5-1-2012

Privacy-preserving techniques for computer and network forensics

Bilal Shebaro

BILAL SHEBARO

Candidate

Computer Science

Department

This dissertation is approved, and it is acceptable in quality and form for publication:

Approved by the Dissertation Committee:

Jedidiah R. Crandall , Chairperson

Dorian C. Arnold

Fernando Perez-Gonzalez

Pedro Comesaña-Alfaro

Privacy-Preserving Techniques for Computer and Network Forensics

by

Bilal Shebaro

B.S., Computer Science, Beirut Arab University, 2003
M.S., Computer Science, Lebanese American University, 2006

DISSERTATION

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Doctor of Philosophy
Computer Science

The University of New Mexico

Albuquerque, New Mexico

May, 2012

Dedication

To my parents where none of this would have been possible without their love & patience.
To my brothers and sisters for their prayers and support.
To my lovely nephews and niece whom I hope this work will inspire them to believe that you can accomplish anything you set your mind to.
To my friends and beloved ones who supported me during my graduate studies.

Acknowledgments

Though only my name appears on the cover of this dissertation, this work was only possible due to the contributions of several people whom I am grateful for.

I would like to express my deepest appreciation to my advisor, Dr. Jedidiah Crandall, for his support, wise advice, and patient encouragement throughout my graduate studies. I would also like to thank Dr. Fernando Perez-Gonzalez for his continuous advice and for the long simulating discussions that were greatly needed and deeply appreciated. I would also like to thank Dr. Dorian Arnold and Dr. Pedro Comesana Alfaro for their guidance during the course of my graduate studies and for being on my defense committee in spite of their busy schedules.

I am also thankful to the Computer Science faculty and staff at the University of New Mexico who made my graduate experience one that I will cherish forever.

I would also like to deeply thank my sister, Dr. Lina Shebaro Germann, and my brother-in-law, Dr. Timothy Germann, for all their help and support that eased my move to the United States of America and helped me adjust to a new country.

A final word of gratitude is expressed to Mohammad Al-Sharif whose support and encouragement has brought greater meaning to this entire work.

Privacy-Preserving Techniques for Computer and Network Forensics

by

Bilal Shebaro

B.S., Computer Science, Beirut Arab University, 2003
M.S., Computer Science, Lebanese American University, 2006

Ph.D., Computer Science, University of New Mexico, 2012

Abstract

Clients, administrators, and law enforcement personnel have many privacy concerns when it comes to network forensics. Clients would like to use network services in a freedom friendly environment that protects their privacy and personal data. Administrators would like to monitor their network, and audit its behavior and functionality for debugging and statistical purposes (which could involve invading the privacy of its network users). Finally, members of law enforcement would like to track and identify any type of digital crimes that occur on the network, and charge the suspect with the appropriate crimes. Members of law enforcement could use some security back doors made available by network administrators, or other forensic tools, that could potentially invade the privacy of network users.

In my dissertation, I will be identifying and implementing techniques that each of these entities could use to achieve their goals while preserving the privacy of users on the network. I will show a privacy-preserving implementation of network flow recording that can allow administrators to monitor and audit their network behavior and functionality for debugging and statistical purposes without having this data contain any private information about its users. This implementation is based on identity-based encryption and differential privacy. I will also be showing how law enforcement could use timing channel techniques to fingerprint anonymous servers that are running websites with illegal content and services. Finally I will show the results from a thought experiment about how network administrators can identify pattern-like software that is running on clients' machines remotely without any administrative privileges.

The goal of my work is to understand what privileges administrators or law enforcement need to achieve their goals, and the privacy issues inherent in this, and to develop technologies that help administrators and law enforcement achieve their goals while preserving the privacy of network users.

Contents

List of Figures **xii**

List of Tables **xiv**

1 Introduction **1**

2 Privacy-Preserving Network Flow Recording **5**

2.1 Abstract 5

2.2 Introduction 6

2.3 Threat Model 8

2.4 Background 11

2.4.1 Traffic flow recording and data collection 11

2.4.2 Identity Based Encryption and AES 11

2.4.3 Statistical database modeling and inference controls 13

2.5 Implementation 13

2.5.1 Data collection 13

2.5.2 Phase 1: Encryption vs. decryption 14

2.5.3 Phase 2: Generating statistical reports 15

2.6 Experimental Methodology 19

2.6.1 Live experiments 20

2.6.2 Offline experiments 21

2.7 Results 22

2.7.1 Live experiments 22

2.7.2 Offline experiments 28

2.8 Discussion and future work 29

2.9 Related Work 30

2.10 Conclusion 32

3 Network Flow Recording with Differential Privacy 33

3.1 Abstract 33

3.2 Introduction 33

3.3 Threat Model 35

3.4 Background 36

3.5 Implementation 39

3.6 Experimental Methodology 41

3.7 Results 42

3.8 Discussion and future work 43

3.9 Related Work 44

3.10 Conclusion 44

4 Leaving Timing Channel Fingerprints in Hidden Service Log Files 45

4.1 Abstract 45

4.2 Introduction 46

4.3 Measurement methodology 49

4.3.1 Tor network delays 50

4.3.2 Existing HTTP GET requests 51

4.4 Measurement results 52

4.4.1 Tor delays 53

4.4.2 Existing HTTP GET requests 54

4.5 Implementation 55

4.5.1 Example 58

4.6 Discussion and future work 59

4.7 Related work 61

4.8 Conclusion 62

5 Exploiting Geometrical Structure for Forensic Applications 63

5.1 Abstract 63

5.2 Introduction 64

5.3 Threat Model 67
5.4 Background 68
5.4.1 MPEGs 69
5.4.2 Video Compression Picture Types 70
5.5 Implementation 72
5.6 Experimental Methodology 74
5.7 Results 79
5.8 Discussion and future work 86
5.9 Related Work 87
5.10 Conclusion 88

6 Conclusion **89**

References **91**

List of Figures

2.1 **Data collection steps, and examples of plaintext and encrypted network flow records.** 14

2.2 **Timings for `/24` Subnets.** 25

2.3 **Tradeoff between time to encrypt and time to import statistical data for `/24` subnets.** 25

2.4 **Timings for `/22` subnets.** 26

2.5 **Tradeoff between time to encrypt and time to import statistical data for `/22` subnets.** 26

2.6 **Timings for `/20` subnets.** 27

2.7 **Tradeoff between time to encrypt and time to import statistical data for `/20` subnets.** 27

2.8 **Timing comparison between `/20`, `/22`, `/24` subnets according to recording time.** 28

4.1 **Web server traffic for a 24-hour period.** 46

4.2 **Histogram for Tor delays in seconds.** 49

4.3 **Histogram for existing HTTP GET requests per minute.** 50

4.4 **Histogram of existing HTTP GET requests per minute before (thin line) and after adding additional requests for fingerprinting purposes (bold line).** 52

4.5 **How a 60-bit codeword appears in the log file.** 53

4.6 **Details on a bit error from the example.** 54

4.7 **The fingerprinting algorithm cycle.** 56

4.8 **Histograms with and without fingerprinting.** 57

5.1 **MPEG Compression Overview, reproduced from page 3 of The MPEG Handbook [1].** 69

5.2 **Examples of different MPEG streams.** 71

5.3 **Binning of movies based on distances of movie vectors from the surface of the sphere. Positive counts are outside the sphere, and negative counts are inside the sphere, binned by distance.** 82

5.4 **Projection of vectors on the constant vector.** 84

List of Tables

2.1 **Rates that my implementation can achieve for different subnet sizes.** 28

3.1 **A table demonstrating the effect of applying the differentially private mechanisms related to link utilization values.** 43

5.1 **Euclidean distances of running the same movie as a sample.** 80

5.2 **Euclidean distances of running different movies.** 80

5.3 **Sample mean vector coordinates over different machines.** 83

Chapter 1

Introduction

The focus of my dissertation work is on forensic techniques that preserve the privacy of individuals' data that exists in the same computer or network where a crime happened. My work will focus on techniques that administrators and law enforcement personnel could use to help identify digital crimes without invading the privacy of the clients. For this reason, I will be showing what privileges administrators or law enforcement need to achieve their goals, and will develop technologies and techniques to preserve clients' privacy.

Computer forensics personnel are settled with the current tools to do technical investigations on stand alone computers, while network forensics is still challenging. Most of these tools will do a brute force scan to recover everything that happened on the computer or network, thus invading the privacy of all participating users in that network for the sake of catching the crime and identifying the suspect.

There are many challenges that must be addressed to achieve my goal. One main challenge is the existence of anonymous networks that hide the identity and location of its users. Suspects behind these networks cause problems for law enforcement to go after them, and sometimes such networks can hide the crime itself which makes the network auditing job much harder. Such networks make tracing back suspects a big challenge. An example of an anonymous network is Tor, which I will be considering in my work.

Another challenge is that of preserving the privacy of individuals that reside on the same network as the suspect, that is, their records, traffic, connections, and services. This is challenging because investigating a crime in such networks might require the network operator to reveal all of the traffic and records on it, that is, those records of innocents as well as suspects. In fact, some existing techniques involve replaying all the recorded data that happened during the time of the crime, where privacy is invaded from both actions, recording and replaying.

Besides the privacy concerns for innocent users, in this work I also offer time savings related to investigations. The law enforcement has now to deal with a smaller data set of records related to the suspected criminal rather than a large amounts of records of participating users in the network. Thus, in many critical cases where quick investigations are required, this approach will save time for such investigations due to making it easier to get a warrant.

Complying with law enforcement requirements forced on networks residing in organizations such as ISPs, academic institutions, and companies could be another challenge. Some of these requirements involve recording their users' traffic for a certain period of time, and investigation of these records in case of a crime would reveal much of the private data of innocent people. These requirements would also leave less trust for users working on these networks, with the belief that everything is traceable and can be recovered.

Possessing child pornography media, an example of a digital crime, is a key research issue in the forensics domain. A lot of research has been done to trace all such users, from those who are downloading/uploading to those who are watching/streaming. Uploading and downloading related torrents behind anonymous networks causes challenges for law enforcement to trace these people, especially with the distributed client to client architecture that bit-torrents work with.

Some clients are not aware of what administrators can do on their machines once they are connected to their network. They are also not aware of how much their privacy is invaded and to what extent these administrators can reveal information out of their computers. This also applies to law enforcement or any other user that is connected to the same network, each has its own techniques and limits as to what information is visible to them.

The work in this dissertation will consider these challenges, and will focus on such privileges that are applied on users of the same network. It will also focus on privacy preserving techniques that will help investigators trace back and identify suspected criminals, even those who are residing behind anonymous networks, with the privacy of individuals preserved. I will show how these techniques comply with the law enforcement requirements as well as provide a privacy-preserving recording technique that will only replay the suspect's crime traffic.

The intention of privacy protection considered in this dissertation may not cover every aspects of privacy invasion that people might consider. In fact, the definition of privacy widely varies according to what is considered private and what is not. This work is more focused on allowing the clients to use the network services in a freedom friendly environment where no traceback of their network traffic can be identified, unless these clients are involved in a digital crime. For the purposes of this dissertation, I consider an invasion of privacy to be any time network administrators or law enforcement learn more information about any individual than is necessary to do their respective jobs.

In Chapter 2 I present a network flow recording technology that considers users' privacy by using Identity Based Encryption in combination with privacy-preserving semantics for on-the-fly statistics. I argue that my implementation supports a wide range of policies that cover many current applications of network flow recording. I also characterize the performance and scalability of my implementation and find that the encryption and statistics scale well and can easily keep up with the rate at which commodity systems can capture traffic.

While the statistical approach in Chapter 2 preserves the privacy of the network users and administrators can achieve their goals from the generated statistical reports, Chapter 3 discusses a different implementation of the statistical part based on differential privacy that is capable of revealing useful information from the statistical database while the adversary fails in predicting individual records, even if given the database.

The second-generation onion routing, Tor, is a very popular anonymous network that allows users to use internet services anonymously. Tor also has a hidden service feature. In Chapter 4 I will show how law enforcement can leave an identifiable fingerprint in the log file of Tor hidden services using timing channels that can be recovered from the timestamps found on the machine hosting the service. Upon confiscating the machine, this is recovered as a proof that a particular machine was hosting the service.

Finally, Chapter 5 discusses a thought experiment about how much information can an unprivileged process learn by just running on a system and observing its own timings, which is an approach to understanding the system's behavior and detecting pattern-like software. I will be using timing inference channels to develop a geometrical interpretation in a high dimensional space of timing signatures for movies as an example of pattern-like software.

The work in this dissertation aims to understand the possibility of building forensic tools that could achieve the exact same goals intended with more efforts to protect the privacy of the people connected to the internet. The existence of such privacy-preserving tools would attract and encourage more people to securely use the internet, which makes the world more connected than ever.

Chapter 2

Privacy-Preserving Network Flow Recording

2.1 Abstract

Network flow recording is an important tool with applications that range from legal compliance and security auditing to network forensics, troubleshooting, and marketing. Unfortunately, current network flow recording technologies do not allow network operators to enforce a privacy policy on the data that is recorded, in particular how this data is stored and used within the organization. Challenges to building such a technology include the public key infrastructure, scalability, and gathering statistics about the data while still preserving privacy.

In this chapter I present a network flow recording technology that addresses these challenges by using Identity Based Encryption in combination with privacy-preserving semantics for on-the-fly statistics. I argue that my implementation supports a wide range of policies that cover many current applications of network flow recording. I also characterize the performance and scalability of my implementation and find that the encryption and statistics scale well and can easily keep up with the rate at which commodity systems can capture traffic, with a couple of interesting caveats about the size of the subnet that data is being recorded for and how statistics generation is affected by the implementation details.

I conclude that privacy-preserving network flow recording is possible at 10 gigabit rates for subnets as large as a `/20` (4096 hosts).

Because network flow recording is one of the most serious threats to web privacy today, I believe that developing technology to enforce a privacy policy on the recorded data is an important first step before policy makers can make decisions about how network operators can and should store and use network flow data. The goal in this chapter is to explore the tradeoffs of performance and scalability *vs.* privacy, and the usefulness of the recorded data in forensics *vs.* privacy.

2.2 Introduction

Network flow recording, such as Cisco's NetFlow [2], is a tool that network administrators use to record information about network traffic sessions. This data can be used for network forensics and traffic accounting, usage-based network billing, network planning, security, denial-of-service monitoring, and network monitoring. Network flows also provide valuable information about network users and applications, peak usage times, and traffic routing. Along with the related practice of clickstream recording, network flow recording is one of the most serious threats to Internet and web privacy today. Network flow data reveals what websites a network user visited, along with timing information, what services they used, and how much data was transferred. An especially important issue is how this data is stored, since a compromise of this data reveals so much about so many users. Another important issue is how the data can be accessed for the purposes it was collected for without enabling accesses that violate the privacy policy. In nowadays forensics, a digital crime occuring in a network will cause the whole network traffic to be analyzed and investigated on, thus time consuming for invesigators as well as invading the privacy of users that are on the network and not involved in the crime.

Most ISPs use or implement network analyzer tools that read network flow data and do their own traffic analysis, such as Orion NetFlow Traffic Analyzer (NTA) which enables network administrators to capture data from continuous streams of network traffic and convert those raw numbers into easy-to-interpret charts and tables that quantify exactly how the corporate network is being used, by whom, and for what purpose [3]. Another tool is NetFlow Analyzer, that uses Cisco NetFlow to monitor bandwidth and to gather information on the top network users, applications, and many other features [4]. These tools produce reports on in-depth traffic analysis and network bandwidth monitoring. Network flow data can also be requested by law enforcement agencies to aid in investigations. Because network flow data is used for so many critical functions, it is important to provide a way for network operators to record and use this data while also enforcing a privacy policy on the data.

While ISPs care about the privacy of their customers as well as providing the most secure and convenient service, in the U.S. there have been many calls from the FBI and multiple members of congress requesting that ISPs perform mandatory data retention, that is, keeping records of their users' activities for later review by law enforcement [5]. This can aid in the investigations of serious crimes such as hosting child pornography. At the same time, the potential for data breaches and abuses by employees of the network operator means that network operators should enforce privacy policies on the data collected, and such enforcement may become mandatory in the future. This chapter is about how to develop technologies that aid in enforcing such privacy policies.

The network flow recording system that I have developed, and described in this chapter, can provide ISPs with common traffic analysis and statistics as well as the ability to retain more detailed information about each session, while encrypting the session data and enforcing privacy-preserving semantics on queries for the statistics. I consider two different threats: threats against the stored session data, and abuses of the statistics. To address the first, I propose to use Identity Based Encryption (IBE) because it limits the public key infrastructure necessary such that the network users do not need to be aware of their public keys nor modify their computers or network traffic in any way. To address the second threat, I define a privacy-preserving semantics for network flow data that provides access to statistical reports about groups of individuals, while restricting access to information about any particular individual unless enough data is aggregated to mitigate any privacy threats.

This chapter is organized as follows. First, I describe my threat model in Section 2.3. This is followed by Section 2.4 where I introduce some basic concepts and background. Section 2.5 shows the implementation of my system. In Section 2.6 I explain the evaluation methodology, and then the results in Section 2.7. A discussion of performance issues and future work in Section 2.8 is then followed by related works and the conclusion.

2.3 Threat Model

First, I must distinguish between forward security and backward security as it applies to network flow recording privacy. Forward security means protecting information that will be recorded in the future, while backward security means that after a compromise the data that was recorded in the past is still secure. Forward security is not possible in the context of network flow recording. At the moment that a gateway router is compromised or an employee of the network operator with the necessary access decides to violate the policy, they can simply record network flow data as plaintext. It is impossible for web users to hide their source and destination IP addresses and timing information from those that they trust to route their packets.

Backward security, on the other hand, is very important for network flow data, and is the focus of this chapter. Network operators reported that they retain network flows and clickstreams for anywhere from 2 weeks to 6 months and even, in some cases, indefinitely [6]. It is this stored data that must be protected, so that exposure of private information is limited to the time that a breach of policy or a router compromise persists undetected, and no longer. So, the first of two threats to be addressed in this chapter is how to encrypt this data in a way that enforces the privacy policy but requires no complicated public key infrastructure or modifications to protocols or implementations outside of the network flow recording system. Using Identity Based Encryption gives three important properties in this regard. First, the public key (or encryption key) for each individual IP address can be the IP address itself, possibly combined with a timestamp. Thus the network users do not need to encode their public keys into the network traffic, nor even be aware that any public key cryptography is involved. The network hosts and network itself operate as they normally would. Second, the network flow recorder can encrypt the session data for an IP address in a given time period using the IP address and timestamp for that period as the encryption key, and does not need to make any queries to a public key infrastructure to know the required public key. Third, the secret that decrypts the traffic does not need to be stored where the network flow data is recorded and encrypted, and this secret can be divided into pieces so that separation of duty requires multiple parties to be involved to decrypt the data.

The second threat that I address in this chapter has to do with the fact that not all of the information can be encrypted in most applications of network flow recording, since statistics about how much and when network links are used are also very important just for day-to-day network operations. The threat is that information about the amount of network traffic an individual used must be stored for statistical purposes for the queries to be accurate (for example, for billing purposes), but this information is often enough to infer certain facts about an individual user's web activities if it is not aggregated properly with other data before being presented. I assume that the stored statistical data is presented to the rest of the organization (the billing department, traffic engineers, quality-of-service experts, *etc.*) through a database interface, and present a privacy-preserving semantics for that database interface that enables a wide range of queries but preserves the privacy of individuals.

As one example of an organization that might want to enforce a privacy policy on network flow data, consider a university. In the U.S., universities collect network flow data for network management reasons but are often also required to retain network flow data for periods of weeks due to state laws about employing state employees or federal laws designed to curb the pirating of copyrighted music and videos. This period of weeks or months is a very long exposure window in which any breach of that data or possible violation of the privacy policy by a university employee is a major vulnerability in terms of the web privacy of the network's users. This proposed system would allow for the network flow recording to support both the legal obligations and network operations of the university, while virtually eliminating this window of exposure. The secret necessary to decrypt traffic for law enforcement or public freedom of information purposes could be divided, for example, among the regents, faculty senate, and university counsel so that the enforcement of the university's privacy policy for this data has the property of separation of duty.

Also consider an Internet Service Provider (ISP). In this case the secret used for decryption could be divided between the customer service department and another department that is tasked with enforcing and auditing the privacy policy of the organization. If a customer service agent needed to look at detailed network flow data for a particular customer to debug a network connection problem, they could obtain the customer's consent and then request the keys to decrypt that customer's network traffic 24 hours into the past and several hours into the future. The department that enforces and audits the privacy policy could confirm customer consent and then use their part of the decryption secret to provide keys for only the requested time period, and these keys could not be used to decrypt traffic for other customers or in other periods of time. More routine queries of data (how much bandwidth is a particular neighborhood using during peak hours, what are the most common applications being used by a large group of customers, how much did a particular customer go over their allowed traffic quota in a billing period, and so on) can be handled by the privacy-preserving statistical database.

I wish to reiterate that I am not attempting to protect web privacy against untrusted network controllers. My goal is to give network controllers the technological tools they need to enforce privacy policies so that network users can place their trust in the network controllers as an organization.

2.4 Background

My system uses the concepts of traffic flow recording, data collection, Identity Based Encryption, and AES as well as statistical database modeling and inference controls. These concepts are key to my implementation so here I describe some background about them.

2.4.1 Traffic flow recording and data collection

The most common form of network flow recording is Cisco's NetFlow, which is a format and tool designed for network administrators that provides a set of services for IP applications [2]. It can capture all traffic that passes through a tapped network interface or router or sample that traffic at some specified rate, and there are tools for combining network flow data from multiples routes. In my implementation, a tool called fprobe is used to record network traffic, which is a libpcap-based tool that records network traffic data in real-time and emits it as session information that can then be turned into network flow records with a tool such as nfcapd. Each network flow record contains the source and destination IP addresses of the recorded session, with the total number of bytes that were transfered, as well as the protocol type, mainly TCP or UDP, with a timestamp and duration for the whole session. Each of these records is collected after the session is closed and these fields will be known and ready to be printed as a whole record line.

2.4.2 Identity Based Encryption and AES

An Identity Base Encryption (IBE) scheme is a public-key cryptosystem where the public key can be any string. Such a scheme was first proposed by Shamir in 1984 [7] but it was not until 2001 that Boneh and Franklin [8] gave a practical way to do identity-based encryption based on the Weil pairing. In IBE the strings for public keys can be email addresses, phone numbers, IP addresses, or any identifier, and can be merged with dates or represented hierarchically so that notions of time and hierarchical trust relationships can be included in the public key. The cryptosystem I used has chosen-ciphertext security in the random oracle model, assuming an elliptic curve variant of the computational Diffie-Hellman problem.

IBE has a lot of interesting features. One is that it allows encryption without the need to look up the public key or a certificate for the identity associated with the encryption key. Also, individual private keys within the scheme need not be generated or stored anywhere until decryption is necessary. If an encryption key is a public key that is a string merged with a timestamp, any attacker who discovers the private key associated with that timestamp will only be able to decrypt the messages for that particular time. Ideally IBE is used in systems where encryption is applied more often than decryption. One last property of IBE that I use in this chapter is that the secret needed to generate private keys associated with identities can be broken up among more than one party for separation of duty. This prevents abuse and can ensure that only trusted authorities combined can generate the private key that decrypts the suspect's records. For example, law enforcement agencies are only given data for the user and time period a warrant specifies, requiring the cooperation of multiple parties within an organization to obtain the necessary keys.

Advanced Encryption Standard (AES) is a symmetric-key encryption standard that comprises several sizes of block ciphers: AES-128, AES-192, and AES-256 [9]. The AES ciphers have been analyzed extensively and are now used worldwide. I use AES to encrypt network flow records for its speed, and then apply IBE to the stored AES key for each individual IP address and time period.

2.4.3 Statistical database modeling and inference controls

Statistics are computed for subgroups of records having common attributes [10]. A characteristic formula is any logical formula over the values of attributes where the set of records that matches these values is called a query set. A statistic is considered sensitive if it discloses too much confidential information about some individual, and with the use of some criteria for each query I prevent this type of query from returning an answer.

Inference control mechanisms are those which protect all sensitive statistics. The problem is that it can be extremely difficult to determine whether releasing a statistic will lead to disclosure of sensitive statistics information or not, but through the concept of query set overlap control I was able to keep the statistical reports from leading to the inference of any individual's sensitive data.

2.5 Implementation

Figure 2.1 shows the steps flow records go through in my implementation. The system is implemented in two main phases. Phase one addresses threats against stored, complete network flow records by encrypting them. Phase two is focused on data collection for producing statistical reports that are privacy-preserving based on specific criteria on statistical queries. This second phase addresses threats where these statistics, possibly combined, can reveal information about individual network users.

2.5.1 Data collection

Starting with data collection, fprobe 1.1 is used to record the network traffic that passes though a tapped network interface as sessions. Then nfcapd is used to collect the recorded data in a series of rotated files in Cisco's NetFlow format, each file containing 5 minutes of recorded data. After the completion of each of these files, the file is read separately and required data for statistics is imported from it and then stored into the statistical database.

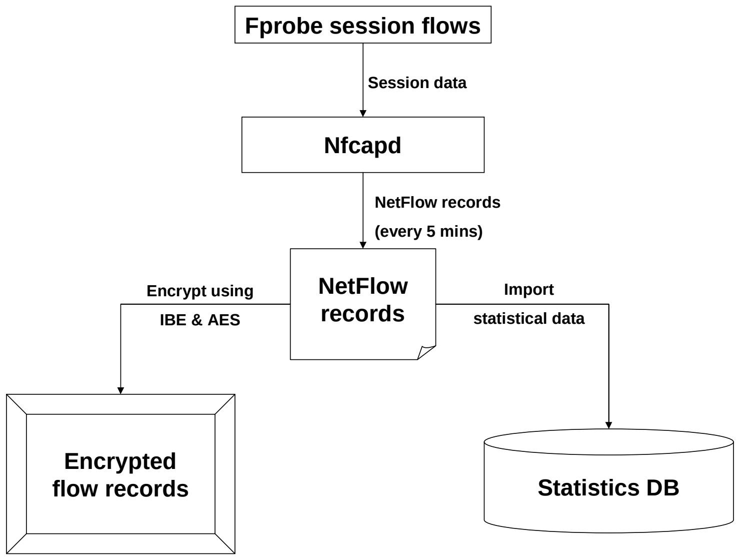


Figure 2.1: **Data collection steps, and examples of plaintext and encrypted network flow records.**

This database stores statistical network data that is needed for queries for statistical reports. Immediately after this, I encrypt the file. Thus, complete network flow data records will not be stored in plaintext for more than five minutes. Each encrypted file contains only the network flow records for a single IP address in a single 5-minute period.

2.5.2 Phase 1: Encryption vs. decryption

The encryption is done using a combination of identity based encryption (IBE) and AES encryption. My IBE implementation is custom written, using the pairing based crypto library [11] as a reference. Since my system privacy guarantee is per-user, I identify in each record the IP address that belongs to the domain (whether it is the source or destination) and merge it with the timestamp associated with this file that includes records for that IP address in that 5-minute period. All flow records for a particular IP address in a 5-minute period are merged into one file and encrypted with AES using a random 128-bit key that is different for each IP-timestamp tuple. After this, each of these AES keys is encrypted with IBE using the IP-timestamp tuple as the public key for this encryption. All plaintext files are immediately deleted, and the end result is a log file where each five minute period has a list of IP addresses with records for that time period. Each record contains the IP address and an AES decryption key that is encrypted with IBE. This information or the information in the statistical database could violate the criteria of my queries for generating statistical reports, but this is a separate threat and I have two separate threat models for the two separate threats. See Section 2.8 for more discussion about this.

When law enforcement requests the decrypted records for a particular IP address within a particular time period, the parties that have pieces of the secret needed for decryption must each aid in generating the decrypting IBE keys. If they all agree and cooperate, they can provide law enforcement with a set of keys that only decrypts the set of 5-minute periods specified and only for the IP address given. This set of decryption keys can be used to obtain the stored AES key for each of the relevant records.

2.5.3 Phase 2: Generating statistical reports

Statistical reports are generated after instantiating certain queries that comply with specific criteria designed to protect the privacy of individual network users. These criteria ensure that the queries not only prevent the direct revelation of information about individual users, but also prevent inferences of such information using any combination of possible queries. At the stage where the timestamped file of plaintext records is complete, before it is encrypted and the plaintext version deleted, some values needed for statistical reports are taken from this file and stored in a database that will be used to generate the statistical reports. This database does contain some individual information but cannot be accessed except through an interface with my predefined queries that are designed in order not to reveal any private information of a particular user, not even through inference. Generating statistical reports allows ISPs to monitor their network performance and the bandwidth of users, generate bills, and many other critical functions, so the major tradeoff is accuracy of aggregated statistics *vs.* privacy of individual statistics.

Before I discuss the queries and how privacy-preserving reports are generated, I need to explain the format of the database that such queries can be executed on. As explained before, every five minutes a new file of network flow records is completed. The statistical values needed are taken from this file for statistics, then encrypt each record in the file as explained before, and then the original plaintext file is deleted. The statistical values are stored in a database as one record per IP address per time period (TP). Note the difference between the recording time interval and the time period TP; a recording time interval is the five minute period in which network flow records are recorded in plaintext and then encrypted in a rotating fashion, a time period TP is a 12-hour period chosen such that the statistical reports generated will consists of at least 12-hours of statistical data records.

Another problem that might lead to the inference of an individual user's private information (such as what websites they visit) is if, for example, they only visited one website in one TP. The number of bytes transferred in this 12-hour time period can be used as a signature to make inferences about what website was visited. For this reason, one of the criteria is that there should be enough bytes transfered by an IP address in a given query so that very little can be inferred about specific websites visited. In this case, however, it is not enough to define some queries as legal and others as not legal. Two legal queries can form two datasets whose intersection leads to some confidential data about an individual. To solve this, I check the query criteria on the records for every TP and decide which IPs do not match the criteria within that TP. For IP records, I merge them with the corresponding IP record in either the pre- or post TP according to where that record was originally found. After merging these records, they are given a timestamp of 24 hours instead of 12 hours if they were merged with one TP, or 36 hours if merged with two TPs, and so on.

The statistical database is populated immediately after the network flow file is completed, new values in this database are entered or merged every five minutes in the following form:

- IP: IP Address
- TP: Time Period (time-stamped)
- TTI: Total TCP bytes In
- TTO: Total TCP bytes Out
- TUI: Total UDP bytes In
- TUO: Total UDP bytes Out
- LPI: List of Ports In
- LPO: List of Ports Out
- BI: Bytes In
- BO: Bytes Out
- PI: Packets In
- PO: Packets Out

The values of these attributes for a single record are updated every five minutes for every TP, per IP address, then a new record for the same IP address is created for the next TP and so on.

Generating the privacy-preserving statistical reports requires to design a set of queries with corresponding criteria for each to generate the required report that ISPs need, as well as not revealing any confidential data about individual users. Below is a list of some of the queries with their corresponding criteria:

$$Q1 : Sum[BI, (TP \geq \alpha) \bullet IP] \ \ \& \ result \geq \beta$$

$$Q2 : Sum[BO, (TP \geq \alpha) \bullet IP] \;\; \& \; result \geq \beta$$

$$Q3 : Sum[BI + BO, (TP \geq \alpha) \bullet IP] \;\; \& \; result \geq \beta$$

Q1, Q2, and Q3 are queries used to calculate the link utilization of a particular IP in a specified time interval for bytes-in, bytes-out, and total bytes respectively. The criteria and conditions for such queries is that $TP \geq \alpha$ and $result \geq \beta$ are satisfied. The values of α and β should be defined by the ISP depending on how large and busy the network is. Basically, the conditions for these queries state that if the number of TPs required is greater than α, a predefined number of 12-hour period statistical reports, the answer for such a query should also be above a certain threshold β. Otherwise confidential data of that particular IP could be inferred. In other words, if that particular IP was not active enough during these TPs, then the results output from this query could make it possible to deduce something about the websites or Internet services being used by a specific user. For example, Hintz discusses the concept of website fingerprinting [12] where it is possible to infer that a website has been visited through a signature based on the number of bytes transferred. So the value of β in the above queries should be chosen to be large enough such that no such patterns exist. One key feature about these conditions in the above three queries is that they must have the same conditional values (*i.e.*, the α values of the three queries must be equal, as must the β values), so that their combination cannot dissatisfy the conditions of any other statistical query.

$$Q4 : 8 \times \frac{\sum_{i}^{n} [BI{+}BO,(TP{\geq}\alpha){\bullet}IP_i}{TP_{\text{sec}}} \; \forall IP_i \in subnet, \; count(IP_{i_s}) > \delta \;\; \& \; result \geq \beta$$

Q4 is another example of a privacy-preserving statistical query that calculates the total bytes per time per subnet. Such queries require all the records of the IP addresses that fall in the given subnet in the given number of time periods such that $TP \geq \alpha$ and satisfying the same conditions that were specified in Q1, Q2, and Q3. This is important for the same reason of avoiding any combination of queries to infer individual queries that would violate their conditions.

$$Q5 : list[LPI, (TP \geq \alpha) \bullet IP_i] + list[LPO, (TP \geq \alpha) \bullet IP_i]$$

$$\forall IP_i \in subnet,\ count(IP_{i_s}) > \delta$$

One very useful query for ISPs is the list of applications used over a time period for particular subnetworks. For simplicity, applications are defined in terms of ports for my queries. Q5 is designed so that queries can return aggregate statistical data about ports used for large-enough groups of network users, but cannot query information pertaining to any single user. This query requires that enough data be included in the aggregation, both in terms of the size of the subnet and the amount of data available for that subnet.

The considered set of queries was designed to demonstrate that useful statistical data can be extracted from network flow records while still preserving privacy. I have focused on basic network administration and billing tasks. Additional queries could be defined for other purposes, the major challenge being the tradeoff between privacy and accuracy of the report.

2.6 Experimental Methodology

Here I explain the experimental setup and methodology. **The purpose of my experiments was to characterize the way that the traffic recording, identity-based encryption, and statistics generation scaled with respect to each other. In particular, I am interested in which of these might present problems as either the rate of traffic that is recorded goes up or the size of the subnet that recording is being performed for goes up.**

Because each IP address is an identity, performance scaling depends not only on the rate of flow records generated but also on the number of local IP addresses involved since the identity based encryption workload is per IP address.

2.6.1 Live experiments

One specific question I wanted to answer is: for a reasonably sized subnet, are my encryption and statistics-generating implementations fast enough to keep up with the rate at which network flow records can be generated on a typical commodity machine. I wanted to test this on a live system that was performing all of the packet capture and processing that a real system with the privacy-preserving capabilities would perform, so that system effects would also be accounted for. To answer this question, I ran live capture experiments for 6 hours each, on subnets of size `/24` (256 hosts), `/22` (1024 hosts), and `/20` (4096 hosts). The machine I ran these experiments on was a Core i7 X980 running at 3.33 GHz, with 24 gigabytes of RAM and a RAID 0 array with three 6 GB/s hard drives dedicated to the partition that I used. However, the motherboard RAID controller, though it uses PCI Express, limits the hard drive bandwidth to 3 gigabits per second so that this is the fastest rate I was able to test for long periods of time, due to the need for a large file so that tcpreplay does not loop too often over the same traffic. The Core i7 X980 has six cores, each with two hardware threads, and supports Advanced Encryption Standard Instructions (AES-NI) on all six cores. For both encryption and statistics generation I used six parallel threads (twelve threads would not have had a significant speedup over six since then pairs of threads would be sharing the AES hardware and memory hierarchies). The test machine was running Linux kernel version 2.6.32. Live experiments were performed at 3 gigabits per second due to the hard drive bandwidth limitation, but the offline experiments confirm that for the subnet sizes I considered more than 10 gigbaits per second is achieved by my implementation.

I generated a file representing enough realistic traffic to saturate a 3 gigabit link for

1 hour, to be looped six times for a six-hour live test. I then created a tuntap interface, which is a virtual network interface that works in a similar way to a loopback interface, but can be dedicated to just traffic for a particular experiment. Using tcpreplay, I filled this tuntap interface with realistic traffic that was generated based on a statistical profile of real network traffic, in terms of the sizes of flows, the timing, rate, protocols, and other factors. For this purpose, I use AnetTest [13] and modified it to support the subnet sizes I was interested in. I used other tools as references [14] in order to generate more realistic traffic rather than just depending on one traffic generator's statistical model. My goal was to consider the worst-case scenario in terms of the amount of sessions that could result from real traffic at a given rate.

By looping this 1-hour, 2 terabyte file six times I was able to do live testing of my privacy-preserving network flow recording implementation for six hours for each of three subnet sizes: `/24`, `/22`, and `/20`. The purpose of these tests was to demonstrate that my implementation could sustain a rate of 3 gigabits per second for network flow recording for a long period of time. The purpose of rerunning or looping this 1-hour files six times is for the purpose of eliminating any interference caused by the system and being more accurate in the results. I also wanted to see how well the different parts (encryption and statistics generation) scaled in terms of the amount of data collected in a 5-minute time period.

2.6.2 Offline experiments

I also sought to answer the question of how fast my implementation could perform for different subnet sizes, without hardware limitations or the inherent limitations of the Linux kernel in capturing network traffic. The reason this is important is that improvements in network traffic recording could lead to network flow recording at 10 gigabit rates, so the encryption and statistics generation for a privacy-preserving network flow implementation would have to keep up with this rate. To answer this question I generated libpcap files of traffic as before and then created the network flow records offline. Then the time that encryption and statistics generation took is measured when computation was the only limiting factor (since network flow records are small and not a significant factor in file system bandwidth). From this I inferred rates that this implementation could perform privacy-preserving network flow recording at for the same three subnet sizes: `/24`, `/22`, and `/20`.

2.7 Results

Recall that for live experiments my main purpose was to demonstrate that privacy-preserving network flow recording could be performed at the 3 gigabit per second rate on a commodity Linux machine. I answered this in the affirmative, but there are a couple of interesting caveats about the scalability of the system that I describe below.

Also recall that I ran off-line experiments to see if my implementation could achieve a 10 gigabit per second rate, assuming that future improvements to the Linux packet capture infrastructure make it possible to record traffic at this rate. I also answered this in the affirmative.

2.7.1 Live experiments

Figure 2.2 shows how encryption and statistics scale compared to the time to record on a `/24` subnet. The straight line that ends at 300 seconds is the amount of time that it takes to record the traffic, with 500,000 flow records being recorded in 300 seconds (5 minutes). This time is the same with or without performing any privacy-preserving operations such as encryption or statistics generation. Note that my implementation of these operations can be done in parallel to the recording of network flow records, so the times will not be added on but just need to be less than the time to record. The x-axis is the number of flow records recorded up to that point in time. For a fixed bandwidth and subnet size, the worst-case scenario for a network flow recording implementation (that is not sampling) is a large amount of very short sessions. 500,000 network flow records in a five minute period is what I considered to be a worst-case scenario for my experiments. Recall that this is a mix of large and small connections but is more biased towards small connections than I would expect normal traffic to be, to ensure that these tests cover all realistic scenarios by considering the worst case. The y-axis is either the time that has passed for recording the traffic, or the amount of time (out of that total time passed) that has been spent so far on encryption, statistics generation, or both.

The dashed line is the amount of time spent on encryption, including both identity-based encryption and AES. The thick black line is the amount of time spent generating statistics, and the thinner black line is the total time spent on privacy-preserving operations (*i.e.*, the sum of the times for encryption and statistics). Note that this time is cumulative, so linear growth means that the performance of that component is the same throughout. In this graph, it is clear that the privacy-preserving operations can be performed in the amount of time required, meaning that adding privacy-preserving operations to a network flow implementation will not change the rate at which it can perform. For a `/24` subnet at 3 gigabits per second, the encryption and statistics only took a fraction of the time allowed for them.

Figure 2.3 is the same as Figure 2.2, except that the scale on the y-axis is different to show more details about how encryption and statistics generation scale with respect to each other. I expected that encryption would scale non-linearly with subnet size because of identity-based encryption being required on a per-IP address basis. I was surprised to see that statistics generation grows nonlinearly with the number of network flow records, however. The reason for this is that merging statistics records for IP addresses that have already been seen in that five-minute period can be an expensive operation if not implemented carefully. In earlier versions of my implementation, where the dynamically allocated memory for operations such as appending port numbers to a list was not managed explicitly because the implementation had been in the Python language, the statistics actually broke the implementation at relatively low rates of traffic, meaning that statistics generation was taking more than five minutes and not keeping up with the network flow recording. I thought that this was due to the size of the hash table for hashing IP addresses that have been seen before in this time period, which does have some effect, but it turned out to be more related to the relatively simple addition and append operations that are performed to merge records for IP addresses that have been seen already in that time period. I found this result, that generating statistics could be more of a performance bottleneck than encryption, to be rather surprising.

Figures 2.4 and 2.5, respectively, show the same for `/22` subnets as what Figures 2.2 and 2.3 show for /24 subnets. Similarly, Figures 2.6 and 2.7 show the same for `/20` subnets. As expected, encryption becomes slightly more dominant in larger subnets since there are more unique IP addresses, as can be seen more clearly in Figure 2.8. However, I also see that statistics is still the main performance limitation in terms of non-linear behavior with respect to the number of flow records recorded. Since the hash table is reset every five minutes and the time it takes to merge a five-minute statistics record into a 12-hour TP is negligible, I do not need to consider how the statistics scales beyond five minutes.

My conclusion from the live experiments was that encryption and statistics are scalable and not performance problems compared to how long it takes to record traffic using pcap-based tools. This means that privacy-preserving network flow recording can be done for any realistic rate, traffic mix, and subnet size that regular network flow recording can be done at. However, special attention needs to be paid to the statistics generation, particularly the memory management for this part, to make sure that it scales well over time within a five-minute recording period.

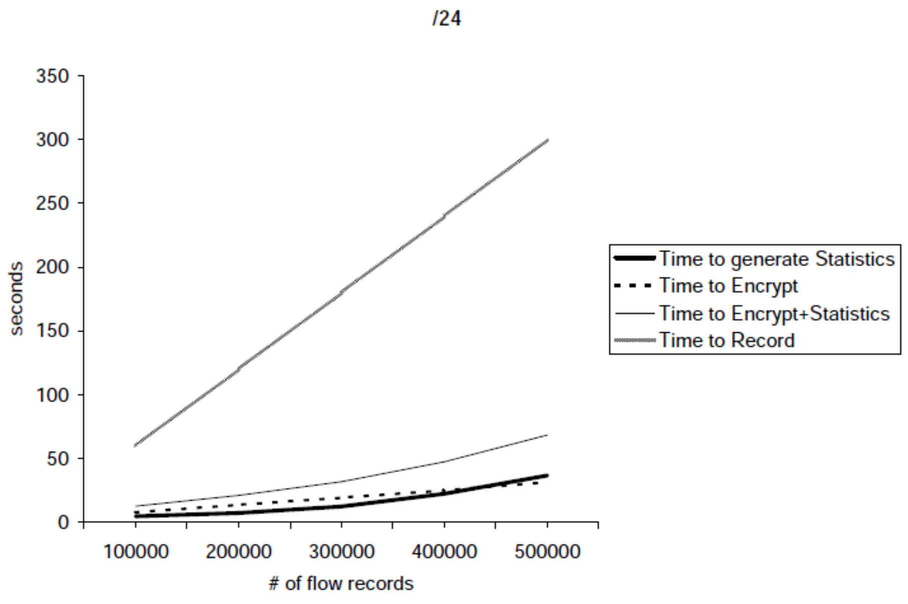


Figure 2.2: **Timings for `/24` Subnets.**

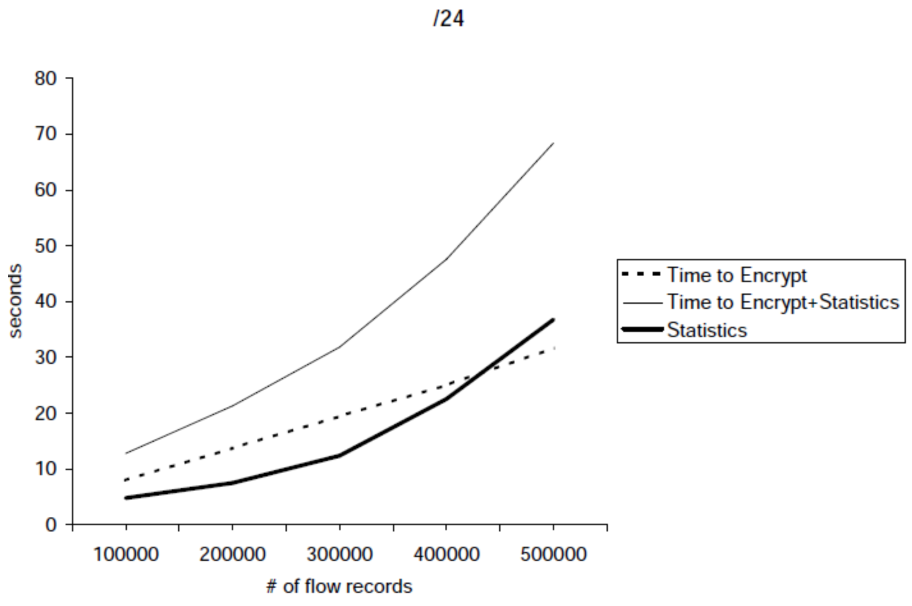


Figure 2.3: **Tradeoff between time to encrypt and time to import statistical data for `/24` subnets.**

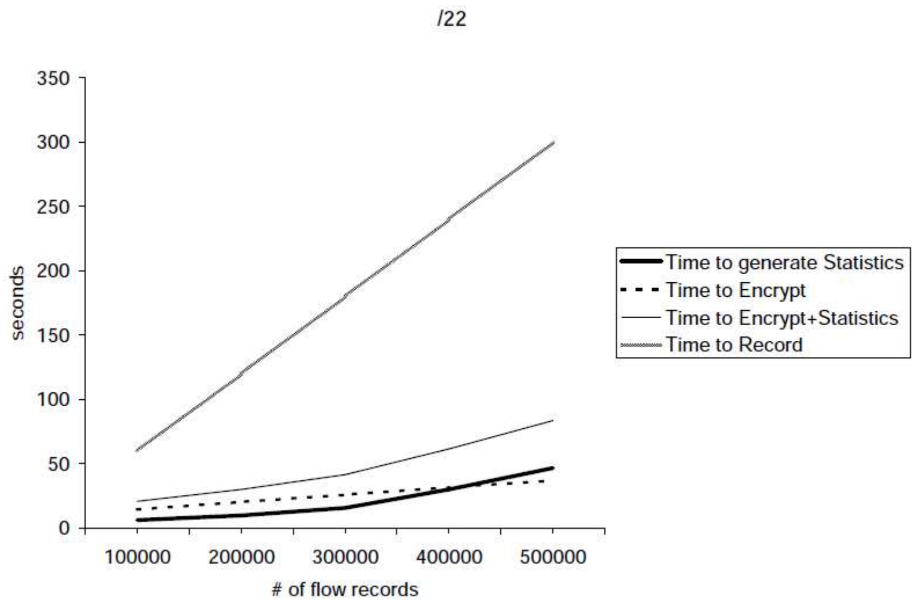


Figure 2.4: **Timings for /22 subnets.**

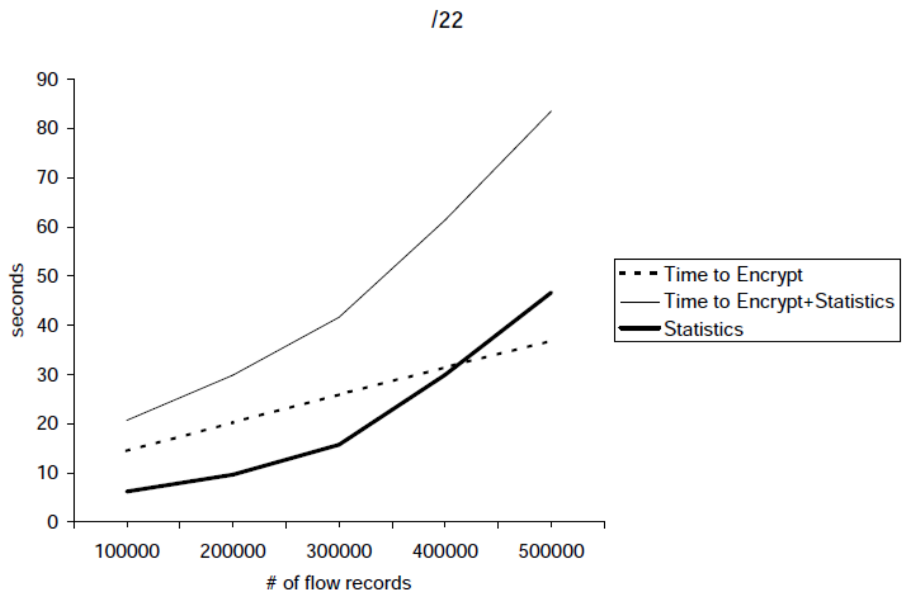


Figure 2.5: **Tradeoff between time to encrypt and time to import statistical data for /22 subnets.**

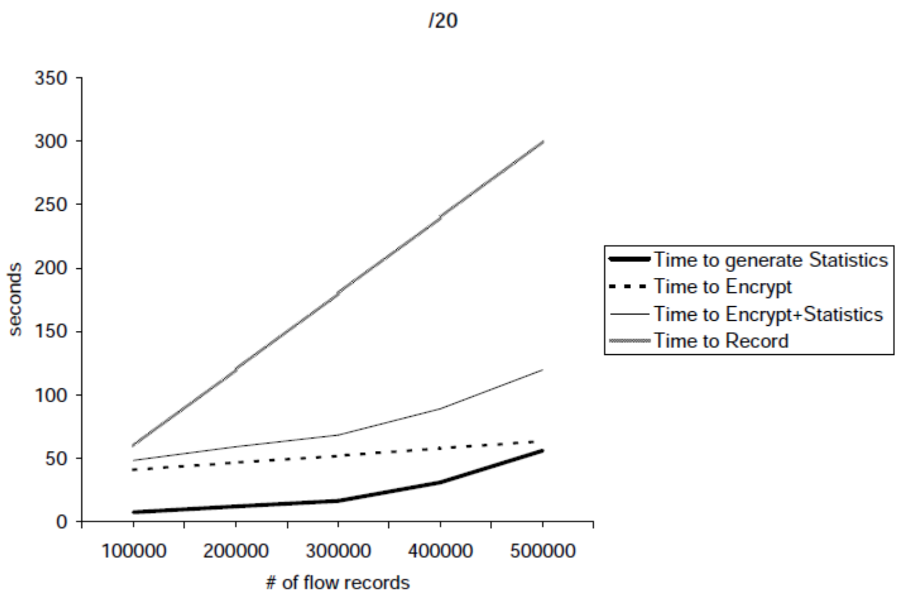


Figure 2.6: **Timings for `/20` subnets.**

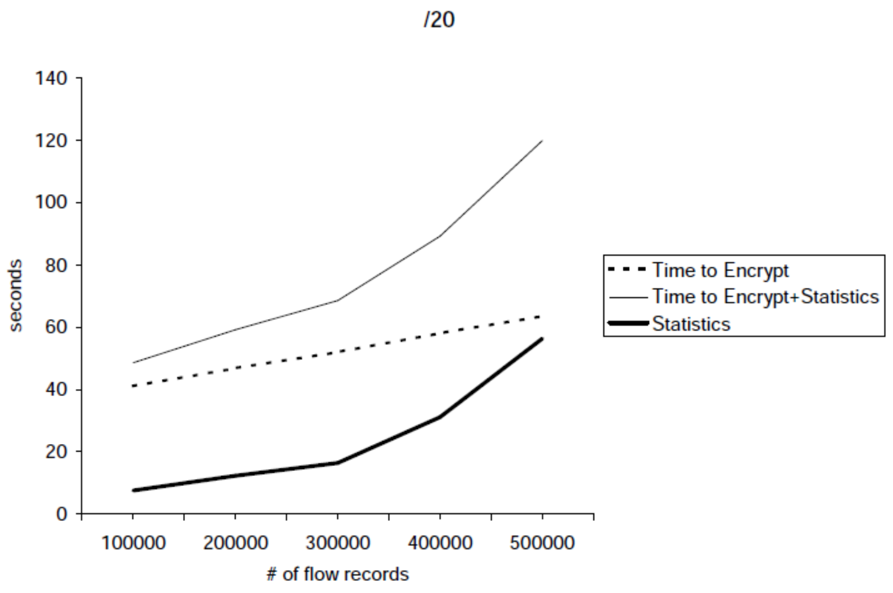


Figure 2.7: **Tradeoff between time to encrypt and time to import statistical data for `/20` subnets.**

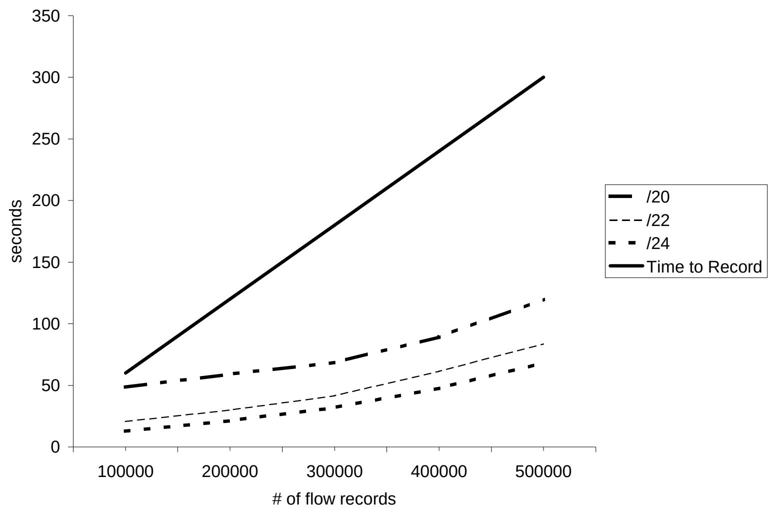


Figure 2.8: **Timing comparison between `/20`, `/22`, `/24` subnets according to recording time.**

Subnet size	Maximum rate (Gbps)
`/24`	23
`/22`	18
`/20`	12

Table 2.1: **Rates that my implementation can achieve for different subnet sizes.**

2.7.2 Offline experiments

Table 2.1 shows the rates, in gigabits per second, that I was able to achieve in the offline experiments for the three subnet sizes. Clearly, my encryption and statistics generation implementation can be used for 10 gigabit connections at any of these subnet sizes, assuming that the network flow recording itself is able to achieve such a rate.

2.8 Discussion and future work

I have shown that privacy-preserving network flow recording can be performed at any reasonable rate and subnet size that regular network flow recording can be performed at. However, there are still some open issues for future work. One is to consider other types of queries that network operators may require statistics for. The existing queries used are basic operations, and are all linear. The current set of queries used cover the most common uses of network flow data, however. Another issue that might be considered in future work would be to consider clickstreams and other data that should be kept in a privacy-preserving manner. Since clickstream data is used in different ways compared to network flow data, it may have different requirements for both encryption and statistics.

Two threats were considered in this chapter: threats against stored network flow data that is stored unencrypted, and threats against statistical data from the network flow recording that is exported to the rest of the organization. I considered these threats separately, and for the second threat I assumed that the statistics were exported only through a database interface. An attacker that can view the encrypted network flow records on the gateway where they are stored cannot violate the protections for the first threat since they are encrypted, but for the second threat many inferences can be made that could not be made through the database interface. The recorded, encrypted network flow data shows every local IP address that communicated in a 5-minute period (IP addresses outside the local network will be encrypted, however). Also, the length of the encrypted records gives some indication of the number of flows during that five-minute period. Since I considered the two threats separately and assumed that for the second threat the attacker would be required to use the database interface, I did not do any padding or attempt to prevent inferences from the encrypted network flow records. This raises an interesting question, however, which is: is it possible to store the encrypted network flow records in a way that prevents the same kinds of statistical inferences if an attacker gains access to the stored network flow records? I plan to explore this in future work.

2.9 Related Work

Different approaches have been taken in order to protect the confidentiality of web users that are applicable to recorded network flow data, even if network flow recording is not their focus. One main approach that users themselves can employ is to anonymize their traffic through anonymity networks. Tor, the second generation onion router [15], allows you to browse the Internet anonymously. Tor is resistant to many threats against privacy, including many of the threats posed by network flow recording. Tor trades off performance of the network connection for anonymity properties, for the users that choose to use it. In the problem domain I focus on in this chapter, I am interested in the threats against the vast majority of users that do not use Tor but whose daily web activities are recorded in the form of network flow records and clickstreams and then stored for weeks, months, or longer.

A research problem that is related to privacy-preserving recording and storage of network flows is the problem of anonymizing network flow data so that it can be shared with others. Foukarakis *et al.* [16] implemented anontool that allows administrators to anonymize network flow data in a highly customizable way. This is more directly related to the statistics generation than to the encryption phase of my implementation. For a discussion of issues concerning external factors in data sanitization, see Bishop *et al.* [17]. Privacy-preserving data aggregation and anonymization have a long history, see, for example, Denning [10]. Related to network flows, there are network trace anonymization attacks that correlate external information with anonymized data and successfully de-anonymize objects with distinctive signatures [18]. Moreover, some work has been done to improve Crypto-Pan by designing and integrating it with an efficient passphrase-based key generation algorithm in order to avoid the risk of exposing sensitive information to potential attackers while sharing network logs among security engineers and network operators for the purpose of security research and network measurements [19]. Koukis *et al.* designed and implemented a generic and flexible anonymization framework, due to the current anonymized tools that offer limited flexibility, thus their tool provides extended functionality, covering multiple aspects of anonymization needs and allowing fine-tuning of privacy protection level [20].

I have also seen research related to privacy-preserving forensic attribution architecture primitive where the authors propose a prototype implementation , called Clue, that attributes individual network packets of the sender's machine to protect the privacy of individual senders from serendipitous use, with the ability of revealing the identity of the criminal actors by a trusted authority [21].

Securing log files by cryptographic means was another considered approach towards hiding sensitive information that is stored in certain log files. Schneier and Kelsey [22] had described a computationally cheap method that makes it impossible for the attacker to read, modify, or destroy log file entries by generating log entries prior to the logging compromised machine.

Much previous work has focused on privacy-preserving data publishing. Denning [10] provides a good overview of early work in this area. More recently the notion of differential privacy was introduced by Dwork [23]. The focus is on privacy-preserving analysis of data, where the concept of differential privacy is used to capture the increased risk to one's privacy incurred by participating in a database. Differential privacy is a useful property for many types of proofs and reasoning about privacy. My work could be more formally analyzed within the framework of differential privacy, but for the current effort I have defined the queries in a way that more closely resembles the inference controls of Denning [10]. Zhou *et al.* [24] target the problem of continuous privacy-preserving publishing of data streams, as opposed to static data. They propose a group of randomized methods and anonymization quality measures used to verify the effectiveness and efficiency of their methods. The statistical reports could be cast as a streaming privacy-preserving data problem. Horizontally partitioned databases also fall into the category of privacy-preserving data publishing, as studied by Jurczyk and

Xiong [25]. They developed a set of decentralized protocols that enable data sharing for horizontally partitioned databases.

To the best of my knowledge, this is the first work to focus on one of the largest threats to web privacy today: the recording and storage of network flows and the related practice of clickstreams.

2.10 Conclusion

In this chapter, I proposed a privacy-preserving method for recording and storing network flow records, that network operators can use to enforce a privacy policy. A performance analysis of my implementation demonstrates that privacy-preserving network flow recording can be performed at any realistic rate, traffic mix, and subnet size that regular network flow recording can be performed at. I also showed that this implementation can achieve more than 10 gigabits a second on reasonably-sized subnets, as large as `/20`. The main tradeoffs and performance considerations that I identified were:

- The tradeoff between accuracy and privacy for defining privacy-preserving database semantics that allow network operators to view statistics about their network.
- The fact that identity-based encryption in my implementation scales with subnet size and not just the amount of traffic.
- The fact that the memory management of statistics generation scales non-linearly as the number of IP addresses and records grows.

Chapter 3

Network Flow Recording with Differential Privacy

3.1 Abstract

In chapter 2 I presented a privacy preserving way of revealing and recording netflow statistical reports using queries. While these queries were sufficient to preserve privacy, an adversary who had access to the database records without such queries could still reveal accurate individual records. In this chapter, I apply the concept of differential privacy applied to the statistical database generated from my netflow records, and I will show that I am still able to reveal useful information from the database as well as the adversary's failure of predicting individual records, given the database.

3.2 Introduction

Network flow recording, such as Cisco's NetFlow [2], is an important tool with applications that range from legal compliance and security auditing to network forensics, troubleshooting, and marketing. Storing such records for statistical purposes could reveal a great deal of private information related to the participating individuals and the network records itself. In chapter 2 I showed a privacy preserving approach of protecting individuals' personal information while still being able to retrieve the required data. This approach failed to address two important issues though. First, any direct access to the database without the built queries designed in Chapter 2 could leak a large amount of private information. Second, a lot of work needs to be done to work out a query every time network administrators require a new report. Thus, I felt the need to change my approach by applying the concept of differential privacy to the statistical database and reports.

Dwork [23] suggests a new measure of the privacy of individuals' data that resides in a database, differential privacy, that determines the risk of revealing such private data to the public. The motivation of her work was based on how to reveal useful information about population's data residing in a database while preserving the privacy of its individuals. According to Dwork, differential privacy attempts to guarantee the outcome of a calculation to be insensitive according to the dataset it resides in.

In general, differential privacy applied to statistical databases creates a possible trade-off between the usability of the database and the secrecy of individual records. Dwork in her survey paper [26] formalized and quantified the notions of private queries and developed analogues for different types of queries that meet the needs of database administrators. The risk is measured by how sensitive each query is, and is illustrated by the adversary's success in predicting an individual record's existence in a database.

This chapter is organized as follows. First, I describe the threat model in Section 3.3. This is followed by Section 3.4 where I introduce some basic concepts and their background. Section 3.5 shows the implementation of my system. In Section 3.6 I explain the evaluation methodology, and my results are presented in Section 3.7. I discuss performance issues and future work in Section 3.8, which is then followed by related works and the conclusion.

3.3 Threat Model

The fact that the statistical database contains some imported data from netflow records in plaintext is of great concern even though the reports generated from this database preserve privacy through my pre-designed queries. The main concern is that if an adversary was able to access the statistical database by bypassing my query interface, the records stored in the database are accurate and reflect a great amount of private information that would defeat the whole purpose of my approach. There were also concerns based on the concepts of forward and backward security as it applies to network flow recording privacy. Forward security means protecting information that will be recorded in the future, while backward security means that after a database has been compromised the data that was recorded in the past is still secure.

Statistics about how frequently network links are used are also very important just for day-to-day network operations. The threat is that information about the amount of network traffic an individual used must be stored for statistical purposes for the queries to be accurate (*e.g.,* for billing purposes). Unfortunately this information is often enough to infer certain facts about an individual user's web activities if it is not aggregated properly with other data before being presented. I assume that the stored statistical data is presented to the rest of the organization (the billing department, traffic engineers, quality-of-service experts, *etc.*) through a database interface, while those attacks that bypass such an interface is my main security threat.

I wish to reiterate that I am not attempting to protect web privacy against untrusted network controllers. My goal is to give network controllers the technological tools they need to enforce privacy policies so that network users can place their trust in the network controllers as an organization. Storing statistical data in a privacy preserving manner will protect users from any inappropriate access and will develop user experience trust.

3.4 Background

My approach is based on securely recording network flow data for forensic reasons, and importing some of the useful data from the netflow records into a statistical database, mainly for generating statistical reports. Recording netflow data is done with a sophisticated encryption algorithm composed of both AES and IBE (Identity Based Encryption). Statistics are computed for subgroups of records having common attributes [10]. Inference control mechanisms can protect all sensitive statistics. The problem is that it can be extremely difficult to determine whether releasing a statistic will lead to disclosure of sensitive statistic information, fortunately through the concept of query set overlap control I am able to keep the statistical reports from leading to the inference of any individual's sensitive data.

I had created a set of privacy-preserving queries, each formed with a set of criterias and constraints, and applied the concept of query set overlap control to check if the results of the queries could overlap and defeat the preservation of an individual's data records.

However, the above approach does not protect individual's privacy in the case that the adversary was able to bypass the pre-designed queries and thus was able to access the database records directly that contain accurate statistical data.

Moreover, my previous approach with the considered design of queries makes it hard for database administrators to create additional queries, as significant care must be taken so that new queries do not overlap another query's datasets.

My choice of applying differential privacy concepts in the statistical database is based on the premise that such methods provide formal privacy guarantees that are independent of the adversary's background knowledge of the values in the database. Using these methods also makes it easier to create new queries for generating new reports that could be applied on past or future data releases.

Dwork [23] has stated several definitions and privacy mechanisms that could be applied in different cases of data retrieval. Mironov et al. [27] has also presented a differentially private protocol for cases where privacy is needed to hold against efficient computationally bounded adversaries.

By definition, consider two data sets D_1 and D_2 that differ on a single element, a randomized algorithm A gives $\varepsilon - differential$ privacy if for all S subset of $Range(A)$, where $Range(A)$ is the output of the randomized algorithm A.

$$\Pr[A(D_1) \in S] \leq \exp(\varepsilon) \times \Pr[A(D_2) \in S]$$

The purpose is to define a differentially private algorithm A such that it will behave the same on both datasets D_1 and D_2. The measure of how private the algorithm A is defined by its sensitivity Δf of a function f such that:

$$\Delta f = \max_{D_1, D_2} ||f(D_1) - f(D_2)||$$

Laplace noise is a controlled noise that is added to the outcome of queries that will make them differentially private especially to those functions with low sensitivity. Specifically, the noise is sampled from the Laplace distribution has a probability density function (p.d.f.) of:

$$Y = \Pr[x] = \frac{1}{(2\delta)} e^{\frac{-|x-\mu|}{\delta}}$$

where μ is the mean of the distribution, and δ is a parameter that controls the degree of privacy protection. Thus the scale of Laplace noise and the sensitivity of the query applied are the factors that measures the privacy of the query results. In principle, the sensitivity measures the maximum change in query results due to changing a single tuple in the database.

Given a standard deviation λ, the output value of a given query A' will be the original output of A plus Laplace distribution noise such as:

$A'(x) = A(x) + Y \ where \ Y = Lap(\lambda)$

Since Y is a probability density function, that makes $A'(x)$ a continuous random variable where

$\frac{pdf(A'(x)_{D_1}=t)}{pdf(A'(x)_{D_2}=t)} = \frac{noise(t-f(D_1))}{noise(t-f(D_2))}$

which is at most

$e^{\frac{|f(D_1)-f(D_2)|}{\lambda}} \leq e^{\frac{\Delta f}{\lambda}}$

The privacy factor ε is now considered to be $\frac{\Delta f}{\lambda}$.

Differential privacy is achieved by adding noise to the outcome of regular queries. The sensitivity of each query should be defined to decide how much noise should be added to the output of the query generating the statistical report. Factors such as minimum TP range, link utilization, and subnet sizes could all affect the amount of noise need to be added to the data saved in the database.

In case of an arbitrary domain D, given a function $f : D \rightarrow \mathrm{R}^d$, the sensitivity of f is

$S(f) = max \parallel f(A) - f(B) \parallel$

With $S(f)$ being the sensitivity function of f, the computation

$M(X) = f(X) + (Laplace(S(f)/\varepsilon))^d$

provides ε differential privacy.

Moreover, let $q : (D^n \times R) \rightarrow \mathrm{R}$ be an equality function with database $d \in D^n$ and a score $r \in R$, and let $S(q) = \max_{r, A\Delta B=1} \parallel q(A,r) - q(B,r) \parallel$

Then a differentially private mechanism M with ε differential privacy is defined by

$M(d,q) = [\mathrm{r}eturn \ r \ with \ probability \propto exp((\varepsilon \ q(d,r))/(2 \ S(q)))]$

3.5 Implementation

Before proceeding to describe my privacy preserving solution, I will describe the format of the statistical database. Each record in the database contains the following fields:

- IP: IP Address
- TP: Time Period
- TTI: Total TCP bytes In
- TTO: Total TCP bytes Out
- TUI: Total UDP bytes In
- TUO: Total UDP bytes Out
- LPI: List of Ports In
- LPO: List of Ports Out
- BI: Bytes In
- BO: Bytes Out
- PI: Packets In
- PO: Packets Out

Every time-period corresponds to 12 hours of recorded data summed up into each corresponding data field. This time-period is considered in order to avoid retrieving data for any record for less than 12 hours old.

In my implementation, I considered three types of major queries needed by network administrators:

1- Link utilization of a particular IP address:

$Sum[BI \bullet IP]$ for bytes-in

$Sum[BO \bullet IP]$ for bytes-out

$Sum[BI + BO \bullet IP]$ for total bytes

2- Total bytes transferred within a subnet, represented in bits per second per subnet:

$$8 \times \frac{\sum_{i}^{n} [BI+BO \bullet IP_i}{TP_{\text{sec}}} \forall IP_i \in subnet$$

3- Lists of all applications used over a time period for particular subnetworks. For simplicity, I define applications in terms of ports for the queries:

$list[LPI + LPO \bullet IP_i] \forall IP_i \in subnet$

Next, I implemented my differentially private mechanisms denoted by S_i that is related to query q_i defined above, where each mechanism S_i has a corresponding privacy parameter ε_i. Each mechanism S_i works by calibrating noise to the sensitivity of each query q_i. The value of the privacy parameter ε_i is calculated for every mechanism S_i according to the level of accuracy that needs to be considered as well as the sensitivity of the sum query q_i, which is my case for the link utilization queries. In order to know the sensitivity of each query, each network user behaves differently and thus administrators need to bound the change in each query that may result in a change of a single entry for that particular query. My chosen queries belong to the non-unique class of query, those that do not have their sensitivity equal to 1, but are still easy to give some bound that is very unlikely to be crossed, (e.g. the link utilization queries). The bound is chosen per time-period, denoted as γ_i. Let γ_1 , γ_2, be the global sensitivity and ε_1, ε_2 be the privacy parameters for the corresponding first two queries (bytes-in and bytes-out) respectively.

Then the noise added to each database entry of bytes-in and bytes-out is Laplace noise $Y_1 = \gamma_1/\varepsilon_1$ and $Y_2 = \gamma_2/\varepsilon_2$ respectively. Thus, the value that is recorded into the database fields is the actual value added to the Laplace noise expressed by the differentially private mechanism $S_i(x) = q_i(x) + Y$, where x corresponds for the particular IP address and $q_i(x)$ corresponds to the actual number of bytes for that IP address. In this similar manner, the data entries in the database have been modified by adding Laplace noise since the queries applied on the database are of the summation form (as defined by Dwork [23]).

My set of queries was designed to demonstrate that useful statistical data can be extracted from network flow records stored in the database. I have focused on basic network administration and billing tasks. Additional queries could be defined for other purposes, with the major challenge being the trade-off between privacy and accuracy of the report.

3.6 Experimental Methodology

My main approach will be, instead of modifying the regular queries applied on the statistical database by adding conditions and constraints, I will rather keep the queries unmodified, and improve the way the statistical records are stored such that the data retrieved from those queries preserves privacy. Typically, differential privacy is achieved by adding noise to the results of a query or to the dataset itself.

Indeed network statistical databases hold sensitive information such as IP addresses and ports used. I consider applying the concepts of differential privacy by adding Laplace noise to database records such that none of the statistical reports can reveal any private information about individuals.

The Netflow recording experiment was repeated, and imported from those records the required statistical values that needed to be stored in to the statistical database. However, these entries are not stored with the same accurate values that were used in chapter 2, rather I use the differentially private mechanisms in order to safeguard private data in the database. In this case, I have added Laplace noise as the differential privacy methods for my type of queries.

Since the database will contain some noise in its values, such values will take care of the privacy preserving concerns of the data, and regular queries generate the needed statistical reports. For this particular purpose, the queries have to be set first before the decision of how much noise should be added to the data as well as the sensitivity of such queries. Factors such as minimum time periods (TP) range, link utilization, subnet sizes, and the sensitivity of the query could all affect the amount of noise that needs to be added to the data saved in the database.

3.7 Results

While I still have the actual values of recorded statistical values in the database from chapter 2, I have repeated the experiments with the new differentially private mechanisms. Table 3.1 shows a sample of the records taken from the database that reflects the change in the value of bytes-in and bytes-out for different IP addresses. Such differences, computed as Laplace noise, reflect the required values in reports that are generated by normal queries while preserving the privacy needed to secure an individual's private data.

Noisy statistical values protect from attacks such as website fingerprinting. Hintz discusses the concept of website fingerprinting [12], where it is possible to infer that a website has been visited through a signature based on the number of bytes transferred. Consider, for example, the link utilization query for a specific IP address, say user X, where the total bytes transferred during a Skype voice call is 6,750 kilobytes. The type of service used in this example is Skype voice call, which has been discovered through the port used, even though this is another private information that violates the privacy policy, but I am violating it just for the sake of demonstrating this example. It is also known through service fingerprinting that Skype voice calls takes between 7 to 18 kbps of

Bytes-In	Y_1	Noisy Bytes-In	Bytes-Out	Y_2	Noisy Bytes-Out
329873	-80	329793	23462	-30	23432
524872	24	524896	72346	42	72388
765538	63	765601	68798	-87	68711
768746	-175	768571	35674	13	35687
980743	144	980887	58796	-80	58716
537457	79	537536	35763	-73	35690

Table 3.1: **A table demonstrating the effect of applying the differentially private mechanisms related to link utilization values.**

bandwidth, depending on the network. The network behavior shows that Skype is using 15 kbps of bandwidth average. So if an adversary was able to see the accurate value of the Skype voice call session for user X to be 6,750 kilobytes, then the adversary could determine how long this voice call took (the length of the voice call tested was one hour). This might be a privacy concern, given that the adversary is familiar with Skype voice call speeds on the monitored network. My improvement in the database records is that it now has noisy values, the adversary will not be able to know the exact length of the user's voice call (in this example, the adversary discovered that the length of the voice call is 1.06 hours), as well as it will get more confusing when user X's total bytes is composed of many sessions of different services used. Thus differential privacy has relevant protection against attacks such as the website fingerprinting attack.

3.8 Discussion and future work

I have shown how crucial it is not to store accurate values in statistical databases since doing so could raise great privacy concerns and cause an individual's private data to be revealed. I applied differential privacy mechanisms on those records so they can preserve the privacy of individuals. However, there are still some open issues to address in future work. One is to consider other types of queries that network operators may require statistics for. Such queries might need other types of differentially private mechanisms rather than the Laplace noise case since not all queries are of a summation form. My existing queries are basic operations, and are all linear. However this current set of queries only cover the most common uses of network flow data. Another issue that might be to consider clickstreams and other data that should be kept in a privacy-preserving manner. Since clickstream data is used in different ways compared to network flow data, it may have different requirements for both encryption and statistics.

3.9 Related Work

There is considerable amount of work where differential privacy has been applied. Lindell and Omri [28] have applied differential privacy concepts on online advertising systems such as Facebook. Friedman and Schuster [29] considered the problem of data mining with privacy guarantees for a given data access interface based on differential privacy. Chaudhuri et al. [30] discussed how the infrastructure of database systems can hold sensitive data and can assist with privacy needs. It is my belief that differential privacy concepts will be applied more in the near future, with many possible applications in data storage and other fields.

3.10 Conclusion

In this chapter, I applied differential privacy concepts for the purpose of improving my privacy-preserving implementation of network flow recording. I have defined the type of queries needed to generate the reports and have moderated the values stored in the statistical database that corresponds to such reports, making great improvement to the netflow recording approach in the privacy-preserving world.

Chapter 4

Leaving Timing Channel Fingerprints in Hidden Service Log Files

4.1 Abstract

Hidden services are anonymously hosted services that can be accessed over an anonymity network, such as Tor. While most hidden services are legitimate, some host illegal content. There has been a fair amount of research on locating hidden services, but an open problem is to develop a *general* method to prove that a physical machine, once confiscated, was in fact the machine that had been hosting the illegal content. In this chapter I assume that the hidden service logs requests with some timestamp, and give experimental results for leaving an identifiable fingerprint in this log file as a timing channel that can be recovered from the timestamps. In 60 minutes, I am able to leave a 36-bit fingerprint that can be reliably recovered.

The main challenges are the packet delays caused by the anonymity network that requests are sent over and the existing traffic in the log from the actual clients accessing the service. I give data to characterize these noise sources and then describe an implementation of timing-channel fingerprinting for an Apache web-server based hidden service on the

Tor network, where the fingerprint is an additive channel that is superencoded with a Reed-Solomon code for reliable recovery. Finally, I discuss the inherent tradeoffs and possible approaches to making the fingerprint more stealthy.

4.2 Introduction

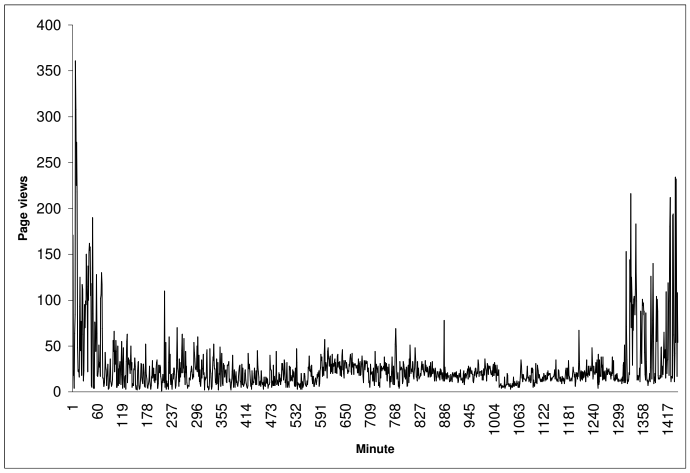


Figure 4.1: **Web server traffic for a 24-hour period.**

In this chapter, I consider the problem of leaving fingerprints in the log files of hidden services so that if the machine hosting the service is recovered by law enforcement the fingerprint can be recovered as proof that that particular machine was hosting the service. My threat model is the following. Illegal content is being hosted on a hidden service of an anonymity network such as Tor [15]. Hidden services allow clients on the anonymity network to access the service while preserving the anonymity of both the client and server. The server's IP address is not revealed to clients, instead clients request the service using a pseudodomain, *e.g.*, `http://gaddbiwdftapglkq.onion/`. There are many ways that the hidden service can be identified, both technical (*e.g.*, the methods [31, 32, 33, 34, 35] that I describe in Section 4.7 where I discuss related works) and non-technical (*e.g.*, the crime is reported). In this chapter I consider the following problem: given a hidden service that is believed to be hosted by a machine that will be confiscated by law enforcement, how can I leave a fingerprint on the machine through the hidden service that can be recovered and identified on the physical machine at a later time[1]?

The threat model I assume in this chapter is a passive observer that does not suspect this form of fingerprinting but does observe bursts in the log file. A stronger threat model where the hidden service host suspects that fingerprinting will be employed is left for future work. The approach I take in this chapter is to assume that the underlying service that is being offered as a hidden service has a log file of client requests that contains a timestamp. Logging can be disabled for hidden services, but the client information and information about traffic can be valuable to those offering the service for many reasons. Furthermore, the fact that suspects can erase their fingerprints in the physical world does not change the fact that dusting for fingerprints is a standard practice for non-digital crimes.

For my implementation, I use an Apache web server. By making requests to the service from a client on the anonymity network that will be logged, I can create an additive timing channel to leave the fingerprint in the log. The main challenge for this type of channel is the tradeoff between stealth and the amount of time it takes to leave the fingerprint. Because IP addresses are hidden by anonymity technologies, I assume that during the process of recovering the fingerprint no distinction can be made between the added requests and requests from real clients. The two main sources of noise that must be accounted for through redundancy in the fingerprint, then, are the delays of requests that are added by the anonymity network and bursts in the actual traffic from real clients for that particular service. I present results that characterize both of these sources of noise, and describe an implementation that can leave an easily recoverable 36-bit fingerprint in an Apache log file over the Tor network in 60 minutes.

There are three main reasons why, among the many information channels various log files afford, I focus on only timing channels using the timestamps:

[1]In practice, more than one fingerprint will typically be left to ensure sufficient evidence for conviction.

- For legal reasons, standardized methods are preferable to ad-hoc methods, because precedents can be established for well-analyzed algorithms for recovering a footprint. This requires that a single method be used for many services, and, while various services log different data that is application-specific, most contain some sort of timestamp.

- Anonymization technologies sometimes hide IP addresses, URLs, and other objects in the log file. For example, when Apache is set up as a Tor hidden service using privoxy [36], the IP address for all logged GET requests is `127.0.0.1` due to local proxying. Timing information, on the other hand, is typically preserved.

- By using exclusively timing and timestamps for leaving the fingerprint, the other channels of information (*e.g.*, the URL of the document being requested) can be reserved for other information that the fingerprinter may want to preserve in the log (*e.g.*, proof of the existence of a file on the server at a given time).

An important property that a forensic technique should have is generality. For Tor hidden services, a unique private key on the confiscated machine will easily identify that machine as the host of the hidden service. Other anonymity networks or future versions of Tor may not have this, however. A technique that can be established and used for a wide variety of cases is preferable to application-specific forensic techniques applied on a case-by-case basis. The suspect might uninstall Tor or disable the hidden service, which will make it harder to recover the unique private key they used to offer the hidden service. Suppose the log files are disabled on the hidden server or the attacker uses an encrypted file system. Depending on the noise model, a timing channel may appear in statistical packet capture logs of the suspect's local network or Internet Service Provider (*e.g., Cisco's Netflow [2]*). Thus, for this chapter I acknowledge that there are many ways to prove that a confiscated machine was hosting a hidden service but my focus is on timing channels.

The rest of this chapter is structured as follows. First, I describe the measurement methodology for characterizing the two main sources of noise in Section 4.3, followed by the results from these measurements in Section 4.4. Then, I describe my implementation of hidden service fingerprinting in Section 4.5. A discussion of stealth techniques and possibilities for future work is in Section 4.6. Related works are discussed in Section 4.7, followed by the conclusion.

4.3 Measurement methodology

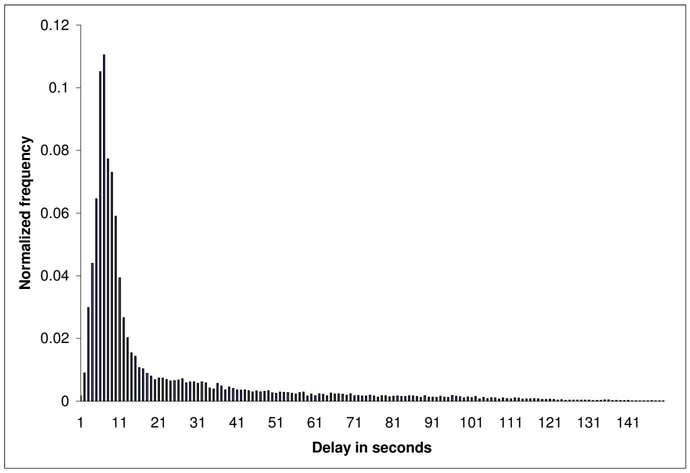


Figure 4.2: **Histogram for Tor delays in seconds.**

In this section I describe my methodology for two sets of measurements: delays of HTTP GET requests in the Tor network, and the traffic characteristics of existing GET requests for a web service. Because my fingerprinting method uses an additive code where GET requests are added to existing traffic in the log file, Tor network delays and existing GET requests are the two main sources of noise that must be accounted for to reliably recover the fingerprint.

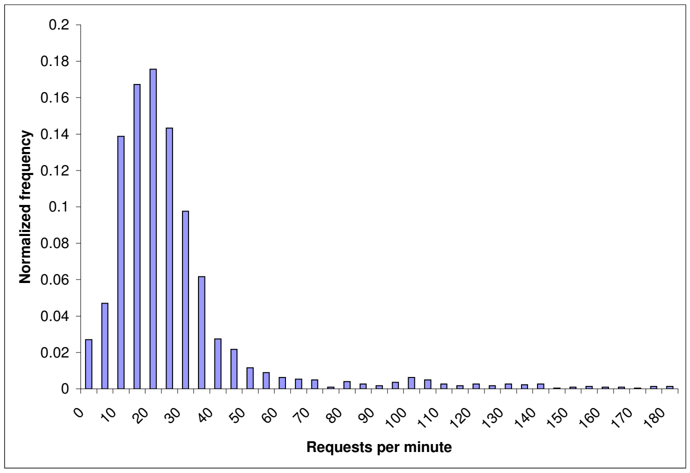


Figure 4.3: **Histogram for existing HTTP GET requests per minute.**

4.3.1 Tor network delays

To measure the delays of (and potentially also dropping of) GET requests that the fingerprinter as a client will send to the hidden web service, I set up two Linux machines. One hosts the Apache web server version 2.2.12 as a Tor hidden service and is configured using privoxy [36], which acts as a local web proxy interface to the Tor network. Thus, all GET requests appear to come from the IP address `127.0.0.1`, which is the loopback interface. The Apache log file logs GET requests when they are received, with a granularity of 1 second. For these experiments, the other machine acts as a web client, also configured with privoxy, with wget as the web client which makes requests for the hidden service. The server and client were located in the same place geographically, but the Tor onion routing default configuration is to set up a new circuit between the server and client every ten minutes[2] where the client chooses different entry, exit, and intermediate relays which are geographically distributed across the globe. *For measurement purposes only*, each GET request was tagged with a unique integer identifier so that I could easily detect dropping and reordering of packets. For my fingerprinting implementation in Section 4.5,

[2]The countdown of these ten minutes starts after the first use of each circuit, and will not cut off an existing session even if it exceeds this time limit.

I assume that no such identifier can be put in the GET request and use only the timing channel.

I sent a GET request from the client to the server every 5 seconds for 55 hours, to ascertain the distribution of delays introduced by onion routing and the connection loss rate. Because different circuits being built on a global scale every 10 minutes (after their first use) accounts for most of the variance in Tor delays, diurnal patterns were not evident in this data so I consider it as a single distribution independent of the time of day.

4.3.2 Existing HTTP GET requests

The other major source of noise in the fingerprint timing channel is existing traffic in the log file. For my implementation, I assume that the fingerprinter, when recovering the fingerprint, cannot distinguish their requests from existing requests (from real clients that are accessing the service). The additive channel is thus prone to errors due to bursts of traffic in the web server, so I sought to characterize existing traffic for a real web server. I obtained the log file for a university department that runs Apache 2.2.3 and receives approximately $35K$ to $40K$ requests per day. The log file I obtained covers a 36-hour time period.

I calculated the distribution of GET requests per minute for a normal web service using this log file. I assert that this distribution shape is representative of typical web services, but the mean of the distribution depends on how popular the web service is. My fingerprinting technique assumes that the fingerprinter has some estimate of this mean for a given web service that they want to fingerprint[3]. It is an open research question to know the mean of a given web service but one possible solution is for law enforcement to obtain this information from the ISP of the suspect.

Figure 4.1 shows the number of requests over time for this department web log file.

[3]This estimate can be a conservative estimate, at the cost of fingerprinting taking a longer time if the mean is overestimated, due to the redundant bits necessary for the error correction code.

There is a spike in traffic at about midnight MDT (Mountain Daylight Time), which corresponds to late afternoon local time. I designed this fingerprinting algorithm to be robust for any hour of the day, but since the fingerprinter can choose when to leave the fingerprint they can also choose a time of day when the traffic for the web service is known to be lower.

4.4 Measurement results

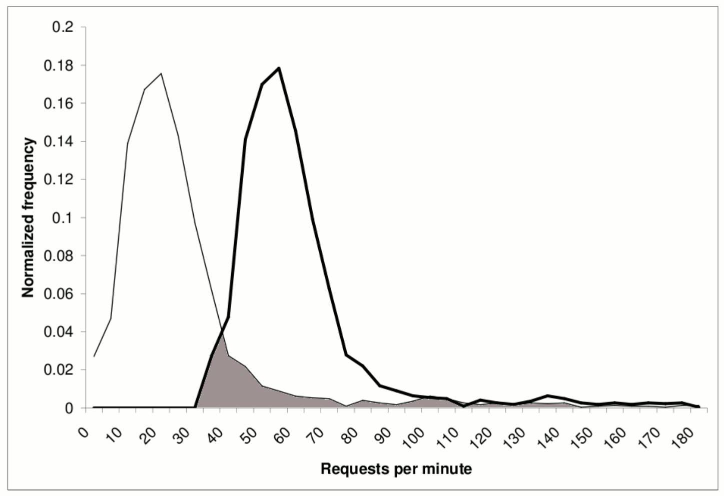


Figure 4.4: **Histogram of existing HTTP GET requests per minute before (thin line) and after adding additional requests for fingerprinting purposes (bold line).**

The purpose of these measurements was to characterize the two main sources of noise for my fingerprinting technique: delays or connection drops by the Tor network and existing traffic in the web service log file. The main tradeoff considered in this chapter is the amount of time it takes to leave the fingerprint *vs.* the number of requests I need to make per minute. The faster fingerprinting is performed, the more requests per minute will be necessary to reliably recover the fingerprint. The main consideration in this chapter is this tradeoff, I do not make any claims regarding the stealthiness of my current implemention. Other stealth techniques besides reducing the rate of requests are discussed in Section 4.6.

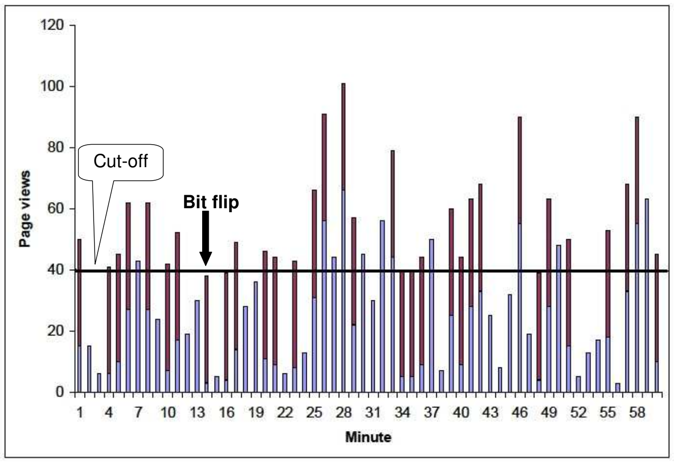


Figure 4.5: **How a 60-bit codeword appears in the log file.**

The main questions regarding the speed *vs.* request rate tradeoff that I sought to answer are:

1. When the fingerprinter sends a request, what is the loss rate and distribution of delays for when the server actually sees and logs the request? This is one of the two major sources of noise in my timing channel fingerprinting. Note that anonymity networks such as Tor deliberately route packets in unpredictable ways, meaning that the delays will also be much more unpredictable than normal Internet traffic.

2. What is the distribution of existing traffic in a web service log file? In particular, the relationship between the number of requests I add per minute and bit error rate when recovering the fingerprint is determined by this distribution.

4.4.1 Tor delays

Figure 4.2 shows the histogram for the delays added to requests by the Tor network from my measurements. The mean delay is 21.1. Not pictured are the 2975 requests that were

Input Codeword	**Page View / Min**	**Page View / Min after Fingerprinting**	**Output Codeword**
:	:	:	:
1	50	83	1
0	7	9	0
0	25	25	0
1	7	38	0
1	28	66	1
1	33	68	1
0	25	25	0
1	8	41	1
:	:	:	:

Figure 4.6: **Details on a bit error from the example.**

dropped. Based on the measured probability density, 83.6% of sent requests will be logged by the server within a minute of being sent, 8.9% will arrive after more than a minute, and 7.5% will be dropped. Based on these results, for the additive timing channel I chose to send all requests for a 60-second period in a burst at the beginning of the 60-second period. They will arrive with roughly the same distribution shown in Figure 4.2. Because of these delays, the server will not see the packets in a burst as the client had sent them. This is important so that conspicuous bursts of traffic do not appear in the log file.

4.4.2 Existing HTTP GET requests

Figure 4.3 shows the distribution of requests per minute for the university department log file. This distribution helps in predicting the rate of errors due to existing requests that can be expected for different numbers of requests per minute added. This is illustrated in Figure 4.4, which shows this error rate for the parameters I chose for the implementation as the shaded area under the intersection between the two curves (the bold line is the same distribution with 35 added to each value). For this chosen parameter of adding 35 requests in a minute for the additive channel, the error rate for errors that are due to existing requests is approximately 11.5% and the optimal cutoff to determine whether requests were added to a given minute is 40. Lowering the parameter of 35 to make the fingerprint less conspicuous will move the bold curve to the left and increase the error rate, meaning that longer error correction codewords are needed and therefore more time is required to leave a fingerprint that can be reliably recovered. Increasing this parameter moves the bold curve to the right which decreases the error rate and makes fingerprinting quicker, but creates a more conspicuous burstiness in the log file.

Figure 4.3 shows the same shape for the histogram as previous work on measuring Tor network delays [37], but the mean of my histogram is different. One possible reason for this is that the Tor network has grown considerably in the past several years. Another possible reason is that in that work changes were made to the Tor client to choose the first introduction point from the hidden service descriptor, whereas my measurements are end-to-end application-layer measurements from when the request was made by the client to when the server logged the incoming GET request. In general, I found the Tor network delays to have a large amount of variance between different circuits. In the additive channel, I can account for this variance, but in future work to improve the stealthiness of fingerprinting a more detailed and up-to-date model of Tor network delays will be needed.

4.5 Implementation

Based on the results in Section 4.4, I developed a prototype implementation of timing channel log file fingerprinting that is based on an additive channel that is superencoded with a Reed-Solomon code [38]. In this section I describe the implementation and present results to demonstrate that it is robust to Tor delays and existing requests in terms of the ability to reliably recover the fingerprint.

My implementation starts with a random 36-bit bitstring that will serve as the fingerprint. In this way, the probability of recovering a matching fingerprint by chance is approximately $\frac{1}{2^{36}} \approx 1.46 \times 10^{-11}$. Before being sent over the Tor network to appear in the server log file, this fingerprint is superencoded as a Reed-Solomon code word, which

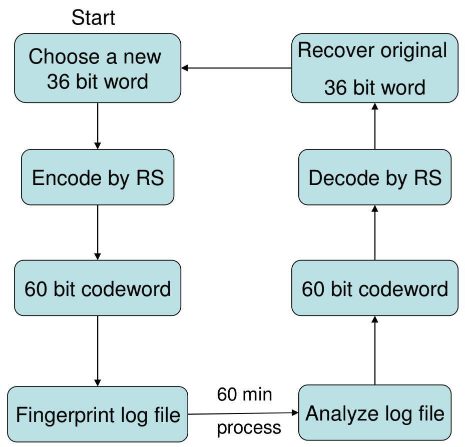


Figure 4.7: **The fingerprinting algorithm cycle.**

is 60 bits in length. Reed-Solomon coding is an error correction code that uses redundancy of the original word in a codeword that is based on an oversampled polynomial to recover from errors automatically (in contrast to an error detection code, in which errors could be detected, but not corrected).

I then transmit this 60-bit codeword, at a rate of 1 bit per minute, to the server's log file as follows. To transmit a 0 in a given minute, I do nothing. To transmit a 1 I make 35 requests for the hidden service over the Tor network at the beginning of the minute. These requests will arrive at the server and be logged with the distribution shown in Figure 4.2. The overall shape of the distribution of requests that will be seen in the log is not conspicuous, as is shown in Figure 4.8 where light bars show the histogram before fingerprinting and dark bars show the histogram after fingerprinting. Note that Figure 4.8 is normalized, the difference in magnitude between the distributions can be noticeable to the attacker but my threat model assumes a passive attacker who is not suspicious that fingerprinting is taking place (stronger threat models are left as future work).

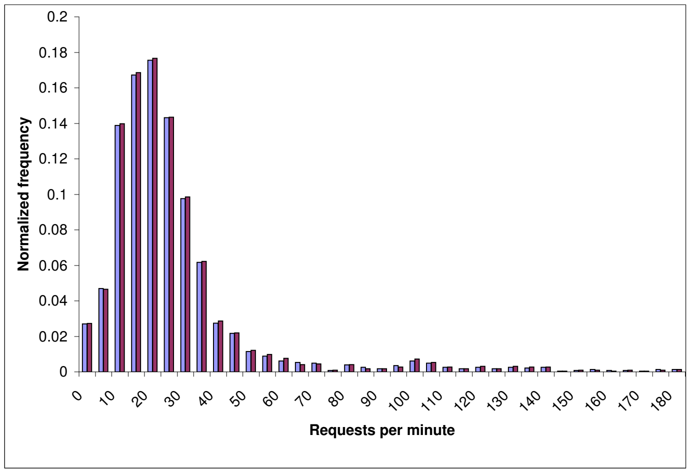


Figure 4.8: **Histograms with and without fingerprinting.**

To recover the fingerprint once the log file is obtained, I scan the log file near the time when the fingerprint was transmitted and attempt to recover the code word as follows. Within a minute, I read a 0 if less than 40 requests appear in the log file within that minute, and a 1 for 40 or more requests. The cut-off value of 40 was chosen because this is the approximate intersection point for the two graphs in Figure 4.4, which minimizes errors. I then apply Reed-Solomon error correction and compare the word that is recovered from the measured codeword to the fingerprint. When a match is found, then I am highly confident that the log file provided is the log file for the hidden webservice that was fingerprinted. Bit errors can occur in two different ways, which are discussed in Section 4.5.1.

I tested this fingerprinting implementation for different hours of the day for the department log file as follows. First, I do the fingerprinting for a Tor hidden service and record the requests as they are received by the hidden service. Then I superimpose this onto an hour of the log file. This is equivalent to doing the fingerprint live, since I am using an additive channel, but allows for repeatability of the results and does not require access to a hidden service that receives a lot of traffic. I tested the fingerprinting 24 times (once for each hour of the day) and were able to recover the fingerprint 22 times. This includes three tests that were performed for the three highest-traffic hours of Figure 4.1.

This was done to test the robustness of the fingerprinting implementation in the limit. The fingerprints recovered in the two cases that failed had a low Hamming distance with the original fingerprint, but my results suggest that leaving multiple fingerprints at different times of day is important in practice.

Note that Reed-Solomon codes perform well for correlated bit errors, which is why they are used in scenarios such as compact disc encoding where bit errors can be due to scratches. Thus, the small correlations between bit errors in my scheme (such as delayed requests showing up in the next minute) are easily handled by the superencoding.

4.5.1 Example

Here I give an example of one iteration of fingerprinting. The process is shown in Figure 4.7. The process can be repeated to leave multiple fingerprints at different hours of the day for added robustness, but here I describe only one iteration, which takes 60 minutes. The first step is to choose a random 36-bit word as the fingerprint. In this example I choose "`1101 1000 1111 0011 1100 0101 1010 0010 1101`". The second step is to apply Reed-Solomon encoding to produce a 60-bit codeword: "`1001 1101 0110 0101 1001 1010 1101 1000 1111 0011 1100 0101 1010 0010 1101`". For each minute in the 60-minute process, if the corresponding sequential bit in the codeword is a 0, I do nothing, if it is a 1 I make 35 requests from a Tor client to the hidden service at the beginning of the minute.

After the fingerprinting process is complete, I assume that the machine that hosted the hidden service has been physically recovered (*e.g.*, by law enforcement) and then the second half of the process is to recover the 36-bit fingerprint. To account for inaccuracies in the hidden server's system time, the process of attempting to recover the fingerprint can be repeated for some number of seconds into the past or future. For a given start time alignment, the 60-bit codeword that is received in the log file is generated as follows. For each sequential minute, I record a 0 if less than 40 total requests appear in the log during that minute, and a 1 if 40 or more appear. If there are 12 bit errors or less in the recovered 60-bit codeword then applying Reed-Solomon decoding to this codeword will produce the original 36-bit fingerprint.

There are two types of bit flips. One is that the number of requests from actual clients during that minute is very low and thus the number of requests I add to the log file for a 1 (which can be less than 35 if Tor drops some connections or delays them for more than 60 seconds) is not sufficient to put the total above the threshold of 40. This will flip a bit in the codeword from 1 to 0. The more common type of bit flip is from 0 to 1. This happens when either the number of existing requests from actual clients already exceeds the threshold, or it is near the threshold and a few delayed requests from a previous bit in the codeword show up during that minute instead of the earlier minute they were intended for.

Figure 4.5 shows how the codeword "`1001 1101 0110 0101 1001 1010 1101 1000 1111 0011 1100 0101 1010 0010 1101`" is added to a log file. The lighter bars are the existing requests from actual clients and the darker bars are the requests added by fingerprinting. This figure has 8 bit flips, one of which is highlighted and shown in more detail in Figure 4.6. Note that there are 7 bit flips from 0 to 1 and only 1 from 1 to 0. The received codeword in the log file will be: "`1001 1111 0110 0001 1001 1010 1111 1101 1111 1011 1100 0101 1110 0010 1111`". By applying Reed-Solomon decoding, I then recover the original fingerprint of "`1101 1000 1111 0011 1100 0101 1010 0010 1101`".

4.6 Discussion and future work

In this chapter I have explored the tradeoff in terms of how long it takes to leave a fingerprint in a hidden service log file *vs.* how much traffic must be added per minute. The threat model I assumed was a passive observer that does not suspect this form of fingerprinting but does observe bursts in the log file. For future work, I plan to explore the tradeoffs in a stronger threat model where the hidden web service host suspects that fingerprinting will be employed so that an extra degree of stealth in leaving the fingerprint is required.

In this work under progress I am modelling the probability distribution of the network delays, which is required for a closed-form expression of the uncoded bit error probability. This expression will in turn constitute the basis for predicting the probability of correct detection when superencoding is used. Furthermore, a time domain analysis of the gathered data will provide useful elements for the design of channel coding mechanisms, as they will depend among other factors on the coherence time. For instance, preliminary results show that the autocorrelation of the observed data can be reasonably modelled by an autoregressive process. The fact that the delays corresponding to consecutive requests are strongly correlated suggests that a differential encoding scheme would be beneficial.

A good model for both the distribution and the second order statistics of the request delay is also crucial for an information-theoretic approach to the problem, which will reveal what the fundamental limits of this delay-based communication scheme are. In addition, this approach will give insights for the design of better channel codes.

An underlying assumption of my current fingerprinting technique is that the fingerprinter has a good estimate of the mean of traffic requests per minute for the hidden service. This estimate can be an overestimate, which will cause the fingerprinter to use more redundant bits than necessary for superencoding and take longer than necessary to do the fingerprint. Overestimation also makes the fingerprinting more conspicuous. For future work, I plan to explore methods for estimating the traffic rate of a hidden service indirectly and accurately. For example, infrequent observations of the last access/modification time have been shown to be useful in estimating the access/modification rate [39]. A more general way to estimate the traffic load of a hidden service is still an open problem, however. What is needed is a way to exhaust a resource on the hidden server without first exhausting some resource within the anonymity network, and then a way to infer traffic load from the scarcity of that resource in a non-conspicuous way.

4.7 Related work

The work most related to mine is efforts to *locate* hidden servers. Overlier and Syverson [32] describe several different methods for locating hidden services in Tor and provide recommendations to prevent them, which were adopted by the Tor project. Murdoch and Danezis [34] consider the problem of traffic analysis, and Murdoch [35] considers Internet Exchange-level adversaries. Bauer *et al.* [33] describe attacks on Tor anonymity based on nodes misrepresenting the amount of resources they have. Murdoch [31] demonstrates that it is possible to determine whether a given IP address is hosting a particular hidden service or not based on clock skew. The basic idea is to send some amount of traffic to the hidden service, and query for TCP timestamps from the IP address suspected of hosting the service. As the server becomes busier, it will heat up causing a timing skew in the quartz crystal that maintains the system time, which will be seen in the timestamps. Murdoch also shows some results suggesting that geolocation is possible based on diurnal patterns associated with heat. In contrast to these works, my work assumes that the hidden server has been located and describes a way to prove that a physical machine (once confiscated) was the one that had been hosting the service.

My work falls in the general domain of covert timing channels [40, 41, 42, 43, 44]. There has been a considerable amount of work on creating and detecting covert timing channels based on TCP/IP [45, 46, 47]. To the best of my knowledge, this is the first work to consider timing channels as a method for leaving fingerprints in hidden service log files.

The relationship of my measurement results for Tor delays to the results of Loesing *et al.* [37] was described in Section 4.4. Other works have considered the timing characteristics of Tor and other anonymity networks in the context of timing attacks on anonymity [48, 49, 50, 51].

4.8 Conclusion

I demonstrated a technique for leaving timing channel fingerprints in hidden service log files. The technique presented in this chapter is robust, even for the random delays introduced by the Tor network and realistic, bursty traffic from existing clients for the web service. I was able to reliably recover a 36-bit fingerprint with a 60-minute fingerprinting process.

Chapter 5

Exploiting Geometrical Structure for Forensic Applications

5.1 Abstract

Timing inference channels are a well-studied area of computer security and privacy research, but they have not been widely applied in digital forensic applications. This is due to the fact that the timing signatures (for example, of movies) are not robust against variations in the machine, the encoder, the environment, and other factors that affect timing, and unfortunately such issues have limited many researchers from using timing inference channels for revealing hidden data, detecting machine behavior, or even forensic analysis.

In this chapter, I develop a geometrical interpretation in a high dimensional space of timing signatures for movies as an example of pattern-like software. My results suggest that timing signatures can be made robust against different machines, different encoders, and other environmental conditions by exploiting geometrical structure in this space. This geometrical structure will help identify the behavior of running pattern-like software that could be useful for identifying digital crimes, privacy invasion matters, as well as network

behaviors.

This chapter is centered around a thought experiment: how much information can an unprivileged process learn by just running on a system and observing its own timing? Although installing administrative software is the most frequent approach for understanding system behavior and detecting running software, my results show that it is feasible that such goals could be still achieved without any administrative privileges.

5.2 Introduction

This chapter is centered around a thought experiment: how much information can an unprivileged process learn by just running on a system and observing its own timing? There are many tradeoffs to be made between the power of forensics recording techniques and privacy, but the question of what privileges are needed is central to all of these tradeoffs. Timing inference channels hold a lot of promise for forensics applications and typically do not require special privileges on the system. In this chapter I focus on timing inference channels for identifying a movie that is playing on the system, and show that a geometrical structure in the space of vectors of timing measurements makes it feasible that these timing inference channels could be made robust to different machines and even potentially different encoders.

The geometrical structure I propose to describe movie timing channels is a sphere. Consider a vector of timing measurements related to a movie over time for some fixed-length snapshot. Imagine that movies put some average load on the CPU, which would be represented by a point on the constant vector. The encoder of the movie must maintain this average load, but must adapt to do more work when there are scene changes and less work when the video has less information to process. Thus all movies would form a sphere around the average CPU load point on the constant vector, with the fact that movies have scene changes in different places pushing the movies away from the center of the sphere in orthogonal directions. The distance that each movie's vector is pushed away from the center is an artifact of the tradeoffs the encoder is trying to make for storage *vs.* picture quality over time, so that this distance is relatively uniform across different movies. A movie with many scene changes, such as an action movie, may be further from the origin than a movie with few scene changes, but both would lie on the surface of this hypothetical sphere. In this chapter I test hypotheses about this geometrical structure and conclude that my geometrical interpretation of movie timing inference channels is supported by my data.

Privacy invasion has become a critical problem for Internet users. This invasion has the potential to cause users to begin avoiding some services due to the risk they may impose. The exact definition of privacy varies widely according to context and environment [52]. Some consider their privacy to be invaded when personal information is collected and used as data. Others consider genetic and drug testing to be an invasion of their privacy. Mail, telephones, emails, and social networks all play a part in communication privacy, and this has been a big concern.

The increased abundance of technology in society has caused an influx of electronic crimes [53]. Thus more rules and regulations are being created to deal with these crimes. Unfortunately evidence exists to suggest that such policies do not provide a higher degree of security. It has been noticed that in many cases the policies for using network services are more related to monitoring the traffic of its clients rather than providing security for their online transactions.

Downloading, possessing, watching, and distributing child pornography are among the fastest growing crimes over the Internet. There are now federal laws for e-crimes requiring corporations to report known incidents of child pornography [54], and digital signature technology has proved to guarantee authenticity, integrity and non-repudiation of critical, digitally stored, evidence of such crimes.

Timing inference channels would also be useful in some forensic cases investigating such e-crimes. Unlike inference channels where the sender transmits information by accident, timing channels are based on passing information by the use of system resources such as CPU time, packet send and receive times, or any other time-related interaction in the system [55]. Timing channels modulate the response time of a system and can create a covert message based on the resulted timings [56]. More interesting are timing inference channels where the modulation of the information being inferred is a property of the system and not necessarily intentional. In my case movie decoders modulate their CPU usage because of tradeoffs the encoder made and the specific structure of the encoded movie.

In principle, each movie would need to finish a certain amount of work (like decoding or constructing frames) in a certain amount of time based on the movie length. Thus the use of timing inference channels that I have developed, and described in this chapter, is targeted to detect the use of certain pattern-like media such as movies. Through these timing channels, I will be constructing a geometrical structure that represents those movies in a vector space as signatures, each with its own relative location to this structure. This approach is robust against the variation of machine hardware that technically should change those timings; still being capable of recognizing the movie signatures. My approach could be used in many forensic cases, such as detecting a child pornography movie that has been played on a particular machine on the network. It is only required that the user execute an unprivileged Java applet while connected to the network.

This chapter is organized as follows: First, I describe the threat model in Section 5.3. This is followed by Section 5.4 where I introduce some basic concepts and background. Section 5.5 shows the implementation of my system. In Section 5.6 I enumerate hypotheses and explain my experimental methodology for testing each, and then the corresponding results in Section 5.7. A discussion of my results and future work in Section 5.8 is then followed by related works and the conclusion.

5.3 Threat Model

In many cases, there is distrust between network administrators and their corresponding clients. On one side, the clients would like to use services that do not monitor their traffic or log their transactions. On the other hand, network administrators would like to keep track of the clients and the clients' corresponding transactions over their network to avoid abuse of their service, as well as to keep monitoring their network for other purposes such as consistency and continuous service. Moreover, network administrators and web developers have been asked by law enforcement to leave a security backdoor for their network and web applications to use in the case of suspected e-crimes [57]. For these purposes, network administrators will implement different types of auditing software that could help them identify and track any misuse of their network services such as e-crimes, or other acts that violate their terms and conditions.

It is also well known that such auditing software could be used to monitor the clients' behavior outside the domain of network misuse and digital crimes. The type of data that such software collects could contain private information of which clients should be aware. Such collected data could be abused to identify personal records and clients' network transactions. Some of this software will grant administrative rights on a client's machine where much more can be done. For example, SafeConnect, Symantec and McAfee network access controls are used by network administrators in many corporations and some educational institutions, where their users are required to install on their machines software for the purposes of network access control and applying network policies. These software programs are closed source and have administrative privileges on clients' machines that could create some privacy concerns for some users.

The very existence of criminal activity on the Internet could cast doubt on the true intentions of clients who unintentionally violate the policies of a network. A story was related to me that the law enforcement in Europe suspected that an Internet user had downloaded a movie of child pornography. The suspect's side of the story was that he had downloaded a legal movie whose content was replaced by scenes of child pornography. In such a case, knowing how much of the video the suspect actually watched could help in determining the suspects culpability.

Another incident happened in Germany where the largest European hacker club Chaos Computer Club (CCC) claimed in October 2011 on its blog [58] that the German government had developed software to gather information from target computers. They reverse-engineered and analyzed a lawful interception malware program called Bundestrojaner, or "government trojan," that can capture screenshots, record keystrokes, and record audio from sources like Skype calls. It is a backdoor that allows the installation and execution of additional software on infected computers.

My goal is to show that movies, as an example of pattern-like software, could form a geometrical structure through timing channels and allow network administrators to detect whether certain movies have been played on their clients' machines via a Java applet, eliminating the need for any client-side software with administrative privileges.

5.4 Background

My experiments were applied on movies that were encoded in MPEG format. It is essential to understand how MPEG movies are encoded and decoded, as well as the standards used for audio and video compression, because this is the CPU load that I am trying to infer through timing analysis. Different algorithms are used in the field of video compression and are used to encode to what is referred to as picture types or frame types. I, P and B frames are three major video frame types that are most commonly used by encoding algorithms for video compression. These concepts are key to my implementation so I shall describe some background about them here.

5.4.1 MPEGs

The MPEG standard was formed by the Moving Picture Experts Group, who were setting standards for audio and video compression and transmission [1]. The MPEG compression methodology is considered to be asymmetric, as the encoder is more complex than the decoder. The encoder needs to be algorithmic or adaptive whereas the decoder carries out fixed actions. The MPEG encoder is not standardized and implementers can supply encoders using proprietary algorithms, giving very little information regarding structure and operation of encoders. This leads to competitions to design better encoders. As a result users have the freedom of choosing encoders of different levels of cost and complexity. Decoders on the other hand are more standardized, and operate with all encoders. This is ideal for broadcasting applications, for example, where the number of expensive complex encoders is small but the number of simple inexpensive decoders is large. In MPEG, it is also possible to combine audio data with video data to produce a movie that includes the metadata used by decoders to demultiplex the movie correctly.

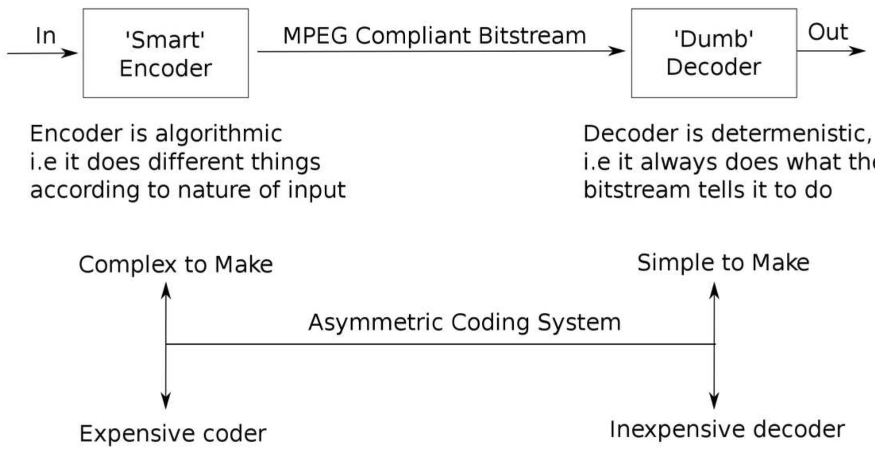

In
'Smart' Encoder
MPEG Compliant Bitstream
'Dumb' Decoder
Out
Encoder is algorithmic i.e it does different things according to nature of input
Decoder is determenistic, i.e it always does what the bitstream tells it to do
Complex to Make
Simple to Make
Asymmetric Coding System
Expensive coder
Inexpensive decoder

Figure 5.1: **MPEG Compression Overview, reproduced from page 3 of The MPEG Handbook [1].**

The MPEG standards include profiles and levels that make up certain aspects of the whole specification [59]. The profile defines the bitstream scalability and the colorspace resolution, while the level defines the image resolution and the maximum bit-rate per profile. MPEG-1 was released in 1993 and was used for encoding moving pictures and associated audio for digital storage. MPEG-2, released in 1995, improved upon MPEG-1 and was also used for generic coding of moving pictures and associated audio information. Advanced standards such as MPEG-4 Part 10 or SMPTE VC-1 have been used for high definition movies like those on Bluray. The difference between MPEG-2 over MPEG-4 is that the latter is used as an encoding method for devices with limited resources since MPEG-4 files are much smaller than those in MPEG-2, and are useful when streaming over the internet. So my preference of using MPEG-2 is because it is more popular and is used in DVDs.

5.4.2 Video Compression Picture Types

Video frames are compressed using different algorithms such as picture types or frame types that are directly related to data compression, each with different advantages and disadvantages, with I, P and B frames being the major picture types:

- I-frames are the least compressible but do not require other video frames to decode.
- P-frames use data from previous frames to decompress and are more compressible than I-frames.
- B-frames use both previous and next frames for data reference to get the highest amount of data compression.

Frames are segmented into macroblocks, and individual prediction types can be selected on a macroblock basis rather than being the same for the entire picture. For example, I-frames can contain only intra macroblocks, while P-frames can contain either intra macroblocks or predicted macroblocks. Finally B-frames can contain intra, predicted, or bidirectionally predicted macroblocks.

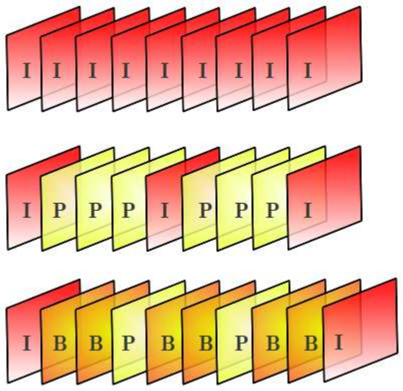


Figure 5.2: **Examples of different MPEG streams.**

In some cases, MPEG compression is considered to be an attempt to overcome some shortcomings of H.261. In H.261 for example, the absence of B-frames will cause many macroblocks to need information that is not in the reference frame. The MPEG solution is to add a third frame type which is a bidirectional frame, or B-frame. The main advantage of the usage of B frames is coding efficiency. In most cases, B frames will result in fewer bits being coded overall. One disadvantage is that the frame reconstruction memory buffers within the encoder and decoder must be doubled in size to accommodate the 2 anchor frames. While both B-Frames and P-Frames are predicted from their reference frames, only B-Frames also reference to a future frame in display order as shown in Figure 5.2. No matter whether a frame was coded as P-Frame or as B-Frame, there will be a difference between the predicted frame and the original frame. That difference is called the residual and will be stored in the file. As B-Frames can predict the frame better, the residual will contain less information and thus take less space. As Figure 5.2 shows, the top version has no P or B-Frames, and is the least efficient format. The second stream records I and P-Frames only, and the bottom stream records I, B, and P-Frames, which is the method commonly used in movies. My experiments will be based on this third type of encoding.

Specifically, I-frames will appear whenever there are scene changes in a movie, and P- and B-frames will cause a CPU load proportional to the action in the scene, leading to a distinctive, movie-specific pattern of CPU usage over time.

5.5 Implementation

In my implementation, I focus on an approach that does not require users to install any software or require any elevated privileges. ActiveX controls, for example, would require permission from users to use them whether they were signed or not. Once installed they can be harmful and can do anything the user can do [60]; since most users avoid running such objects on their machines due to the high risk they pose, it is definitely not a good choice. I was also interested in a script that is multi-threaded and could run within webpages, due to my interest to run multiple threads on all CPU cores of a given machine. JavaScript is also not an option since it is not multi-threaded.

Java applets, however, are multi-threaded and are used by many websites, especially those that require secure connection from clients such as banks, hotels or car rentals. While Java applets and ActiveX controls are both mini-applications developed to perform a specific task, there is great significant difference in their security. ActiveX controls are all or nothing: the user has made the decision to trust the control to do anything. On the other hand, Java applets have security built into their design [61]. The user can often define what the applet can and cannot do. Moreover, once ActiveX controls are loaded, there is no boundary to what the control can do. Once downloaded they remain on the computer until removed by the user [62, 63]. By comparison, Java applets are downloaded into the users' RAM and once the RAM is flushed the applet is no longer present [64].

In my case, I have coded a Java applet that maximizes the CPU load on all cores available in a given machine. Using the CPU this aggressively is not necessary in practice, see Section 5.8 for discussion about how this can be ameliorated. The applet contains simple threads, each with infinite loops that can simply do a job. I have also created a simple pop-up webpage that contains my Java applet, which is loaded once the machine is connected to the network. My approach assumes that once the user is connected to the Internet, she will receive this pop-up window and it must stay loaded as long as she wishes to be connected to the network. This pop-up window is considered a session window, and her connection to the network will stop if it is closed. The Java applet that is loaded by this pop-up window, however, will load all its threads into all CPU cores and count the number of loops executed in a given time frame. Those values are also sent gradually to the server and saved for further analysis.

In my experiments I played 80 movies, each of length 10 minutes, and recorded the number of loops executed every 50 msec. Thus 12,000 values were recorded for each movie in each run. The movies were encoded using ffmpeg with the same MPEG-2 encoder, and were played on 7 different machines with 3 different types of hardware specifications using Windows Media Player. The runs were repeated 10 times for each movie on every machine to characterize the variance. It was my intent to use the same encoder and media player, assuming that this configuration would be the worst case scenario compared with playing two different movies processed by different encoders. The 7 different machines with 3 types of hardware had the following specifications: three machines ran Windows Vista on a 2.40 GHz Intel Core 2 Duo with 4 GB RAM. Two machines ran Windows 7 with on 2.13 GHz Intel Atom processors and 2 GB RAM. Finally two machines ran Windows XP with 2.66 GHz Intel Core 2 Duos and 8 GB RAM.

I measured the number of loops executed by the Java applet that fully loads the CPU cores while the machine was idle, that is, without any other software interference. I also measured the CPU load per second when I just played each of these movies, while the machine is idle. Finally, I ran the Java applet together with each movie and recorded the number of loops executed by the Java applet script for every 50 milliseconds. This number of loops varied due to the applet and movies' competition for the CPU cores and other hardware resources.

5.6 Experimental Methodology

I performed several experiments for the purpose of understanding how movies affect the execution of the Java applet code as a result of their competition for hardware resources (mainly the CPU). Below I list a set of hypotheses that I tested for, each with its corresponding purpose and conditions to whether each of these hypotheses is shown to be supported by my data or not supported by my data, where testing is performed in Section 5.7.

Hypothesis 1 *On a given machine, playing each movie will produce a different pattern denoted by the CPU load.*

- Purpose:
  The purpose is to test whether different movies of the same encoder and length will produce the same pattern on the CPU load or not.

- Testing:
  This hypothesis will be shown to be supported by my data if the Euclidean distances of the CPU load values per second while running the same movie multiple times are at least 3 standard deviations away from the closest point of any other movie. I will calculate the mean and standard deviation of each movie, and compute the distances between the closest points of different movies, then compare and check if the closest distance between different movies is at least 3 standard deviations away.

Hypothesis 2 *On a given machine, playing each movie will reflect a different pattern denoted by the number of loops executed in the Java applet code.*

- Purpose:
  This is the same purpose as that of hypothesis 1, but the testing is now measured indirectly by number of loops executed rather than the CPU load.

- Testing:
  This hypothesis will be shown to be supported by my data if the Euclidean distances of the number of loops executed per $50msec$ between multiple runs of the same movie was again much smaller than the distances of two different movies. I will calculate the mean and standard deviation of each movie, and compute the distances between the closest points of different movies, then compare and check if the closest distance between different movies is at least 3 standard deviations away from the movie itself.

Hypothesis 3 *Each movie represents a vector in space for the number of loops executed every $50msec$, with a mean for all movies close to the constant vector.*

- Purpose:
  To locate the mean of all movies in space which determines the average CPU load.

- Testing:
  This hypothesis will be shown to be supported by my data if the cosine similarity between the mean vector of all movies and the constant vector is ≈ 1. In fact, if it is close to 1, then the mean vector is pointing in the same direction as the constant vector.

Hypothesis 4 *Movies of the same encoder and length played on the same player will form a regular simplex with vertices being the movie vectors, and I can construct an approximate spherical structure with the center being the mean value (calculated in the previous hypothesis) of all these movies.*

- Purpose:
  To understand the geometrical structure formed by all the movie vectors tested.

- Testing:

  I will experiment if the majority of the movies lie at a close distance from the surface of the sphere, whether inside or outside the sphere, where the mean distance between the mean point (described in the previous hypothesis) and each movie vector (among the tested movies that resulted in this mean) will be considered as the radius of the spherical like-structure in the space of $12,000$ dimensions. Then this hypothesis will be shown to be supported by my data if the movies are separated from the surface of the sphere by less than 0.5% of the radius size computed.

Hypothesis 5 *Running different movies on different machines moves the center of the spherical structure closer or farther from the origin along the constant vector.*

- Purpose:

  To understand the behavior of the geometrical structure upon testing on different machines.

- Testing:

  I will consider this hypothesis to be supported by my data if the mean vector is still moving along the constant vector, that is, the cosine similarity between the mean vector and the constant vector is ≈ 1.

Hypothesis 6 *The movie vectors that result from playing the same movie on similar machines (CPU, RAM, OS) will have some correlation and thus movies can be identified.*

- Purpose:

  To check if it is possible to identify movies running on different machines of similar specifications.

- Testing:

  I will only consider this hypothesis to be supported by my data if I can find some correlation of the playing of the same movie on physically distinct machines with similar specifications (CPU, RAM, OS). Measuring the cosine similarity of each movie vector from the center of the sphere with the same vector on a different machine will demonstrate this correlation.

Hypothesis 7 *The projection of any movie vector onto the constant vector is the center of the sphere for that particular machine inscribed by all other movies of the same length and encoding.*

- Purpose:
  I want to make sure that the movie vectors are not lying in the plane that is orthogonal to the constant vector. Another purpose is to see if it is possible to approximate the location of the center of the sphere from the values of a single movie.

- Testing:
  This hypothesis will be shown to be supported by my data only if the majority of movies projected on the constant vector meet in nearly one point, where the distance between the furthest points will be less than 1% of the radius length.

Hypothesis 8 *The standard deviation between the mean of different movies is very small compared to the standard deviation of the number of loops within the same movie.*

- Purpose:
  To understand the behavior of the execution of the Java applet code while playing different movies.

- Testing:
  This hypothesis will be shown to be supported by my data only if the standard

deviation of the mean of all movies on a given machine, denoted by the number of loops executed in the Java applet, is one tenth of the standard deviation within the values of each movie vector.

Hypothesis 9 *Based on hypotheses 4 and 5, the center and the radius of the spherical like-structure represented by movies with the same encoding can be determined using CPU benchmarking and a greedy algorithm that tests the CPU speed and Java applet code execution.*

- Purpose:
  To determine the center and radius of the sphere of movies with the same encoding on different machines.

- Testing:
  To test this hypothesis, I will compare the new sphere with the original one where all movies are played by measuring the cosine similarity between the corresponding vector from the center to each movie using the calculated center as well as the original. This hypothesis will be shown to be supported by my data for a particular movie if the cosine similarity value is greater than 0.85. As for the radius, I will measure the ratio of the calculated radius over the original radius and accept if it is within 5% of the original radius whether longer or shorter.

Hypothesis 10 *Based on hypotheses 5 and 9, playing the same movies on machines that vary from each other (different CPU speed and RAM) can have correlated signatures that can be detected through the spherical like-structure determined in hypothesis 4.*

- Purpose:
  To test if movies can be detected over machines that are slightly different.

- Testing:
  This hypothesis will be shown to be supported by my data if I was able to identify the movies based on normalizing, projecting, and measuring the ratio of the projection of the movie vector over the radius of the sphere. Specifically, for a given movie there should be a non-negligible correlation.

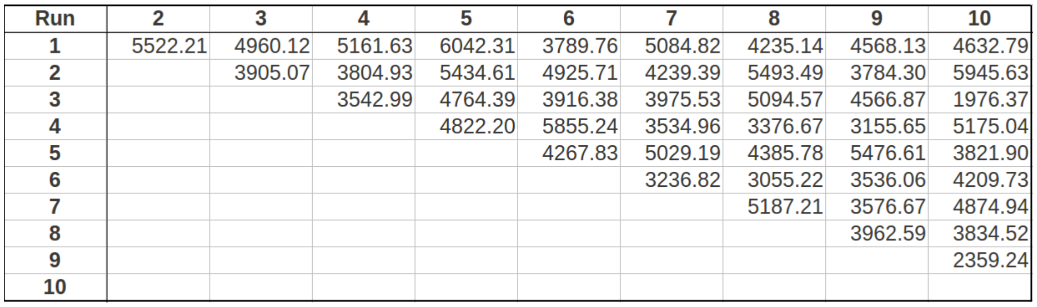
In the next section, I test each of these hypotheses and show whether each hypothesis is shown to be supported by my data or not supported by my data.

5.7 Results

Recall that the purpose of these experiments is to understand how the pattern of execution of the Java applet code changes according to the movie that is being played simultaneously. Below I describe each of the tests that determines whether the hypotheses I enumerated in Section 5.6 are shown to be supported by my data or not supported by my data.

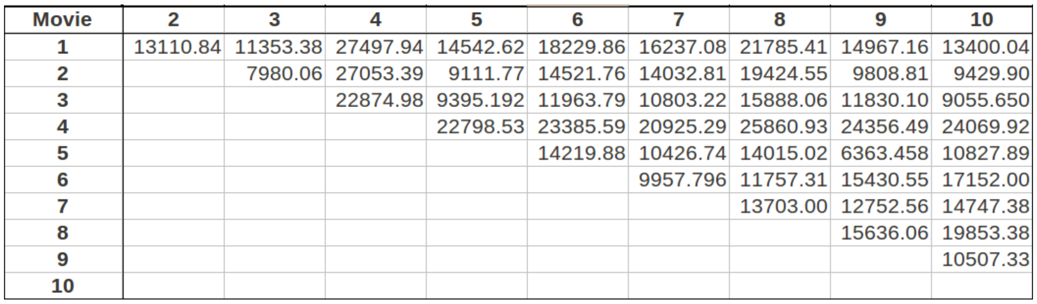
For Hypothesis 1, I examined 50 movies, each of which had the same length of 10 minutes and were encoded using the same MPEG-2 encoder. I ran each movie 10 times and calculated the Euclidean distances of their CPU loads as shown in table 5.1. Then I calculated the Euclidean distances of the CPU load of the different movies as shown in Table 5.2. I also computed the mean and standard deviation for each movie, and calculated all possible distances from each movie to the closest point of every other movie. Comparing the values in these tables, the Euclidean distance values in table 5.1, which fall between 1976.37 and 6042.31, and the values in table 5.2, which fall between 6363.46 and 27497.94. With the exception of 4 movies, those distances were 3 standard deviations away from the closest point of the other movies, thus I claim that **Hypothesis 1 is shown to be supported by my data,** specifically that the distances between different movies are much higher than the distances produced by playing the same movie repeatedly.

For Hypothesis 2, I have examined 80 movies, each 10 minutes in length and encoded



Run	2	3	4	5	6	7	8	9	10
1	5522.21	4960.12	5161.63	6042.31	3789.76	5084.82	4235.14	4568.13	4632.79
2		3905.07	3804.93	5434.61	4925.71	4239.39	5493.49	3784.30	5945.63
3			3542.99	4764.39	3916.38	3975.53	5094.57	4566.87	1976.37
4				4822.20	5855.24	3534.96	3376.67	3155.65	5175.04
5					4267.83	5029.19	4385.78	5476.61	3821.90
6						3236.82	3055.22	3536.06	4209.73
7							5187.21	3576.67	4874.94
8								3962.59	3834.52
9									2359.24
10									

Table 5.1: **Euclidean distances of running the same movie as a sample.**



Movie	2	3	4	5	6	7	8	9	10
1	13110.84	11353.38	27497.94	14542.62	18229.86	16237.08	21785.41	14967.16	13400.04
2		7980.06	27053.39	9111.77	14521.76	14032.81	19424.55	9808.81	9429.90
3			22874.98	9395.192	11963.79	10803.22	15888.06	11830.10	9055.650
4				22798.53	23385.59	20925.29	25860.93	24356.49	24069.92
5					14219.88	10426.74	14015.02	6363.458	10827.89
6						9957.796	11757.31	15430.55	17152.00
7							13703.00	12752.56	14747.38
8								15636.06	19853.38
9									10507.33
10									

Table 5.2: **Euclidean distances of running different movies.**

using the same MPEG-2 encoder. I ran the same experiments as for Hypothesis 1, but the number of loops executed were recorded for every $50msec$, giving $12,000$ records. I calculated the Euclidean distances for repetitions of the movies, as well as those for different movies. I also calculated the mean and standard deviation of each, and I measured the Euclidean distance of the closest point of each movie to the other. The mean of the Euclidean distances for the same movie is $52,517.82$ with a standard deviation of 5851.80, while the mean distance between all different movies is $227,941.36$ with a standard deviation of $78,182.41$ on a particular machine. With the exception of 8 movies, those distances were 3 standard deviations away from the closest point of the other movies, thus **Hypothesis 2 is shown to be supported by my data,** specifically that the distances between different movies are much higher than the distances produced by playing the same movie repeatedly.

I also calculated the mean vector for all movie records (both for the 80 movies, and for the $12,000$ records resulting from the numbers of loops executed in the Java applet code) and I measured the cosine similarity with the formula:

$$Cosine\ Similarity(\overrightarrow{A}, \overrightarrow{B}) = \frac{\overrightarrow{A}.\overrightarrow{B}}{\|A\|\|B\|} = \frac{\Sigma A_i \times B_i}{\sqrt{\Sigma A_i^2} \times \sqrt{\Sigma B_i^2}}$$

Cosine Similarity is a good measure for the correlation between two vectors. In a high dimensional space, it is very useful for measuring the proximity of vectors [65]. These movies resulted in a cosine similarity value of $0.996 \approx 1$. Thus I claim that **Hypothesis 3 is shown to be supported by my data,** as the cosine similarity value is close to 1. For confirmation, I performed the same experiment over all machines, which further supported Hypothesis 3.

For Hypothesis 4, and to minimize the noise in the movie records, I ran each movie 10 times. Doing so meant that each movie now represents a vector of $12,000$ dimension space. I calculated the mean vector of all 80 movies, and I calculated the mean vector of the individual movies. I then measured the distance from the total vector to each movie vector. The distances vary from one machine to another with a different CPU speed, so I focused on testing this hypothesis per machine.

Hall *et al.* [66] has shown that a low sample size data in a high dimensional space would lie deterministically at the vertices of a regular simplex. In my case, I have a high dimensional space of $12,000$ dimensions, and I represented the 80 movies as vectors, which would form a simplex as Hall *et al.* defined. I then calculated the mean value of all these movies which would lay in the middle of all vectors of the movies, that represents a center of a spherical-like structure in this high dimensional space. I also calculated the mean distance between the center of the sphere and all other movies and I considered this as the radius of the sphere. Finally, I measured the distance from each movie vector to the

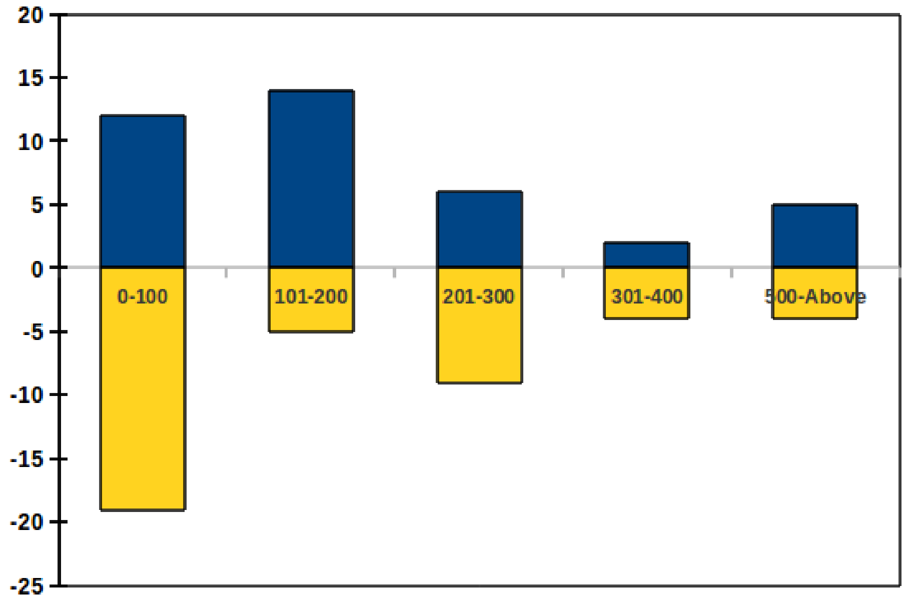
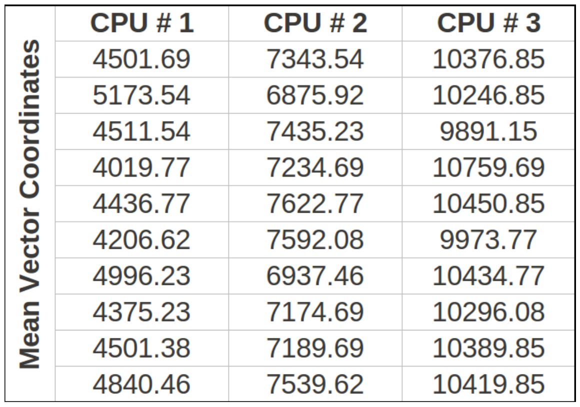


Figure 5.3: **Binning of movies based on distances of movie vectors from the surface of the sphere. Positive counts are outside the sphere, and negative counts are inside the sphere, binned by distance.**

surface of the considered sphere. Figure 5.3 shows how far the movie vectors are away from the center of the sphere with radius $\approx 862,600$, with positive values representing the number of movie vectors outside the sphere, and negative values representing the number of movie vectors inside the sphere. By comparison, my data had 64 of the 80 movies separated from the surface by less than 0.5% of the radius considered, while the rest of the movies were either further from or closer to the surface.

From this hypothesis, I considered the mean point in space to be the center of the sphere whose radius is the average distance between the mean vector and the movie vectors that complied with this hypothesis. **Hypothesis 4 is shown to be supported by my data.**

Testing Hypothesis 5 was done by applying all of the above experiments over different machines with different CPU speeds. As shown in table 5.3. The faster the CPU is, the faster the Java applet code is executed, thus the number of loops executed increases while the movie is being played, and vice-versa. Indeed, there is still a competition for CPU resources, but faster CPUs can manage these tasks more quickly. Table 5.3 shows a section of the mean vector coordinates (which are, in fact, $12,000$) over different machines. In fact, I can imagine that each movie needs to finish a certain amount of work (like decoding or constructing frames) in a certain amount of time based on its length. It then seems that this amount of work is the same across all machines, and the difference is when, and how fast, it is done. **Hypothesis 5 is shown to be supported by my data.**



	CPU # 1	CPU # 2	CPU # 3
Mean Vector Coordinates	4501.69	7343.54	10376.85
	5173.54	6875.92	10246.85
	4511.54	7435.23	9891.15
	4019.77	7234.69	10759.69
	4436.77	7622.77	10450.85
	4206.62	7592.08	9973.77
	4996.23	6937.46	10434.77
	4375.23	7174.69	10296.08
	4501.38	7189.69	10389.85
	4840.46	7539.62	10419.85

Table 5.3: **Sample mean vector coordinates over different machines.**

For Hypothesis 6, and in my initial setup, I had three sets of similar machines that have the same CPU, RAM, and operating system, and I tested this hypothesis on these sets separately. I constructed the vector from the center of the sphere (the mean of all movies) to the movie vector in each machine for each movie. I then measured the cosine similarity of the two vectors of each movie resulting from machines of the same set, as well as between different movies.

In all cases, the cosine similarity of two different movies in two different machines was almost zero, that is, they are orthogonal and there is no correlation. When it comes to the same movie, the cosine similarity of 63 movies out of 80 showed better values with a minimum value of 0.327. **I assert that this supports Hypothesis 6** that the resulting vectors are correlated. This hypothesis will be very useful for detecting if there is a possibility for two movies, running on similar machines, to be the same.

As for Hypothesis 7, this hypothesis is *not* supported by my data. For 20 movies, I calculated the mean vector of the sphere from all movies, and the distance between the center of the sphere and the projection point of each movie on the constant vector. On a particular machine one movie projection was as far as $150,000$ unit distance. Even though I am in a $12,000$ dimensional space, Figure 5.4 illustrates the intuition of why this hypothesis would not be true in two dimensions. Still, I tested this hypothesis because of the possibility that movies exhibit an interesting behavior in $12,000$ dimensions of being equidistant from the origin. My data did not have this property, indicating that the total amount of work over time to play a movie is not a constant.

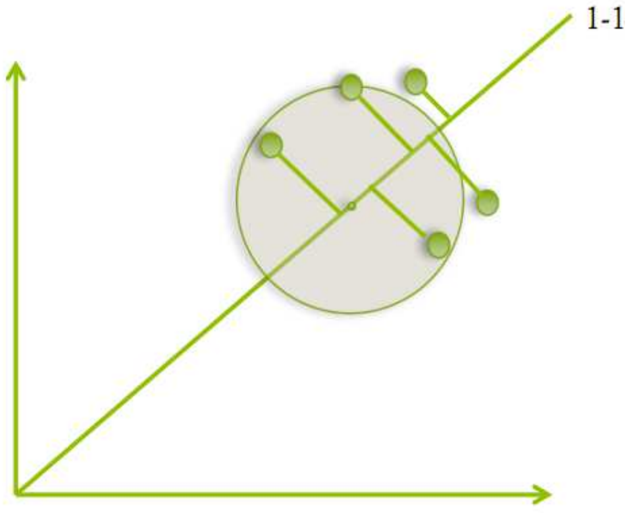


Figure 5.4: **Projection of vectors on the constant vector.**

After calculating the standard deviation of the mean of all movies on a given machine, its value tends to be around 50 while the standard deviation within the movie coordinates is about 700, which is greater than ten times more than those within each movie, thus **Hypothesis 8 is shown to be supported by my data.**

I then focused on determining the center and radius of the sphere of any machine before playing all the movies I was testing. Based on my assumption that all of the movies were processed with the same encoder, I wrote a CPU benchmark algorithm that runs for 5 minutes. During this time the machine needs to be mostly idle, but can have some background noise in terms of CPU usage. In using this algorithm, I continue to count the number of loops executed by the Java applet code while simultaneously running another process that loads the CPU at 0% for the first minute, 10% for the next 30 seconds, and for every 30 seconds thereafter increases the load to $20, 30, 40, 50, 60, 75$ and 100. This algorithm could be repeated to get more accurate results and better chances of catching the machine in an almost idle state.

The trade-off here is accuracy over time. The more I repeat this 5-minute algorithm, the more accurate my results are, but the longer it takes. While running this algorithm, I handle the noise and false positives caused by other system processes that might be running using a Reed Solomon error correcting code. This is robust to erasures due to the unexpected drop of the Java applet's loop execution at a certain loop counter.

Based on the values of this algorithm, I was able to estimate the ability of the CPU of the tested machine to execute the Java applet on different CPU loads caused by other simultaneous processes. The same algorithm was executed on a machine where the sphere specifications (center location and radius length) were known by running all the movies, and on other machines where such parameters were unknown. I then used a greedy algorithm that compares the values calculated by this technique between these machines, and were able to estimate the location of the center of the sphere with the corresponding radius for each machine. I then compared the sphere computed by running all movies, to the sphere computed by the CPU benchmark and greedy algorithm. The location of the center comparison was performed by measuring the cosine similarity between the computed center and the original center, which resulted in a minimum value of 0.847 for 58 movies out of 80. As for the radius comparison, I measured the ratio of the calculated radius over the original one, which resulted in values ranging from 0.957 to 1.03, which is less than 5% of the original radius, **thus I assert that Hypothesis 9 is supported by my data.**

Since I was able to estimate the center and radius of the spherical-like structure on different machines, I then tried to determine if I was able to detect the same movie that played on those machines. I considered each movie separately and played it on different machines. I normalized the vector of each movie on each machine and projected its space vector onto the constant vector, and I calculated the ratio between the vector from the center of the sphere to the projection with the radius. I then compared this ratio with the one from each machine and I was able to identify 52 movies out of 80, thus **I assert that Hypothesis 10 is supported by my data.** In fact, I can imagine that each movie would need to finish a certain amount of work (like decoding or constructing frames) in a certain amount of time based on the movie length. It seems that this amount of work is the same on all machines, and the difference arises in when, and how fast, the work is done. This also explains that, based on Hypothesis 10, playing the same movie on different machines would cause rotation and expansion of the movie vector in a high dimensional space, $12,000$ dimensions in my case.

5.8 Discussion and future work

I have shown that it is possible to model movies in a geometrical structure that could help identify them in different machines. Without administrative privileges on one's machine, CPU computations could reveal some information to network administrators which could be analyzed to reveal the execution of pattern-like software such as movies. My model's application could vary from forensic uses to studies for privacy preservation and protection of clients from such information extraction, and can increase the understanding of what CPU computation values could reveal without any administrative rights on the machine.

In this work, however, I have a few caveats and implementation issues that are left for future work. My focus in this chapter was on the geometrical characterization part rather than on engineering an actual timing inference framework. I am planning to work on a better approach for recording movie signatures in such a way that does not saturate the CPU at 100% load as I did in the Java applet. Information-theoretically speaking, there is no need for the CPU to be heavily loaded for information to flow through the timing inference channel, so long as there is at least some interference. Furthermore, it has been shown to be possible to hide a processes true CPU load from the operating system kernel's accounting mechanisms is Tsafrir *et al.* [67].

Moreover, the alignment of movie records with one another after they have been retrieved from a particular machine is another element on which I will be working. When it comes to understanding the CPU execution speed of different machines, more efficient algorithms than my CPU benchmarking and greedy algorithm are possible. An example of performance prediction was done by Oskin *et al.* [68] by using a new simulation technology, HLS, that combines statistical profiles with symbolic execution which can define the relationship between different machines with varied parameters.

5.9 Related Work

A great deal of previous work has focused on approaches for multimedia behavioral analysis and movie event detection. Enev *et al.* [69] defined electromagnetic interference (EMI) signatures of the power supplies that modern TVs produce based on the video content that is being displayed. Chua *et al.* [70] introduced MovieBase, a large-scale movie corpus that covers full length movies as well video clips downloaded from YouTube. For the purpose of gaining more control for movie decoders, Chen *et al.* [71] proposed an action movie segmentation and summarization framework based on movie tempo, which represents as the delivery speed of important segments of a movie. A new high-level video segmentation has been proposed by Chasanis *et al.* [72] that detects most scene and chapter boundaries by using temporal smoothing of visual word histograms of video shots. Finally, the classification of movies, mainly animated movies, was done by Ionescu *et al.* [73] by using temporal and color based categories of content descriptors.

Moreover, many approaches over the past years have been targeting the issue of privacy invasion when it comes to analyzing the data recorded by network administrators over their network. The most common technique used to protect the privacy of clients was to apply different privacy preserving approaches against the auditing software. Many privacy preserving techniques have been implemented, like those done by Agrawal and Srikant [74], that develop accurate models for aggregated data without access to precise information in individual data records. Further work by Evfimievski *et al.* [75] aimed for the same goal by using data randomization techniques, and in a later publication in 2007 using suppression, randomization, cryptography and summarization [76].

To the best of my knowledge, my approach is the first work using timing inference channels to classify pattern-like software in the context of forensics applications, in a way that is robust to different machines.

5.10 Conclusion

In this chapter, I proposed a geometrical model of how a movie's encoding interferes with the CPU usage of other running processes, and defined a structure that helps in identifying movies. CPU-related analysis allowed me to understand the behavior of such geometrical structures over different machines, and how a movie's location in space could be predicted when shifted from one machine to another. I have also shown that my approach does not require administrative privileges and thus could be very useful in detecting any software that has a pattern-like structure, and this can subsequently have many applications in the fields of forensics, privacy preservation, and fingerprinting.

Chapter 6

Conclusion

In many cases in the world of forensics one technique or piece of evidence is not enough to prove that someone is guilty or innocent. In fact, the combination of different techniques can reveal more evidence and clears up the picture of who is responsible for a certain crime. While using different techniques is relevant, the main concern still stands of how to protect the privacy of the people involved in the case and those who are related to it.

The techniques that are described in this dissertation are not in particular certain to prove the innocence and guilt of individuals, but more to help the investigation of identifying digital crimes as well as the responsible individuals.

I proposed a privacy-preserving method for recording and storing network flow records, that network operators can use to enforce a privacy policy. It is very useful to traceback the traffic of the suspect without any privacy invasion to the users on the same network. I used Identity Based Encryption (IBE) together with AES to encrypt the NetFlow records, and applied the differential privacy techniques on the statistical data for the privacy preservation.

I also demonstrated a technique for leaving timing channel fingerprints in Tor hidden service log files. The technique presented has shown to be robust, even with the random delays introduced by the Tor network and realistic, bursty traffic from existing clients for the web service. This technique helps in forensic cases where evidence is needed to prove if a machine was hosting illegal content through Tor hidden services.

Finally, I proposed a geometrical model that could help in identifying pattern-like software such as movies using inference timing channels which could be inferred without giving administrative privileges to the machine involved. CPU-related analysis allowed me to understand the behavior of such geometrical structures across different machines, and how a movie's location in space could be predicted when shifted from one machine to another.

These developed techniques and their proven results can subsequently have many applications in the fields of forensics, privacy preservation, and fingerprinting that could aid in many investigations where similar digital crimes occur.

In this dissertation, I showed how forensic tools could be implemented and used at in different levels of privilege that could affect the privacy of users involved in the forensic investigation. In most cases, the administrative privileges given to forensics personnel is not needed to achieve their goals, and thus if given, it could cause the users a privacy invasion risk. For this reason, this work showed how these investigations can be held without administrative privileges and how much information could an unprivileged task reveal from the users' machines.

References

[1] J. Watkinson, *The MPEG Handbook*. Focal Press, Woburn (MA), USA, 2001.

[2] "Cisco IOS Netflow," http://www.cisco.com/en/US/products/ps6601/products_ios_protocol_group_home.html.

[3] Solarwinds, "Orion NetFlow Traffic Analyzer," http://www.solarwinds.com/products/orion/nta/.

[4] Manage Engine, "NetFlow Analyzer," http://www.manageengine.com/products/netflow/cisco-netflow.html.

[5] CNET News, "Data Retention," http://news.cnet.com/8301-13578_3-9926803-38.html#ixzz0zTejcqJw.

[6] J. Angwin and T. McGinty, "Sites Feed Personal Details To New Tracking Industry," Wall Stree Journal, available at http://online.wsj.com/.

[7] A. Shamir, "Identity-based cryptosystems and signature schemes," in *Proceedings of CRYPTO 84 on Advances in cryptology*. New York, NY, USA: Springer-Verlag New York, Inc., 1985, pp. 47–53.

[8] D. Boneh and M. K. Franklin, "Identity-based encryption from the weil pairing," in *CRYPTO '01: Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*. London, UK: Springer-Verlag, 2001, pp. 213–229.

[9] Vincent Rijmen, Joan Daemen, "Advanced Encryption Standard," http://en.wikipedia.org/wiki/Advanced_Encryption_Standard.

[10] D. E. Robling Denning, *Cryptography and data security*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1982.

[11] Standford Crypto Group, "IBE Secure Email," http://crypto.stanford.edu/ibe/.

[12] A. Hintz, “Fingerprinting websites using traffic analysis,” in *Workshop on Privacy Enhancing Technologies*, 2002.

[13] Anton Titov, SourceForge, “AnetTest,” http://anettest.sourceforge.net/.

[14] pstavirs, Google Project Hosting, “ostinato,” http://code.google.com/p/ostinato/.

[15] R. Dingledine, N. Mathewson, and P. Syverson, “Tor: The second-generation onion router,” in *In Proceedings of the 13th USENIX Security Symposium*, 2004, pp. 303–320.

[16] M. Foukarakis, D. Antoniades, S. Antonatos, and E. P. Markatos, “Flexible and high-performance anonymization of netflow records using anontool.”

[17] M. Bishop, J. Cummins, S. Peisert, A. Singh, D. Agarwal, D. Frincke, and M. Hogarth, “Relationships in Data Sanitization: A Study in Scarlet,” in *Proceedings of the 2010 New Security Paradigms Workshop*, Concord, MA, September 21–23 2010.

[18] M. Burkhart, D. Schatzmann, B. Trammell, E. Boschi, and B. Plattner, “The role of network trace anonymization under attack,” 2010.

[19] C.-P. T. Cisco, “Network log anonymization: Application of.”

[20] D. Koukis, S. Antonatos, D. Antoniades, E. P. Markatos, and P. Trimintzios, “A generic anonymization framework for network traffic,” in *IEEE International Conference on Communications*.

[21] M. Afanasyev, T. Kohno, J. Ma, N. Murphy, S. Savage, A. C. Snoeren, and G. M. Voelker, “Network support for privacy-preserving forensic attribution,” University of California - San Diego and University of Washington, Tech. Rep. CS2009-0940, March 2009.

[22] B. Schneier and J. Kelsey, “Cryptographic support for secure logs on untrusted machines,” in *Proceedings of the 7th conference on USENIX Security Symposium - Volume 7*. Berkeley, CA, USA: USENIX Association, 1998, pp. 4–4. [Online]. Available: http://portal.acm.org/citation.cfm?id=1267549.1267553

[23] C. Dwork, “Differential privacy,” in *in ICALP*. Springer, 2006, pp. 1–12.

[24] B. Zhou, Y. Han, J. Pei, B. Jiang, Y. Tao, and Y. Jia, “Continuous privacy preserving publishing of data streams,” in *EDBT ’09: Proceedings of the 12th International Conference on Extending Database Technology*. New York, NY, USA: ACM, 2009, pp. 648–659.

[25] P. Jurczyk and L. Xiong, “Privacy-preserving data publishing for horizontally partitioned databases,” in *CIKM ’08: Proceeding of the 17th ACM conference on Information and knowledge management*. New York, NY, USA: ACM, 2008, pp. 1321–1322.

[26] C. Dwork, “Differential privacy: a survey of results,” in *Proceedings of the 5th international conference on Theory and applications of models of computation*, ser. TAMC’08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 1–19. [Online]. Available: http://dl.acm.org/citation.cfm?id=1791834.1791836

[27] I. Mironov, O. Pandey, O. Reingold, and S. Vadhan, “Computational differential privacy,” in *Proceedings of the 29th Annual International Cryptology Conference on Advances in Cryptology*. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 126–142. [Online]. Available: http://dl.acm.org/citation.cfm?id=1615970.1615981

[28] Y. Lindell and E. Omri, “A practical application of differential privacy to personalized online advertising,” *eprintiacrorg*, 2011. [Online]. Available: http://eprint.iacr.org/2011/152

[29] A. Friedman and A. Schuster, “Data mining with differential privacy,” in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, ser. KDD ’10. New York, NY, USA: ACM, 2010, pp. 493–502. [Online]. Available: http://doi.acm.org/10.1145/1835804.1835868

[30] S. Chaudhuri, R. Kaushik, and R. Ramamurthy, “Database access control and privacy: Is there a common ground?” in *CIDR*, 2011, pp. 96–103.

[31] S. J. Murdoch, “Hot or not: revealing hidden services by their clock skew,” in *CCS ’06: Proceedings of the 13th ACM conference on Computer and communications security*. New York, NY, USA: ACM, 2006, pp. 27–36.

[32] L. Overlier and P. Syverson, “Locating hidden servers,” in *SP ’06: Proceedings of the 2006 IEEE Symposium on Security and Privacy*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 100–114.

[33] K. Bauer, D. McCoy, D. Grunwald, T. Kohno, and D. Sicker, “Low-resource routing attacks against tor,” in *WPES ’07: Proceedings of the 2007 ACM workshop on Privacy in electronic society*. New York, NY, USA: ACM, 2007, pp. 11–20.

[34] S. J. Murdoch and G. Danezis, “Low-cost traffic analysis of tor,” in *SP ’05: Proceedings of the 2005 IEEE Symposium on Security and Privacy*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 183–195.

[35] S. J. Murdoch and P. Zieliński, "Sampled traffic analysis by Internet-exchange-level adversaries," in *Proceedings of the Seventh Workshop on Privacy Enhancing Technologies (PET 2007)*, N. Borosov and P. Golle, Eds. Ottawa, Canada: Springer, June 2007.

[36] "privoxy," http://www.privoxy.org/.

[37] K. Loesing, W. Sandmann, C. Wilms, and G. Wirtz, "Performance measurements and statistics of tor hidden services," in *SAINT '08: Proceedings of the 2008 International Symposium on Applications and the Internet*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 1–7.

[38] S. B. Wicker, *Reed-Solomon Codes and Their Applications*. Piscataway, NJ, USA: IEEE Press, 1994.

[39] N. Matloff, "Estimation of internet file-access/modification rates from indirect data," *ACM Trans. Model. Comput. Simul.*, vol. 15, no. 3, pp. 233–253, 2005.

[40] J. C. Wray, "An analysis of covert timing channels." in *IEEE Symposium on Security and Privacy*, 1991, pp. 2–7.

[41] B. W. Lampson, "A note on the confinement problem," *Communications of the ACM*, vol. 16, no. 10, pp. 613–615, 1973. [Online]. Available: citeseer.ist.psu.edu/lampson73note.html

[42] S. B. Lipner, "A comment on the confinement problem," in *SOSP '75: Proceedings of the fifth ACM Symposium on Operating Systems Principles*. New York, NY, USA: ACM Press, 1975, pp. 192–196.

[43] M. H. Kang and I. S. Moskowitz, "A pump for rapid, reliable, secure communication," in *CCS '93: Proceedings of the 1st ACM conference on Computer and Communications Security*. New York, NY, USA: ACM Press, 1993, pp. 119–129.

[44] S. Gianvecchio and H. Wang, "Detecting covert timing channels: an entropy-based approach," in *CCS '07: Proceedings of the 14th ACM conference on Computer and communications security*. New York, NY, USA: ACM, 2007, pp. 307–316.

[45] S. Cabuk, C. E. Brodley, and C. Shields, "Ip covert timing channels: design and detection," in *CCS '04: Proceedings of the 11th ACM conference on Computer and communications security*. New York, NY, USA: ACM, 2004, pp. 178–187.

[46] J. G. N, R. Greenstadt, P. Litwack, and R. Tibbetts, "Covert messaging through TCP timestamps," in *in Workshop on Privacy Enhancing Technologies*, 2002, pp. 194–208.

[47] S. J. Murdoch and S. Lewis, “Embedding covert channels into TCP/IP,” in *Information Hiding: 7th International Workshop, volume 3727 of LNCS*. Springer, 2005, pp. 247–261.

[48] G. Danezis, “The traffic analysis of continuous-time mixes,” in *Proceedings of Privacy Enhancing Technologies workshop (PET 2004)*, ser. LNCS, vol. 3424, May 2004, pp. 35–50.

[49] C. Diaz, L. Sassaman, and E. Dewitte, “Comparison between two practical mix designs,” in *Proceedings of ESORICS 2004*, ser. LNCS, France, September 2004.

[50] V. Shmatikov and M.-H. Wang, “Timing analysis in low-latency mix networks: Attacks and defenses,” in *Proceedings of ESORICS 2006*, September 2006.

[51] S. J. Murdoch and R. N. M. Watson, “Metrics for security and performance in low-latency anonymity networks,” in *Proceedings of the Eighth International Symposium on Privacy Enhancing Technologies (PETS 2008)*, N. Borisov and I. Goldberg, Eds. Leuven, Belgium: Springer, July 2008, pp. 115–132.

[52] C. Laurant, “Privacy International: Privacy and Human Rights 2003: an International Survey of Privacy Laws and Developments, Electronic Privacy Information Center (EPIC),” https://www.privacyinternational.org/survey/phr2003/index.htm, 2003.

[53] P. Wright and W. Fone, “Designing and managing networks to aid the capture and preservation of evidence to support the fight against e-crime.”

[54] R. P. Prabhu and R. P. Prabhu, “Child pornography on the internet, federal laws . . .” 2003.

[55] P. Kanellis, E. Kiountouzis, N. Kolokotronis, and D. Martakos, *Digital Crime and Forensic Science in Cyberspace*. Idea Group Publishing, 2006. [Online]. Available: http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Digital+Crime+And+Forensic+Science+in+Cyberspace+(N/A)#0

[56] A. Patel, M. Shah, R. Chandramouli, and K. P. Subbalakshmi, “Covert channel forensics on the internet: Issues, approaches, and experiences,” *I. J. Network Security*, vol. 5, no. 1, pp. 41–50, 2007.

[57] D. McCullagh, “Police want backdoor to web users’ private data,” 2010. [Online]. Available: http://news.cnet.com/8301-13578_3-10446503-38.html

[58] Chaos Computer Club, “Chaos Computer Club analyzes government malware,” 2011. [Online]. Available: http://www.ccc.de/en/updates/2011/staatstrojaner

[59] K. Diepold and S. Moeritz, *Understanding MPEG 4*. Focal Press, Woburn (MA), USA, 2004.

[60] R. Anand, N. Islam, T. Jaeger, and J. R. Rao, “A flexible security model for using internet content,” in *Proceedings of the 16th Symposium on Reliable Distributed Systems*, ser. SRDS ’97. Washington, DC, USA: IEEE Computer Society, 1997, pp. 89–. [Online]. Available: http://dl.acm.org/citation.cfm?id=829522.830930

[61] *Guide to Sun Microsystems Java Plug-in Security*, Network Applications Team of the Systems and Network Attack Center, 12 2003.

[62] J. Hurst, “Comparison of java applets and activex controls.” [Online]. Available: http://www.giac.org/cissp-papers/252.pdf

[63] J. M. MARE, “Risks of Java Applets and Microsoft ActiveX Controls,” Sans, Tech. Rep., 03 2002.

[64] S. Oaks, *Java Security*. O’Reilly, Sebastopol (CA), USA, 1998.

[65] S. Zhua, J. Wua, H. Xiongb, and G. Xiaa, “Scaling up top-k cosine similarity search,” *Data and Knowledge Engineering*, vol. 70, pp. 60–83, 2011.

[66] J. S. M. Peter Hall and A. Neeman, “Geometric representation of high dimension, low sample size data,” *Journal Of The Royal Statistical Society Series B*, vol. 67, pp. 427–444, 2005.

[67] D. Tsafrir, Y. Etsion, and D. G. Feitelson, “Secretly monopolizing the cpu without superuser privileges,” in *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*. Berkeley, CA, USA: USENIX Association, 2007, pp. 17:1–17:18. [Online]. Available: http://dl.acm.org/citation.cfm?id=1362903.1362920

[68] M. Oskin, F. T. Chong, and M. Farrens, “Hls: combining statistical and symbolic simulation to guide microprocessor designs,” in *Proceedings of the 27th annual international symposium on Computer architecture*, ser. ISCA ’00. New York, NY, USA: ACM, 2000, pp. 71–82. [Online]. Available: http://doi.acm.org/10.1145/339647.339656

[69] M. Enev, S. Gupta, T. Kohno, and S. N. Patel, “Televisions, video privacy, and powerline electromagnetic interference,” in *Proceedings of the 18th ACM conference on Computer and communications security*, ser. CCS ’11. New York, NY, USA: ACM, 2011, pp. 537–550. [Online]. Available: http://doi.acm.org/10.1145/2046707.2046770

[70] T.-S. Chua, S. Tang, R. Trichet, H. K. Tan, and Y. Song, "Moviebase: a movie database for event detection and behavioral analysis," in *Proceedings of the 1st workshop on Web-scale multimedia corpus*, ser. WSMC '09. New York, NY, USA: ACM, 2009, pp. 41–48. [Online]. Available: http://doi.acm.org/10.1145/1631135.1631143

[71] H.-W. Chen, J.-H. Kuo, W.-T. Chu, and J.-L. Wu, "Action movies segmentation and summarization based on tempo analysis," in *Proceedings of the 6th ACM SIGMM International Workshop on Multimedia information retrieval*, ser. MIR '04. New York, NY, USA: ACM, 2004, pp. 251–258. [Online]. Available: http://doi.acm.org/10.1145/1026711.1026752

[72] V. Chasanis, A. Kalogeratos, and A. Likas, "Movie segmentation into scenes and chapters using locally weighted bag of visual words," in *Proceedings of the ACM International Conference on Image and Video Retrieval*, ser. CIVR '09. New York, NY, USA: ACM, 2009, pp. 35:1–35:7. [Online]. Available: http://doi.acm.org/10.1145/1646396.1646439

[73] B. Ionescu, C. Vertan, P. Lambert, and A. Benoit, "A color-action perceptual approach to the classification of animated movies," in *Proceedings of the 1st ACM International Conference on Multimedia Retrieval*, ser. ICMR '11. New York, NY, USA: ACM, 2011, pp. 10:1–10:8. [Online]. Available: http://doi.acm.org/10.1145/1991996.1992006

[74] R. Agrawal and R. Srikant, "Privacy-preserving data mining," in *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, ser. SIGMOD '00. New York, NY, USA: ACM, 2000, pp. 439–450. [Online]. Available: http://doi.acm.org/10.1145/342009.335438

[75] A. Evfimievski, R. Srikant, R. Agrawal, and J. Gehrke, "Privacy preserving mining of association rules," 2002, pp. 217–228.

[76] A. Evfimievski and T. Grandison, "Privacy perserving data mining," in *Encyclopedia of Database Technologies and Applications*, V. E. Ferraggine, J. H. Doorn, and L. C. Rivero, Eds., 2007.