# Subverting Exit Censorship in Anonymous Networks

Student Name: Mridul Malpotra
Roll Number: 2012061

Student Name: Tarun Verma
Roll Number: 2012112

BTP report submitted in partial fulfillment of the requirements
for the Degree of B.Tech. in Computer Science & Engineering
on April 19, 2016

**BTP Track**: Research

**BTP Advisor**
Dr. Sambuddho Chakravarty

Indraprastha Institute of Information Technology
New Delhi

## Student's Declaration

We hereby declare that the work presented in the report entitled **"Subverting Exit Censorship in Anonymous Networks"** submitted by us for the partial fulfillment of the requirements for the degree of *Bachelor of Technology* in *Computer Science & Engineering* at Indraprastha Institute of Information Technology, Delhi, is an authentic record of our work carried out under guidance of our adviser **Dr. Sambuddho Chakravarty**. Due acknowledgments have been given in the report to all material used. This work has not been submitted anywhere else for the reward of any other degree.

**Mridul Malpotra** **IIIT Delhi, April 19, 2016**
**Tarun Verma**

## Certificate

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

**Dr. Sambuddho Chakravarty** **IIIT Delhi, April 19, 2016**

**Abstract**

Anonymity on the Internet, while a very convenient concept, is also quite powerful. It enables users to utilize the facilities enabled by global networks without leaking any sensitive information that could lead to exposure of their real life identity. A lot of web services therefore (understandably) prefer blocking such users to prevent abuse. However, it has been observed that some web services block anonymous users even in the absence of abuse, thus disallowing a genuine user (who wishes to maintain his/her anonymity). We aim to verify existing and build schemes that bypasses the blocking mechanism that is currently in use against Tor, the most widely used service for enabling anonymous communication. We primarily focus on Single Packet Authorization (SPA) and location based Hidden services in Tor as a medium to subvert exit censorship.

Keywords: Anonymity, Security, Onion Routing, The Onion Router, Censorship

## Acknowledgments

We would like to thank our project advisor, Dr. Sambuddho Chakravarty, for his invaluable guidance throughout the course of the project. Our friends, for their suggestions as well as advice which helped solve some non-trivial issues which arose from time to time. Lastly, we'd also like to convey our gratitude to the evaluation committee for their honest and able judgment of our work.

## Work Distribution

### Mridul Malpotra

- Setting up the private Tor network on PlanetLab using CDNs and automating creating exit relays. *Concerns **section 4.1.2**.*
- Developing and implementing the Fwknop scheme for accessing hidden exits using Fwknop *Concerns **section 4.2**.*
- Testing the scheme with circuit randomizations to measure and record performance in circuit creation and extension. *Concerns **section 4.2**.*
- Working with Tor Stem library for testing operations including polling network stats and custom circuit creation on public Tor network. *Concerns **section 3.1**.*
- Server configuration and creating alternate private subnet for testing purposes. *Concerns **section 4.1**.*
- Testing Single Packet Authorization using Firewall knock operator (Fwknop). Client and server tested directly on private subnet and through private Tor network. *Concerns **sections 3.3 and 4.2**.*
- Troubleshooting Fwknop client and server communication by connecting to Tor socks proxy using socat; used packet and connection state monitoring. *Concerns **sections 3.1 and 4.2**.*
- Writing custom socks client/server application using Socksipy module for connecting Fwknop client to server through private Tor network. *Concerns **section 4.1**.*
- Contribution to coming up with an alternative scheme to employ hidden exit nodes (using Tor hidden services mechanism). *Concerns **section 4.4**.*

### Tarun Verma

- Surveying available literature on Tor and notable contributions to creation of a local Tor network, namely the automation of the entire process of construction, listing and tear-down of Tor circuits. *Concerns **section 3.1**.*
- Surveying existing literature on **Anonymous Authentication** to get a clear idea of available techniques. *Concerns all of **section 3.2**.*
- A primitive design for the anonymous authentication scheme with (suggested) usage in the project. Implementation and testing still remain, although it is expected that the scheme would (supposedly) perform better than a public key approach. *Concerns all of **section 4.3**.*
- Classifying websites as either Tor blocking, detecting or allowing. *Concerns all of **Chapter 5**.*
- Contributions in testing and debugging of firewall knock protection service. *Concerns **section 4.1**.*
- Consequential contributions in respect to the design and implementation of the hidden services scheme. *Concerns **section 4.4***

- Survey of web proxying mechanisms as well as configuration and implementation of various proxy servers like *Squid3, Nginx etc. Concerns* ***section 3.4***.
- Other minor contributions and suggestions, notably the use of a proper three node Tor network that fixed issues with packet forwarding.

# Contents

# Chapter 1

# Introduction

As more users have migrated solely to online services as their preferred means of communication, surveillance by various government as well as private institutions has become an issue of major concern [13]. This poses a problem to eavesdroppers, for example, who may wish to expose unethical practices of said institutions without risking identity exposure. Anonymity may also be beneficial to users who face bullying in real as well as virtual spheres strictly for voicing their opinions on public forums.

Some popular web services facilitate user anonymity by dropping any pre-registration requirement. Image-boards like 4chan [**?**, 4ch]re prime examples of the same. Such a design decision, however, is far from universal, and most services (apart from demanding registration) also track user activities through various means. Onion Routing is a novel scheme that enables anonymous communication by encapsulating messages in layers of encryption, analogous to layers of an onion. The encrypted data is transmitted through a series of network nodes called onion routers, each of which "peels" away a single layer, uncovering the data's next destination. When the final layer is decrypted, the message arrives at its destination. The sender remains anonymous because each intermediary knows only the location of the immediately preceding and following nodes [7].

The first implementation of onion routing wasn't very optimal for real world usage. It was a fragile proof-of-concept that ran on a single machine and processed thousands of connections from machines all over the world every day. Apart from latency issues, the scheme also had critical design flaws. Thus, it paved the way for development of the second generation onion router, now known popularly as Tor. Tor not only fixed some critical design issues that were present in the original scheme, but also provided a more robust deployment mechanism along-with easy availability and open design. It has since become the most powerful and popular distributed anonymity network.

Even though Tor has found itself quite a large user-base, some web services still prefer banning Tor users to prevent abuse. Wikipedia, for example, does not allow anonymous users to edit articles for obvious reasons. A service wishing to block Tor users may simply block Tor exit nodes, thus disallowing any connection to be established. Since Tor users can be blocked quite easily, some web services may choose to block them regardless of their use-case, confiscating

functionality from them in an unfair manner.

We aim to build a scheme that circumvents this kind of unfair censorship. The basic idea that enables us to tackle such a problem would be to portray an anonymous user as an authentic one. For this purpose, we present the concept of a hidden exit node, an authentication server which forwards the traffic of an anonymous client (after authentication) to the Tor blocking web service, while at the same time causing no harm to the client's anonymity. To the web service, this client would appear as a regular user, and thus (ideally) it would not employ any blocking mechanism. We do not address the problem of a malicious user also benefitting from our service (by abusing the censors). Although it's an unreasonable assumption, but we expect a respectable behavior from the clients on their end.

Apart from building such robust, simple, scalable scheme, we also aim to devise a distinguishing algorithm that is able to predict (with a fair amount of accuracy) whether a particular website blocks/detects Tor or not. This will enable us to quickly shortlist websites on which we will ultimately test our anonymous authentication scheme, and (hopefully) produce convincing results.

# Chapter 2

# Goals and Non-Goals

Here we give a concise description of what we are aiming to build and clearly mention the problems/issues that we will *not* be addressing (for now). We briefly mention some extensions that we may explore in the latter half of the project.

## 2.1 Goals

1. Building a novel proof of concept for circumvention of Tor censors through anonymous authentication.
2. Devising a cryptographically secure anonymous authentication scheme that is scalable and suited to our case.
3. Making the overall circumvention process robust and easily incorporable.
4. Devising a distinguishing algorithm that classifies websites according to their behavior towards Tor (blocking/preventing/allowing).
5. Using (4) to generate a test set for verification of the bypassing mechanism, as well as generate a (considerably large) list of websites that aids future research on Tor censors.

## 2.2 Non Goals

1. Detecting malicious parties who may abuse the circumvention mechanism.
2. Defense mechanisms against attacks on the scheme itself (taking control of hidden exit node, for example).

## 2.3 Extensions

1. Alternative strategy to anonymous authentication scheme for achieving censorship circumvention by using the Tor hidden services mechanism.
2. Detection of malicious parties who may abuse the scheme.
3. Addressing different attacks that may be mounted against the scheme to deanonymize the user.
4. Implementing defense mechanism(s) for (3).

# Chapter 3

# Background

## 3.1 Tor

The Onion Router (Tor) is a second iteration over the Onion Routing network. It is a distributed, scalable, resilient and secure network that allows either sender-only anonymity or both sender-receiver anonymity. The design of Tor uses the concept of intermediate mixes as proposed by David Chaum [3] to allow for untraceability and prevent sender-to-receiver association. [4]

Tor improves upon the traditional Onion routers by having additional features installed, which include hidden services through introduction and rendezvous points, perfect forward secrecy in terms of session tracking, flexible exit policies and circuit creation and more effective congestion control. [1]

### 3.1.1 Design elements

We describe relevant elements that are a part of the Tor network and whose understanding will be used in our implementations.

- **Onion Router (OR):** The most critical unit of the network, an onion router is a volunteer based server that serves to anonymously forward TCP traffic to a predefined set of nodes as dictated by the client. The router follows onion routing, implying that over the standard Diffie Hellman key exchanges, there exists another layered structure that envelopes the data in an onion-layer like manner. The router can either be a *entry/middle* node and relay traffic, or can be a *exit* node and allow traffic to exit the Tor network through it.
- **Onion Client (OC):** The client is majorly responsible for creating the circuit by choosing Onion router nodes from a prefetched list of participating members. A client creates multiple circuits and is responsible for the connection negotiation happening at each of the selected ORs.
- **Circuit:** A circuit is a connection negotiation happening between two participating nodes

where Diffie Hellman key exchanges lead to temporary TCP connection creation and forwarding of data through these connections. An OR can have multiple circuits which in turn have multiple TCP streams of data. An ideal circuit length is three hops as specified by the Tor design paper [1].

- **Directory Authority (DA):** A semi-centralized authority that keeps the record of participating Onion routers, performance characteristics and onion addresses data structures. There are 11 directory authorities that remain synchronous with respect to data that each DA possesses. Their location is hard coded onto the Tor software bundle. The DAs periodically send and sign a consensus containing the current list of active Tor exit relays in the network that is voted and agreed and published.

- **Socks proxy:** An underlying protocol capable of routing any TCP packet through a proxy server. This protocol is used by Tor to allow custom application plugin support, unlike I2P.

### 3.1.2 A Private Tor Network

Tor provides developers and researchers with the capability of creating custom private Tor networks through configuration modifications or using one of their prebuilt tools for some use cases. This tor network has its own directory authority, onion routers and onion clients with no data allowed to go out and vice versa. This is to prevent the real Tor network from obtaining erroneous data that could possibly lead to loss in anonymity for the nodes involved.

Tor is actively under the process of creating utilities like Shadow[1] and Chutney[2] for better private network creation, but they did not fit our use case of modifiable circuits and directory authorities and did not give us the freedom to tweak parameters a lot, hence our decision to stick with using the Tor daemon and modifying the `.conf` files to create a simple private network for testing purposes.

### 3.1.3 Custom Circuits

We now explore the ControlPort[3], the telnet utility provided to us by the Tor Daemon to view various process parameters including the available Onion routers, created circuits and also allow us to change these parameters and create or extend our own circuits. This is especially useful in a private Tor network where circuit modifications is needed for effective experimentation.

We use the Python `Stem` library[4] created by the Tor Open source community, which is an extension to the now deprecated `TorCtl` library. The library involves creation of a `Controller` object that can be used to execute functions. These functions are wrappers for the telnet

[1] https://shadow.github.io/
[2] https://gitweb.torproject.org/chutney.git
[3] http://www.thesprawl.org/research/tor-control-protocol/
[4] https://stem.torproject.org/

commands alongside better exception handling and execution. We use the `Stem` library to create scripts that help us build, list and remove circuits in our private Tor network.

### 3.1.4 Tor Hidden Services

Tor makes it possible for users to hide their locations while offering various kinds of services, such as web publishing or an instant messaging server. Using Tor's "rendezvous points", other Tor users can connect to these hidden services, each without knowing the others network identity. This enables a feature known as *responders anonymity*. Location-hidden services allow users to offer a TCP service, such as a webserver, without revealing their IP addresses. This type of anonymity also protects against distributed DoS attacks: attackers are forced to attack the onion routing network since they do not know the target's IP address [1].
A hidden service needs to advertise its existence in the Tor network before clients will be able to contact it. Therefore, the service randomly picks some relays, builds circuits to them, and asks them to act as introduction points by telling them its public key. In a step-by-step manner, the hidden services mechanism may be explained as (assume Bob is the hidden services provider) [5]:

1. Bob picks up some introduction points and builds circuit to them.

2. The hidden service assembles a hidden service descriptor, containing its public key and a summary of each introduction point, and signs this descriptor with its private key. It uploads that descriptor to a distributed hash table. The descriptor will be found by clients requesting XYZ.onion where XYZ is a 16 character name derived from the service's public key. After this step, the hidden service is set up.

3. A client that wants to contact a hidden service needs to learn about its onion address first. After that, the client can initiate connection establishment by downloading the descriptor from the distributed hash table. If there is a descriptor for XYZ.onion (the hidden service could also be offline or have left long ago, or there could be a typo in the onion address), the client now knows the set of introduction points and the right public key to use. Around this time, the client also creates a circuit to another randomly picked relay and asks it to act as rendezvous point by telling it a one-time secret.

4. When the descriptor is present and the rendezvous point is ready, the client assembles an introduce message (encrypted to the hidden service's public key) including the address of the rendezvous point and the one-time secret. The client sends this message to one of the introduction points, requesting it be delivered to the hidden service. Again, communication takes place via a Tor circuit: nobody can relate sending the introduce message to the client's IP address, so the client remains anonymous.

5. The hidden service decrypts the client's introduce message and finds the address of the rendezvous point and the one-time secret in it. The service creates a circuit to the rendezvous point and sends the one-time secret to it in a rendezvous message.

6. The rendezvous point notifies the client about successful connection establishment. After that, both client and hidden service can use their circuits to the rendezvous point for communicating with each other. The rendezvous point simply relays (end-to-end encrypted) messages from client to service and vice versa.

## 3.2 Anonymous Authentication

Authentication is the act of confirming the truth of an attribute of a single piece of data (a datum) claimed true by an entity. In contrast with identification which refers to the act of stating or otherwise indicating a claim purportedly attesting to a person or thing's identity, authentication is the process of actually confirming that identity [14]. The idea of *Anonymous Authentication* then sounds quite paradoxical: how can someone verify his/her identity without revealing it? Before answering this question, we first look at a few cases where anonymous authentication might be of use.

1. Consider leakage of sensitive secret information related to some organization by a *member of the organization itself*. How can one in such a case leak information anonymously while at the same time proving that the information leaked is genuine? To do so, some sort of authentication mechanism is required wherein the whistle blower is able to prove to the third party his/her authenticity without revealing any personal information.

2. Anonymous authentication can also be employed in cases wherein a governing body may restrict the usage of a particular resource to a set of users having a defining trait without necessarily caring about the identity of each individual in such a set. For example, a department head at some educational institution may want to restrict access to the department lab only to his/her own students and assistant(s), with no interest in their identity.

3. Private forums requiring their users to possess certain knowledge/skills (without any knowledge of their identity) before granting access might also benefit from such schemes.

Having made a case for anonymous authentication, we would now like to discuss how one may go around building such a scheme. Most of the foundations for anonymous authentication have been discussed pretty well by Andew Lindell [15], so our discussion is more or less a brief discussion of his work.
There are two security requirements on an anonymous authentication protocol:

1. **Secure authentication:** No unauthorized user should be able to fool the server into granting it access (except with very small probability).

2. Anonymity: The server should not know which user it is interacting with.

To achieve the first requirement, we rely on asymmetric and challenge-response authentication measures that are commonly used for key exchange in various ubiquitously used protocols.
For the second requirement, we need to first and foremost realize that there is no concept of anonymity when we have only a single user to consider. We need to have a group of users with a certain defining trait. We refer to such a group as a $K$ anonymity set, referred to as $K_s$ from here on. This set basically consists of all users which we know to be genuine, but do not require the identities of. We say that a server learns that it is interacting with a user that belongs to a defined set of authorized users, but nothing more about which user it is in that set. A protocol satisfying above two requirements is called a secure protocol for anonymous authentication.

### 3.2.1 A Simple Secure Protocol for Anonymous Authentication

The idea that we use is to ensure the same behavior by every user, irrespective of its identity. Informally speaking, the protocol works by the server first encrypting a random challenge $w$ under all of the users public keys. The user then decrypts the ciphertext associated with its private key and obtains the plaintext $w$. The user then returns $w$ to the server, proving its ability to decrypt. Since only authorized users can decrypt, this in turn proves that the user is authorized. If the server follows the protocol, this provides perfect anonymity (because all users would obtain the same $w$). We now describe this protocol formally just like it has been described in Lindell's paper.

- **Input:** The user $U_i$ and server $S$ both have $l$ public keys $pk_1, pk_2, ..., pk_l$ for an encryption scheme, and $U_i$ has the private-key $sk_i$ associated with $pk_i$, for some $1 \leq i \leq l$
- **Protocol:**
  1. The server $S$ chooses a random string $w$ of length $n$ and random coins $r_1, ..., r_l$ such that each $r_j$ is of the length needed to serve as randomness for encrypting $w$ under $E$.
     For every $j = 1, .., l$, the server $S$ computes $c_j = E_{pk_j}(w; r_j)$.
     $S$ sends $c_1, .., c_l$ to the user $U_i$.
  2. Upon receiving $c_1, .., c_l$, the user $U_i$ computes $w = D_{sk_i}(c_i)$ and sends $w$ back to $S$.
  3. Upon receiving a string $w_0$ from the user, the server $S$ grants access if and only if $w_0 = w$. If access is g ranted, $S$ sends $r_1, ..., r_l$ back to $U_i$.
  4. User $U_i$ verifies that for every $j = 1, .., l$ it holds that $c_j = E_{pk_j}(w; r_j)$. If not, it outputs $cheat_S$ and halts. Otherwise, it is granted access and continues.

## 3.3 Single Packet Authorization (SPA)

For achieving selective remote access to a particular system, it is necessary to be able to convey pre-shared information in a manner that allows only the intended hosts to connect, while not

allow the attackers an opportunity to even passively observe the communication. Port knocking was devised earlier as a method to securely transmit data through a known set of port connection sequences. [8]

Single Packet Authorization (SPA) is an advancement made over the traditional Port knocking technique for effective and simple remote administration. It is primarily a protocol that allows for a 'stateful' application of remote authentication and usage of a particular resource on a closed system [11]. It uses passive OS fingerprinting and the strengths of Port knocking to create a scheme allowing for restricted remote access to a server while allowing for data to be transmitted through the application payload.

SPA also inherits the use of a passive filter from Port knocking, where all connection attempts are scanned through `libpcap`[5] or similar libraries. All protected ports have a default DROP policy for packets and the passive filtering is done for the majority of packets not associated with the knock sequence, or in the case of SPA, the authorization attempts [11]. Critical services can hence be safeguarded by default and zero-day attacks on services, if present can be avoided all together. SPA also avoids port scanning, which could send off alarms on some Intrusion Detection Systems, allowing for a cleaner approach to remote user authorization.

### 3.3.1 Advantages over Port Knocking

Comparison leads us to observe that the SPA protocol has been designed to offer advantages over the traditional method of port knocking.

- **Larger size of payload transmitted:** Transmission of 2 bytes per attempt limits the data transmitted over each attempt to 2 bytes. If we are insisting on using encrypted payloads, a secure AES-128 would require a minimum of 8 attempts at least to get across the intended message. An SPA packet on the other hand allows for the identifying information to be within the application payload.
- **Single packet transmitted:** For sending a large chunk of information, port knocking insisted on using multiple sequences and hence multiple packets. In an asynchronous and jitter-prone environment, reordering is an unavoidable situation. SPA eliminates all these issues with a single authentication packet.
- **Replay protection:** Having a more sophisticated support of session keys, HMAC usage and encryption allows for prevention against replaying of packets. The packet is only valid for the time window within which the SPA enabled server allows it to, thereby preventing an instance of freshness attack coming into the picture.

[5] http://www.cipherdyne.org/fwknop/docs/SPA.html

### 3.3.2 Firewall Knock Operator (Fwknop)

The Fwknop software [12] is an implemented client server setup that sees SPA in action. The Fwknop server typically runs on port `62201` and is run as root. This server has control over the firewall binary (typically `iptables` is used). It uses either of the standard Rijndael scheme of encryption or can rely on GPG keys to help with the packet encryption. The encrypted payload is sent to to the server where the keys have already been transmitted earlier. The Fwknop client in turn transmits a packet that has the following data fields.

1. Local username
2. Local timestamp (must be approximately in sync with server)
3. Fwknop version
4. mode (Access or command)
5. desired access information (transport layer protocol, port)
6. 16 bit of random data
7. MD5 sum of resultant payload

The Fwknop server typically accepts UDP connections, but can be configured to obtain TCP requests as well. The datagram is sent directly or can be made to send across a `socks` proxy. The packet received is encrypted with the keys that were generated previously (alongside an optional HMAC). This piece(s) of information are stored already on the fwknop server. The keys can either use standard Rijndael scheme of encryption or can be mapped to configured GPG keys. The server tries all keys and on successfully decryption of the payload, it modifies rules as required for a given time period.

Fwknop server by default insists on knowing the IP address from which the request has come to make the connection stronger in terms of security. This can be omitted, but leads to a lesser secure implementation as per the SPA protocol.

We will use the Fwknop client and server to allow our Onion Client to anonymously authenticate itself with some preshared information it had received.

## 3.4 Web Proxying

In computer networks, a proxy server is a server (a computer system or an application) that acts as an intermediary for requests from clients seeking resources from other servers. A client connects to the proxy server, requesting some service, such as a file, connection, web page, or other resource available from a different server and the proxy server evaluates the request as a way to simplify and control its complexity. Proxies were invented to add structure and

encapsulation to distributed systems. Today, most proxies are web proxies, facilitating access to content on the World Wide Web and providing anonymity.
A discussion on web proxying is necessary since our new hidden services scheme is dependent upon it. Broadly, web proxying can be of three different types:

1. **Forward (Regular/Caching) Proxying:** A regular caching proxy server is a server which listens on a separate port (e.g. 3128) and the clients (browsers) are configured to send requests for connectivity to that port. So the proxy server receives the request, fetches the content and stores a copy for future use. So next time when another client requests for the same webpage the proxy server just replies to the request with the content in its cache thus improving the overall request-reply speed.

2. **Transparent Proxying:** A transparent proxy server is also a caching server but the server is configured in such a way that it eliminates the client side (browser side) configuration. Typically the proxy server resides on the gateway and intercepts the WWW requests (port 80, 443 etc.) from the clients and fetches the content for the first time and subsequently replies from its local cache. The name Transparent is due to the fact that the client doesn't know that there is a proxy server which mediates their requests. Transparent proxy servers are mostly used in big corporate organizations where the client side configuration is not easy (due to the number of clients). This type of server is also used in ISP's to reduce the load on the bandwidth usage.

3. **Reverse Proxying:** A reverse proxy is totally different in its usage because it is used for the benefit of the web server rather than its clients. Basically a reverse proxy is on the web server end which will cache all the static answers from the web server and reply to the clients from its cache to reduce the load on the web server. This type of setup is also known as Web Server Acceleration.

# Chapter 4

# Scheme Design and Implementation

We try to implement the idea of extending a regular Tor circuit to have an additional Tor exit node. This node is somehow not known to the directory authority servers and hence the regular Tor users, only those who have somehow obtained the information through different sources. We will first try to implement an hidden exit node using Single Packet Authorization through the Tor private network to allow for only a Fwknop enabled Onion Client to connect to this hidden node.

## 4.1 Setting up Private Tor Testbeds

### 4.1.1 Internal Tor Network

For the initial testing, we use 6 Linux VMs (Operating system: Ubuntu 12.04 LTS) and assign each VM as either an Onion Client, Onion Router and/or a Directory Authority. The VMs have an alternate subnet with IP addresses with the range `10.0.0.217 - 10.0.0.222`. These VMs are provided with a Tor daemon (capable of running both as an Onion Client and an Onion Router) and are configured accordingly.

A brief summary of the configuration is as follows:

- `IP: 10.0.0.217`: Onion Client
- `IP: 10.0.0.218 and 10.0.0.219`: Onion Routers with default Exit policy accepting all IP addresses in subnet
- `IP: 10.0.0.220`: Onion Router with default Exit policy accepting all IP addresses in subnet and only Directory Authority for network
- `IP: 10.0.0.221`: Hidden Onion Router (OR port disabled by default in iptables)
- `IP: 10.0.0.222`: Regular HTTP Server for testing purposes

### Creating a Custom Circuit

We now begin by creating a custom Tor circuit from OR1 to OR2 to DA1 as shown in the diagram. The HTTP requests through regular Tor work perfectly. We now try and send an SPA packet through DA1 to the Fwknop server.

Once the Fwknop should receive the packet, it will open the hidden Tor Onion router, which will be used to extend the circuit to access the HTTP Server. We can test this in the future by only allowing HTTP Server access through the hidden exit node, thereby corroborating our attempt at subverting censorship in a basic implementation. Note that the last part is yet to be completed, as explained below.

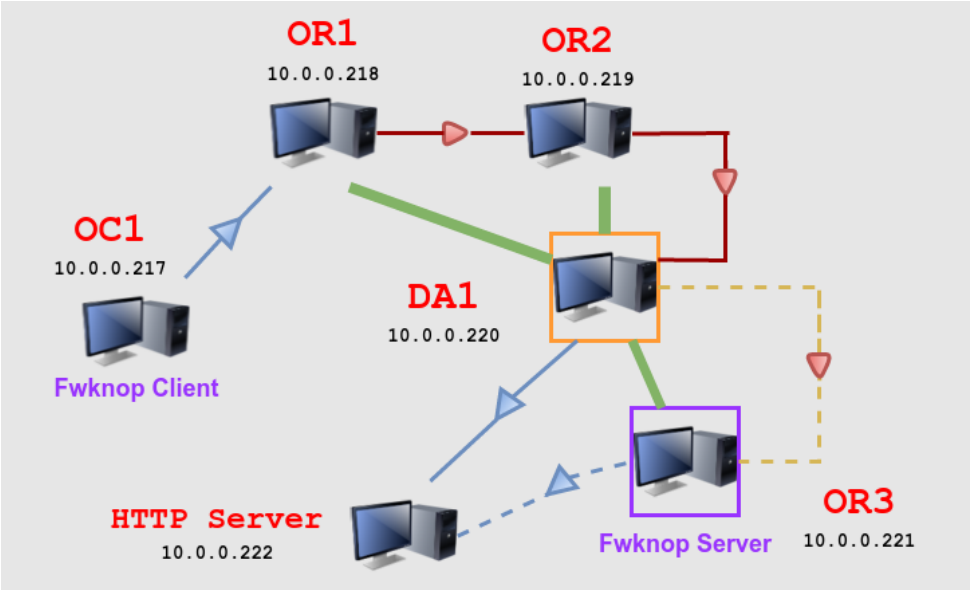


Figure 4.1: The private Tor network visualized with all nodes and their respective goals. OC: Onion Client, OR: Onion Router and DA: Directory authority

### 4.1.2 Using PlanetLab for a private Tor testbed

The internal Tor network had many limitations with respect to testing and developing the scheme viz. the circuit creation could not be randomized beyond a certain limit. This was due to the shortage of exit relays provided to the onion client for circuit creation. Besides, with all the nodes inside the same subnet, Tor behavior was unpredictable, having been designed for large scale distributed networks.

A necessary decision was to migrate the next step of testing to **PlanetLab**, which is an Overlay Testbed for broad-coverage services [9]. The PlanetLab network comprises universities and organizations around the world that volunteer to provide node(s) for testing purposes. For every user who wants to use the network, he/she is assigned a *slice* which is a separate modified VM instance on each of these distributed nodes. The users public key is also distributed to each and every one of these nodes, because of which the user is able to SSH to all of these machines.

| | |
|---|---|
| Total deployed nodes in PlanetLab network | **170** |
| Total nodes active at any given instant | **65-70** |
| Total no. of directory authorities present | **2** |
| Minimum consensus time for DAs | **300 seconds** |

Table 4.1: Active node and other specifications of the network.

For configuring the network, we used a Content Distribution Network (CDN) already deployed on the PlanetLab network called as **Codeploy** [10] that allows for simultaneous file transfers and execution operations to hundreds of servers. After deployment, we have the following characteristics of the testing network.

We create two directory authority servers, one on `planetlab-03.cs.princeton.edu` and the other on `planetlab01.cs.washington.edu`. The directory consensus is updated every 5 minutes.

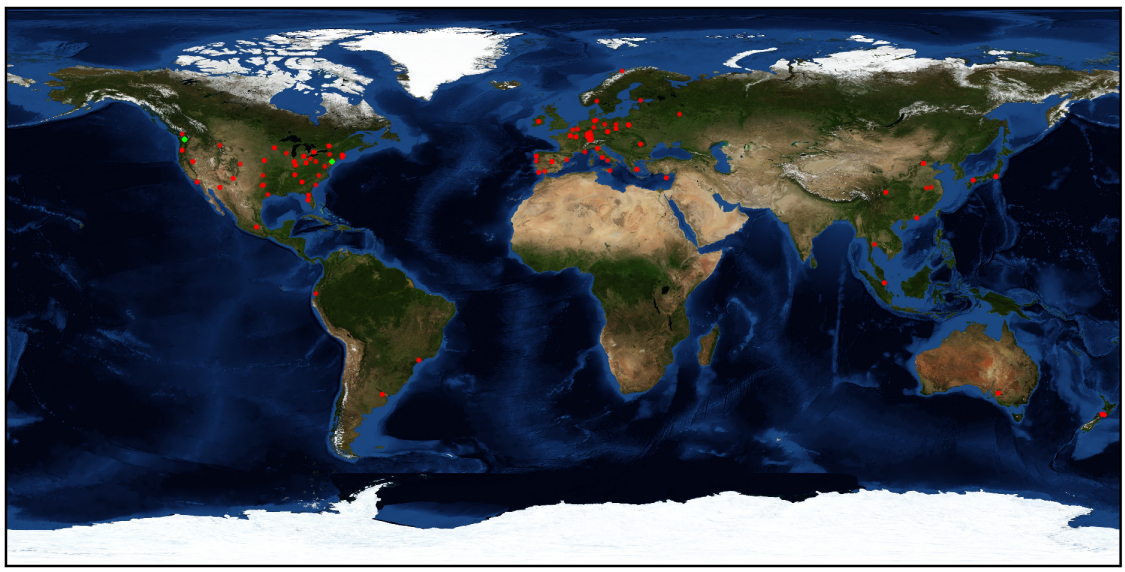

Figure 4.2: All active nodes of the current PlanetLab private Tor network distributed throughout the world, mainly in US and EU universities. The red dot represents all active Tor exit relays and the green dots represent the directory authorities.

The Tor network created provides us with the needed randomization and global consensus distribution to allow for better testing of our scheme. Unlike an internal Tor network with 6 nodes, we can have a multitude of circuits created (a total of C(70,3) at any given instant). This network can also be used for testing performance of other software over the Tor network.

There are still some similarities and differences when we compare this network to the actual public Tor network:

- Both support hidden services, which will be helpful later on while trying to implement our 2nd scheme.
- Private Tor network has a minimum allowed consensus gap of 5 minutes, as compared to an hour for the public Tor network.
- Both networks are distributed throughout the world and have circuit creations from any exit node that is active.
- The private Tor network will provide a lesser latency as compared to the public Tor network. This is because unlike public Tor, our testbed is not being used for common networking applications.

## 4.2 Hidden Exit Design and Implementation

### 4.2.1 Using Fwknop on the internal network

For the internal network, we will assume the Onion Client (`10.0.0.217`) to also act as the Fwknop client who sends the SPA packet and a hidden Onion Router (`10.0.0.221`) to act as the Fwknop server. The Fwknop server will by default disable the Onion Router port and disallow any Tor connections. Only when an SPA packet has been passed through and successfully decrypted and checked, will the Fwknop server open that particular port in a stateful manner for a given time frame only for the specified IP address (in this case, the initial Tor exit node).

- `IP: 10.0.0.217`: Fwknop client and Onion Client
- `IP: 10.0.0.221`: Fwknop server and hidden Onion Router (OR port disabled by default in iptables)

We use `socat` for the public Tor network as mentioned in Fwknop's official documentation. However, while using Fwknop on a private Tor network, we encountered issues with socat and instead decided to create a custom socks client using the Python `socksipy` module. The socks utility we created uses the socks port provided by Tor (usually `9050`) to send TCP packets across the Tor network with the custom circuits created.

The Fwknop server, as mentioned above, uses UDP datagrams by default as it is faster. However, the Tor design demands having only TCP traffic inflows to counter packet bursts and congestion. This requires using the server to be configured to be used in the TCP mode with the `PCAP` filter being set for TCP ports instead.

The keys generated at the client end have already been shared (with an optional HMAC entry as well). These keys reside with the Fwknop server for one use case. Multiple keys can be generated accordingly.

Now, with the socks client in place, we create a connection from the host to the intended receiver using the socks proxy. This creates a bi directional tunnel that allows packets from the source to travel to the destination. We then send the SPA packet through the Tor network to the Fwknop server. The Fwknop packets reach the destination and the payload is to be then processed by the Fwknop server.

We need to ensure that the IP address for which the packet is valid needs to be the current Tor exit node in order for this to work. The SPA packet is anonymously sent across the private Tor network and the payload is received anonymously.

### 4.2.2 Scheme details and flowchart

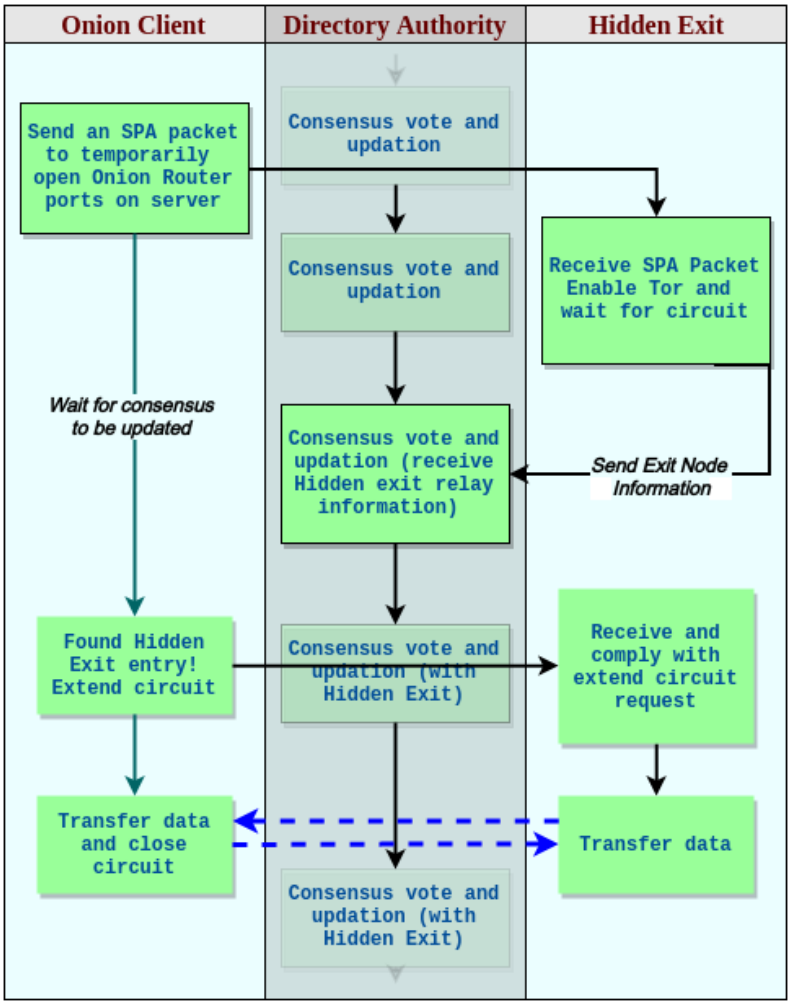


Figure 4.3: A flowchart to represent the scheme as seen from the onion client, the directory authorities and the hidden exit.

For using Fwknop effectively across the PlanetLab Tor network and subsequently across the public Tor network, we devise a scheme that involves using Single Packet Authorization (SPA). We send an SPA packet across the Tor network to an exit node which is disabled by default. The incoming and outgoing traffic of the exit relay is moderated using Fwknop, which only allows

temporary established connections to be created *immediately after* an SPA packet is received and successfully deciphered.

The server runs a Tor daemon that has its port traffic stopped by the Firewall Knock operator. This prevents any connections to be made as well as the hidden exit to be made known through the public consensus.

The client asynchronously creates new Tor circuits to be extended later. It then sends an SPA packet through one of the circuits to temporarily open the ports for the Tor server daemon. This is for the onion relay to successfully communicate with the Directory Authorities (DA). These ports are *only* opened to the directory authorities and to the exit node of the circuit used. The client then checks the consensus at regular intervals to see if the hidden exit has been added or not.

The Directory Authorities have a consensus after a fixed interval (which is set to 300 seconds in the private Tor network). As soon as the directory authorities update the consensus, the client detects changes in the consensus and updates its own. Then it tries to extend the same circuit used for sending the SPA packet to the hidden exit.

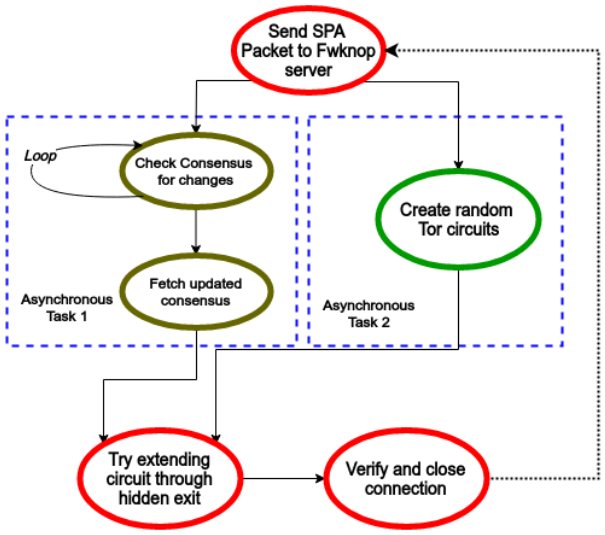


Figure 4.4: Client flowchart for connecting to the Hidden Exit.

### Shortcomings of scheme

The scheme is still currently being tested, but over the first period of trials, we face the following shortcomings.

- A major constraint of this scheme is that we are bounded by the periodic consensus times for the directory authorities, making the scheme slower to use.
- Since the SPA packet must specify the duration for which the ports are kept open, estimation of how time the previous states might take could go wrong and a connection could fail.

## 4.3 Anonymous Authentication Design

Traditional anonymous authentication scheme (as discussed in Chapter 3, section 3.2) requires public-private key operations to function as it's supposed to. Indeed, the importance of PKI cannot be doubted. However, asymmetric cryptography has always faced issues related to speed. Even in modern day scenarios, it's used (mostly) for key exchange purposes. The verification step, that encrypts or decrypts using either the public or private key is a *costly one*, and therefore we can't really use it directly for our scheme since Tor itself creates noticeable latency issues. For this purpose, we need to come up with an alternative approach, one that should be scalable and easily implementable. For this purpose, we present a brief discussion of challenge response authentication mechanisms, and then proceed towards a description of our (primitive) scheme.

### 4.3.1 Challenge-response Authentication

Challenge response authentication [16] protocols require one party to authenticate itself to another by solving a simple challenge. The simplest examples of the same are CAPTCHAs, which only need to authenticate whether the user is human or not by posing a test that cannot be solved by a classical computer. We chose a challenge response authentication system because it necessarily eliminates all the complex computations and still manages to produce a fairly secure scheme. We thought that Merkle's puzzles were a really novel idea, and we plan on adapting that for our use case. Said puzzles allow two parties to agree on a shared secret by exchanging messages, even if they have no secrets in common beforehand. In a sequence of simple steps, this challenge response mechanism can be laid out as:

1. Let $A$ and $B$ be two communicating parties.
2. $A$ creates n puzzles and sends them to $B$.
3. Each of these puzzles have a unique random ID and a secret code $k$. They can be solved with a moderate effort $m$.
4. $B$ randomly solves any one of these and sends the ID back to $A$.
5. Both of the parties now possess a shared secret $k$ using which they can communicate easily.

$A$ requires $n \times m$ effort, $B$ requires $n + m$ effort. Also, the asymmetry in this system is quadratic compared to the case of **RSA** where it was exponential.

For our scheme, we propose a simple solution which has a challenge response kind of scheme at its heart. First, however, we'd like to mention that our scheme has to be **necessarily** centralized. Basically, we need a definition of authenticity that can be conveyed to the hidden exit node in a way that also retains anonymity of users. We depend on a *secret certifying authority* for the same, referred from here on as **SCA**. Assuming the hidden exit node trusts our SCA, we

can proceed with authentication by defining a certain heuristic which may be understood only by 3 parties:

1. Hidden exit node.
2. SCA.
3. User who demands anti-censorship.

We do not address the problem of how a user might approach SCA, simply because it is not of concern to us. SCA may be advertised in private forums, or through IRC channels etc. The only two requirements are:

1. Address of SCA must not be made public, to prevent common denial of service as well as more sophisticated access-oriented attacks.
2. This address may be communicated only after some sort of verification has been done through communication (or other means) to ensure that:
   (a) User is a human.
   (b) User may not cause harm to the SCA in any manner (although it's impossible to have such a guarantee, we can still secure some amount of trust).

Considering our aforementioned presumptions hold, we can specify the idea broadly as:

- User $A$ contacts the SCA for access to the hidden exit node.
- SCA, after procuring some amount of confidence that $A$ is genuine and may not harm the SCA, exchanges the address of the hidden exit along with a simple sequence of *unique* steps that will help $A$ in solving SCA's challenge.
- $A$ extends its Tor circuit to incorporate the hidden exit.
- The hidden exit node now sends a challenge back to $A$. If $A$ had been able to verify itself to the $SCA$, it must also have the knowledge of how to solve the challenge.
- $A$ solves this challenge and sends the result back to hidden exit using a secure channel.
- The hidden exit, now seeing that a *valid* result arrived from some user in the distributed Tor network, knows that whoever pinged it must *also know the secret*, and hence must have been verified by the SCA itself.
- Above conditions fulfilled, the hidden exit now accepts connections from $A$ without caring about its identity.

It's easy to see from here that $A$ may be just another user in the worldwide Tor network, thus no harm may come to its anonymity.
We again remark that this scheme is pretty much speculative right now, and we intend on making it concrete, implementable and scalable along with presenting proofs of correctness of the same in the second half of our project.

## 4.4 Hidden Services Design

We propose to exploit the hidden service design by using the hidden server as a layer 2 relay, or a transparent proxy. Assuming that the end user has a browser that handles onion URLs properly and the hidden service has been set up properly, the scheme works as follows:

1. The anti-censorship service provider first enables IPv4 forwarding on the hidden server and sets up `iptable` rules as follows:
   `iptables -t nat -A POSTROUTING -o eth1 -j MASQUERADE`
   `iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 80 -j REDIRECT --to-port`
   Where the `--to-port` directive is followed by the port number which has a proxy server listening.

2. The client crafts the request for a particular web resource as: `http://<16 digit onion URL>.onion/website.com`.

3. The URL is first resolved to the actual hidden server, which sees the request for the resource after the delimiter ('/').

4. The traffic has now been redirected to the proxy server, which should be advanced enough to carry out three important tasks on the fly:

   (a) Append the proper protocol in front of the requested web resource after '/', since the said resource isn't local to the hidden server.

   (b) Change the source IP to that of itself, so that when a particular censored service is requested, it replies to the hidden server (and not to the client running Tor directly).

5. The proxy server now acts as a layer 2 NAT middlebox that relays the traffic back and forth between the client and the end web service.

It is easy to infer that the end web service cannot deny service to the original Tor user since it sees the hidden server as the effective client. This hidden server interacts with the requested web service directly, without going through the public Tor network, and hence appears as a legitimate user to the web service.

# Chapter 5

# Website Classification

A considerable part of the project also aims at categorizing a website's behavior as either Tor blocking or allowing. This not only helps us in creating a data set on which we can apply our anonymous authentication scheme to produce convincing results, but may also serve as a test-bed for future research along the same lines.
When they detect Tor users, web services may either:

- Block Tor users. Yelp for example, is a popular web service that exhibits such behavior.
- Detect and give warning(s)/take away functionality.
- Allow full access without any penalties.

In cases when services outright reject Tor, one may directly compare the HTTP status codes to derive a conclusion. However, most of the services that we've come across only *cripple* the anonymous user by taking away more advanced functionality and allowing only very basic facilities to be used. Amazon, as an example, allows Tor users to fully browse the catalog of products and add them to their wish list, but does *not* allow said users to buy them. This significantly harms the end user, since Amazon markets (and functions) itself as an online marketplace. Other web services like Wikipedia do not allow editing of articles by Tor users. While such a move is understandable (since anonymous users might modify the existing content in a malicious manner), some might want to genuinely contribute valid information, resorting to anonymity only for social or political reasons. Such services pose a problem in constructing an effective classifier. Intuitively, it's not possible to construct a deterministic algorithm in this case, since the variables are too many. Some websites may choose to only put up a warning on the web page (requiring scanning of raw text), others may modify various components of the response headers. More sophisticated ones might behave normally for most of the services offered, only choosing to take action when more sophisticated parts are visited.
It's fairly easy to see, therefore, that the complexity of making such a classification may not be polynomial. Also, since all parameters must be compared for all websites, we have to resort to brute force. It is possible, however, to speed up the process by usage of multi-threading for

concurrency, since classification of a particular website does not depend on some other (presumably distinct) website. Here we present the algorithm that we have employed to make such a classification:

**Data:** List of websites
**Result:** List of websites along with their behavior towards Tor
initialization;
**for** *each website in the list of websites* **do**
    read url of website;
    **r1** = url response without Tor;
    **r2** = url response with Tor;
    **if** *r1.status_code != r2.status_code* **then**
        classify website as Tor blocking;
        fetch url for next website from the list;
    **end**
    **else if** *r1.header_length != r2.header_length* ***or*** *r1.content_length != r2.content_length* ***or*** *r1.content_location != r2.content_location* ***or*** *r1.server_type != r2.server_type* ***or*** *"Tor" in r2.raw_text* **then**
        report website as Tor allowing;
        fetch url for next website from the list;
    **end**
    report website as Tor allowing;
    fetch url for next website from the list;
**end**

**Algorithm 1:** Website Classification Algorithm

This can be sped up by computing results for multiple websites in parallel by usage of multithreading libraries, such as **OpenMP**. We were able to classify around 1000 websites using this algorithm. The results were satisfactory, although we do aim to further improve this algorithm to check for more parameters, ultimately aiming for lesser false positives and negatives.

# Chapter 6

# Further Work

With respect to the initially devised milestones and the future work from the last semester, here is what milestones are currently in the midst of being finished:

- Collecting more data and testing the Fwknop Hidden Exit scheme further over the private and public Tor network. Through this data, we can present some specifications as to the time taken for the connection to be established.
- Successfully implement the Hidden Services model and begin testing on the public and private Tor networks.
- (Optional) Implementation of both, as well as measures of efficiency, scalability etc. using C or equivalent for better integration into the current software bundle provided without considerable tweaking.

# Bibliography


[1] Roger Dingledine, Nick Mathewson, Paul Syverson, **Tor: The Second Generation Onion Router**, *SSYM'04 Proceedings of the 13th conference on USENIX Security Symposium.*

[2] Whitfield Diffie, Susan Landau, **Internet Eavesdropping: A Brave New World of Wiretapping**, *Scientific American*

[3] Chaum, David L. **Untraceable electronic mail, return addresses, and digital pseudonyms.**, *Communications of the ACM 24.2 (1981): 84-90.*

[4] Chaum, David L. **Blind signatures for untraceable payments.**, *Advances in cryptology. Springer US, 1983.*

[5] Tor Hidden Services Blog Entry

[6] 4chan

[7] Goldschlag D., Reed M., Syverson P.,**Onion Routing for Anonymous and Private Internet Connections**, *Onion Router, 1999.*

[8] Krzywinski, Martin. **Port knocking from the inside out.**, *SysAdmin Magazine 12.6 (2003): 12-17.*

[9] Chun, Brent and Culler, David and Roscoe, Timothy and Bavier, Andy and Peterson, Larry and Wawrzoniak, Mike and Bowman, Mic., **Planetlab: an overlay testbed for broad-coverage services**, *ACM SIGCOMM Computer Communication Review*

[10] Park, KyoungSoo and Pai, Vivek and Peterson, Larry, **CoDeploy: A scalable deployment service for PlanetLab**, *2004*

[11] Rash, Michael. **Single packet authorization with fwknop.**, *The USENIX Magazine 31.1 (2006): 63-69.*

[12] Rash, Michael. **Combining port knocking and passive OS fingerprinting with fwknop.**, *USENIX; login: Magazine 29.6 (2004): 19-25.*

[13] Whitfield Diffie, Susan Landau, **Internet Eavesdropping: A Brave New World of Wiretapping**, *Scientific American*

[14] Authentication Wikipedia Entry.

[15] Andew Y. Lindell, **Anonymous Authentication**, *Aladdin Knowledge Systems Inc.*

[16] Challenge Response Authentication Wikipedia Entry.