to the idea of the Eternity Service [And96], an attack-resistant storage medium. Freenet [CSWH01,CMH+02] was the first approach that tries to combine both types of networks: an anonymous, decentralized peer-to-peer network and a censorship-resistant network. Freenet can perhaps be described best as "distributed file system storing replicated content in an obfuscated form". One obvious drawback of Freenet is that it "forgets" infrequently requested content, which not only contradicts censorship-resistance but also makes the retrieval of content inconvenient as downloads often fail.

Recently, GNUnet [GPB+02,BGHP02,BG03] has been proposed, an alternative approach that claims to be much more practical and efficient. Therefore, we

analyze GNUnet and point out some weaknesses. We start in section 2 with a short introduction to GNUnet. In section 3 we show how an attacker can exploit GNUnet's shotcut feature to degrade anonymity. The shortcut feature was originally designed to optimize routing in case of high load. We describe how this feature can be misused for a special intersection attack. Afterwards, we discuss in section 4 how censorship can be applied to GNUnet. We present a content filter that makes use of GNUnet's encoding scheme to apply censorship. Finally, in section 5 we disscuss the exposed problems and suggest some improvements to GNUnet.

Disclaimer. The goal of this paper is **not** to suborn sharing of copyrighted or illegal content but to discuss, whether efficient anonymous, censorship-resistant networks are technically feasible.

2 A Description of GNUnet

GNUnet consists of nodes that communicate with each other. Every node chooses a key pair, that is used for identification, authentication, and to encrypt the communication between the nodes. To hide which nodes are indeed communicating with each other, GNUnet uses a MIX-like [Cha81] approach: nodes are intermediaries that send messages to other nodes.

In the following, we present the encoding scheme and the routing mechanism of GNUnet, which will both be exploited for our attacks.

2.1 Encoding[1]

GNUnet transfers blocks of fixed size (1Kb). There are three types of blocks: *Data Blocks* (DBlocks), *Indirection Blocks* (IBlocks), and *Root Blocks* (RBlocks).

Files of arbitrary size are split into DBlocks. For every DBlock D_i of a file the 160 Bit RIPE-MD hash value $H(D_i)$ is calculated. Then a tree of IBlocks is calculated recursively, where every (leaf) IBlock contains up to 25 query-hashes (see below), a superhash and a CRC32 checksum. The superhash of an IBlock is calculated as the hash of the concatenation of all query-hashes in the blocks below this IBlock. Every block B_i is encrypted with the hash value of its own content to $E_{H(B_i)}(B_i)$ and is stored under $H(E_{H(B_i)}(B_i))$. Thus, the query-hashes included in the IBlocks are pairs of the form $(H(B_i), H(E_{H(B_i)}(B_i)))$ that are necessary to retrieve the remaining blocks.

Finally, the RBlock is generated, containing a description of the file and the query-hash to retrieve the root of the tree of IBlocks. One or more keywords are used to encrypt and to store the RBlock. For every keyword K_j the RBlock R is encrypted to $E_{H(K_j)}(R)$, then both the encrypted RBlock and $H(H(K_j))$ are stored under $H(H(H(K_j)))$.

[1] The encoding scheme used by GNUnet to store and retrieve files has changed recently. The original scheme can be found in [BGHP02].

2.2 Routing

Retrieving content from GNUnet is a two step process:

1. **Discover RBlocks:** RBlocks are discovered using search queries containing a list of triple hashed keywords $H(H(H(K_j)))$. A node that receives a search query and has a RBlock stored under one of those values returns the encrypted RBlock $E_{H(K_j)}(R)$ together with $H(H(K_j))$ to prove that it indeed possesses the requested content stored under this keyword. The node that has issued the query is able to decrypt the RBlock using $H(K_j)$.
2. **Download Content:** After an RBlock is discovered, the node can download the corresponding file by issuing a download query. The node first uses the query hash included in the RBlock to retrieve and decrypt the root IBlock. Then the node uses the query-hashes included in the root IBlock to retrieve and decrypt all additional IBlocks and finally, the query hashes included in the leaf IBlocks are used to retrieve and decrypt the DBlocks of the file. The node may also use multi-queries to retrieve several I- or DBlocks at once. A multi-query consists of several download hashes plus the corresponding superhash to speed up lookups.

Thus, there are two types of queries: search queries and download queries. Both types of queries contain three additional values that are used to process the query: a return address, a priority, and a time-to-live (TTL).

Processing Queries. Each node sets up a message queue for every neighbor node. Before a query or a response is sent to a neighbor node, it is temporarily stored in the queue representing this node. Each queue is flushed at random intervals, then all messages waiting in this queue are sent to the corresponding node. A node that receives a query proceeds as follows:

1. If the requested content (RBlock, IBlock or DBlock) is found locally, a response is enqueued in the message queue corresponding to the return address given in the query. Otherwise (but always in the case of a search query), the query is sent to some other nodes.
2. Depending on the priority and the available bandwidth the node randomly selects n nodes and enqueues the query in their message queues. The query is adjusted as follows:
   - The new priority is calculated as $\frac{p}{n+1}$, where p is the old priority.
   - The node either keeps the return address of the predecessor node or replaces it with its own address. In the latter case, the node adds the query to its local routing table.
   - The TTL is continuously decreased by a reasonable amount. If the TTL falls below zero, the query is dropped and removed from the queues.

Credit. GNUnet is based on an economic system, where every node associates any other known node with a local value called *credit*. The goal of this economic system is to prevent flooding attacks by limiting the resources available to an attacker:

Let A and B be two nodes. A associates B with credit c_B and vice versa. After B received a query with priority p from A, it first checks whether A has enough credit.

- If $c_A = 0$ the query is dropped, otherwise set the new priority p' to p.
- If $p' > c_A$ the priority of the query is reduced to $p' = c_A$.

Then B processes the query and charges A by reducing A's credit to $c_A = c_A - p'$. However, if B has excess bandwidth, it may decide not to charge A. The decision whether and how much B decreases A's credit is invisible to A.

If B returns a valid response, B expects A to increase his credit to $c_B = c_B + p$. Again, A may decide not to pay B for returning a valid response. The decision whether and how much A increases B's credit is invisible to B.

Zero Priority Queries. If a node indirects a query to another node without charging the preceding node, it may assign zero priority to the forwarded query, because it does not want to be charged by the following node.

Excess bandwith means, that the load on the node is low. Thus, if a node has excess bandwidth, responding to a zero priority query does not hurt this node. We doubt that zero priority queries work:

- From an economic point of view, it does not make any sense to respond to a zero priority query, as the responding node cannot gain any credit.
- A node with excess bandwith cannot hide its own traffic (i.e. responding to the query) very well. Thus, the best thing the node can do is either drop or indirect the query to another node, which may proceed similarly until the query is finally dropped.

3 Deanonymization in GNUnet

An informal definition of anonymity can be found in [PK01]:

> *"Anonymity is the state of being not identifiable within a set of subjects, the anonymity set".*

We consider a network of several nodes, where some nodes may be malicious as shown in Figure 1. A malicious node is behaving correctly, but it tries to acquire as much information about the network as possible. Furthermore, we assume that all malicious nodes are controlled by the same attacker.

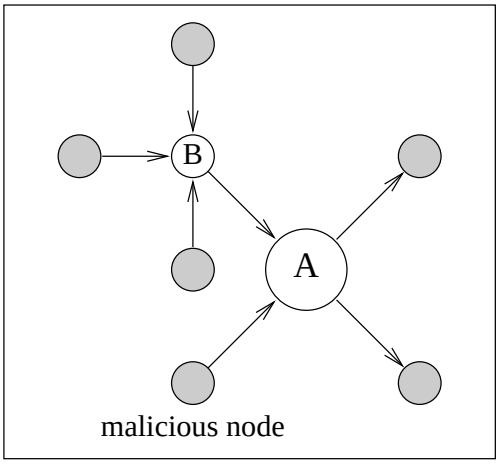

Figure 1. Malicious nodes in a network.

3.1 Identifying Sessions

To successfully apply our attacks, we first have to identify sessions. This can be done very easily: GNUnet splits larger files into small DBlocks. If we know the corresponding IBlocks, we are able to identify DBlocks that belong together. Thus, the attacker prepares a dictionary of interesting keywords and their corresponding hash values, queries the network with those keywords and receives a number of encrypted RBlocks $E_{H(K)}(R)$ which he is able to decrypt. Then the attacker can also retrieve the corresponding IBlocks. As the IBlocks contain the query-hashes of all DBlocks, the attacker is able to use his malicious nodes to observe the following:

- Queries containing one of the prepared keywords.
- Queries for known I- or DBlocks.
- Responses containing known I- or DBlocks.

Note that the attacker can also observe responses to queries with non-trivial keywords if the file also has been inserted (probably by a different user) under trivial keywords. Inserting content under a non-trivial keyword renders the content useless as only few people are able to retrieve the content again. This is a contradiction to the idea of a censorship-resistant network, where everybody should be able to access any possible content. Therefore, we expect that most content is available to the attacker.

3.2 Degrading Anonymity

An attacker can exploit the linkability of related I- and DBlocks to execute two types of attacks on the anonymity:

1. **Intersection Attacks:** The intersection attack [BPS01] exploits the fact that not all users of a MIX network participate in every batch. In the setting of GNUnet, if a node has not been involved in routing linkable traffic, the attacker will remove the node from the anonymity set.
2. **Predecessor Attacks:** The predecessor attack [WALS02] extracts information from the setup of dynamically chosen, linkable paths. In the setting of GNUnet, the attacker logs the identity of the preceding node. The expected number of times that the initiator is logged is greater than the expected number of times that any other node is logged (assuming that all nodes are logged with equal probability).

In the following, we focus on an improved intersection attack, which seems to be most effective to determine the initiator of a query. If GNUnet's routing turns out to be much more dynamic than we expect, the predecessor attack should be reconsidered. While this attack has be used in [WALS02,Shm02] to attack Crowds [RR98], it should be mentioned that the results cannot be applied directly to GNUnet, as the setup of paths is different.

3.3 Intersection Attacks from Shortcuts

While intersection attacks also benefit from dynamically chosen paths, in the setting of GNUnet this attack is successful even with very static paths. Our *shortcut attack* is a special intersection attack that exploits GNUnet's shortcut feature. Depending on the current load, a node uses this feature to decrease both bandwidth utilization and latency:

- If the node is idle, queries are *indirected*. The node replaces the return address with its own address.
- If the node is busy, queries are *forwarded*. The node keeps the return address of the preceding node.
- If the node is very busy, queries are *dropped*. The node does not process the query.

In short, a busy node simply optimizes the routing of queries and responses by removing itself from the anonymity set. As the adjacent nodes learn that this node is not necessary to route related queries it will be removed completely to gain more performance [BG03].

It is claimed by the authors of GNUnet that shortcuts do not hurt anonymity: A node that stops indirecting some traffic can only hurt itself, as hiding its own activities is worse with lower traffic. We show that this is not the case. We exploit this optimization to discover the initiator of a query as shown in Figure 2, where node C is an attacker who receives queries from node B that have been issued by node A.

Shortcut Attack. The attacker tries to entice an attacked node to forward traffic instead of indirecting it by increasing the load of the node:

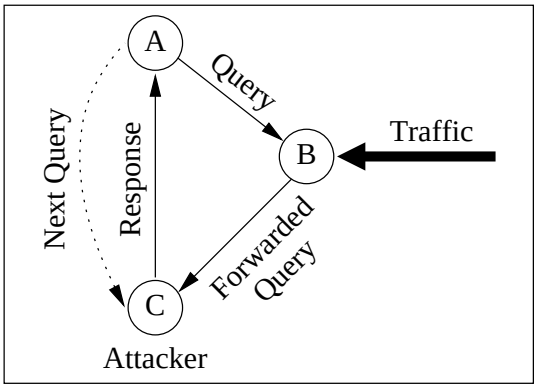

Figure 2. The shortcut attack.

1. The attacker floods node B with traffic. The attacker may hide his attack by using several malicious nodes to flood the attacked node. Note that even if the traffic received from the attacker is immediately dropped, the load of the attacked node yet increases. Thus, the attack is independent of credit.
2. The load on the flooded node B increases and the node decides to forward instead of indirecting some traffic: The node keeps the return address of node A when forwarding queries, so that A will directly receive responses from node C (the attacker).
3. To improve performance after node A has received a valid response from node C, it will address further related queries directly to C.

This attack enables the attacker to eliminate honest nodes one after the other until the attacker has identified the initiator of the query.

Improved Shortcut Attack. We can further improve the shortcut attack by removing the requirement to flood the attacked node.

1. The attacker guesses that node A is the predecessor of node B and returns a response directly to node A.
2. If the attacker has guessed correctly, then he will receive further related queries directly from node A.

Thus, the attacker can efficiently test, which nodes are required to route responses until the attacker has identified the initiator of the query.

4 Censoring GNUnet

Assuming that GNUnet turns out to be practical and is in wide use, we expect that there will be attempts to censor GNUnet. One possibility for censorship is

to apply "Rubber-Hose Cryptanalysis" by forcing the owner of a node to remove certain content from the cache.

Constructing an anonymous network that is resistant against rubber-hose cryptanalysis is very difficult. A short discussion why Publius [MWC00], Freenet [CSWH01], and Tangler [WM01] fail to resist such attacks can be found in [Ser02]. There, a censorship-resistant network is also sketched. Besides this approach, the only approach achieving a similar goal is Free Haven [DFM01].

In the following sections, we show that GNUnet also is not censorship-resistant.

4.1 Rubber-Hose Cryptanalysis

To be able to apply rubber-hose cryptanalysis, we assume that the content to be censored is only infrequently requested and thus stored in only few locations. To force nodes to remove such content, the attacker first has to determine which nodes provide this content. The goal of the following attacks is to limit the anonymity set of the responding node.

Distance Attack. The attacker manipulates the time-to-live value[2] of queries:

1. The attacker sends or forwards queries with an artificially low (zero) TTL value. Thus, if the succeeding node is not able to respond, the query will be dropped immediately. The attacker resends the query with an increased TTL until he receives a response. Note that the attacker must make sure that his query is not dropped due to low bandwidth. Thus, the attacker has to send queries either with a high priority (which requires credit), or the attacker must have knowledge of the excess bandwidth available to the attacked node.
2. After the attacker has received a response, he can use the TTL value to guess the approximate distance to the content provider. The attacker restarts the attack with a different but related query (e.g. the next DBlock) and a neighbor node of the current node. If the neighbor node is closer to the content provider, then a lower TTL succeeds and the attacker proceeds similarly. Otherwise he queries another neighbor node.

Similar attacks are already known from Freenet. Freenet tries to prevent such attacks by not always dropping queries with zero TTL. While this offers only weak protection against such attacks, it is even worse with GNUnet. Due to the splitting of large file into small linkable blocks, a node can be tested more effectively whether the node only caches some blocks or provides the whole file (i.e. stores the file unencrypted).

Shortcut Attack. The shortcut attack introduced in section 3.3 can similarly be used to determine the responding node by exchanging the roles of node A and node C. The attacker can successively get closer to the responding node by eliminating intermediary nodes one after the other.

[2] The priority information can be used to construct a similar attack.

4.2 Enforced Content Filtering

Rubber-hose cryptanalysis is neither suited for large scale censorship nor to censor frequently requested content that is stored in many locations. Therefore, we present a content filtering mechanism that can be legally enforced to make sharing illegal content nearly impossible.

Licenses. Our filter mechanism makes use of the fact that I- and DBlocks are stored and retrieved under the hash value of their encrypted content. Thus, indexing illegal content is possible and censorship can be applied very efficiently by allowing nodes to only deliver root IBlocks together with a valid license (it is sufficient to filter root IBlocks). A censoring authority issues licenses by signing the query-hash $H(E_{H(I)}(I))$ of the root IBlock. There are two types of licenses, positive and negative licenses. A positive license allows, a negative license prohibits delivering the corresponding content.

Positive License: A positive license certifies that $H(E_{H(I)}(I))$ is currently not on the index. Positive licenses are always time restricted. Thus, to issue a positive license, the censoring authority need not check the actual content.

Negative License: A negative license certifies that $H(E_{H(I)}(I))$ is on the index. Negative licenses are not time restricted.

A node that wants to respond to a query, but does not possess a positive license for the IBlock (e.g. the content is new or the positive license is not valid anymore), first tries to retrieve a license from GNUnet. If no license is available, it directly requests a license from the censoring authority. Thus, for frequently requested legal content the permission to deliver this content is wide distributed in GNUnet and can be retrieved efficiently.

Censoring. Searchability and censorship-resistance exclude each other to a certain extent. If content is easy to find for users, it is similarly easy for others to find illegal content:

1. Content owners can search for their copyrighted content. After finding such content, the corresponding root IBlock is sent to the censoring authority together with a proof of ownership.
2. The censorship authority itself searches for illegal content and issues negative licenses for all found IBlocks.

To check whether a node applies content filtering, the censoring authority only has to search for indexed content. The owner of a node that returns such content (without a valid positive license) may be prosecuted for e.g. copyright infringement. As every owner of a node can be prosecuted, GNUnet may discourage users to share illegal content even more than any non-anonymous approach.

Multiple Censoring Authorities. Instead of using only a single censoring authority, it is also possible to use multiple censoring authorities, depending on the jurisdiction of the node indirecting the content.

5 Discussion

We have exploited two properties of GNUnet for our attacks: shortcuts and unique identifiers. In the following sections, we discuss those properties again and try to give give some suggestions for possible improvements of GNUnet.

5.1 Shortcuts

The idea of using shortcuts in case of high load is not bad at all: Nodes try to compensate high load by removing themselves from routing related messages. Thus, only one node is removed from the anonymity set. The alternative to this approach is to drop messages, which is unfortunately done in case of very high load. In this case it is very likely that the download fails and the initiator has to setup a completely new path. It directly follows that this often results in a completely different anonymity set, which can be exploited for both a predecessor attack and a plain intersection attack.

Although shortcuts are perhaps the best solution to compensate high load, they should be used very carefully. To make shortcuts more secure, the following two provisions should be considered:

1. A node removing itself from the anonymity set should make sure that both affected neighbor nodes are informed about this optimization.
2. A node should be enabled to differentiate ordinary traffic from dummy traffic that is used to increase the load of the node.

While the first provision can be implemented easily, the second one is more difficult. In the following, we will show that the economic model is not suited to solve this problem. Instead, we propose a trust based approach.

Credit. GNUnet's economic model does not help much to differentiate both types of traffic and currently GNUnet regards all traffic as good. If credit was used to differentiate good from bad traffic, the shortcut attack would be even more powerful: If nodes always indirect queries originating from nodes with high credit but sometimes forward queries from nodes with lower credit, this behavior can be exploited by an attacker to better influence the routing of queries. As the attacker can gain credit by behaving correctly, this credit can be used to have the attacked node indirecting dummy traffic and forwarding ordinary traffic. Thus, the shortcut attack will become even more effective.

Trust. To enable a node to distinguish ordinary traffic from dummy traffic, we suggest integrating a trust-metric in GNUnet. As every GNUnet node is identified by a public key, we propose to make use of the web of trust used by *Pretty Good Privacy* [Zim95] for public key certification. While this approach has advantages, it also has a disadvantage:

- A web of trust allows a node to better distinguish between good and bad traffic.
  - In case of high load a node should begin to forward traffic originating from less trusted nodes.
  - In case of low load a node may try to acquire (additional) traffic from trusted nodes.
- Using a web of trust introduces pseudospoofing. An attacker that is close enough in the web of trust is able to create an infinite number of identities that are fully trusted.

Thus, integrating a trust-metric in GNUnet may improve anonymity, but should be carefully used. An attack resistant trust-metric is proposed in [LA98].

5.2 Unique Identifiers

Content filtering is possible as every file in GNUnet is identified by (at least) one unique identifier, the hash value of the encrypted root IBlock. As long as the content of the file is unchanged, this identifier doesn't change either. The purpose of such an identifier is not only to retrieve the corresponding file, but also to automatically replicate this file while it is delivered. Thus, those identifiers serve as obfuscated filenames in a distributed filesystem.

The drawback of associating files with unique identifiers is that those identifiers can be used to index illegal content. As every intermediary node is aware of those identifiers, censorship can be applied very efficiently by introducing licenses to allow delivering content. To make GNUnet robust against censorship, intermediary nodes must not be aware of the content of an indirected block. Every block should be indistinguishable from any other block. Thus, unique identifiers have to be removed completely, which renders automatic replication impossible.

6 Conclusion

GNUnet is an interesting approach, but it still contains some weaknesses. A problem that is inherent to the design of GNUnet is the decision to split larger files into many small pieces. To download a file all those linkable pieces have to be retrieved separately. An attacker can exploit this property for intersection and predecessor attacks. But an even more serious problem is the shortcut feature of GNUnet. This feature has been designed to improve performance, but it can be misused for an improved intersection attack. To prevent this attack, we have suggested some modifications to the shortcut mechanism.

Furthermore, we have discussed censorship-resistance of GNUnet. Interestingly, the replication mechanism used as basis for censorship-resistance does not enhance censorship-resistance but paradoxical decreases it. We have shown that GNUnet is vulnerable to rubber-hose cryptanalysis, but we also have presented an efficient content filter for GNUnet. If content filtering is legally enforced, censoring GNUnet is possible at a very large scale. Finally, we have pointed out that censorship-resistance and automatic replication seem to be complementary goals.

An open problem is to ascertain how much GNUnet is really affected by the predecessor attack. A formal specification of the node behavior would ease such an analysis.

References

[And96] Ross J. Anderson. The eternity service. In *Proceedings of Pragocrypt '96*, 1996.

[BG03] Krista Bennett and Christian Grothoff. Gap – practical anonymous networking. In *Designing Privacy Enhancing Technologies – International Workshop on Design Issues in Anonymity and Unobservability 2003*. This proceedings, 2003.

[BGHP02] Krista Bennett, Christian Grothoff, Tzvetan Horozov, and Ioana Patrascu. Efficient sharing of encrypted data. In *Information Security and Privacy – 7th Autralasian Conference (ACISP 2002)*, pages 107–120. Springer-Verlag, 2002.

[BPS01] Oliver Berthold, Andreas Pfitzmann, and Ronny Standtke. The disadvantages of free MIX routes and how to overcome them. In *Designing Privacy Enhancing Technologies – International Workshop on Design Issues in Anonymity and Unobservability 2000*, Lecture Notes in Computer Science 2009, pages 30–45. Springer-Verlag, 2001.

[Cha81] David Chaum. Untraceable electronic mail, return adresses and digital pseudonyms. *Communications of the ACM*, 24(2):84–88, 1981.

[CMH+02] Ian Clarke, Scott G. Miller, Theodore W. Hong, Oskar Sandberg, and Brandon Wiley. Protecting free expression online with freenet. *IEEE Internet Computing*, pages 40–49, 2002.

[CSWH01] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Designing Privacy Enhancing Technologies – International Workshop on Design Issues in Anonymity and Unobservability 2000*, Lecture Notes in Computer Science 2009, pages 46–66. Springer-Verlag, 2001.

[DFM01] Roger Dingledine, Michael J. Freedman, and David Molnar. The free haven project: Distributed anonymous storage service. In *Designing Privacy Enhancing Technologies – International Workshop on Design Issues in Anonymity and Unobservability 2000*, Lecture Notes in Computer Science 2009, pages 67–95. Springer-Verlag, 2001.

[FP00] Justin Frankel and Tom Pepper. Gnutella v. 0.56. Nullsoft, 2000.

[GPB+02] Christian Grothoff, Ioana Patrascu, Krista Bennett, Tiberiu Stef, and Tzvetan Horozov. GNET. Whitepaper version 0.5.2, 2002.

[LA98] Raph Levien and Alex Aiken. Attack-resistant trust metrics for public key certification. In *Proceedings of the 7th USENIX Security Symposium*, 1998.

[MWC00] Aviel D. Rubin Marc Waldman and Lorrie Faith Cranor. Publius: A robust, tamper-evident, censorship-resistant, web publishing system. In *Proc. 9th USENIX Security Symposium*, pages 59–72, 2000.

[PK01] Andreas Pfitzmann and Marit Köhntopp. Anonymity, unobservability, and pseudonymity – a proposal for terminology. In *Designing Privacy Enhancing Technologies – International Workshop on Design Issues in Anonymity and Unobservability 2000*, Lecture Notes in Computer Science 2009, pages 1–9. Springer-Verlag, 2001.

[RR98] Michael K. Reiter and Aviel D. Rubin. Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, 1998.

[Ser02] Andrei Serjantov. Anonymizing censorship resistant systems. In *Peer-to-Peer Systems, First International Workshop – IPTPS 2002*, Lecture Notes in Computer Science 2429, pages 111–120. Springer-Verlag, 2002.

[Shm02] Vitaly Shmatikov. Probabilistic analysis of anonymity. In *15th IEEE Computer Security Foundations Workshop*, pages 119–128. IEEE Computer Society Press, 2002.

[WALS02] Matthew Wright, Micah Adler, Brian N. Levine, and Clay Shields. An analysis of the degradation of anonymous protocols. In *Network and Distributed System Security Symposium – NDSS 2002*. Internet Society, 2002.

[WM01] Marc Waldman and David Mazi. Tangler: a censorship-resistant publishing system based on document entanglements. In *ACM Conference on Computer and Communications Security*, pages 126–135. ACM Press, 2001.

[Zim95] Philip R. Zimmermann. *The Official PGP User's Guide*. MIT Press, 1995.