Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science

6.829 Fall 2002 **Anonymity in the Internet** November 22, 2002

---

1 Overview: Why Anonymity?

Why do we need anonymity? Typically it's to help certain people "have a voice" that wouldn't ordinarily be able to do so, considering their circumstances. For example, in light of the recent corporate scandals, anonymous communication could have provided a useful channel for whistle-blowers to alert authorities of unscrupulous practices. Anonymous retrieval of information can be beneficial to people who are subject to oppressive government censorship. On an everyday basis for you and me, anonymous communication can provide a means for a student to provide feedback to a professor about a class without fear of repercussions. Often people just want their statements to be judged on their own merit. An example of this in the non-technical world is *The Economist*, where the editors and authors of the magazine are listed inside the front cover, but articles are run without bylines.

Of course, anonymity is a double-edged sword that can be used for bad things as well; including harassment. Pseudonymous communication, such as provided by `nym.alias.net`, can facilitate other illegal activities, such as the trading of child pornography, etc. We assume that people who build these systems do so because they feel that the benefits outweigh the potentially bad effects.

Because of the nature of these services, anonymous services can be subject to abuse and attacks in an attempt to shut them down. As a result, an anonymous system that is designed to operate in the real world must be able to withstand many varied types of attacks. We'll look at these systems in the context of how they might stand up to various types of attacks.

One type of anonymity is called pseudonymity—where it's possible for one to correspond with the anonymous party via a *pseudonym*, but it is not possible to discover the actual identity of the pseudonym.

2 Email Pseudonym Servers

The first email pseudonym server was `anon.penet.fi`—which was a one-hop pseudonym server that operated completely in cleartext. It performed a simple pseudonym/email-address substitution, exactly as you might expect. What are the problems with this type of system? First, it was a "single point of compromise"—compromising their server exposed all identities. Also, it's certainly an invitation for the authorities to come knocking! Additionally, it became a bottleneck.

In response to these shortcomings, people developed type-1 *anonymous* remailers, which were typically not used for pseudonymous communications. The type-1 remailers simply allowed a message to be sent through a series of remailers, the message having been encrypted successively with the public key of each remailer along the chain. The catch here is that pseudonymous communication is not possible, because it's not possible to send a message in return—someone is certainly not going to provide a valid email address for reply! Type-2 (mixmaster) remailers also provide some improvements over type-1 remailers: notably, they make traffic analysis harder by using fixed size messages and better reordering of messages at these remailers (i.e., using a mixnet). Problems here: delay.

What's the key for allowing pseudonymous communication? *Reply Blocks*—i.e., a way for someone sending mail back to you to somehow be able to use a pseudonym and have it get to you.

So how does `nym.alias.net` work? First, you have to pick a pseudonym and generate a public/private key pair for that pseudonym. Next, you have to select a series of remailers through which you will send your message (there are many remailers in operation, although the method for getting these outlined in the nym help file is grossly out of date; in addition, the `premail` tool does not seem to be maintained).

The nym server keeps track of three pieces of information: a public key for your nym (used for authentication—all emails from your nym must be signed by your private key), a reply block, and some configuration information.

What does the reply block do? It basically provides a path for getting the mail from the pseudonym server back to you. You can think of it has hop-by-hop instructions, so that each hop can only see the next hop for where to send the piece of mail. Note a cool thing here is that you don't always have to choose your personal email address as the final destination: you could, for example have the mail show up as a post on usenet.

Note that the digital signature requirement allows mail to be authenticated, while at the same time coming from anywhere.

How does one create a reply block? Create the first hop with your real email address (or something else), and specify a symmetric encryption key, k_1. Take that block, encrypt it for the public key of the *last* hop along the remailer chain that's coming back to you. Why the last? Because it's the last hop that's going to have your email address in there? That's going to be the first chunk of your reply block. To that hop, you them pick a second remailer, set the `To:` address in this chunk to the address of the remailer of the last hop, pick a new symmetric key, and then PGP encrypt that part of the reply block for the second-to-last reply block. Rinse, lather, repeat.

Note that, at no point do any of the hops see the plaintext of the message. The last hop sees the real email address of the user. Is this a problem? Not necessarily, since that last hop doesn't necessarily know that it's the last hop! It seems actually you could start to make a pretty good guess based on the size of the reply block— as it traverses progressively more hops, the size of the reply block gets smaller, so... Also, everytime a nym receives mail, identical reply block ciphertexts fly across the network.

Why is the super-encryption of each part of the reply block necessary? The complete path (thus, the last hop) that the message will take becomes very obvious. Note, it's not really to protect confidentiality of the message itself, since that's provided with encryption of the nym's public key. The reason seems to be that the ciphertext arriving at and leaving a remailer cannot be correlated; delay helps with this as well, as mixing can be performed.

Don't you have to communicate with *someone* to set up the nym in the first place? No – can just set up your public key and reply blocks with nym using an anonymous remailer! Nyms can provide multiple reply blocks to get better redundancy. Note that the user never has to communicate directly with the nym server; cool.

This type of system is obviously subject to a bunch of attacks. Let's highlight just a few of them. Harassment; suitable solution is destination blocking. Exponential mail loop (this actually happened): include the nym itself in a reply block. Now you've got a looping issue; solution here is simply to limit the amount of mail one nym can receive in one day. Reverse mail bomb to `help@nym.alias.net`, with forged headers: solution is to include headers of original message that prompted the reply. Encrypted mail-bomb. Point a reply-block at someone you don't like, and sign that nym up for lots of high-traffic stuff. Mail-to-news gateway that allows pasting arbitrary from headers creates a hell of a lot of commotion – spam-baiting with people's email addresses. People's complaints to the remailers caused many remailers to shut down...

Lessons: abuse can have a very detrimental effect on anonymity. solve problems without logging (i.e., solution to the reverse mail bomb). Rate limiting (also, by putting the human in the loop) can help reduce abuse.

3 Freenet

Attempts to provide "anonymity for both producers and consumers of information", deniability, etc. Note, again, that a chiefly stated design goal is the resistance to attacks – i.e., attempts by attackers to block access to certain information.

Freenet: p2p network that query each other to store and retrieve data. The data is named by location-independent *keys*. Each node contains: local datastore, and a "routing table" that contains addresses of other nodes and the keys that they should hold. Queries are passed from node to node; each node makes a local routing decision about where to send the query next. Queries are given pseudorandom IDs to prevent looping of queries.

Let's first consider how data is retrieved from Freenet. The storage story is pretty much analogous, so we'll briefly discuss that second.

How is data retrieved? First, you need to have the name or description of the content you are trying to retrieve (finding the appropriate description or filename is another story that we'll consider briefly. Assume for now that we can get this information.) The query also contains a hops-to-live value to prevent it from looping, eventually guaranteeing an answer, etc. If the node that receives a query contains the key in question, then it returns the data, saying also that it was the source of the data. The data is returned along the query path, and cached at each point along the way.

What's the point of this caching business? Nodes become specialized in locating sets of similar keys, since requests in that part of the keyspace will be forwarded to other nodes that contain similar keys. Because answering these queries also implies getting a copy of the file upon reply, nodes should become specialized in storing clusters of files that have similar keys. Popular data also automatically gets replicated closer to the requesters. Nodes also discover nodes that contain files, thus resulting in increased connectivity.

What if a loop is encountered? Then a node can try its second closest key, and so forth. Thus, the search is simply a hill climbing

How is anonymity for the data-sources obtained? Any node along the retrieval path for data can arbitrarily decide to change the reply message to claim itself as the data source.

How is anonymity for the requester obtained? At any point along the path, the node only knows the immediately preceding node that forwarded the request, not that it was the requester.

Inserting is analogous. Pick a descriptive string, hash it to get a key, and send an "insert" message with a particular TTL value. If collision found, return a fail back to the person attempting the insert (actually, treat it as a query...return the actual data back along that path, copies and all). Obvious problems? Of course we could have hash collisions. We could also have duplicate keys all over the network if the TTL is not large enough. Cool because newly inserted files get inserted on nodes that already have similar keys. Copy hack means an attacker is likely to just cause further replication of the file he's trying to clobber.

How to deal with the flat namespace? Use a level of indirection. Directory files, for example, contain pointers to keys that are lower level in the hierarchy (similar to filesystem structure). How to update these without permitting corruption, etc. ? Who knows.

How to deal with collisions in the namespace? Indirection again. Basically, insert files based on their *content-hash*. Users retrieve these files by first retrieving a file under a semantically-meaningful name, which then contains a list of content-keys, along with other metadata (author, etc.).

What about search? When someone inserts a file, they also insert "keyword files", that contain pointers to the file itself. Must make two relaxations to the original Freenet protocol: 1.) collisions are possible, and 2.) retrieval can return more than one of these keyword files.

4 Infranet

Please see:

- USENIX Security 2002 Paper: `http://nms/~feamster/papers/usenixsec2002.pdf`
- Talk Slides: `http://nms/~feamster/talks/infranet/UsenixSec02/html/`

