5-20-2011

Private Information Retrieval in an Anonymous Peer-to-Peer Environment

Michael Miceli
University of New Orleans

Private Information Retrieval in an Anonymous Peer-to-Peer Environment

A Thesis

Submitted to the Graduate Faculty of the
University of New Orleans
in partial fulfillment of the
requirement for the degree of

Master of Science
in
Computer Science
Information Assurance

by

Michael Vincent Miceli

B.S. Louisiana State University, 2009

May, 2011

Acknowledgment

This work would not have been possible without the guidance of Dr. Mahdi Abdelguerfi and a grant from the Louisiana Board of Regents. I'd like to thank the members of my committee: Dr. Bilar, Dr. Richard III, and Dr. Tu. Also, my experience at Louisiana State University provided a solid foundation. Specifically, I'd like to thank Dr. Hartmut Kaiser, for his profound mentoring abilities and programming expertise; and Dr. Shantenu Jha, for his ability to push me to research and success. Many talks with a personal friend, Rodrigo Farnham, have also helped produce this research.

Table of Contents

List of Figures

Abstract

Private Information Retrieval (PIR) protocols enable a client to access data from a server without revealing what data was accessed. The study of Computational Private Information Retrieval (CPIR) protocols, an area of PIR protocols focusing on computational security, has been a recently reinvigorated area of focus in the study of cryptography. However, CPIR protocols still have not been utilized in any practical applications. The aim of this thesis is to determine whether the Melchor Gaborit CPIR protocol can be successfully utilized in a practical manner in an anonymous peer-to-peer environment.

Keywords: Anonymous peer-to-peer, Private information retrieval (PIR), peer-to-peer

Chapter 1

Introduction

1.1 Motivation

The first amendment of the United States constitution guarantees freedom of speech, which allows citizens to express their opinions without fear of persecution. Freedom of speech is necessary for a functioning democracy. However, not every country has this freedom and there also have been many instances in the United States where freedom of speech has been restricted. For instance, after the publication of the over 250,000 leaked United States diplomatic cables in 2010, there was rash reaction from lawmakers. Sen. Lieberman introduced the "Protecting Cyberspace as a National Asset Act of 2010", which would have allowed for government control of Internet service providers (ISP)s, arguing that in a national emergency the government should be able to 'shut down' the Internet. This vague law with no checks and balances would have greatly destroyed freedom of speech. Only in 2011, when Egypt was able to prevent Internet access to citizens in an attempt to unsuccessfully prevent a revolution was the bill revised to remove this 'shut down' idea. This ebb and flow of censorship and regulation has shown a need for the ability to communicate freely without government control.

1.2 Importance of an Anonymous Peer-To-Peer Network

An anonymous peer-to-peer network is a network that allows peers to communicate while remaining anonymous. Creating such a network is very difficult because the network has to immune against any attack that would even help determine original authors of data. Freenet is a prominent anonymous peer-to-peer network. They argue that freedom of speech cannot be guaranteed without anonymity. This is because it is easy to curtail free speech by punishing those who exercise it. Doing so instills fear and

stifles further free speech by others. By allowing anonymity it becomes impossible to punish anyone exercising free speech, because it is unknown who exercised the freedom in the first place. Freenet argues that to ensure the availability of freedom of speech, the government should not be able to control its population's ability to share information freely at all. If knowledge is stifled and censored, then people are uninformed and are not able to adequately determine a complete and accurate conclusion. Therefore, no information should be censored in any way. It is difficult to take such a firm choice when it comes to freedom of speech. There are many arguments for stifling freedom of speech. Child pornography, hate speech, racism, and terrorism are all examples where preventing such information flow seems like a good idea. However, it is too hard to determine whether certain knowledge can fall into these categories, and governments have a record of overreaching in censorship. Also, they have been known to hide information that would shine them in a negative light. Instances of this can be seen in the WikiLeaks releases and the seizure of 84,000 sites, many of which were perfectly legal. [1] However, even WikiLeaks itself censors data to protect citizens that would be in harm's way if uncensored. So, for me it is hard to agree with Freenet completely, but the idea is interesting academically and in many countries Freenet is a viable way to spread knowledge that would otherwise be censored.

1.3 Private Information Retrieval

Private Information Retrieval (PIR) protocols allow a client to access data from a server without allowing the server to know which data was accessed. Until recently, it was accepted that all PIR protocols today and in the near future are impractical. The study of Computational Private Information Retrieval (CPIR) protocols were developed shortly afterwards. However, they also were not practical to use. Recently, there have been many attempts to create a practical protocol. This thesis determines whether CPIR protocols can be successfully used to create a practical anonymous peer-to-peer network.

1.4 Goals of this project

To develop and determine the success of the prototype network, there were 3 main goals are laid out. The first goal is to understand anonymous peer-to-peer networks and why they are anonymous. Then, the next goal is to find a CPIR protocol that would be most successful in with constraints placed on it by the anonymous peer-to-peer network. Finally, a test suite has to be developed to analyze the performance and anonymity of the network. By the end of this thesis, the ultimate goal is to show whether a CPIR protocol can be used in a practical manner for anonymity networks.

Chapter 2

Anonymous Peer-To-Peer Systems

Any distributed application where peers process tasks or work between them is considered a peer-to-peer network. There are 3 different types of peer-to-peer networks: centralized, semi-centralized, and decentralized. A centralized peer-to-peer system uses a central server either to distribute work, or help locate information among nodes. Examples of a central system are Napster and BitTorrent. Semi-centralized peer-to-peer systems do not have a central server; however, they do have nodes that are more important than other peers. These peers accept more traffic and may control a part of the total network. Semi-centralized systems include Kazaa [2] and Skype [3]. These more important nodes are usually called super nodes. Finally, there are fully decentralized peers where all nodes have the same priority and are equally valuable to the network. Freenet and GNUnet are examples of fully decentralized peer-to-peer networks. Semi-centralized and fully decentralized networks have the advantage of being more fault-tolerant. Anonymous peer-to-peer networks are peer-to-peer applications in which peers are anonymous to each other. Peers cannot know who is requesting data and who is sending data.

There are many different roles in a peer-to-peer environment and each role has a different view of the system, including senders, receivers, and intermediate nodes. An attacker could be any of these roles or an outsider such as an ISP. Different anonymity protect against different attackers. For instance, some anonymous peer-to-peer networks use proxies. In this sense senders and receivers are anonymous, but the proxy knows everything. Within each view of the system, there are three levels (See Figure 1). [4]

Beyond suspicion	Appears no more likely to have acted than any other
Probable innocence	Appears no more likely to have acted than not have
Possible innocence	Nontrivial probability that it was not the user

Figure 1: Three types of anonymity (continued)

The purpose of anonymous networks is usually to provide freedom of speech. By limiting the ability of censorship, more information can be available. Anonymous networks are a gray area of the law. While information about corrupt governments can be spread throughout the network without the whistleblower being held responsible, so can child pornography and other illegal information.

The most popular schemes to provide anonymity utilize either proxies (mix nets) or intermediate nodes (crowds, onion routes). The proposed anonymous peer-to-peer system in this thesis is very similar to Freenet and Crowds. These systems provide possible innocence to both senders and receivers of data on the network. The reason that the network is similar to Freenet and Crowds is because there is a performance and bandwidth overhead in these networks which can be optimized by CPIR protocols. Also, they are the most successful anonymous networks. The other types of networks have a central point(s) of failure, which could allow a strong enough opposition to pressure the central point(s) of failure to shut down services, crippling the network.

2.1 Freenet

The prototype peer-to-peer network in this thesis is heavily influenced by Freenet. In the original paper [5], Clarke et al. lay out a framework for a distributed, censorship-resistant, peer-to-peer network called Freenet. It provides both sender anonymity and receiver anonymity to the level of possible innocence. The network was created with five goals in mind: provide anonymity for producers and consumers, provide deniability for maintainers of data, be resistant to attempts to deny access to information, have efficient routing and storage, and be decentralized. In Freenet, every peer contributes part of his hard disk drive space for use in a large distributed data store, where nodes store other nodes' data. Each data item stored on each node's hard disk drive is encrypted using AES (Rijndael), a symmetric algorithm, where the passphrase determined by the creator of the data. While the decryption keys are theoretically available to the node, it is not obliged to find these keys and determine the contents on its local data store.

This provides plausible deniability for nodes that are holding incriminating content, because the node does not and cannot easily know what data it is holding – it is hard to determine what the decryption key is unless the node knows the passphrase. So, the peer holding the data does not know specifically what information he is holding. He only knows that he is holding data that has been inserted into Freenet. A common concern among people using Freenet is that they do not want to support illegal actions, such as storing child pornography, or hate speech for example; however, it is impossible to provide free speech if you are unable to tolerate speech that you do not agree with. The main idea of Freenet is that if the user running a Freenet node is discovered to be holding illegal content, there is no way to know for sure the user knew this. Freenet has yet to be tested in court. This could be because persecutors are aware of how hard a case would be with so much plausible deniability.

When a peer requests a file, it will spread throughout the network (See 2.2 Routing). Therefore, the more popular an item is, the more available it will be for other nodes in the network. As nodes' caches become full, least recently accessed items will be removed. The only way a file will be removed from all of Freenet is if no node requests the file until all caches purge the item. This satisfies Freenet's goal of providing resistance to deny access to information and it also removes ownership of sensitive information.

2.2 Routing

Freenet is anonymous because requests are routed through intermediate nodes, which prevents knowledge of where the original request came from. When a request for data is received by a node in the Freenet network, it will first look in his local data store. If the data is found, then the node will respond saying it has the data. If the item is not in the local data store, the node will ask a neighbor that it thinks is most likely to know which node has the data. This is determined by nodes that have returned "similar" items before. Similarity is defined as the lexicographical difference between the hashes of the description of the data described previously. If a node cannot find a close neighbor, he will send the data requested to a

random neighbor. All nodes will recursively do this until the data is found or the number of hops is exceeded. If a destination is found that has the specified hash, all nodes on the route back to the requestor will cache the data to provide faster access for subsequent requests for that data. They may also randomly decide to change the source to themselves. This provides source anonymity, by preventing a requestor from knowing who originally had the data exactly. If the number of hops is exceeded, an error will be sent back. Note that on a successful request all nodes in the successful route will cache the data. To prevent loops each message has an ID and each node keeps track of recent IDs it has received. Also, the hops to live (HTL) – number of hops – value can be set to any value in a small range by the requesting node to create requestor anonymity. The main idea with this routing algorithm is that the network will adapt to requests, and over time become very efficient. The aim of the routing algorithm is to create a small world network.

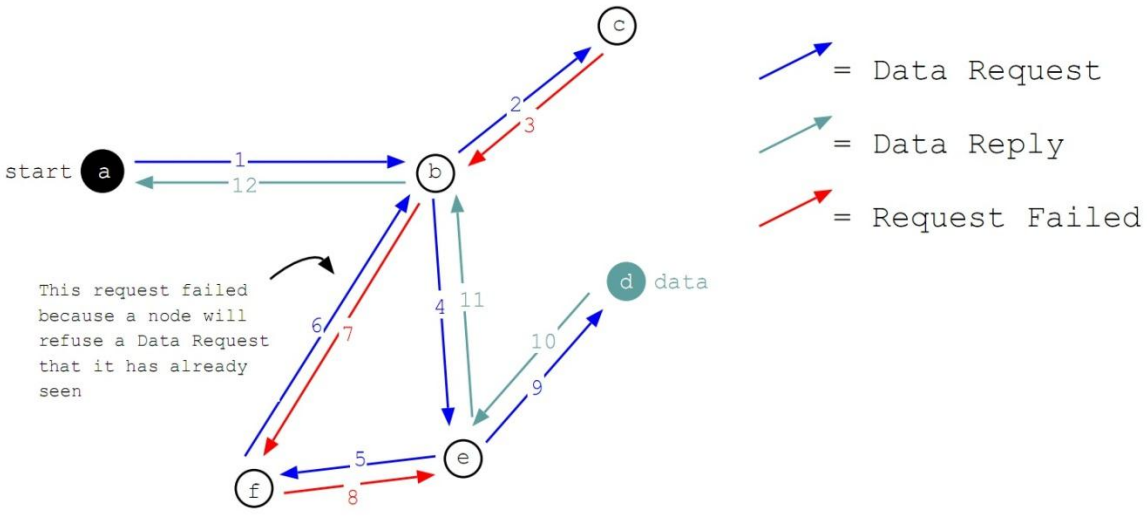

Figure 2: Example of a data request

Freenet relies on the idea of a small world network, which is defined as a network where most nodes are not neighbors of one another, but can be reached by a small number of hops. Small world networks have proven to be a very interesting area of study. In 1967 Dr. Stanley Milgram conducted an experiment that widely popularized the idea of a small world network. He sent several packages to 160 random people

living in Omaha, Nebraska, asking them to forward the package to a friend or acquaintance that they thought would bring the package closer to an individual in Boston, Massachusetts. Milgram chose Omaha and Boston, because of geographic and cultural distance. Surprisingly, the packages that successfully reached Boston, only took on average 5 links. [6] This is the same algorithm as a traveler without a map. The traveler wants to go to a place, but doesn't know how to get there. He can ask a local who knows more about a location how to get to the destination. Although the local may not know how to get to the destination, he probably knows an area that is closer. There the traveler can ask another local, until he gets to the destination. Small world networks are resistant to malicious nodes. In the travel example, a local may give wrong directions (intentionally or unintentionally), but the next local will most likely not give wrong directions again. The traveler will recover and be set in the right direction. Also, note that the traveler must keep track of where he's been to prevent going in a circle.

2.3 Managing Data

This section describes how data is stored, inserted, and removed from Freenet's network. Unlike other peer-to-peer networks, Freenet does not try to store data indefinitely. It acts as a large cache. If an item is popular, it will not be deleted from the network. However, since space is limited by the number of users and the size of each datastore the users set, the items that have been least accessed will be removed.

There are two types of data stored in Freenet: static and dynamic. Static files don't change. Examples of these include mp3, pdf, historical documents, and video. To find these types of files on Freenet a node uses a CHK. A Content Hash Key (CHK) is a three part URI. The first part is a 160 bit SHA-1 hash of a descriptive string identifying the data. The second part is an encrypted decryption key to decrypt the file. The last part contains some decryption settings. Suppose a Freenet user obtains a CHK and its descriptive string, which is used to decrypt the decrypted encryption key, by an out of bounds means, such as IRC, mailing lists, or fproxy. To obtain the file associated with the key, the user sends a request for the key,

which is then routed into Freenet. When the user receives the file he decrypts the key and then the document. All nodes that have helped obtain the file have also successfully cached the file. However, these users do not know what the contents of the file represented by the CHK actually are, because they do not know the descriptive decryption key string.

CHK	1. http://localhost:8888/CHK@SVbD9~HM5nzf3AX4yFCBc-A4dhNUF5DPJZLL5NX5Brs, 2. bA7qLNJR7IXRKn6uS5PAySjIM6azPFvK~18kSi6bbNQ, 3. AAEA—8
SSK	1. http://localhost:8888/SSK@GB3wuHmtxN2wLc7g4y1ZVydkK6sOT-DuOsUoeHK35w, 2. c63EzO7uBEN0piUbHPkMcJYW7i7cOvG42CM3YDduXDs, 3. AQABAAE/ 4. testinserts-3/

Figure 3: Types of keys in Freenet

Content hash keys work well for static content, but suppose a user wishes to publish a website with weekly news. If the user changes the data and reinserts it into Freenet it will have a new CHK. Also, another person could create a very similar website and claim that he is the original author. To prevent this there is another type of file used in Freenet, a Signed Subspace Key (SSK). A SSK will allow a user to have a subdomain. There are 4 parts to an SSK. The first part is a hash of the publisher's public key, which is used for signing. This is the only part of the SSK stored on intermediate nodes. The second part of an SSK is the decryption key used to access the data. The third part contains the decryption settings. The fourth part is a human readable name of a file followed by the version number. The second and third parts are not stored on intermediate nodes. If they were, then the nodes would be able to determine the data and no longer have plausible deniability. The public key serves as the domain. A user can publish many things all in the same domain and viewers know they the same user published all the information by verifying the data was signed with the private key associated with the hashed public key. Although these are the fundamental types of keys in Freenet, there are actually two more that won't be discussed in detail: USK, and KSK. Updateable Subspace Keys are just a wrapper around SSK's to hide the version number.

KSK's are used to access some data using only human-readable URI's. These keys aren't relevant to this thesis.

To insert an item into Freenet, a user simply uses a special message that will attempt to insert the data. However, if there is a collision, the insert will fail. It will also spread the data around more in the network. This prevents malicious uses from trying to override a hash (if, for instance, they found a hash collision). A request is sent to neighbors and if no neighbors report a collision, all nodes will store a copy of the data. The data will be inserted into the network and will remain there until all local datastores drop the file.

Chapter 3

Private Information Retrieval

3.1 Introduction

Private information retrieval (PIR) protocols are protocols that provide a way for clients to request data from a database without the database knowing which data was requested. They are useful in several application domains, such as stock market databases, location based services, and medical databases. PIR protocols are a weakened form of 1 out of n oblivious transfer - introduced in 1981 - where database privacy is not a concern. PIR protocols were first introduced in 1995 by Chor, Goldreich, Kushilevitz and Sudan in [7]. The paper proposed a system that was information theoretic secure and relied on multiple non-communicating servers. They also proved that to be information theoretic secure, you must have multiple non-communicating servers. Later in 1997, the idea of Computational Private Information Retrieval (CPIR) schemes was introduced by both Chor and Gilboa, and Ostrovsky and Shoup. [8] [9] Computational privacy is defined as privacy that is guaranteed against computationally bounded attackers. The first successful implementation was by Kushilevitz and Ostrovsky in [10]. CPIR protocols relax the requirement that PIR protocols must be information theoretic secure; they are computationally secure, i.e., CPIR protocols are secure assuming computation power of today and the near future. Since CPIR protocols are not information theoretic secure, they do not have to rely on multiple non-colluding servers. After Kushilevitz and Ostrovsky proposed the first successful CPIR scheme, there have been many papers describing new schemes. Each protocol focuses on reducing the communication complexity considering the number of elements in database n, block size of each element d, and security parameter k (See Figure 4: Communication complexities of select (C) PIR protocols.).

Scheme	Approximate Communication Complexity
Kushilevitz and Ostrovsky	$O(n^{\epsilon})$ for $\epsilon > 0$
Cachin, Micali, and Stradler	$O(\log^8 n)$
Lipmaa	$\theta(\log^2 n + d \cdot \log n)$
Gentry and Ramzan	$\theta(k + d)$

Figure 4: Communication complexities of select (C) PIR protocols. (cont.)

In the rest of this thesis PIR will be used to represent both PIR protocols as well as CPIR protocols, unless otherwise noted. When studying PIR protocols, the privacy of the user considers the ability of the server to determine which elements are queried. It is not the confidentiality of the client.

3.2 Practicality

There has been a debate over the usefulness of CPIR protocols. An influential paper in 2007 by Sion and Carbunar argue that all single server CPIR protocols are not only impractical today, but also will not be useful in the near future assuming both Nielson's law and Moore's law are stable. [11] While the main goal of (C) PIR protocols has been to minimize communication, Sion and Carbunar show that a low communication complexity is worthless if the computation complexity is so large that the trivial implementation of PIR would take less time. The trivial implementation of a CPIR scheme is to respond to every request with the entire database. This is very bandwidth inefficient; but does ensure client privacy. However, it is very computationally efficient compared to PIR protocols at the time of Sion and Carbunar's paper. Chor, et al. first proposed the idea of a trivial solution and consequently, set the goal of all PIR protocols to minimize communication. Sion and Carbunar's methodology revealed that this is not correct. To prove this they relied on the time a server takes to process (multiply) two integers in the database versus sending a bit in the database over modern networks (Figure 5). This ratio shows that it is much more efficient and always will be in the future to send a bit over the network than to process a CPIR request.

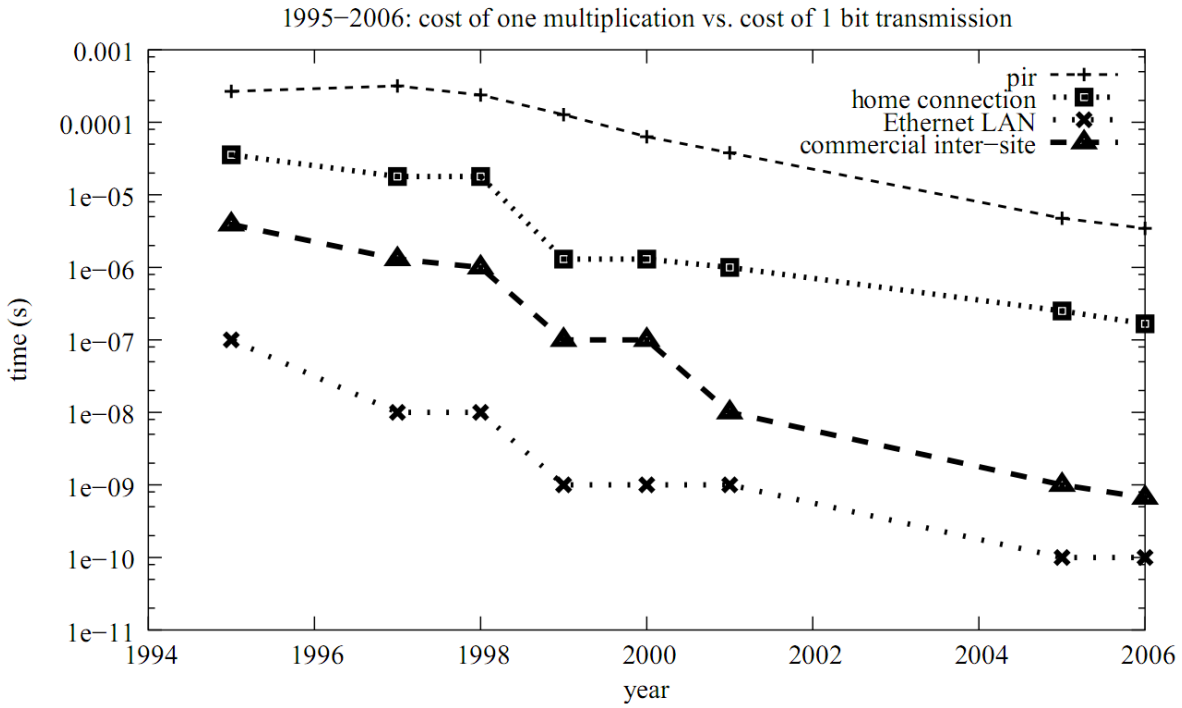

Figure 5: Time to multiply one bit versus time to transfer one bit in recent years.

Pitting CPIR protocols against the trivial solution seemed like a good idea during the PIR's initial development, but computational complexity was ignored. Figure 6 shows the typical amount of time two typical PIR protocols at the time of Sion and Carbunar's paper. The database size was 2.9 GiB and all servers were equal in processing power. The query was for a 3 MiB file.

PIR Protocol	Query Plus Download Time
Limpaa	33 hours
Gentry and Ramzan	17 hours

Figure 6: Common times a typical CPIR scheme would take at the time of Sion and Carbunar's paper

Obviously, these numbers are unreasonable considering today's network speeds. According to Sion and Carbunar, an average home computer can download about .75 Mb per second (6 Mbps). So, downloading a 2.9 GiB database (trivial solution) would take about 66 minutes or about an hour. Consequently, Sion and Carbunar's paper has been used by other researchers to dismiss single server PIR protocols as impractical. [12] In response to this, two papers developed new protocols that were much more efficient

overall, because of increased network utilization: the Melchor and Gaborit protocol, and the Trosle and Parrish protocol. [13] [14] This paper focuses on Melchor and Gaborit's protocol, because they had faster transfer times. Melchor and Gaborit focused on removing the extreme bound on communication, allowing more information to be sent between the CPIR server and client. Their scheme is one hundred times faster in terms of time than previous single database schemes. On top of that, Melchor, et al. takes advantage of the linear algebraic characteristics of their protocol to create further optimizations. Using a GPU and CUDA, they were able to obtain several orders of magnitude faster than their original implementation. [15] With these optimizations, the same query as Figure 6 takes close to 10 minutes. A recent paper by Olumofin and Goldberg, confirm that Melchor and Gaborit's scheme is indeed practical, refuting their own earlier claims. [16]

3.2 Melchor and Gaborit's Scheme

In response to Sion and Carbunar Melchor and Gaborit devised a scheme based on lattices. Lattices were first shown to be useful in cryptography in 1996 by Ajtai. [17] The Melchor Gaborit protocol relies on a new assumed hard problem: Differential Hidden Lattice Problem. This new scheme is very efficient and the one that is used in the experimental anonymous peer-to-peer system this thesis presents.

3.2.1 Details

In Melchor Gaborit's scheme, there are three global parameters: N, p and q. The database is represented as a set of n elements. i_0 is the database element the client is requesting. Each element of the database is encoded as a $L * N$ matrix where L is chosen to be large enough to encode the largest database element. The minute details aren't important in this thesis, but a decent overview is required to understand the security.

3.2.1.1 Query Generation

First, a secret random $N \times 2N$ matrix $M = [M_1|M_2]$ over $\mathbb{Z}_p$ of rank N is used to create a set of matrices. M is multiplied to the left by random invertible matrices to obtain a set of matrices B' of order $N * 2N$. All matrices in the set B' except for the matrix associated with i_0 is combined with an $N * N$ soft noise matrix D_i by $B_i = \left[B'_{i,1}|B'_{i,2} + D_i\right] * \Delta$ where $B'_{i,1}$ is the N leftmost columns of B', $B'_{i,2}$ is the N rightmost columns of B', and Δ is a random $2N \times 2N$ scrambling matrix over $\mathbb{Z}_p$. B_i is called a soft disturbed matrix (SDM). The matrix associated with i_0 is combined with an $N * N$ hard disturbed matrix D_{i_0} in the same manner. This B_{i_0} is called a hard disturbed matrix (HDM). A soft noise matrix is a matrix consisting entirely of $\{-1,1\}$. A hard noise matrix is a soft noise matrix whose diagonal is multiplied by q. The set B is sent to the server along with the prime modulus p.

3.2.1.2 Response encoding

Let $A = [A_1|\cdots A_n]$ be the column concatenation of all database element matrices. First, the server transposes each B_i and then transposes the entire set B. The server then returns $V = A \times B$ over $\mathbb{Z}_p$ a $L \times 2N$ matrix.

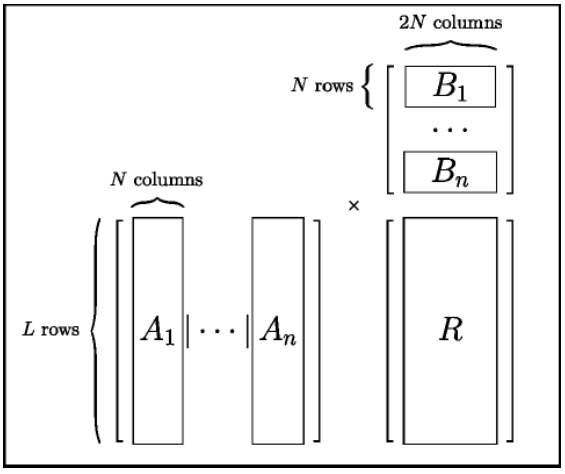

Figure 7: Response encoding

3.2.1.3 Response decoding

Unscramble the result by $V' = V\Delta^{-1}$ to remove the scrambling matrix. Then let $E = {V_{x,1}}' - {V_{x,2}}' * {M_1}^{-1}M_2$. For every element e in E, if $e > p/2$ compute $e' = p - e$. Then, for each e' compute $e'' = e' - \epsilon$ with $\epsilon = e' mod\ q$ if $e' < q/2$ and $\epsilon = e' mod\ q - q$ else. Finally, for $y \in \{1 \cdots n\}$ compute $a_{i_o,x,y} = e'' * q^{-1}$

While these steps for decoding may seem arbitrary they are used because the soft noise inserted into the query should not be more than $q/2$ and the sum of the soft noise and hard noise should not exceed $p/2$. Then the rest is to make sure the sign of the final result is correct. Melchor and Gaborit prove the correctness in their paper.

3.2.2 Lattices

Lattices are a discrete additive subgroup of $\mathbb{R}^m$, i.e. $\Lambda \subset \mathbb{R}^m$ that satisfies:

1. Λ is closed under addition and subtraction (subgroup)
2. $\exists\ \epsilon > 0$ such that any two distinct lattice points $x \neq y \in \Lambda$ are at least distance $\left||x - y|\right| \geq \epsilon$.

Given a set of vectors $b_1, b_2, \cdots, b_n$ in $\mathbb{R}^m$; $n \leq m$ we can define an integer lattice to be the set:

$$L = \sum_{i=1}^{n} x_i \cdot b_i : x_i \in \mathbb{Z}$$

Where the set of vectors b form the basis for the lattice $L(B)$. L is the set of all linear combinations of b. Note that lattices are very similar to vector spaces. The difference is the set x must be from the integers for lattices, where a vector space can be from the real numbers. Usually lattices are represented by a basis matrix B, where the rows are the vectors from b; then to calculate a vector in the lattice, one multiplies B and x^T.

3.2.3 Differential Hidden Lattice Problem

Melchor and Gaborit developed a new cryptographic primitive based on lattices. They admit themselves that the system has not been extensively peer reviewed and may be broken more easily than well-tested cryptographic primitives. The premise is to recognize a special type of lattice between two lattices that have both been modified by. In the Melchor Gaborit protocol, a client essentially gives the server many lattices that they use to send back to the client. If the server can determine which lattice is has certain unique qualities, then the cryptosystem is broken. Each element in B (from the Melchor Gaborit protocol) is a basis of a lattice. The server should not be able to tell which lattice is the HDM lattice.

3.3 CUDA

Compute Unified Device Architecture (CUDA) is an architecture that enables parallelization on Graphics processing units (GPU). The architecture was developed by NVIDIA. GPUs have an architecture that is designed to work on many threads concurrently. As long as applications are able to be written efficiently using stream processing, then CUDA is able to improve performance using concurrency. Stream processing is a programming paradigm where a series of operations are performed on each element in a stream of data independently. Using the GPU has been a highly successful paradigm for embarrassingly parallel tasks. In the Melchor Gaborit scheme these tasks are matrix multiplications. For every element in the database, two matrices are multiplied over integers modulo a prime. While using a GPU is very efficient and effective for many tasks, not all computers have a high end video card that supports CUDA and is powerful enough. So, a peer-to-peer system must also be able to function without a GPU, utilizing CPU resources.

3.3.1 Performance

In Melchor-Gaborit scheme, enabling CUDA provided large performance increases (See Figure 8: Server Processing Time (GPU vs. CPU)). To test the performance, a number of variables had to be isolated: database size, number of items, and item size, database heterogeneity, and whether or not a GPU was used.

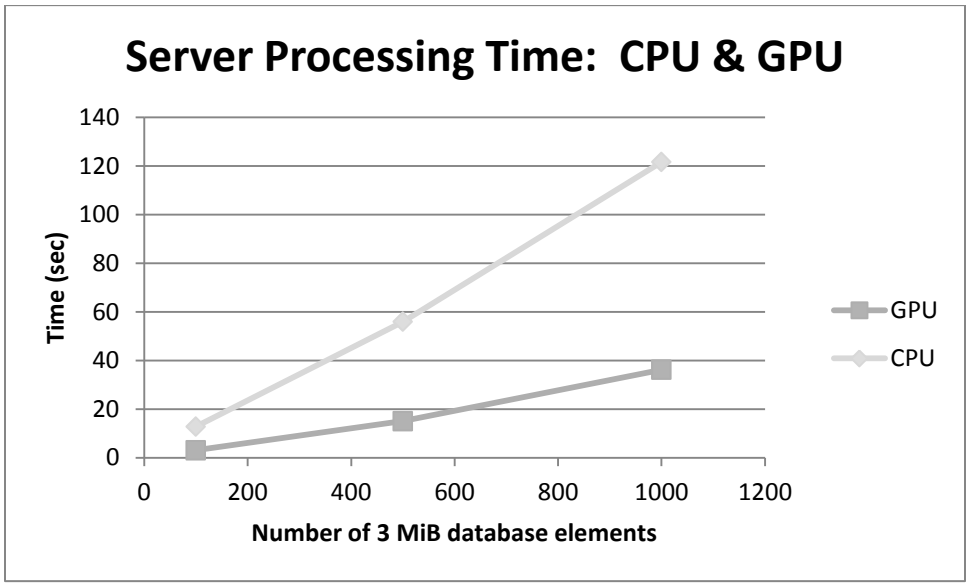

Figure 8: Server Processing Time (GPU vs. CPU)

Upon further experimentation, it became clear that the number of elements does not influence the server processing time as long as the database size is fixed. Also, the size of the data queried does not influence server processing time for homogeneous databases, again as long as the database size is fixed. This is because while the number of files is the database is larger, the total database size is the same and thus requests take the same amount of time to process. (See Figure 9: Server processing time with respect to number of files)

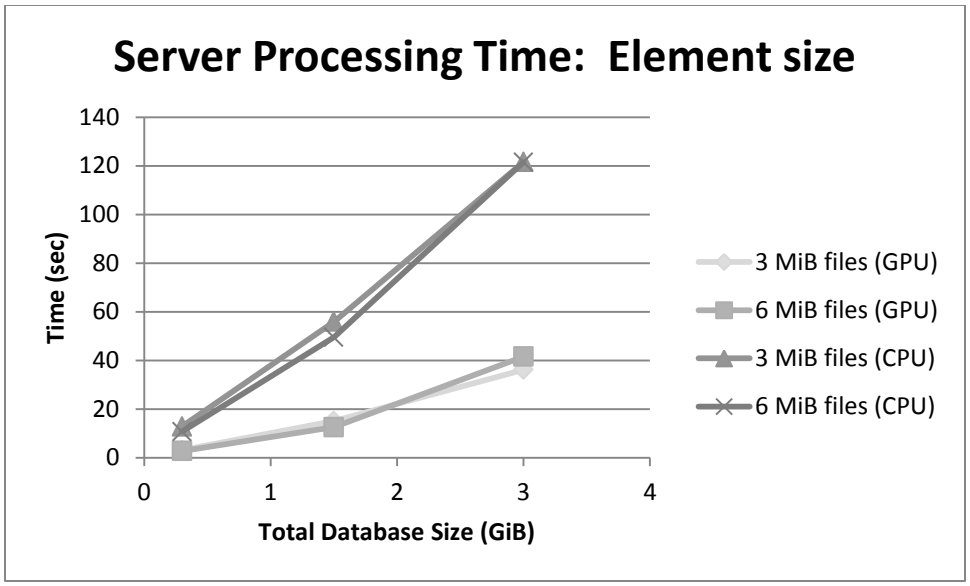

Figure 9: Server processing time with respect to number of files in a homogenous database.

3.4 Overhead

The goal of this section is to determine which parameters would be the most practical in a network where downloading a random file is a common operation. For each query, there is a noticeable bandwidth overhead, which depends on security parameters. In a database with very heterogeneous file sizes, large files will ruin any performance gains on small files. Remember in Melchor Gaborit scheme, the database is encoded as a set of n elements which are encoded as a set of $L * N$ matrices where L is chosen to be large enough to encode the largest database element and N is a security parameter (suggested to be around 50). This means that the matrices representing smaller database elements are filled up using a standard padding technique. This prevents query and response sizes from leaking information about which of the database elements was queried. For instance, querying a 32 KiB file on a database that also has a 500 MiB file will require downloading at least 500 MiB of data. Since requests are nontrivially small, the total bandwidth used would be larger than 500 MiB. This overhead is too much for such a relatively small file. This is a major problem and area of concern on the practicality of CPIR schemes in general. However, in application domains with homogenous data sets like stock pricing data, this is not a

concern. For an anonymous peer-to-peer network that can store any size of data, this is of upmost concern. Outlined below are four common ideas discussed in research literature to solve problems associated with heterogeneous databases and how the Melchor Gaborit CPIR protocol reacts to them.

3.4.1 Solution 1: Split the database into many databases

The total size of database elements affect the server processing time: it greatly increases bandwidth usage as the number of elements increases. Partitioning a single database into many seems like a decent compromise. This lowers the total number of elements in each database, keeping efficiency up. A disadvantage to this idea is that some privacy will be lost because the entire database is no longer being searched. And thus, the server will know which elements weren't queried. Depending on level of paranoia this could be a security concern. As it will be seen in the chapter on the prototype CPIR network, this is not a concern. If during a CPIR request, the server realizes that the only possible element that the requestor is obtaining is incriminating content, the requestor could still claim that it is caching to improve network performance and someone has originally requested the data. This will always be a valid excuse because this is how both Freenet and this prototype CPIR-based protocol work. There is no anonymity in requests on Freenet. Using this CPIR-based protocol only strengthens the requests. There are many ways to partition databases. To optimize bandwidth, databases should be partitioned by filesize, since CPIR protocols need to pad the smallest element to match the largest element in the database. If the range of the elements in each database is small, then the overhead will be reduced.

3.4.2 Solution 2: Each node only stores a certain file size range

Another interesting solution is to partition not the databases on a node but the nodes on a network. This would effectively create multiple networks where each node only stores certain file sizes. This would

increase burden on node storing popular files, because there are less nodes to help caching. Also, nodes will still have to store original requests, and if these are different than the node's file size range, it would be an incriminating factor. Because of this, the former solution is better. It allows all nodes to have an opportunity to cache data. For these reasons, it is ill advised to try to create this type of network, while still trying to maintain the main goals of Freenet, which is what this prototype CPIR network aims to do.

3.4.3 Solution 3: Split each file into chunks

Freenet splits every file into a 32 KiB chunks to provide anonymity from timing attacks, and to improve downloading of larger files by simultaneously obtaining chunks from multiple peers. CPIR schemes usually start to notice their performance gains on larger files, but when splitting a file then a small chunk size should be chosen to prevent internal fragmentation. Internal fragmentation is wasted space due to large chunk sizes. For instance, if Freenet chose a 1 MiB chunk size and had to chunk a 32 KiB file, then 96.875% of the chunk fragment is wasted space. The desire to have both large chunks and small chunks make it hard to find a right balance for such a heterogeneous network like anonymous peer-to-peer networks tend to be.

A compromise scheme could be chunking files as well as partitioning the database. Each node would have many databases of chunked, homogeneous files. This way there is less overhead for the query. This has the same privacy issue in Solution 1, because the entire database is not being searched. However, the overhead is so large on small files that there is no practical way to partition 32 KiB chunked files while still providing proper security. (See Figure 10: Network download speed of CPIR Scheme causes for databases of small files). The average network download speed is assumed to be 6 Mbps and the average upload speed is assumed 2.64 Mbps. [18] The limiting factor would be a node's uploading speed. So, the time it takes to upload a 32 KiB chunk is 94.34 milliseconds with no other delays, such as the load on a node, and propagation delay; consequently, a node requesting data would have to wait at least this time to obtain a chunk. This time is then measured to determine how long a download takes using CPIR for

various database sizes. The download speed decreases exponentially with number of 32 KiB files in a database. In the future a larger chunk size may make more sense as people use more rich data, but right now it is impossible for a CPIR protocol using small chunk sizes to be not only more efficient than Freenet currently is, but also practical. This is true for all practical chunk sizes on one homogeneous database.

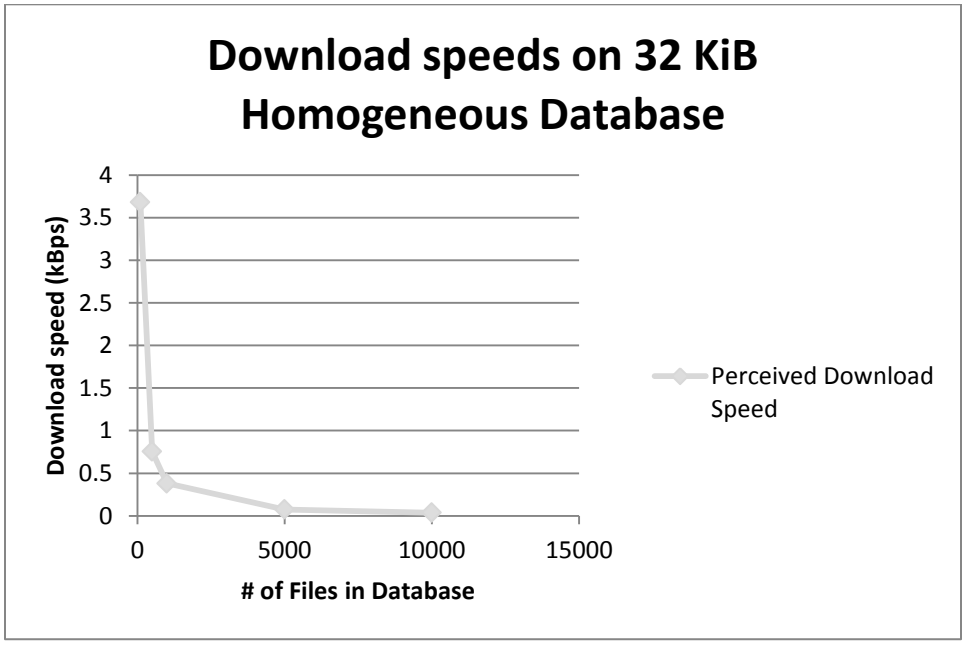

Figure 10: Network download speed of CPIR Scheme causes for databases of small files

3.4.4 Solution 4: Multiple partitions and chunk sizes

As shown above, determining how to deal with heterogeneous databases is very tricky. While chunking creates homogeneous databases, which helps keep query response sizes small, it also increases the number of files in a cache, which negatively affects the query size. Partitioning reduces the number of files in a database. However, partitioning a database will reduce anonymity. A combination of partitioning and chunking is a scheme that has both acceptable download speeds and acceptable requestor anonymity. The first idea is to determine what to chunk for instance, if we allow a larger chunk size and only chunk larger files there will be no internal fragmentation. If the files are smaller than the chunk size,

then they should not be chunked to avoid internal fragmentation. This improves bandwidth by removing unnecessary padding, which hopefully will speed up requests. Smaller requests are most likely requests that would need the lowest latency. Things like html pages and xml documents are probably being used to access larger documents like pdfs and images. Larger documents most likely do not need to be as responsive. Freenet makes this distinction as well.

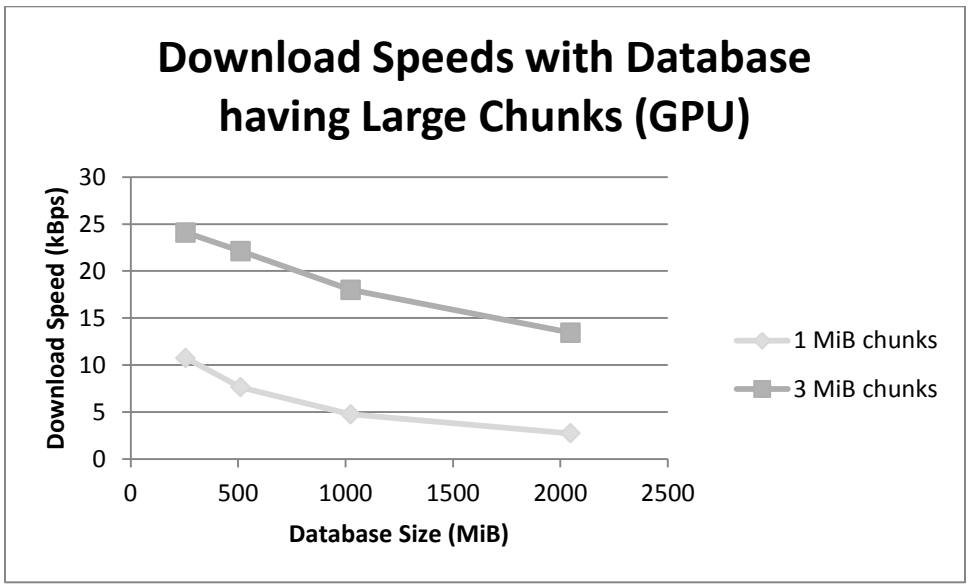

Figure 11: Download speeds of databases with large chunk sizes on a GPU enabled node.

Figure 11 show that having a large chunk size (3 MiB) provides decent download speeds. To determine whether 3 MiB chunk sizes are acceptable, the download speed on files less than 3 MiB should have to be acceptable as well. Figure 12 shows the minimum and maximum download speeds recorded on a database of either 512 MiB or 1 GiB with files less than 3 MiB whose file sizes were distributed evenly using a GPU enabled node. To create Figure 12 every file in the database was downloaded and then the total download time was observed. For each timespan, the largest and smallest files downloaded in that time range were obtained and this determined the minimum and maximum download speeds. Figure 12 shows that the total database size can be relatively large and still have decent performance.

A decent tuning to optimize a GPU node's download speed and bandwidth is to have two separate databases: one for large files, which will be chunked into 3 MiB chunks; and one for files smaller than 3

MiB, which will not be chunked. The database for larger files is partitioned into 1 GiB partitions and the database for smaller files is partitioned into 512 MiB partitions.

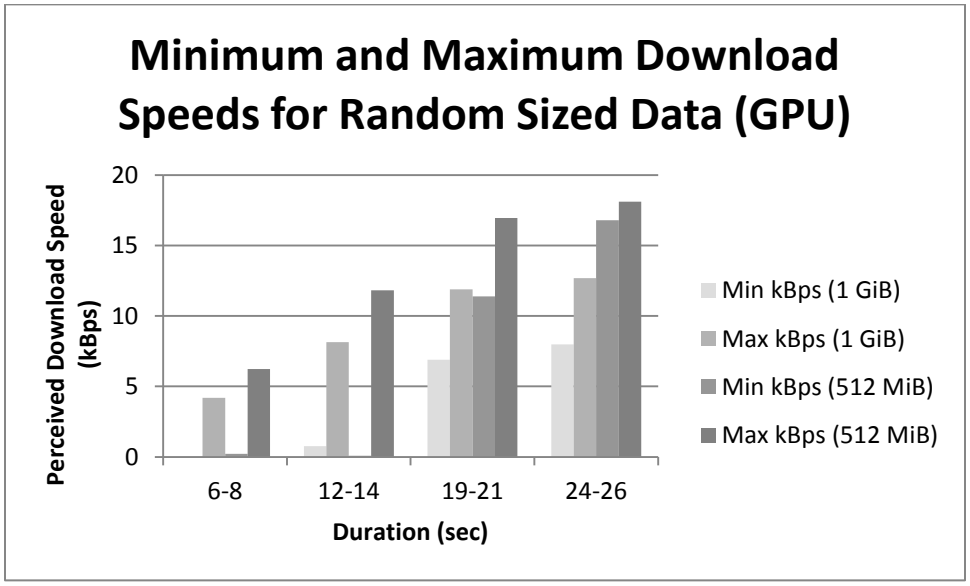

Figure 12: Download speeds for random sized data under 3 MiB for a GPU enabled node.

Testing found that using a CPU wasn't able to achieve significant download speeds while still maintaining privacy. This issue casts a dim light on the practicality of using this scheme as a peer-to-peer network because the average user does not have a CUDA enabled video card; however, this is becoming increasingly less likely.

3.4.5 Comparison of Solutions

Solutions 2 and 3 are discredited in their respective sections. The only viable options for this CPIR-based peer-to-peer network are Solutions 1 and 4. Although Solution 1 is viable, it isn't as efficient as Solution 4. Solution 4 has the ability to break larger files into chunks, which will allow for the ability to resume downloads, if a connection is interrupted. Also, different chunks can be downloaded from multiple peers to improve download speed. Smaller files do not suffer from internal fragmentation, because they are not

chunked. Thus, the settings that this prototype CPIR-based network uses are the tuned parameters from Solution 4.

Chapter 4

Experimental CPIR Based Scheme

Freenet is very popular but has a certain bandwidth and latency overhead. All data requested is cached at intermediate nodes to improve network performance and ensure privacy. This thesis proposes the idea that a new network could be created using CPIR protocols to obtain information from a node, instead of forwarding data through many intermediate nodes. When a node has a hash being requested, instead of sending the data to the requesting node (which is most likely an intermediate node), it simply responds by saying it does have the requested data. These responses are forwarded back until the original requestor obtains the response. The original requestor then requests the data by using the CPIR protocol (See (b) of Figure 13). To prevent timing attacks and to spread data throughout the network, intermediate nodes may intervene and cache the data before responding (See (c) of Figure 13).

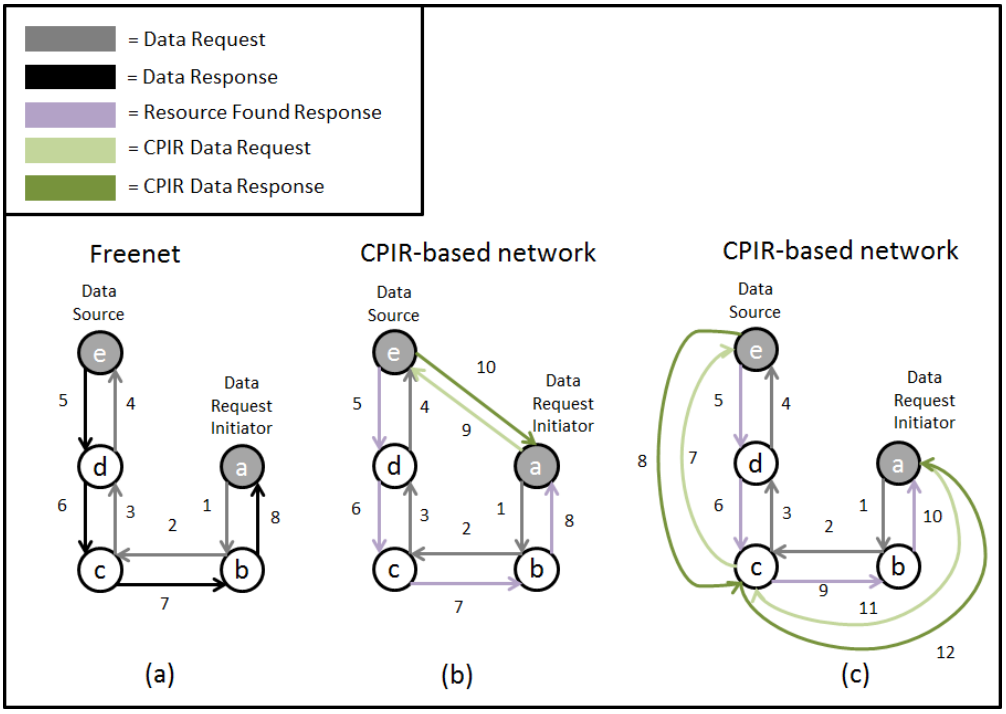

Figure 13: Comparison of Freenet request with CPIR-based request.

By using CPIR, requestor anonymity is protected without the need of going through intermediate nodes. Remember, when using a PIR protocol the sender does not know which file the requestor is requesting. In the Freenet paper Clarke et al. discredit the use of PIR protocols in an anonymous peer-to-peer environment. They argue that in most cases, the act of contacting a particular server itself is incriminating and should be avoided. In Freenet a node never knows when someone is actually requesting data for itself. The data may always be for another node. The authors may have overlooked using PIR along with intermediate nodes to provide plausible deniability for requesting nodes, ensuring there is no more risk in contacting a Freenet data node than being a member of Freenet itself.

The goal of this thesis was to create a system similar to Freenet with lower latency and requiring less overall network bandwidth, while still providing the same level of anonymity for the sender, receiver and all participating nodes in the network. This system would have to be practical. Currently, it is possible to create a network using CPIR systems with much less bandwidth usage than Freenet, but the computational costs would be too high to be practical.

4.1 Details

This scheme is very similar to Freenet, but with a few notable changes. The most obvious change is the way files are downloaded from the network. All file downloads will be through Alguilar and Gaborit's CPIR protocol. This hides which information is being obtained from the server. With this in place, there is much less need to cache data in a network. Freenet caches data in its network for three reasons: to provide anonymity when sending data to a peer, to provide plausible deniability for requesting data, and to provide plausible deniability for holding the data. When sending incriminating data, the node can claim that it is simply forwarding data. It does not know what the data is nor does he know who uploaded the data. When requesting incriminating data, he can claim that he is caching it locally and did not request the data. He can also claim he did not know what the data was (with plausible deniability).

Finally, when caught storing incriminating data he can claim that he has only cached it for someone else and that he never requested the data. By using Melchor and Gaborit's CPIR protocol, the first two reasons for caching data become unnecessary. So, caching is still necessary because we need to provide plausible deniability for storing data; however, not every node needs to cache requests. This can decrease latency dramatically. The percentage of nodes in a request route that request data will also affect the spread of data in a network, which is necessary for network usefulness.

Routing will be very similar to Freenet. Like Freenet this network uses the steepest-ascent hill-climbing search with backtracking, requesting data from nodes that have returned data closest to the requested hash before. On a successful route though, most nodes will not cache the data. This can be done randomly or by a certain preference. Nodes that do cache the data can arbitrarily change the source to themselves. All caching and storage is done through CPIR schemes. The sender is much less responsible for spreading incriminating data in this network compared with Freenet, because the sender is not able to determine which data was requested and consequently which data the sender actually sent. The criteria for determining whether a successful request should be cached could be file size, or computational intensity at the moment, or a strict percentage. In the prototype network tests, a strict percentage is used, but there are advantages of other methods. Using computational intensity to determine whether to cache can prevent a node that is very busy from adding more strain. This keeps the node responsive and the network healthy. File size could help keep node's databases more homogenous and improve performance of CPIR schemes. However, both of those options must be overridable with some certainty to keep requestor anonymity, or else an attacker could force nodes to not cache data and find the original requestor.

Finally another major change is in premix routing. Premix routing uses onion routes at the beginning of a `find` request. Onion routing is the idea of encrypting a message many times in layers and each time has a different key (See Figure 14: Demonstration of onion route.). To decrypt each layer, you must have the

associated key. This technique was created in 1996 by Goldschlag, Reed, and Syverson to provide real-time anonymous communication over application layer protocols. [19] Using Figure 9, we can see that that once hop #3 receives an onion, he can decrypt it but cannot see the contents. Hop #3 knows to send the message to hop #2. This is repeated until hop #1 receives and decrypts the package. Hop #1 reads the message and acts accordingly.

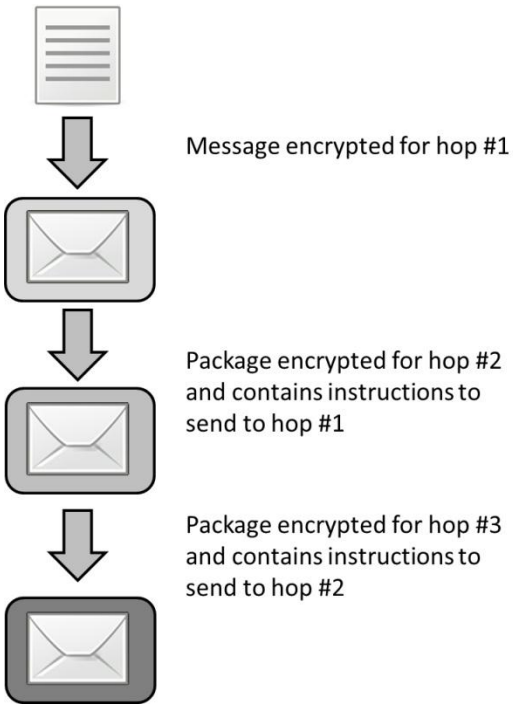

Figure 14: Demonstration of onion route.

The goal of incorporating premix routing into an anonymous peer-to-peer network is to prevent correlation attacks using the HTL value. Assume for instance, that a node received a `find` request with a HTL value of MAX_HTL. Then, with high probability, a node can conclude that this request is the originating request. Right now, Freenet randomizes the HTL value when very high, but a correlation can still be statistically evident. In the prototype anonymous peer-to-peer network, all `find` requests (which include hash to find) are premix routed with onion routes. Now, nodes that receive `find` requests are unable to say with any degree of certainty who initiated the request. If there was no premix routing, then nodes requesting data only had plausible deniability of being incriminated for requesting data. With

premix routing, any requesting node has very strong deniability (beyond suspicion) with regard to initiating a `find` request. Freenet wishes to add this feature, but has not because they are still trying to determine the best way to do so.

4.2 Implementation

To test the usability of CPIR in an anonymous peer-to-peer network, a network that utilized the Melchor Gaborit CPIR protocol and the routing algorithm mentioned above was created. The network is implemented in C++ and uses xml-based messages to exchange information. When a request for data is found, it is automatically downloaded using the CPIR protocol. When a success is being forwarded back to the originator, each node may or may not cache the request, blocking the message until downloaded. Every node listens on a different port and messages are sent asynchronously. Shell scripts were written that would start the nodes on different ports with different neighbors and with different initial data.

4.2 Experiments

The goal of these experiments is to show that this prototype network is practical. Many full scale tests were run on a local machine to determine some statistics. These results were compared against a typical Freenet node running on the same machine. There are many things to determine. One limiting factor is the computation. Each node will have requests that it must fulfill and CPIR operations are computationally expensive. Statistics of Freenet are published by nodes who wish to help determine the health of Freenet. After running a Freenet node whose cache was 2 GiB with the default settings on a 2 GHz machine with 2.00 GiB of RAM for twenty-four hours (See Figure 15), the total load on the node was determined. Bulk requests are requests whose latency isn't as important. A real-time requests demand faster latency. Most of the Freenet requests that were measured were real-time requests. When a Freenet node is busy, it will drop requests; so the standard deviation within each hour is very little. Each request is for a 32 KiB segment. Of course, the results depend on configuration settings. The default

settings were chosen for all options, except the cache size. The default cache size depends on the total size of the hard disk drive a node has, but a 2 GiB cache was chosen. During the most demanding hour of the test (hour 20), the node had to process 620 requests: 600 real-time and 20 bulk requests.

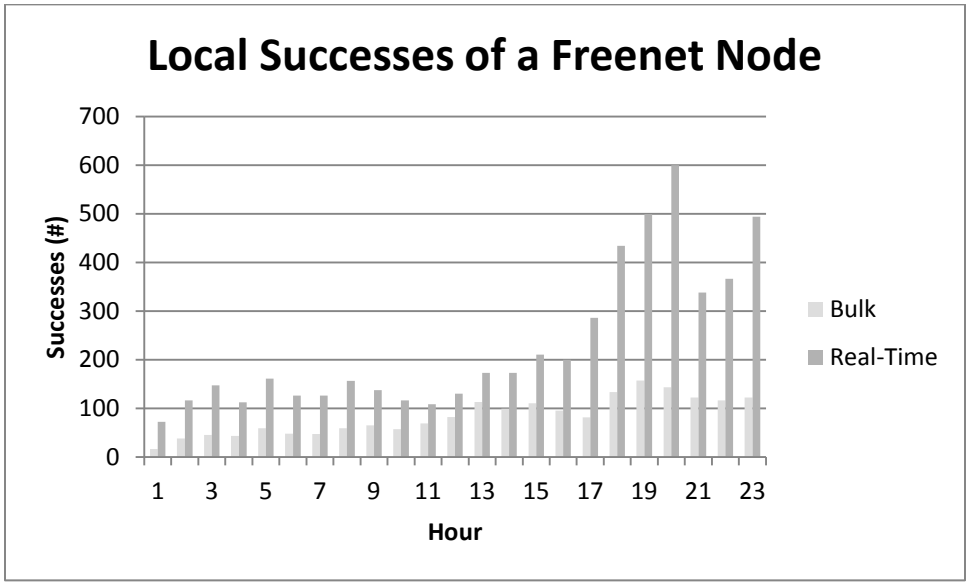

Figure 15: Load on a 2 GHz Machine with 2.00 GiB of RAM in Freenet in the first 24 hours.

I took the peak usage hour and determined that 620 local successes are successfully retrieved from that node's local database. This means that every minute of the peak hour there were, on average, 11 `find` requests being successfully responded to: 10 from real-time requests and 1 from bulk-requests. Stress testing the machine used for Figure 15, which has a CUDA enabled GeForce 8800 GTX video card, was able to produce upwards of 8 real-time CPIR requests per minute. Limiting real requests per minute to 7 real-time requests per minute and limiting bulk requests to 2 per minute, the node is able to process all requests. This would limit the total requests that a node could handle to be 420 real-time requests per hour. The recorded measurement of 620 requests is too high and the node would have to limit these requests for peak hours. This restriction does not stop the network from functioning but it does limit the amount of requests that a node can handle. Testing this on the prototype network did not seem to affect network conditions greatly. The network was able to recover and route data around busy nodes. Not using a CUDA enabled video card, a node can only handle almost 1 bulk-request per core per minute. The

same goes with real-time requests. This is because the request uses almost all CPU for almost an entire minute. The server processing time is almost 50 seconds alone.

It is impractical to think that a node would be willing to spend all CPU/GPU processing time to handle requests for other nodes in a network. Most nodes have ample hard drive space and asking to give up a small percentage makes sense. Limiting the number of requests per minute to such a low number where CPU/GPU is not being constantly worked at near maximum capacity makes the network very unresponsive. This is because one request consumes an entire process for almost 40 seconds on a CPU device. Almost no packets are successfully found if requests are limited to 1 or 2 per minute.

By using experimentation, there was not a way that could be found to create a network that would be practical using CPIR exclusively for downloading files. Every method that would have decent download speeds required too much CPU/GPU use. Average users do not want to sacrifice their computer for freedom. Besides the crippling factor of the CPU/GPU utilization, the bandwidth overhead was too large to be even close to what Freenet's bandwidth has.

Chapter 5

Conclusion

5.1 Observations

The idea of an anonymous peer-to-peer network based on a computational private information retrieval protocol is very intriguing. This thesis evaluated the practicality of such a network through many means. Many ideas were changed over time to keep the network secure and anonymous. Many changes were also made from the ideas of Freenet, mainly that the act of communicating with a Freenet node itself is not dangerous. On top of fundamental differences, many technical changes were made. To spread data around on a network, nodes on a path had to randomly download from the source; this happens in Freenet but for many more reasons. In my original idea, this wasn't here, but there must be a way to carry popular information through a network.

To develop this project there were many tasks that had to be complete. The first step is mainly background reading and analysis. Finding a CPIR protocol was a lot of researching, because there are many different protocols, but it was hard to determine the practicality of each one. Finally, developing a peer-to-peer network was a challenge. It is written in C++ and heavily utilizes the boost library: boost asio for handling networking, boost thread, boost filesystem for writing and reading data, and boost program options for handling program arguments.

After developing the network, it had to be tested to ensure that it worked according to specifications of other anonymous networks. The network had to converge like Freenet, and there had to be a framework for testing the network. This took time to determine errors and bugs. However, I'm sure not all bugs are missing and this thesis project should not be used to guarantee anonymity. It should be used to conduct further research and if CPIR protocols improve, the code can swap out CPIR protocols pretty easily.

5.2 Practicality

The network this thesis proposes, while not being more efficient than Freenet, can be practical and in some sense more secure. It is possible to use this network even though it has very high bandwidth overhead by limiting the total requests handled per hour, making the computation usage acceptable. However, most computers do not have CUDA enabled video cards, yet. So, while CUDA enabled video cards are not the norm, it is very difficult to imagine this network being practical. Also, overall there is much more bandwidth overhead than originally anticipated. For instance a 3 MiB request for a 1 GiB database – the recommended parameters – required at least a total of 41.039 MiB of bandwidth. In Freenet this would have been, on average, significantly less. So, while this network is practical it is not very useful unless a target audience would be willing to accept such performance penalties for added anonymity. This may be the case in some situations.

5.3 Future Work

The prototype network uses a lot of bandwidth and computational power and probably isn't too useful in the near term. However, there are a few projects that could be researched that would allow this network to be practical.

One exciting idea is a Freenet plugin that would allow 'extra' security by downloading an item over CPIR as well as using intermediate nodes. This would block the server from knowing what was obtained while creating a small overhead in both bandwidth and computation. Since not every request uses CPIR, a node can handle CPIR requests with only a CPU and not notice much overhead. This could be done on top of Freenet's current network and allow nodes to optionally participate in the protocol when they either have free computation or a very sensitive request is going though.

Another avenue of research should be in CPIR requests today. Only recently was computational complexity taken into account for CPIR optimizations. So, maybe a more efficient protocol could be developed that uses both less computation and less bandwidth. If so, then this scheme may become even more practical on its own.

References

1. **Masnick, Mike.** Homeland Security Seizes Spanish Domain Name That Had Already Been Declared Lega. *techdirt.* [Online] February 1, 2011. [Cited: March 19, 2011.] http://www.techdirt.com/articles/20110201/10252412910/homeland-security-seizes-spanish-domain-name-that-had-already-been-declared-legal.shtml.

2. *Understanding Kazaa.* **Liang, Jian, Kumar, Rakesh and Ross, Keith W.** 2004.

3. *An Analysis of the Skype Peer-to-Peer Internet Telephony.* **Baset, Salman A. and Schulzrinne, Henning.** 2004 : s.n.

4. *Crowds: anonymity for Web transactions.* **Reiter, Michael K. and Rubin, Aviel D.** 1, New York : ACM, November 1998, ACM Transactions on Information and System Security (TISSEC), Vol. 1, pp. 62-92. 1094-9224.

5. *Freenet: a distributed anonyous information storage and retrieval system.* **Clarke, Ian, et al.** Berkeley : Springer-Verlag New York, Inc., 2001, International workshop on Designing privacy enhancing technologies, pp. 46-66. 3-540-41724-9.

6. *Small World Experiment.* **Milgram, Stanley.** May 1967, Psychology Today, Vol. 1, pp. 61-67. 0033-3107.

7. *Private Information Retrieval.* **Chor, B, et al.** Washington, D.C. : IEEE Computer Science, 1995, FOCS '95: Proceedings of the 36th Annual Symposium on Foundations of Computer Science, pp. 41-50. 0-8186-7183-1.

8. *Computationally Private Information Retrieval.* **Chor, Benny and Gilboa, Niv.** El Paso : Association for Computing Machinery, May 4, 1997, Proceedings of the 29th Annual Symposium on the Theory of Computing, pp. 304-313. 0-89791-888-6.

9. *Private Information Storage (Extended Abstract).* **Ostrovsky, Rafail and Shoup, Victor.** 1997.

10. *Replication Is Not Needed: Single Database, Computationally-Private Information Retrieval.* **Kushilevitz, Eyal and Ostrovsky, Rafail.** Miami Beach : s.n., October 20, 1997, FOCS '97: Proceedings of the 38th Annual Symposium on Foundations of Computer Science, pp. 364-373. 0-8186-8197-7.

11. *On the Computational Practicality of Private Information Retrieval.* **Sion, Radu and Carbunar, Bogdan.** 2007, In Proceedings of the Network and Distributed Systems Security Symposium.

12. *Privacy-preserving Queries over Relational Databases.* **Olumofin, Femi and Goldberg, Ian.** July 2010, 10th Privacy Enhancing Technologies Symposium, pp. 75-92.

13. *Efficient Computationally Private Information Retrieval From Anonymity or Trapdoor Groups.* **Trosle, Jonathan and Parrish, Andy.** [ed.] Mike Burmester, et al. s.l. : Springer Berlin / Heidelberg, 2011, Lecture Notes in Computer Science, Vol. 6531, pp. 114-128. 10.1007/978-3-642-18178-8_10.

14. *A Fast Private Information Retrieval Protocol.* **Melchor, Carlos Alguilar and Gaborit, Phillippe.** Toronto : s.n., July 6-11, 2008, ISIT 2008. IEEE International Symposium on Information Theory, 2008, pp. 1848-1852. 978-1-4244-2256-2.

15. *High-speed Private Information Retrieval Computation on GPU.* **Alguilar-Melchor, Carlos, et al.** Washington DC : s.n., 2008, SECURWARE '08 Proceedings of the 2008 Second International Conference on Emerging Security Information, Systems and Technologies, pp. 263-272. 978-0-7695-3329-2.

16. *Revisiting the Computational Practicality of Private Information Retrieval.* **Olumofin, Femi and Goldberg, Ian.** June 2010, CACR Tech Report, p. 16.

17. *Generating hard instances of lattice problems (extended abstract).* **Ajtai, Miklós.** Philadelphia, Pennsylvania : Association for Computing Machinery, 1996, Proceedings of the twenty-eighth annual ACM symposium on Theory of computing, Vol. 96, pp. 99-108. 0-89791-785-5.

18. Household Upload Index. *Net Index.* [Online] Ookla. [Cited: February 13, 2011.] http://www.netindex.com/upload/allcountries/.

19. *Hiding Routing Information.* **Goldschlag, David M., Syverson, Paul F. and Reed, Michael G.** [ed.] Ross Anderson. s.l. : Springer Berlin / Heidelberg, 1996, Information Hiding: Lecture Notes in Computer Science, Vol. 1174, pp. 137-150. 10.1007/3-540-61996-8_37.

20. *Lectures on the NTRU encryption algorithm and digital signature scheme.* **Pipher, Jill.** Grenoble, France : s.n., June 2002.

Appendix

Building the software

1. Obtain the prerequisite software
   - Boost Program Options
   - Boost Lexical Cast
   - Boost Asio
   - Boost Thread
   - Boost Filesystem
   - CUDA (optional)
   - Crypto++
   - GNU Multiple Precision Arithmetic Library
   - Number Theory Library (NTL)
   - Xerces-C++ XML Parser

2. Extract the archive and build

```
tar xf thesis-michael.tar.gz
make
```

3. Running the node

```
./node -h ip address -p port -v
```

4. Using the node

```
[port] 1. Send public key request
[port] 2. Send find request packet
[port] 3. Send PIR request
[port] 4. ls
[port] 5. wait (debugging)
```

Option 1:

This option essentially finds neighbors. By obtaining a neighbor's public key, you are able to send `find` requests to them. Also, you are able to send encrypted content to them by encrypting a symmetric key with their public key.

Option 2:

This option allows you to request a hash that you wish to obtain.

Option 3:

This option allows you to download a hash when you know where it is. This isn't done automatically after a find is successful to ensure data is wanted.

Option 4:

Display hashes that you currently have. Mainly for debugging.

Option 5:
Wait is for running test networks on the localhost. It allows all nodes to be up and running before requests are made.

Vita

The author was born in Covington, Louisiana and grew up in Slidell, Louisiana. He obtained his bachelor's degree in computer science from Louisiana State University in 2009. He joined the University of New Orleans computer science graduate program to pursue a MS in computer science focusing on information assurance.