

Number 706



**UNIVERSITY OF  
CAMBRIDGE**

Computer Laboratory

## Covert channel vulnerabilities in anonymity systems

Steven J. Murdoch

December 2007

15 JJ Thomson Avenue  
Cambridge CB3 0FD  
United Kingdom  
phone +44 1223 763500  
<http://www.cl.cam.ac.uk/>

© 2007 Steven J. Murdoch

This technical report is based on a dissertation submitted August 2007 by the author for the degree of Doctor of Philosophy to the University of Cambridge, Girton College.

Technical reports published by the University of Cambridge Computer Laboratory are freely available via the Internet:

*<http://www.cl.cam.ac.uk/techreports/>*

ISSN 1476-2986

# Covert channel vulnerabilities in anonymity systems

Steven J. Murdoch

## Summary

The spread of wide-scale Internet surveillance has spurred interest in anonymity systems that protect users' privacy by restricting unauthorised access to their identity. This requirement can be considered as a flow control policy in the well established field of *multilevel secure* systems. I apply previous research on *covert channels* (unintended means to communicate in violation of a security policy) to analyse several anonymity systems in an innovative way.

One application for anonymity systems is to prevent collusion in competitions. I show how covert channels may be exploited to violate these protections and construct defences against such attacks, drawing from previous covert channel research and collusion-resistant voting systems.

In the military context, for which multilevel secure systems were designed, covert channels are increasingly eliminated by physical separation of interconnected single-role computers. Prior work on the remaining network covert channels has been solely based on protocol specifications. I examine some protocol implementations and show how the use of several covert channels can be detected and how channels can be modified to resist detection.

I show how side channels (unintended information leakage) in anonymity networks may reveal the behaviour of users. While drawing on previous research on traffic analysis and covert channels, I avoid the traditional assumption of an omnipotent adversary. Rather, these attacks are feasible for an attacker with limited access to the network. The effectiveness of these techniques is demonstrated by experiments on a deployed anonymity network, Tor.

Finally, I introduce novel covert and side channels which exploit thermal effects. Changes in temperature can be remotely induced through CPU load and measured by their effects on crystal clock skew. Experiments show this to be an effective attack against Tor. This side channel may also be usable for geolocation and, as a covert channel, can cross supposedly infallible air-gap security boundaries.

This thesis demonstrates how theoretical models and generic methodologies relating to covert channels may be applied to find practical solutions to problems in real-world anonymity systems. These findings confirm the existing hypothesis that covert channel analysis, vulnerabilities and defences developed for multilevel secure systems apply equally well to anonymity systems.

## Acknowledgements

I would like to thank my supervisor, Markus Kuhn, for his advice and support throughout my time at Cambridge and also Ross Anderson, for fostering an environment which allowed me to explore a range of topics. The technical staff at the Computer Laboratory also have been of great assistance.

I was supported by a Carnegie Scholarship, from The Carnegie Trust for the Universities of Scotland.

I thank the members of the Security Group, both past and present, for our interesting and fruitful discussions and particularly Richard Clayton, Markus Kuhn and Robert Watson, for proofreading and suggesting improvements to this thesis. I am also grateful for the comments from my examiners, Virgil D. Gligor and Steven Hand. However, the responsibility for all the remaining errors and omissions, of course, rests with me.

I would also like to thank my co-authors, especially those of the papers on which this thesis is based: George Danezis, Stephen Lewis and Piotr Zieliński.

I am indebted to my family, and especially my parents, for their support and advice throughout my education. I also thank my friends for making my time here in Cambridge enjoyable as well as intellectually stimulating.

I would finally like to thank the anonymity community: the researchers and developers for creating a strong foundation to build upon, the service operators who have allowed me to evaluate and refine my ideas, and the users, whose need for privacy has motivated this work.

# Contents

<b>1</b>	<b>Introduction</b>	<b>12</b>
1.1	Covert channels . . . . .	12
1.1.1	Security policies and threat model . . . . .	13
1.1.2	Covert channel terminology . . . . .	15
1.1.3	Identification of covert channels . . . . .	17
1.2	Anonymity . . . . .	18
1.2.1	Anonymity terminology . . . . .	19
1.3	Structure of the thesis . . . . .	20
1.4	Anonymity and links to covert channels . . . . .	22
<b>2</b>	<b>Local covert channels in games</b>	<b>25</b>
2.1	Background . . . . .	25
2.2	Competition structures . . . . .	27
2.2.1	League tournaments . . . . .	28
2.2.2	Knockout tournaments . . . . .	29
2.3	Identification mechanisms . . . . .	30
2.3.1	Timing . . . . .	31
2.3.2	Choice of equivalent moves . . . . .	32
2.3.3	Analysis of identification mechanisms . . . . .	32
2.3.4	Identification key . . . . .	33
2.4	Real world example . . . . .	34
2.4.1	Rules of the game . . . . .	34
2.4.2	Collusion strategy chosen . . . . .	35
2.4.3	Game strategy . . . . .	36
2.4.4	Implementation . . . . .	37
2.4.5	Optimisation . . . . .	38
2.4.6	Effects of poor players . . . . .	38
2.4.7	Results . . . . .	39

2.5	Defeating collusion . . . . .	41
2.5.1	Covert channel prevention . . . . .	41
2.5.2	Detecting collusion . . . . .	41
2.5.3	Collusion resistant competitions . . . . .	42
2.6	Conclusion . . . . .	43
<b>3</b>	<b>Embedding covert channels into TCP/IP</b>	<b>44</b>
3.1	Introduction . . . . .	45
3.2	Threat model . . . . .	47
3.3	TCP/IP-based steganography . . . . .	48
3.3.1	Type of service . . . . .	49
3.3.2	IP identification . . . . .	50
3.3.3	IP flags . . . . .	50
3.3.4	IP fragment offset . . . . .	51
3.3.5	IP options . . . . .	51
3.3.6	TCP sequence number . . . . .	51
3.3.7	TCP timestamp . . . . .	52
3.3.8	Packet order . . . . .	53
3.3.9	Packet timing . . . . .	53
3.4	IP ID and TCP ISN implementations . . . . .	53
3.4.1	Linux . . . . .	54
3.4.2	OpenBSD . . . . .	56
3.5	Detection of TCP/IP steganography . . . . .	57
3.5.1	IP ID characteristics . . . . .	57
3.5.2	TCP ISN characteristics . . . . .	58
3.5.3	Explicit steganography detection . . . . .	59
3.5.4	Accuracy . . . . .	59
3.5.5	Results . . . . .	61
3.6	Detection-resistant TCP steganography . . . . .	63
3.6.1	Linux . . . . .	63
3.6.2	OpenBSD . . . . .	64
3.7	Conclusion . . . . .	64
<b>4</b>	<b>Low-cost traffic analysis of Tor</b>	<b>67</b>
4.1	Introduction . . . . .	67
4.2	Understanding Tor . . . . .	69
4.2.1	Architecture . . . . .	69
4.2.2	Threat model . . . . .	72

4.3	Attacking Tor . . . . .	73
4.3.1	Traditional traffic analysis . . . . .	73
4.3.2	Traffic analysis of Tor . . . . .	75
4.3.3	Traffic-analysis methodology . . . . .	77
4.4	Experimental setup and results . . . . .	78
4.4.1	Results . . . . .	80
4.5	Discussion . . . . .	82
4.5.1	Linkability attack . . . . .	84
4.5.2	Variants of the attack . . . . .	85
4.5.3	Attack costs . . . . .	87
4.5.4	Understanding the traffic artifacts . . . . .	88
4.6	Conclusion . . . . .	91
<b>5</b>	<b>Temperature-based channels</b>	<b>93</b>
5.1	Introduction . . . . .	93
5.2	Hidden services . . . . .	95
5.2.1	Threat model . . . . .	96
5.2.2	Existing attacks . . . . .	97
5.3	Clock skew and temperature . . . . .	98
5.3.1	Background and definitions . . . . .	99
5.3.2	Impact of temperature . . . . .	102
5.4	Attacking Tor . . . . .	105
5.4.1	Results . . . . .	107
5.4.2	Discussion . . . . .	109
5.5	Extensions and future work . . . . .	110
5.5.1	Classical covert channels . . . . .	110
5.5.2	Cross-computer communication . . . . .	112
5.5.3	Geolocation . . . . .	113
5.5.4	Noise sources and mitigation . . . . .	114
5.6	Conclusion . . . . .	117
<b>6</b>	<b>Conclusions</b>	<b>118</b>
6.1	Future research directions . . . . .	121

## Published work

In the course of my studies, I have published the following papers, book chapter and poster. Some discuss topics covered in this thesis but several others are from separate threads of research. My work has been published at three of the top four information security conferences (IEEE Symposium on Security and Privacy, ACM Conference on Computer and Communications Security, USENIX Security Symposium), which comprise the major outlets for research in this field. The paper “Low-cost traffic analysis of Tor” was one of the nominees for the 2006 PET Award for outstanding research in privacy enhancing technologies, and won the 2006 University of Cambridge Computer Laboratory award for most notable publication. The paper “Keep your enemies close: Distance bounding against smartcard relay attacks” was awarded the prize for best student paper at the 2007 USENIX Security Symposium.

### OPERATING SYSTEM AND NETWORK SECURITY

Tuomas Aura, Michael Roe, and Steven J. Murdoch. Securing network location awareness with authenticated DHCP. In *3rd International Conference on Security and Privacy in Communication Networks (SecureComm)*, Nice, France, September 2007. IEEE.

Markus G. Kuhn, Steven J. Murdoch, and Piotr Zieliński. Compounds: A next-generation hierarchical data model. Microsoft Research Academic Days, Dublin, Ireland, April 2004. Poster presentation.

### FINANCIAL SECURITY

Saar Drimer and Steven J. Murdoch. Keep your enemies close: Distance bounding against smartcard relay attacks. In *Proceedings of the 16th USENIX Security Symposium*, Boston, MA, US, August 2007. USENIX.

Ben Adida, Mike Bond, Jolyon Clulow, Amerson Lin, Steven J. Murdoch, Ross Anderson, and Ron Rivest. Phish and chips: Traditional and new recipes for attacking EMV. In *Security Protocols, 14th International Workshop, Cambridge, England*, LNCS, Cambridge, UK, March 2006. Springer. (to appear).

Ross Anderson, Mike Bond, and Steven J. Murdoch. Chip and spin. *Computer Security Journal*, 22(2):1–6, 2006.

Mike Bond, Daniel Cvrček, and Steven J. Murdoch. Unwrapping the Chrysalis. Technical Report UCAM-CL-TR-592, University of Cambridge, Computer Laboratory, June 2004.

#### CENSORSHIP RESISTANCE, ANONYMITY AND INFORMATION HIDING

Steven J. Murdoch and Ross Anderson. *Tools and Technology of Internet Filtering*, a chapter in Ronald J. Deibert, John G. Palfrey, Rafal Rohozinski and Jonathan Zittrain editors, *Access Denied: The Practice and Policy of Global Internet Filtering*. Information Revolution and Global Politics. MIT Press, November 2007. (to appear).

Steven J. Murdoch and Piotr Zieliński. Sampled traffic analysis by Internet-exchange-level adversaries. In Nikita Borisov and Philippe Golle, editors, *Privacy Enhancing Technologies (PET)*, volume 4776 of *LNCS*, pages 167–183, Ottawa, Canada, June 2007. Springer.

Steven J. Murdoch. Hot or not: Revealing hidden services by their clock skew. In *CCS '06: Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 27–36, Alexandria, VA, US, October 2006.

Richard Clayton, Steven J. Murdoch, and Robert N. M. Watson. Ignoring the great firewall of China. In George Danezis and Philippe Golle, editors, *Privacy Enhancing Technologies (PET)*, volume 4258 of *LNCS*, pages 20–35, Cambridge, UK, June 2006. Springer.

Steven J. Murdoch and Stephen Lewis. Embedding covert channels into TCP/IP. In Mauro Barni, Jordi Herrera-Joancomartí, Stefan Katzenbeisser, and Fernando Pérez-González, editors, *Information Hiding Workshop*, volume 3727 of *LNCS*, pages 247–261, Barcelona, Catalonia (Spain), June 2005. Springer.

Andrei Serjantov and Steven J. Murdoch. Message splitting against the partial adversary. In George Danezis and David Martin, editors, *Privacy Enhancing Technologies (PET)*, volume 3856 of *LNCS*, pages 26–39, Cavtat, Croatia, May/June 2005. Springer.

Steven J. Murdoch and George Danezis. Low-cost traffic analysis of Tor. In *IEEE Symposium on Security and Privacy*, pages 183–195, Oakland, CA, US, May 2005. IEEE Computer Society.

Steven J. Murdoch and Piotr Zieliński. Covert channels for collusion in online computer games. In Jessica Fridrich, editor, *Information Hiding Workshop*, volume 3200 of *LNCS*, pages 355–369, Toronto, Canada, May 2004. Springer.

## Selected talks and seminars

I have also attempted to disseminate my research through presenting talks at venues without formal proceedings. A selection of these talks is included below. In the case of the Chaos Communication Congress and the Inter-Disciplinary China Studies Forum conference, talks are put through a competitive peer-review process before acceptance. The talk “Hot or not: Fingerprinting hosts through clock skew” was awarded 2nd prize for best talk at EuroBSDCon 2007.

Steven J. Murdoch. Hot or not: Fingerprinting hosts through clock skew. *EuroBSDCon*, Copenhagen, Denmark, September 2007. (invited talk).

Steven J. Murdoch. EMV flaws and fixes: vulnerabilities in smart card payment systems. *COSIC Seminar*, Belgium, June 2007. K.U. Leuven.

Steven J. Murdoch. Internet censorship in China. *Inter-Disciplinary China Studies Forum Annual Conference*, Cambridge, UK, April 2007.

Steven J. Murdoch. Detecting temperature through clock skew. *21st Chaos Communication Congress*, Berlin, Germany, December 2006. Chaos Computer Club e.V.

Steven J. Murdoch. Censorship resistant technologies. *Horizon seminar: Risk, Threat & Detection*, Cambridge, UK, December 2006. Research Services Division, University of Cambridge.

Steven J. Murdoch. Out of character: Are the Chinese creating a second Internet? *Inter-Disciplinary China Studies Forum workshop: China in the UK*, Cambridge, UK, June 2006.

Steven J. Murdoch and Stephen Lewis. Covert channels in TCP/IP: attack and defence. *22nd Chaos Communication Congress*, Berlin, Germany, December 2005. Chaos Computer Club e.V.

Steven J. Murdoch and Maximillian Dornseif. Hidden data in Internet published documents. *21st Chaos Communication Congress*, Berlin, Germany, December 2004. Chaos Computer Club e.V.

Steven J. Murdoch and Ben Laurie. The convergence of anti-counterfeiting and computer security. *21st Chaos Communication Congress*, Berlin, Germany, December 2004. Chaos Computer Club e.V.

## Work done in collaboration

Chapter 2 is based on the paper “Covert channels for collusion in online computer games” [110]. The Connect-4 algorithm was designed and implemented by Piotr Zieliński, and the strategy collaboratively developed between Piotr Zieliński, Stephen Lewis and myself.

Chapter 3 is based on “Embedding covert channels into TCP/IP” [109]. In this, Stephen Lewis worked on both the analysis of OpenBSD and the design of the encoding mechanism for OpenBSD initial sequence numbers.

Chapter 4 is based on the paper “Low-cost traffic analysis of Tor” [108]. The initial attack idea was proposed by George Danezis, however the refinement and implementation were mine.

# Chapter 1

## Introduction

Covert channels transfer information in violation of a security policy. Multilevel secure systems (MLS) aim to protect against unauthorised flows of information, through mandatory access control, backed by covert channel prevention. Similarly, anonymity systems aim to protect a user's actions and identity from observation by an adversary; an implicit information flow control policy. This thesis will discuss the congruence between covert channel prevention and securing anonymity systems, showing how techniques for attacking and protecting MLS can be used in the analysis and improvement of anonymity systems.

### 1.1 Covert channels

The term *covert channel* was introduced by Butler Lampson [93], although with a slightly different definition to later common usage. He described the generic problem of preventing a program from leaking information it processes but, spurred on by government imposed military standards, most following research dealt with the problem of multilevel secure (MLS) systems. In these, information is categorised at confidentiality levels and the system enforces a policy that only individuals rated at that level or higher may read the item. Another model is multilateral security, where information is placed into compartments and may only flow between them in approved ways. Both of these are examples of mandatory access control systems, where the system administrator sets the policy, in contrast to discretionary access control systems, where the owner of a data item is permitted to choose how access to that item is restricted.

Although MLS systems are not the focus of this thesis, so much of the covert-channel research is concerned with them and their threat model, that to properly understand this thesis in context, it is necessary to summarise some aspects of MLS. However, this section should not be seen as a complete introduction to covert channels in MLS systems; for this I refer the reader to McHugh [98] and its references.

### 1.1.1 SECURITY POLICIES AND THREAT MODEL

By definition, communication through covert channels violates a security policy, so the systems of interest must have an information flow policy, which defines permitted sources and destinations for particular classes of data. Moreover, the policy must incorporate mandatory access control.

Covert channels can exist in all mandatory access control systems which restrict information flow, so are relevant to both confidentiality and integrity policies, as described in the US Department of Defense (DoD) requirements for covert channel analysis [65] (the “Light-Pink Book”). In systems which aim to preserve confidentiality, covert channels can leak information to unauthorised individuals, while in the case of a mandatory integrity policy, covert channels can be used to introduce unauthorised changes to protected objects. However, the remainder of discussion will concentrate on confidentiality policies.

The Bell-LaPadula model (BLP) [16] is a formal description of the MLS policy and is illustrated in Figure 1.1. It was motivated by government and military requirements, where objects are labelled with security levels such as (from low to high), Unclassified, Confidential, Secret and Top Secret. The BLP model ensures that information may not flow “down” security levels, except under approved circumstances. It can be summarised by two properties:

**No read up:** A process at a given security level may not receive information from a higher level than its own. For example, if a file is classed at Secret, an application at Confidential could not read the contents.

**No write down:** A process at a given security level may not send information to a lower level than its own. In the same example, an application at Top Secret could not modify the Secret file.

The first property protects against users who improperly try to read information at a security level higher than their clearance. So a malicious user without clearance (Malory) can only read information produced by applications at the Unclassified level and will be prevented from reading restricted

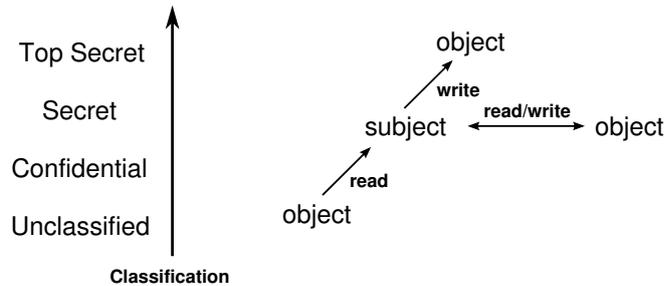


Figure 1.1: DoD classifications of information and legal flows in the Bell-LaPadula model. Based on a diagram in Pfleeger and Pfleeger [122, p256]

information. If only malicious users are to be considered, then the second property might seem unimportant – someone cleared to read Top Secret information could leak it by simply writing it down or remembering it. However, this property is of critical importance when applications can be compromised, when users might unknowingly introduce malicious applications, or simply when users may make mistakes about classification.

Malory might cause a user, Alice, cleared to Top Secret, to run a malicious application (a *Trojan horse*). Alternatively Malory could insert the Trojan by compromising an application running at Top Secret. The Trojan can read Top Secret information but due to the “No write down” property, cannot output this information to any application that Malory can read from. It is in this situation that a covert channel is a threat. If the system has some way for the Trojan to signal to the Unclassified application then Malory can receive the leaked information.

An example of a covert channel from the Light-Pink Book assumes a MLS system based on Unix. In these, directories inherit the classification of their parent, but the owner may upgrade the classification, provided the directory is empty, creating an *upgraded directory*, as described by Whitmore *et al.* [156]. Normal Unix semantics apply, so if Malory owns a directory, he can remove empty subdirectories but not ones that contain any files. If there is an upgraded subdirectory in Malory’s directory, labelled Top Secret, then Alice can create files in it, but Malory cannot read its contents. However, Malory can infer whether it contains any files by attempting to delete it. The Trojan can signal a “1” by creating a file, and Malory receives it by observing the directory cannot be deleted; similarly the Trojan can signal a “0” by deleting all files.

Covert channels are also relevant to integrity models, such as Clark Wilson [33] and Biba [21]. The latter model is the dual of BLP, protecting integrity

rather than secrecy. Here, applications at a particular level are prevented from writing up or reading down. As an example, if a Trojan is inserted at a high level, it cannot receive input from any application that Malory can write to. However, if the Trojan can use a covert channel to receive instructions from Malory then the security policy is violated. While integrity is clearly important, for anonymity purposes the confidentiality of identity information is the primary asset. For this reason, further discussion of covert channels will deal with secrecy and, for simplicity's sake, examples will assume the BLP model.

### 1.1.2 COVERT CHANNEL TERMINOLOGY

While intuitively the definition of the term “covert channel” is clear, in the literature the precise definition varies. For clarity, in this thesis, I will use the following definitions, based common usage and the description by David Wagner [154]. As the field of covert channels overlaps with other areas in information hiding, I will also define some associated terms.

**covert channel:** A means of communicating on a computer system, where both the sender and receiver collude to leak information, over a channel not intended for the communication taking place, in violation of a mandatory access control security policy.

**side channel:** A communication channel which violates a security property, but where the sender unintentionally leaks information and only the receiver wishes the communication to succeed.

**steganographic channel:** A means of communication on an open channel, where sender and receiver collude to prevent an observer being able to reliably detect whether communication is happening.

**subliminal channel:** A covert channel in a cryptographic algorithm, typically provably undetectable.

The above categories are linked. An subliminal channel is special type of steganographic channel. A steganographic channel can be used to construct a covert channel, but not all covert channels are steganographic and there are other uses of steganography, for example watermarking.

Note that the definition of covert channel both requires that the information is transferred using a channel not intended for this use and that this violates a security policy. This is largely consistent with common usage, and similar to the Common Criteria definition [35]:

An enforced, illicit signalling channel that allows a user to surreptitiously contravene the multi-level separation policy and unobservability requirements of the [target of evaluation].

However, there are alternate definitions. For example, when coining the term, Lampson [93] excluded the use of existing communication mechanisms in a way which was not intended, and McHugh [98] omits the connection to a security policy:

A covert channel is a mechanism that can be used to transfer information from one user of a system to another using means not intended for this purpose by the system developers.

This allows a channel to be covert despite being consistent with the security policy, so he uses the term “innocuous covert channel” to define such channels, and “harmful covert channel” for those which do violate the policy.

In the DoD Trusted Computer System Evaluation Criteria [25], commonly known as the “Orange Book”, the definition used is:

A covert channel is any communication channel that can be exploited by a process to transfer information in a manner that violates the system’s security policy.

This clearly states the policy violation constraint, but does not consider whether the communication channel was envisaged as a communication channel by the system designer.

The Light-Pink Book [65] uses a more formal variant of the definition used in the Orange Book, as introduced by Tsai *et al.* [149]:

Given a nondiscretionary (e.g., mandatory) security policy model  $M$  and its interpretation  $I(M)$  in an operating system, any potential communication between two subjects  $I(S_h)$  and  $I(S_i)$  of  $I(M)$  is covert if and only if any communication between the corresponding subjects  $S_h$  and  $S_i$  of the model  $M$  is illegal in  $M$ .

This definition is designed to be consistent with the Orange Book and so the same comments apply. In particular, it explicitly does not differentiate between unintentional communication channels and flaws in the implementation of security policies. As pointed out by both the Light-Pink Book and

McHugh, this is problematic for TCSEC certification levels B1 and below, which do not require covert channel management, as flaws could be classed as covert channels and thus ignored.

The Light-Pink Book therefore further divides covert channels into “Fundamental” channels that would exist in any interpretation of the security model, “Specific” channels that appear only in some interpretations, and “Unjustifiable” channels that appear only in some interpretations but no justifiable reason for their existence can be demonstrated. The guidelines recommend that while systems aiming for certification at B1 do not need to manage covert channels, they should eliminate unjustifiable channels and manage specific channels, but need not deal with fundamental channels.

### 1.1.3 IDENTIFICATION OF COVERT CHANNELS

In order for a system to be granted certain forms of certification, its designers must justify that there has been a systematic approach to identifying covert channels and mitigating them if necessary. This goal has motivated the study of techniques to find potential covert channels and estimate their bandwidth. The Light-Pink Book [65] contains a summary of the most important techniques, for use in TCSEC certification. Many of the approaches are specific to multi-level secure operating systems, but the more generic ones are appropriate for the analysis of anonymity systems.

One very general technique is Kemmerer’s shared resource matrix methodology [87]. It consists of identifying all objects which are accessible to more than one process. Then, a comprehensive list of operations that may be performed on the system is created. For each pair (object, operation), it is established whether any attribute of the object may be read or modified by that operation. Finally, the result is represented as a matrix of object attributes versus operations. Any object attribute that may be both modified and read by system operations is a candidate for a covert channel. Some of the candidates could be of no concern, because either there exists no means to synchronise the attribute modification, or where processes which may communicate are in the same protection domain. The remainder of the candidates are a potential risk and merit further investigation.

A similar approach may be applied to anonymity systems, by considering them as a group of processes and objects. If a covert channel exists that permits two processes,  $A$  and  $B$ , to communicate, where  $A$  knows the identity of a system participant and  $B$  knows the pseudonym, then a potential

covert-channel risk exists. Should an attacker be able to cause  $A$  to modify an attribute of a shared resource, in such a way that it can be read by  $B$ , then the anonymity of a user may be compromised. Chapter 4 shows an example of where the shared attribute is the network and CPU resources of a host, and in Chapter 5 it is the temperature.

## 1.2 Anonymity

Anonymity, the ability to communicate without revealing your identity, has been widely considered as a pre-requisite for the democratic process, as demonstrated by the US Supreme Court's protection of anonymous speech [152]:

Accordingly, an author's decision to remain anonymous, like other decisions concerning omissions or additions to the content of a publication, is an aspect of the freedom of speech protected by the First Amendment.

In addition to political speech, anonymity has protected whistle-blowers, reporting market abuse, from retribution by their employer or government. Human rights defenders can protect the identity of their contacts working in repressive regimes. Many more examples are possible.

As more communication has moved to the Internet, the need for anonymity has moved from the physical to the virtual world. The Internet opens up the opportunity for wider distribution of information than previous mechanisms, such as anonymous leafleting or through classified adverts in newspapers. The social, political and corporate situation across the world demonstrates that there is still a very real need for anonymity systems.

Research into the anonymity of electronic mail began with Chaum's paper on the *mix* [31], which proposed a service to anonymise messages by hiding correspondences between sender and receiver. However, initial implementations, such as the popular anon.penet.fi "Type-0" remailer [75] did not adopt any of cryptographic techniques that Chaum proposed. Later generations, such as the "Type-1" Cypherpunk remailer [118] and "Type-2" Mixmaster systems [116], were closer to Chaum's original design.

Subsequent research developed new attacks against such systems, corresponding defences, and techniques to improve usability. To validate proposed improvements there has also been considerable effort expended on clarifying terminology and quantifying concepts, as described in the following section.

### 1.2.1 ANONYMITY TERMINOLOGY

As defined by Pfitzmann and Hansen [119], “Anonymity is the state of being not identifiable within a set of subjects, the anonymity set”, based on information available to a defined attacker. The attacker is permitted to know an item’s current state, for example in anonymous messaging whether it is sending or receiving, but not the relationship between items or between an item and an identity. For anonymous messaging, “sender anonymity” means that the attacker does not know which sender sent a particular message; similarly for “receiver anonymity”. A different property is “relationship anonymity” where an attacker cannot link senders to recipients. Relationship anonymity is weaker than sender anonymity, because in the former the attacker may know who sent a particular message, but not the message’s recipient. Similarly, relationship anonymity is also weaker than receiver anonymity.

A stronger concept than anonymity is “unobservability”: the state of any item being indistinguishable from any other item in the unobservability set. In a similar way to anonymity, “sender unobservability” means that an attacker does not know whether a particular participant sent a message and similarly for “receiver unobservability”. While unobservability implies anonymity, achieving the former in the general case requires expensive cover traffic.

Unlike encryption, anonymity cannot be achieved in isolation. A subject must hide within a group of other subjects – as Reiter and Rubin say: “anonymity loves company” [132]. For anonymous messaging, the most common mechanism is the mix [31]. Conceptually a mix receives messages, re-orders them and sends them out again, so providing relationship anonymity. Encryption is used to hide the correspondence between the content of incoming and outgoing messages and a delay is added to hide timing characteristics. Similar systems can be used for real-time traffic, by omitting the delay, substantially reducing their resistance to attack. Chains of mixes are used to provide some resistance to a partially corrupt network. There has been a great deal of research on this area, and for further details I refer the reader to Serjantov [139] and Danezis [39]. More information on the specific implementations of anonymity systems discussed in this thesis are provided in their respective chapters.

For anonymous messaging, “sender anonymity set” and “receiver anonymity set”, are the potential senders and the recipients of a message, respectively. The simplest way to quantify anonymity is to calculate the cardinality of the anonymity set, or by taking its logarithm, translate this to bits, as

used by Berthold *et al.* [20]. However, this does not take into account that some members of the set may be more likely to be the actual sender/receiver than others. Crowds [131], the anonymous web browsing system, proposes a 6-point scale, measuring how non-uniform is the probability of each anonymity set member being the true sender or receiver. It ranges from “absolute privacy” (unobservability), “beyond suspicion” (all members of the set have equal probability), “probable innocence” (the subject’s probability is at most 50%), “possible innocence” (the anonymity set has more than 1 member), “exposed” (the anonymity set has one member) to “provably exposed” (the attacker can prove to a third party that a subject is the only member of the anonymity set).

Crowds’ approach does not take into account the size of the anonymity set, but Serjantov and Danezis [140] proposed combining both the size and probability distribution by calculating the entropy. This expresses the effective size of the anonymity set in bits, whereas the approach taken by Díaz *et al.* [44] is to divide this value by the actual size of the anonymity set, resulting in a value between 0 and 1, representing how skewed the probability distribution is from uniform.

### 1.3 Structure of the thesis

Attacks on unobservability occur when information about an item’s state is leaked to the attacker. Attacks on anonymity occur when a subject’s identity or relationships are leaked. These information flows are contrary to the security policy of anonymity systems so they may be considered covert channels or side channels, depending on whether the sender of the information colludes with the attacker or not.

Many existing attacks on mix networks can be seen as side channels. For example, intersection attacks [20, 39, 88] reduce the relationship anonymity of communication partners who are using a mix network to send multiple messages. Each time a subject of interest sends a message, there is a finite recipient set, which will be different for each message sent. Assuming the sender sends to only one recipient, the effective anonymity set will be the intersection of the anonymity sets for each message. Over time, this will reduce to one, completely breaking the anonymity of the sender. Similar attacks exist for receivers.

This attack works because sending a message does leak information about the recipient, although the mixing introduces noise. This noise is not totally

dependent on the sent messages so can be eliminated by considering repeated interactions. This is analogous to de-noising signals through periodic averaging of data from the TEMPEST analysis of computer displays [91], another type of side channel attack.

Anonymity systems need many users for any of them to achieve anonymity. This motivates the establishment of publicly accessible anonymity networks. By giving access to potentially malicious parties, the possibility of covert channels exists. While allowing the execution of arbitrary code on anonymity systems, even sandboxed, is unusual (Chapter 2 describes an exception), the nodes do process untrustworthy data and so can have their behaviour altered.

The goal of this thesis is to show that by viewing an anonymity system as enforcing information flow restrictions, previous work on covert channels can be applied so as to develop new attacks and defend against them.

When military systems primarily used expensive mainframes, it was common for data at several classification levels to be processed on the same computer. The security policy may thus prohibit local inter-process communication. Chapter 2 describes such a case, where a multilateral security policy prevents collusion by enforcing anonymity. Opportunities for using covert channels to break this policy are discussed and a practical example is given.

Chapter 3 describes how covert channels can be used to leak information over a network, where an information flow policy is being enforced through traffic monitoring or protocol cleaning. Several covert-channel techniques have been proposed to break such policies by signalling in TCP/IP headers, but many assume only casual observation. This chapter describes how such schemes can be detected and proposes enhancements to prevent detection.

Chapter 4 shows how side channels can be used to attack a distributed anonymity system – Tor [50]. In order to maintain unlinkability, Tor must prevent one end of a communication channel from discovering the identity of the other end, or the path that the data is taking. The attacker cannot run arbitrary code on Tor nodes but, by sending legitimate data, can induce and measure traffic patterns. As two connections through a Tor node share many resources, these patterns can be used to leak information between them, breaking the isolation property and allowing anonymity to be compromised.

These techniques are extended in Chapter 5, which shows that covert channels and side channels can extend into the hardware, not just software. In Tor, the attacker can send arbitrary data to a hidden service, but must be prevented from learning its identity. The load-modulation attacks of Chapter 4 could be

resisted through fixed response times, but during periods of increased load, the hardware will still warm up. Chapter 5 shows how this increased temperature has an effect on the clock crystal, which can be reliably measured remotely. Using this technique, an attacker can create a side channel between two connections, one using the anonymous channel and the other connecting directly. An attacker can thus discover the identity of a target.

## 1.4 Anonymity and links to covert channels

There is a small literature on attacks against anonymity systems that can be seen as exploiting covert channels. However, in general, previous work has not made this connection. In particular, traffic analysis, the study of extracting information from logs of communication patterns in the absence of knowledge about the content, has been a fruitful source of attacks against anonymity.

The field of traffic analysis originated from the military analysing radio traffic to infer troop location, strength or role, as discussed by Herman [76]. Information extracted from such traffic analysis is useful in its own right, but it is especially suitable for automated analysis and selecting targets for more laborious content analysis. Where the content is encrypted, gaining access to the plaintext may be expensive or in some cases impossible, so traffic analysis may be the only viable method for extracting intelligence.

Patterns of communication on the Internet, such as the duration, volume and endpoints of connections, may also leak information about the nature of the content, sender or receiver. Such attacks are especially relevant when the traffic is encrypted, as while content is obscured to foil eavesdropping, the traffic patterns are preserved. The unintentional leakage of data volume and timing is sufficient to violate a security policy. Traffic analysis thus may be viewed as a special case of side-channel analysis, a concept closely linked to the covert channels discussed in this thesis.

Intersection attacks were shown earlier to be a case of a side channel which eventually links senders to receivers. Other traffic-analysis attacks can extract content from encrypted data streams. For example, Song *et al.* [145] showed that the timing of SSH keystroke patterns, coupled with knowledge of the keyboard layout, is adequate to extract the keys pressed. Another example is by Sun *et al.* [146], who showed that the web pages browsed over an encrypted tunnel can be inferred from pages sizes, which the encryption does not disguise. The potency of the attack may be further improved by also taking into account

links between pages, as demonstrated by Danezis [38]. Even if the boundaries between individual objects downloaded are hidden by a VPN, Bissias *et al.* [22] showed that this is not adequate to defeat traffic analysis.

The distinguishing factor between side channels and covert channels is that the latter requires collusion between sender and receiver. While traffic analysis of intercepted radio transmissions is passive, on the Internet, active traffic analysis is also possible. Pfitzmann and Pfitzmann [121] showed how streams may be *tagged* by corrupting anonymised messages and observing the resulting effects elsewhere. In the closely related field of detecting *stepping stones* (computers relaying control commands to compromised hosts), Zhang and Paxson [161] showed how streams may be passively correlated. Wang and Reevesbut [155] extended this attack by embedding a watermark, improving the detection rate. The work in Chapter 4 may be viewed as a variant of stepping-stone detection using watermarks, with one extra level of indirection.

Another active traffic-analysis attack is proposed by Juels *et al.* [54], who introduce *cache cookies*, which are files placed on users' machines through normal web browser caching behaviour. Their existence can subsequently be queried in order to link a user's pseudonym with their real identity. Less maliciously, Felten and Schneider [84] show that these can act as a replacement to normal cookies, providing an additional factor in authentication protocols.

The number of packets sent by a host can be derived from observing increments in the IP ID field, for hosts which have a global counter. This fact can be exploited in an active attack, known as *idle scanning*, to covertly port scan a victim host, and is implemented in Nmap [60]. The attacker sends TCP SYN packets to the victim, but spoofs the source address to be that of a *cut-out*. If the port is open, the victim responds by sending a SYN-ACK to the cut-out, which responds with a RST packet. If the victim sends a RST or no packet at all, signifying the port being closed, the cut-out sends no response. The attacker can distinguish these cases by probing the cut-out and observing the IP ID counter. At no point does the victim being scanned see the IP address of the attacker, and so the attack hides the source of the scan.

One defence against being used as the cut-out in such a scheme is to randomise the IP ID, or at least use a different counter for each routing table entry. This is the approach adopted by more modern operating systems, which prevents the attack but also creates a covert channel. Since the IP ID is now at least partially random, a firewall cannot predict the expected value, so a host can undetectably embed encrypted information. This topic and related issues

are explored in more detail in Chapter 3, showing that defences against certain types of traffic analysis may open additional vulnerabilities.

The link between anonymity, traffic analysis and covert channels was most notably discussed in a series of papers by Moskowitz and Newman *et al.* [104, 105, 114]. These propose measuring the lack of anonymity provided by a mix as the amount of information that can be leaked from Alice, who is sending messages into the mix, to Eve who can only view the output.

Moskowitz and Newman's model can be used directly for analysing a system which explicitly implements a mandatory information flow control policy. In one scenario they present, Alice is permitted to send messages but only through a *mix firewall* shared with other users. Eve can only count messages emitted by the mix, or in another scenario can also see their recipient. A perfect anonymity system is defined as one where Alice cannot leak any information to Eve over this covert channel.

The metric for anonymity of a mix firewall is different from those used in the well studied anonymous remailer model, as in the latter Alice and Eve are adversaries, not co-operating. However, covert channels can still be used to analyse this case. If Alice cannot leak any information to Eve then Alice cannot give away her identity. Thus the capacity of the covert channel can be used as a measure of its lack of anonymity.

While closely related to the Moskowitz and Newman papers, the emphasis of this thesis is different. In the threat model that remailers and other deployed anonymity systems were developed under, the existence of the covert channels described by Moskowitz and Newman does not imply a valid attack, only that that optimal anonymity is not being provided. The goal of this thesis is to show that even using more realistic threat models, covert channels do exist and can be used as attacks.

# Chapter 2

## Local covert channels in games

When covert channels were first discovered, one concern was that they could be used to transfer information between processes in a computer processing data at multiple classification levels and enforcing a MLS policy. In this scenario, an attacker was assumed to be capable of inserting arbitrary code, in the form of a Trojan, at two levels and their goal was to transfer data from high to low security levels. The trusted computing base (TCB) of the system is relied upon to prevent such unauthorised information flows.

This chapter discusses a similar situation: the two processes may execute arbitrary code, are running on the same computer and are subject to communication restrictions. However, there are differences; rather than leaking secrets against a MLS policy, they are leaking identity information. The scenario is also unusual – a programming competition, where collusion is necessary to win, but is prevented by a special purpose anonymity system<sup>1</sup>.

### 2.1 Background

Contract Bridge is a game in which collusion is an well established technique. This is a card game for two teams, each of two players. In the initial stage, each player takes turns to make a bid of how strong the cards he has been randomly dealt (his hand) appears to be. A player's goal is to bid as high as possible, while not exceeding the capabilities of his team's combined hands. Players cannot see their partner's cards so there needs to be some other way for a player to deduce the nature of his partner's hand.

---

<sup>1</sup>As far as I am aware, this is the first published case of how a covert channel in a computer system was used for profit.



Photograph: Marie-Lan Nguyen

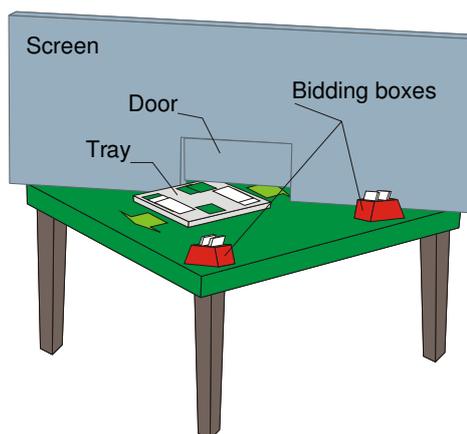


Diagram: Jugoslav Dujic

Figure 2.1: Bidding box (left) allowing Bridge players to signal bids while preventing unauthorised verbal communication. These may also be used with a screen (right), additionally preventing visual communication

Systems for transmitting information between partners during the bidding stage are legal and can provide a great advantage to the team more adept in their use. These schemes provide a means by which one player can encode information about his hand in the cards that he plays. For example, a bid of 1NT often means a balanced hand of intermediate strength. His partner (who he is not allowed to communicate with through any other means) can then make a more precise bid. However, other means of communication, such as facial expressions, are not permitted, so physical barriers to prevent them, as shown in Figure 2.1, are used in high-level competitions.

One complication in Bridge is that while communication is permitted by the rules, if the partner of a player making a bid is asked what the meaning of a bid is, then he must answer truthfully [7, 8], so the information sent through the channel cannot be secret. However, the two members of a team do share a secret, e.g. if one player holds all the aces then he knows that his partner holds none, but the opposing team does not know this [157]. If this secret is used as a key, then it is legal, in certain types of Bridge tournaments, for the recipient of the information to announce the multiple possible meanings of the bid. He does not need to tell his opponent what the bid means when combined with knowledge of the player's own hand.

In Bridge, the collusion is between two members of a team, where communication, other than through bidding, is not permitted. In Section 2.2 we discuss a different situation, where the colluding parties are considered to be

independent players. Here, communication is simply unexpected, since in a competition it is normal for each player to try to optimise his own performance, so there would be no need for communication with other opponents.

In this chapter, we examine the situation where several independent players cannot win the competition acting by themselves, but one of them can win if they collude. If the value of the prize can somehow be divided up between the winner and colluders, this option is attractive for all parties. Conversely, to ensure a “fair” competition, the organisers must enforce an anonymity policy, since then players cannot identify when their opponent is a co-colluder.

In order for collusion to work, there must be some means of communicating. If collusion is not expected, then it may be the case that communication is easy, but the case where it is banned is both plausible and more interesting. In Section 2.3, we discuss how communication can be established, and in particular we show how covert channels can be used for identification. A number of possibilities are presented and compared, including a scheme which draws on techniques used in low-probability-of-intercept spread spectrum radio, to increase the confidence that identification has been performed correctly.

In Section 2.4, an example is given of where these techniques were successfully applied. This was an online programming competition where contestants were required to write a program to play *Connect-4* against the other programs entered. While it was in fact impossible to guarantee a win in any individual game, by developing a collusion based system it was possible to win the contest, subject to reasonable assumptions about other contestants.

Finally, in Section 2.5, defences against such types of collusion are discussed. These include prevention, detection, and modifying the competition so that the benefits of collusion are reduced. One option considered is to use the similarities between elections and competitions so as to design better tournament structures.

## 2.2 Competition structures

The type of competition dictates how effective collusion can be and also how it can best be used. In this section, we introduce two simple but popular tournament arrangements (league and knockout) and show how collusion can be exploited in both. In Section 2.4, these two arrangements are combined to form the hybrid structure that the techniques described in this chapter were designed to win.

Table 2.1: Summary of winners in matches between Fox, Chicken and Optimal players (“—” denotes a draw)

	Fox	Chicken	Optimal
Fox	—	Fox	—
Chicken	Fox	—	—
Optimal	—	—	—

### 2.2.1 LEAGUE TOURNAMENTS

In a league, each of the  $n$  players competes against every other player, resulting in  $n(n - 1)/2$  matches. The structure of a game is not important, only that there are two participants and that it may lead to one of three outcomes: win, lose, or draw. A win earns a player more points than a draw and a draw earns more points than a loss.

We assume that the game is *fair*, that is, neither of the players has an advantage, because any game can be made fair by playing it twice with the roles of the players exchanged the second time. Fairness implies that a perfect player must draw against himself, therefore, no winning strategy exists for the player. Since the opponent has no winning strategy either, the perfect player must have a strategy that guarantees at least a draw.

In order to calculate a lower bound for the benefit of collusion, we assume the worst case scenario – that non-colluding, *independent* opponents are optimal, i.e. they will win a match where possible and draw otherwise. Similarly, we make conservative assumptions for colluding players, namely that they will never lose, but also will never win against independent players. If every player played optimally, then each will gain the same number of points. Where some players collude – *Chickens* who aim to draw against all players except that they lose to *Foxes* – then the Foxes will get more points than would be possible without collusion.

In a competition, let us assume there are  $x$  Optimal players along with  $c$  Chickens colluding with  $f$  Foxes whom the Chickens want to win. A match between an Optimal player and a Chicken, or between two Chickens, will result in a draw since the Chicken will play to draw. However, a match between a Fox and a Chicken will result in a win for the Fox, since the Chicken will recognise that it is playing a Fox. A win gains the winner  $p_w$  points, a draw  $p_d$  points,

and a loss  $p_1$  points (as noted above,  $p_w > p_d > p_1$ ). We assume each player will also compete against himself and draw. This is summarised in Table 2.1.

In this competition, the scores for players within each class are shown below. A Fox will hence score higher in the competition than any Optimal player, since  $c \geq 1$ .

$$\begin{array}{l} \text{Optimal} \quad p_d x + p_d c + p_d f \\ \text{Chicken} \quad p_d x + p_d c + p_1 f \\ \text{Fox} \quad p_d x + p_w c + p_d f \end{array}$$

A variant of this competition is the Axelrod Iterated Prisoner's Dilemma Competition [12]. At each move, a player can "defect" or "cooperate", and is aware of its opponent's previous moves, but not the current one. If both players make the same move, they are awarded 3 points for cooperation and 1 for defection. If they make different moves then the player who cooperates gets 0 points and the one who defects gets 5. Jennings and Ramchurn discovered [69], independently of myself [110], that a similar type of collusion to that presented in this chapter could be used to win such a tournament.

### 2.2.2 KNOCKOUT TOURNAMENTS

The impact of collusion on knockout tournaments is much less than for league tournaments. The result of a match must be a win for one player so as to decide who will continue to the next round. This will require some kind of tie-breaking system, such as the penalty shootout in soccer.

The only way for a player to win in all arrangements of initial matches is if he can beat all other participants. Likewise, if a player can beat all other players then he will win the competition regardless of the initial arrangement. However, it may be advantageous for a player to influence the arrangement of initial matches if there are cycles in the directed graph of game results, for example Figure 2.2(a). Here, Alice and Bob are equivalent players, who both can beat Carol but will be beaten by Dave. Also, Carol can beat Dave. As shown in Figure 2.2(b), if Alice plays as well as possible, then while Alice will win the first round she will be eliminated by Dave in the next round. Then Dave will eliminate Bob and go on to win the tournament. However, if Alice and Bob collude then the result can be as shown in Figure 2.2(c), allowing Bob to win. Alice can deliberately lose the first match and so Carol will go through. In the next round, Carol will eliminate Dave but in the final round Bob can beat Carol. This example shows that there are cases where, if a player

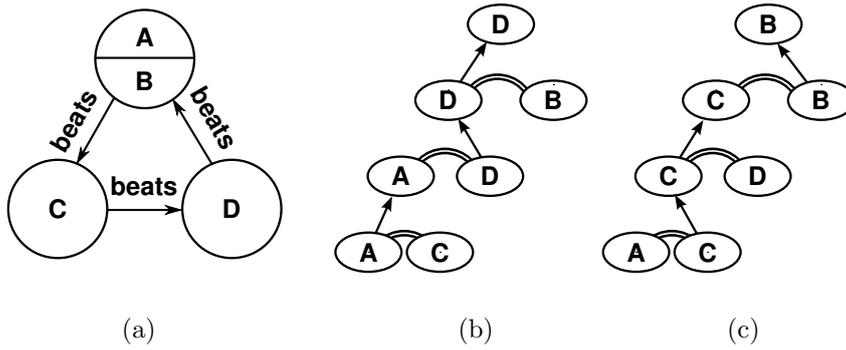


Figure 2.2: Knockout tournament collusion example

is colluding with others in a knockout tournament, it may be advantageous for a colluder to play less well than they are capable of.

Unlike the league tournament, it is clear that the result of a match between co-colluders does not contribute to the final result, if we assume that all players colluding with each other have equal abilities. However, in situations such as those described above, it is useful for a colluder to lose against an opponent who possesses an ability that the colluders do not. We do not explore this further, and the rest of this chapter concentrates on league-like tournaments.

## 2.3 Identification mechanisms

To manipulate a knockout tournament, the abilities of the opponents must be known in advance; however, in a league all that is necessary is for colluding players to perform normally against independent opponents, but to selectively play poorly against other colluding players.

For one player to identify a colluding player when the order of games is not known, there must be some form of identification that occurs before or during each game. This should be reliable and should identify the case where one player must lose before the result of the game is decided.

It may be the case that communication is easy, for example in a face-to-face game the players may recognise each other or be allowed to speak to each other. If the players are computer programs (the case which the rest of this chapter will concentrate on), a standard program-to-program identification protocol (e.g. TLS [45] over TCP/IP [125]) may be executed.

However, there may be times when an overt channel is either not possible because of the constraints of the competition or not permitted by the competi-

tion rules. In these situations, a covert channel can be used, though of course this may also be illegal. A variety of techniques have been developed for such communication channels; however the majority of them are described in the literature for the analysis of multilevel secure computer systems, so while not directly relevant, they can be modified for use within games.

Such techniques could also be used to steganographically send information other than identification. For example two people playing an online game could communicate an encrypted message, as implemented by Hernandez-Castro *et al.* [77]. If the mechanism were designed properly, the fact that a message is being sent could be hidden from an observer.

### 2.3.1 TIMING

In the literature on multilevel secure systems, one frequent way to create a covert channel is for a program to signal to another by varying some kind of system-wide property. For example, this could be modifying the CPU load [81], hence changing scheduling patterns, or it could be modifying timing of acknowledgements to messages which may flow in only one way [85]. These techniques could be used directly, but there are also timing based covert channels that are specific to games.

One such channel would be to use the timing of moves to carry information by causing the sender to delay making a move and the recipient to measure this period. Such schemes are easy to create and can have a relatively high bandwidth i.e. can transfer a significant amount of information. However, if the transport mechanism is affected by latency and/or jitter, then this covert channel may be unreliable or even eliminated completely.

Covert channel encoding schemes can easily cancel out fixed latency, but jitter is more problematic. If the jitter is sufficiently small, then it can be removed, at the cost of reducing bitrate. If the CPU time to make a move is limited by the competition rules rather than “wall clock time” (the amount of time to have passed in the real world), then the maximum delay can be fairly large, since in most operating systems the time that a program is paused is not counted towards the CPU time.

One form of jitter specific to a competition is if the time for a move to be sent is constant and greater than the maximum allowable time for deciding a move. This may occur if the competition is to be shown live and the organisers wish to slow the competition to a speed that humans can watch. If this is done, then the move-timing channel would be eliminated.

### 2.3.2 CHOICE OF EQUIVALENT MOVES

The timing based mechanisms mentioned above require careful coding in the presence of jitter. An alternative to this is to encode the identification data in the moves themselves. In person-to-person games, this could be, for example, the way the pieces of a board game are held, or where a card is placed in a card game (this is why there are complex physical arrangements in Bridge tournaments to prevent such communication). In contrast, for the case of an online competition, the move will likely be expressed in an unambiguous form, hence allowing no extra information to be carried.

At a stage in the game, if there is more than one move which can be shown to not change the outcome of the game when compared to the best move, information can be transmitted. In order to send  $r \in \{1, \dots, n\}$ , the  $r$ -th move is chosen from a set of  $n$  equivalent options. After receiving a move from its opponent, a player can identify which move, out of the opponent's possible moves, was chosen, and hence recover  $r$ .

The order chosen for moves can be arbitrary, but often there is an obvious solution, for example in the Connect-4 situation described in Section 2.4, ordering moves by column number would be one possibility.

### 2.3.3 ANALYSIS OF IDENTIFICATION MECHANISMS

In order for a collusion strategy to succeed, a reliable covert channel must be established to allow a Chicken to identify when it is playing a Fox and thus should deliberately lose.

For the simple case of where a Chicken needs to identify whether its opponent is a Fox or not (Section 2.2.1), the purpose of the channel is to transfer a single bit while the result of the game is still undetermined. Although the required capacity of the channel is low, the reliability requirements are high, since a false positive will result in a Chicken losing to an independent opponent and so reduce the chance of the Fox winning.

Much research on bandwidth estimation of covert channels, for example Millen [99], concentrated on finding upper bounds for the data rate of the channels. These techniques can be used to design a coding system which approaches these upper bounds.

In the case where timing information is used for identification, it is possible that the communication channel will corrupt information being sent. However, where the move itself carries the information, it is reasonable to expect that

the signal will be received intact. For this reason, a message sent using this covert channel will always be received correctly.

However, this does not mean that the channel is noiseless, since the receiver cannot differentiate between the case where information is being sent, and the case where the moves carry no meaning (this is also true for timing channels). The moves of independent players are analogous to noise. The situation is similar to low-probability-of-intercept spread-spectrum [9, p327] radio in that the “amplitude” of the signal cannot be any more than the noise (a particular move is either made or not, there is no concept of “magnitude”).

In order to reliably transmit a single bit of information, a technique based on frequency-hopping can be used. For each move, the number sent is chosen according to a keyed pseudo-random number generator. The receiver shares the key and so knows what move to expect from a colluding player. If, after a specified number of moves, the receiver has found that the opponent has made every move as expected, then it can assume that the opponent is colluding with it and act accordingly. The confidence level of the decision being correct can be improved by increasing the number of possibilities at each move or by increasing the number of moves before a decision is made. While waiting longer before making a decision is preferable, if the player waits too long, then by the time a decision is made, it may be no longer possible to change the game result.

#### 2.3.4 IDENTIFICATION KEY

The goal of the generator is to distinguish itself from the “background noise” of other players. Where little or nothing is known about the game strategies of independent players, it is difficult to make any assertions about the characteristics of the noise. For this reason, it may be safe to assume that at each turn every move is equally likely – analogous to white noise. This assumption is particularly useful since it greatly simplifies the design of the generator, and allows a fast implementation so as to reduce CPU usage (which may be a factor in deciding a winner).

For spread-spectrum radio, typically a cryptographically secure pseudorandom number generator, such as a stream cipher, is used. In the case of spread-spectrum radio the transmission is effectively public but in a game the moves are typically only seen by the opponent. One threat in spread-spectrum radio is an adaptive adversary, whereas in a game the opponents are not permitted to be changed during the competition. When coupled with the fact that other

opponents are probably not aware of the collusion strategy, it is reasonable to assume that cryptanalytic attacks are unlikely. This assumption simplifies the design of the generator and so reduces processor time requirements.

The only goal of the generator is to appear different from a white noise source, so a repeating constant could be used, such as always picking the first move. However, it is plausible that an opponent could accidentally pick the same strategy. A small change can be made where the move chosen depends on the stage in the game. For example,  $r$  could simply be the result of a pseudorandom number generator (PRNG) seeded by a shared secret. This simple identification system could also be used with the timing based covert channels. A linear congruential PRNG is very fast and simple, and with well chosen parameters [89, Section 3.2.1] meets all the requirements (assuming no cryptanalytic attacks).

## 2.4 Real world example

The above techniques were developed for and used with the 2002–2003 Cambridge University Computing Society (CUCS) Winter Competition [29]. This was a programming competition where entrants submitted one or more programs which played a variant of Connect-4. These programs then played against each other and a winner was decided.

### 2.4.1 RULES OF THE GAME

As with normal Connect-4, the game is played on a  $7 \times 6$  board. Each player takes turns to choose a column and places his token at the lowest free square. The first player to have four tokens in a row, either horizontally, vertically or at a  $45^\circ$  diagonal, wins the game. In the standard game, a player must place exactly one token at each turn, but in the variant used in the competition, the player also has the option to pass. This change was made so that standard Connect-4 strategies [4] would not work and thus force entrants to invent with their own techniques. However, an unforeseen result of the modification to the rules was that the possibility of a guaranteed winning strategy was eliminated since a move cannot be forced.

The competition was split into two stages, a league followed by a knock-out tournament. The league proceeds by every entered program being played against every other entered program. Each match consisted of six games, with each player alternately starting first. The winner of the match was the player

Table 2.2: Summary of winners in matches between Fox, Chicken, Rooster, Rabbit and Optimal players (“—” denotes a draw)

	Fox	Rooster	Chicken	Rabbit	Optimal
Fox	—	Fox	Fox	—	—
Rooster	Fox	—	Rooster	—	—
Chicken	Fox	Rooster	—	—	—
Rabbit	—	—	—	—	Optimal
Optimal	—	—	—	Optimal	—

with the most number of wins and was awarded two points. If both players had an equal number of wins in the match, then each player is awarded one point; no points are awarded for a loss.

The five programs with the highest scores in the league were selected for the knockout tournament. Firstly, the fourth and fifth programs were played in a match of six games as in the league. However, if this match was a draw, then the winning program would be the one with the lower CPU usage, and if that was equal, then memory usage and finally code size were to be considered.

Then, the remaining four programs were played in a standard knockout tournament, with each match following the rules for the fourth/fifth playoff, i.e. fourth/fifth vs. first, second vs. third. Finally, the winners of the previous two matches played each other and the winner of the competition declared.

#### 2.4.2 COLLUSION STRATEGY CHOSEN

In this competition, overt communication was not permitted in order to prevent programs communicating with more able humans or more powerful computers. Also, the only information that a program received from its opponent was the move number, in ASCII, so there was no redundancy in the encoding. However, the rules did not explicitly prohibit collusion between opponents.

For these reasons, a covert channel was required for communication, but it would not break the rules. There were plans for the final stages of the competition to be run live so timing information may have been jittered, even unintentionally. Because of the advantages in reliability and simplicity of the *Choice of Move* covert channel described in Section 2.3.2, we adopted this approach for our implementation.

One refinement to the identification method described in Section 2.3.4 was rather than having only two types of colluding player (the Fox and the Chicken, where a Fox always wins against a Chicken), three were used. The additional category, *Rooster* would beat a Chicken but would be beaten by a Fox (see Table 2.2). This was because collusion is ineffective in the knockout stage, so the only way to be certain of a win was for all five participants to be our colluding players. This could be achieved by having five Foxes and the rest Chickens, but there remained the risk that another independent player would get into this stage (due to *Rabbits*, the category which will be introduced in Section 2.4.6).

Since, by applying the strategy described in Section 2.4.3, our players will never lose, CPU usage would be the decider and so this should be optimised. Hand optimising a program is time consuming so it is preferable to minimise the number of programs that this needs to be done on. If only one of the five Foxes was optimised, then there is the risk that another will knock it out of the tournament before it has a chance to play the independent player. To mitigate this risk, two optimised Foxes were entered, along with four Roosters, so the optimised Foxes would be guaranteed to play any remaining independent players. Two Foxes were entered to reduce the impact of any programming errors. This reduced the number of points given to the Roosters and Fox slightly, but it was decided to be worthwhile.

### 2.4.3 GAME STRATEGY

In order for collusion to be feasible, it is necessary to have a strategy which guaranteed a draw in every game. It is also desirable to design the strategy such that all the outcomes of the game remain possible for as long as feasible, so that the decision as to whether to lose or not can be delayed. Finally, so as to optimise the bandwidth of the covert channel, the number of possible moves at each turn should be maximised.

Piotr Zieliński developed a very efficient strategy which allowed a draw to be forced, regardless of who made the first move. For completeness we include its description and his associated diagrams here.

The strategy relies on finding a subset of the squares on the board, such that every winning line must pass through at least one of the marked squares, and preventing the opponent from occupying any of them. This is achieved by designing a pattern of non-overlapping rectangles on the board as is illustrated in Figure 2.3(a).

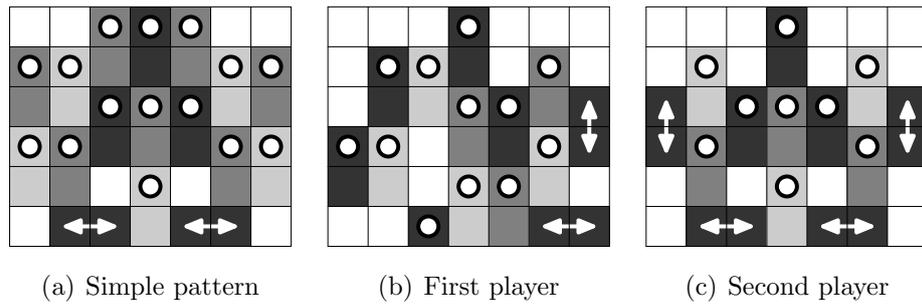


Figure 2.3: Possible board patterns used for the game strategy. The different shades of grey have no semantic meaning; they are used only to differentiate the rectangles from each other

- 

If the opponent plays on the bottom square, then our player plays on the top square. Our player never plays on the bottom square. Therefore, the opponent can never occupy the top square.
- 

If the opponent plays on one of the squares, then our player plays on the other. Therefore, the opponent can never occupy both squares.
- 

If our player moves first, then it plays on this square, thereby preventing the opponent from occupying it.

Three possible patterns are shown in Figure 2.3. Since the rectangles do not overlap, the strategy forces our player to play on at most one square per move, thereby guaranteeing at least a draw.

#### 2.4.4 IMPLEMENTATION

The competition allowed ten entries per person and three people entered from our research group. While the rules explicitly stated that it was permitted to implement an algorithm developed by someone else, using someone else's code was not allowed. For this reason, each member of the group entered a program written independently in a different language (Ada95, C and Java – unfortunately due to the lack of a good debugger the PostScript version had to be dropped).

As intended, no players lost other than by design, against another colluding player. While there was some risk that this (false positive) could have

happened by accident, the design of the covert channel reduced this to an acceptable level. As shown in Figure 2.4, after ten moves (the point at which a decision was made) the number of possible move sequences ranged between 480 and 51 840, with a mean of 6 380. Therefore, even if an opponent happened to choose an identical game strategy, the probability of a false positive was at least 1 in 480 (subject to previous assumptions). In contrast, the risk of a false negative (that one colluding player, who should lose to its colluding opponent, fails to identify in time) can be reduced to the risk of programming error. This is because the covert channel used can be assumed to introduce no noise. Furthermore, for deterministic players, all possible games between colluding opponents can be (and in our case were) exhaustively tested in reasonable time, before entry to the competition.

#### 2.4.5 OPTIMISATION

The final stage of the competition would take CPU usage into account, so there was a potential advantage to optimise the Foxes. Aside from standard code efficiency improvements, one domain-specific optimisation was to remove all detection code from the Foxes. This simplification was possible since it was not necessary for a Fox to identify that it is playing a colluding player, as the responsibility for the match result can be given to the losing player. To achieve this, a player who has identified that it must lose will continually pass until the game has ended. Additionally, no evidence of collusion can then be found by inspecting the source code of the Foxes.

To ensure the game will result in a win for the Fox when the Chicken passes, the game strategy must be changed slightly. Firstly, the Chicken must start playing losing moves sufficiently early in the game such that it is still possible to lose. Secondly, a different pattern must be used for the player starting first and the player starting second. This is because if both players have the same pattern they would draw the game by default after playing four passes before the identification could be completed. Thirdly, more flexible patterns (Figure 2.3(b) and Figure 2.3(c)) give the players more equivalent moves, thereby increasing the reliability of the identification procedure.

#### 2.4.6 EFFECTS OF POOR PLAYERS

In the simple example of Optimal players and colluding players, it was seen that only one Chicken was necessary for the Fox to win, however, the situation is not so simple when not all independent players are Optimal. That additional

Table 2.3: Summary of results at the end of the league stage. Players are ordered in descending order of points

No	Category	Won	Drew	Lost	Points
1	Fox	58	26	0	142
2	Fox	58	26	0	142
3	Rooster	51	29	4	131
4	Rooster	49	31	4	129
5	Rooster	49	31	4	129
..... cut-off point .....					
6	Rooster	48	32	4	128
7	Semi-Optimal	16	67	0	99
⋮	⋮	⋮	⋮	⋮	⋮
13	Semi-Optimal	12	64	8	88
14	Chicken	3	69	12	75
⋮	⋮	⋮	⋮	⋮	⋮
37	Chicken	0	72	12	72
38	Semi-Rabbit	4	63	17	71
⋮	⋮	⋮	⋮	⋮	⋮
43	Semi-Rabbit	1	52	31	54

worst-case category of players (so as to find a lower bound) is a *Rabbit*, which will play poorly, so lose to Optimal players, but draw with everyone else. From Table 2.2 it can be seen that an Optimal player will act as if it is colluding with any Rabbits in the tournament. Therefore, the only way to win the tournament is to have a greater number of Chickens than there are Rabbits, no matter how many Optimal players exist. While it was likely that several players entered would be approximately Optimal, our strategy assumes that there would be a small number of contestants who would enter a player that would play so badly that the chances of winning would be low.

#### 2.4.7 RESULTS

A summary of the final league table is shown in Table 2.3.

Since the algorithm used by the Fox, Rooster, and Chicken would only win in exceptional circumstances, the actual results for colluding players in

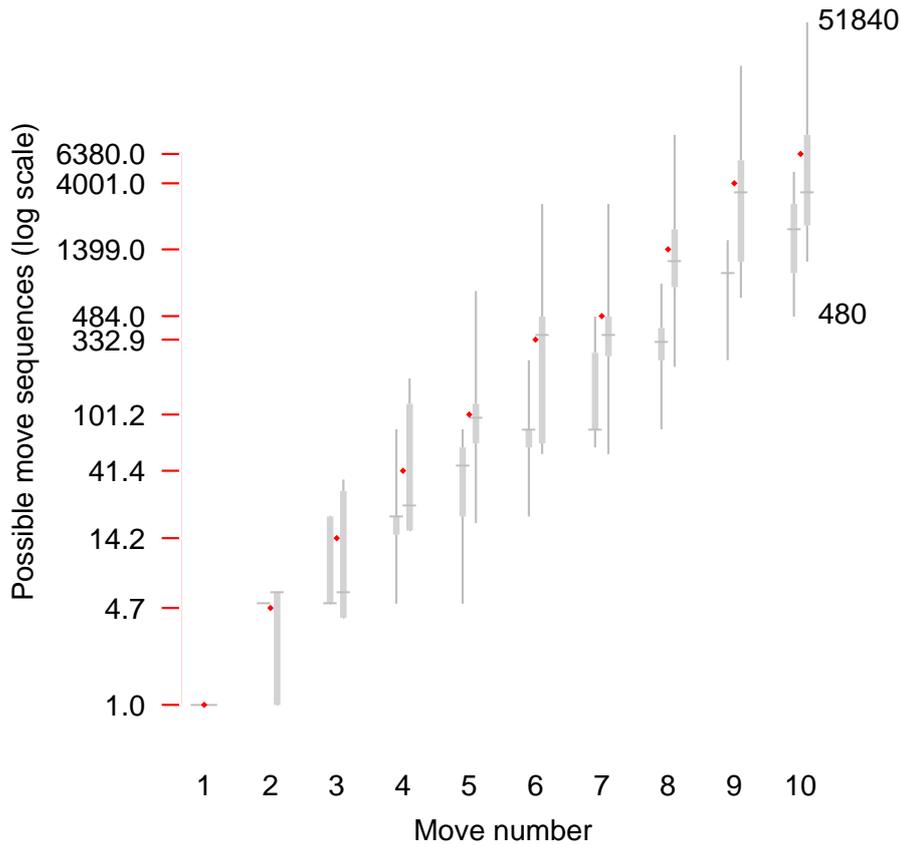


Figure 2.4: Number of possible move sequences after a given number of moves. For each move number, the left line is for the player making the first move and the right line is for making the second move. Three classes of colluding players were used (Fox, Rooster, and Chicken) so for each move number, the lower limit, first quartile, median, third quartile and upper limit of the nine possible matches are plotted using a boxplot variant. The mean over all classes of players, and over both first and second players is indicated by  $\blacklozenge$ , and the absolute values shown as the  $y$  axis tick-marks

the competition were very similar to the worst-case scenario estimates. Some players appeared to play randomly, so when played against programs using a tree-searching algorithm the tree-searching algorithm won. This behaviour approximates the expected results from Rabbits and Optimal players, so the random players are classed as Semi-Rabbits and the tree-searching players are classed as Semi-Optimal. However, as expected, only six Semi-Rabbits were entered by other participants and 28 Chicken/Roosters were entered by our group, so we won the competition by a safe margin of 30 points.

## 2.5 Defeating collusion

In the above example, neither non-colluding participants nor the competition management expected collusion to be used. In the case where collusion is expected and not desired, there exist interesting possibilities for preventing collusion from being effective. These fall into three categories. Firstly, covert channels can be prevented, secondly, the use of collusion or a covert channel can be detected, and finally, the competition could be designed to give no advantage to colluding players.

### 2.5.1 COVERT CHANNEL PREVENTION

To prevent collusion, the game framework can enforce anonymity between players by preventing the communication of identity information through covert channels while minimising the impact on game play. Chapter 1 and the works it references discuss covert channels in MLS systems and similar techniques can be used to both create and block covert channels in games. For example, as mentioned in Section 2.3.1, adding jitter reduces the capacity of the move timing covert channel. Non-application-specific channels, such as CPU load modulation, can be prevented in the same way as on MLS systems.

However, the choice-of-move channel is harder to defend against, because altering the move made would harm the integrity of the competition. One possibility for reducing the capacity of this channel is to prohibit the program from maintaining state. At each stage, the program would be restarted, be given the board position, and asked for the next move. Without move history, the signalling technique used in this chapter would not be effective. However, the board state would allow some communication, and this restriction would prevent legitimate strategies based on determining opponent characteristics.

### 2.5.2 DETECTING COLLUSION

In some games, it may not be desirable or possible to eliminate covert channels. Here, the only alternative may be to detect collusion and disqualify players if caught. One possible technique for identifying collusion is to watch the moves the player makes and look for efforts to communicate. However, a PRNG is not suspicious by itself, even if it is not cryptographically secure, since it is arguable that every method of choosing one of several moves is equally valid.

Watching what happens when a player loses could be more revealing. In the example given, the players would simply pass once they decided to lose.

If the competition rules stated that each participant must play at the same standard with every player, then this behaviour could be identified from game analysis. Another way of detecting this behaviour would be to examine the source code of players, but the code responsible for losing could be hidden. Additionally, if the optimisation described in Section 2.4.5 is used, then no evidence of collusion can be found by inspecting the source code of the winning players (Foxes).

To detect more subtle collusion, an expert could examine the match results [159], and in a similar way that a Bridge expert would look for players being exceptionally lucky in a tournament, an expert suspecting collusion would look for players being exceptionally unlucky. The expert could also monitor the games in progress looking for a suspicious change in apparent skill. If a player is aware of such monitoring, then countermeasures to both techniques could be taken, such as Chickens losing to Foxes with a probability of less than one, and playing to lose in a manner more plausible than consistently passing.

### 2.5.3 COLLUSION RESISTANT COMPETITIONS

An alternative way to deter collusion is to design the competition such that it provides no advantage. During discussion of the problem one observation made was that the problem of deciding a winner in the competition is similar to that of electing a candidate in an election<sup>2</sup>. While there are some differences, for instance, that the number of candidates is identical to the number of voters, there are also many similarities.

One possibility investigated was of a game tournament similar to the Single Transferable Vote (STV) system. Here, every player plays every other player, in a similar fashion to a league tournament. However, the winner evaluation is more complex. At each stage, the normal league rules are applied and an ordering established, but then the players with the lowest score are eliminated, along with their contribution to all other players' scores. The process is repeated until no more players can be eliminated.

This system has the advantage that Chickens will be eliminated before Foxes, so the Chickens' scores can have no effect on the final result, however, they can control the order in which players are eliminated so it is not clear that this system is free from manipulation. Additionally, the number of "voters" is identical to the number of "candidates" so the final stage will likely result in more than one winner. This was confirmed by running the results of

---

<sup>2</sup>I thank Ian Jackson for making this contribution

the above example competition through this algorithm. As expected, all the Chickens were eliminated but the final result included the Foxes and all the Semi-Optimal players. Since all these players will draw against each other, deciding a winner is difficult, but this scoring system has at least destroyed the advantage of collusion.

Not only should competitions be resistant to collusion but they should be fair and this is a very difficult quantity to measure. There are a variety of proofs which state, given certain assumptions, that it is not possible to design an *ideal* election. These include Arrow's theorem [11], Gibbard-Satterthwaite [62, 138] and Gärdenfors' extension [61]. These primarily deal with manipulation by voters, but there has been some work on manipulation by candidates, such as a general result of Dutta *et al.* [52] and an analysis of the particular case where the election is made out of a series of pair-wise comparisons, in a later paper [53]. These state that, given certain assumptions, non-dictatorial elections<sup>3</sup> are manipulable by candidates deciding whether or not to participate in the election. This result is not directly applicable since it assumes that each candidate who votes will vote himself the highest, and the stronger version of the result also assumes that no candidates vote. However, it may still be partially applicable. Whether these theories imply that an *ideal* competition is impossible depends on a formal definition of fairness and collusion resistance, which is outside the scope of this thesis.

## 2.6 Conclusion

In this chapter, we have shown that collusion can offer significant advantages in tournaments that are based around leagues. We presented a simple algorithm for acting on the basis of identification information, which will guarantee winning a competition, assuming only one team is using a collusion strategy and the standard of players is good. We have also introduced a covert channel built using only redundancy in the moves of a game and show how this can be used to identify colluding players, in spite of the game infrastructure enforcing anonymity. We demonstrated these techniques being successfully applied in order to win a real world competition. Finally, options for resisting and detecting collusion are explored, including drawing parallels between the design of competitions and the design of elections.

---

<sup>3</sup>The result of a non-dictatorial election must be decided by more than one voter

## Chapter 3

# Embedding covert channels into TCP/IP

The previous chapter described techniques for two processes on one computer to communicate, in violation of a security policy. Early interest in covert channels focused on this scenario, as the computing environment at the time was based around a small number of large, multi-use mainframes. There, it would not have been feasible to buy separate computers for each security level, but now, as computers are far cheaper, this option is available. An advantage of this approach is that it avoids the covert channels that come about through resource sharing (e.g. CPU, hard disk and memory).

However, as a consequence of the physical separation, now data and processing is distributed, so a network is needed to share information. To ensure that flow control policies are observed, enforcement must be built into the network. The hope was that such hardware would be simpler and thus easier to secure than the complex operating systems on mainframes. An example of such a device is the NRL Pump [85], which enforces that data must only flow in one direction, in compliance with the MLS policy.

As each computer only manipulates data at a particular security level, covert channels on a host are no longer a concern, but ones over the network are. Covert channels may be used to transmit information in a way which is otherwise prevented by protection mechanisms, or they can hide from auditing systems which detect violations of policy. The general-purpose covert channel introduced in this chapter can be used in a wide-variety of situations and will serve as an introduction to the special-purpose channels designed to break anonymity systems, to be described in the following two chapters.

Due to the ubiquity of the protocol, covert channels in TCP/IP have been a popular research topic. It is commonly believed that undetectable steganography within TCP/IP is easily achieved by embedding data in header fields seemingly filled with “random” data, such as the IP identifier, TCP initial sequence number (ISN) or the least significant bit of the TCP timestamp. Not only would these techniques allow information to be transferred in violation of a security policy, but even detailed logs would not reveal the contravention.

This is not the case; these fields naturally exhibit sufficient structure and non-uniformity to be efficiently and reliably differentiated from unmodified ciphertext. Previous work on TCP/IP steganography does not take this into account and, based on TCP/IP specifications and open source implementations, this chapter provides tests to detect the use of naïve embedding. Finally, it describes reversible transforms that map block cipher output onto TCP ISNs, indistinguishable from those generated by Linux and OpenBSD. The techniques used can be extended to other operating systems. A message can thus be hidden so that an attacker cannot demonstrate its existence without knowing a secret key.

## 3.1 Introduction

Steganographic covert channels based on modification of network protocol header values are best understood by considering a scenario with three actors; in keeping with the existing literature, we shall call them Alice, Bob and Walter. Alice can make arbitrary modifications to network packets originating from a machine within Walter’s network. Allowing Alice to generate new packets creates further opportunities [64], but we do not consider those here. She wants to leak a message to Bob, who can only monitor packets at the egress points of this network. Alice aims to hide the message from Walter, who can see (but not modify) any packet leaving his network. This is analogous to a passive warden within the threat model introduced by Simmons [142].

In a practical instantiation of this problem, Alice and Bob may well be the same person. Consider a machine to which an attacker has unrestricted access for only a short period of time, and which lies within a closely monitored network. The attacker installs a key-logger on the machine, and wishes to leak passwords or cryptographic keys to himself. *Rootkit* techniques can hide the Trojan from virus-detectors, but the communication channel must additionally ensure that the owner of the network does not observe anything untoward.

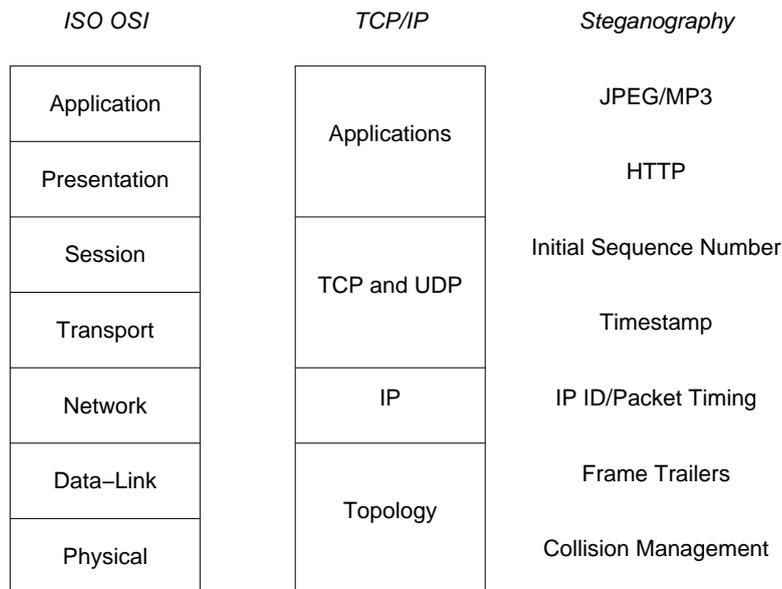


Figure 3.1: OSI model layers and approximate mapping to TCP/IP

Alice can choose which layer of the protocol stack she wishes to hide her message in. Each layer has its own characteristics, which indicate the scenarios in which it can best be used. Handel and Sandford [73], discuss opportunities for embedding across all layers of the OSI model, as shown in Figure 3.1.

At the bottom of the stack, in the Physical and Data-Link layers (e.g., Ethernet), there is some opportunity for embedding data. However, it requires low-level control of the hardware, which Alice may find difficult to obtain. Also, if she chooses to signal to Bob at this layer, her messages will be stripped out if they reach a device that connects networks at a higher layer (e.g., an IP router). This requires Bob to be on the same LAN. An example of a steganography system that relies on embedding at the Physical layer is described by Szczypiorski [148].

Alice might also choose to embed data at the Presentation or Application layers of the network stack (e.g., in Telnet or HTTP/FTP traffic). If, however, she only has brief access to the machine from which she is leaking data, she needs to anticipate which applications are likely to be used on it; she can then modify them to carry her messages in the traffic they generate.

Similarly, the format of files sent over HTTP or FTP (such as JPEG or PDF) may also be viewed as protocols in which steganographic data can be embedded. These provide Alice with a high-bandwidth channel, but only if she is confident of being able to modify these files without arousing suspicion.

The only remaining layers to consider in the OSI model are Network, Transport and Session. TCP and IP (specified in RFC 793 [125] and RFC 791 [124]) fall within these layers, and are common to the vast majority of Internet applications. A message embedded in these protocols has the advantage that it will survive unchanged on its journey out of Walter's network. Here we consider only IPv4-based embedding; IPv6-based covert channels are discussed by Lucena *et al.* [95].

This chapter studies a number of previously proposed schemes for embedding data within the TCP and IP protocol headers, thus creating a steganographic covert channel. It shows how the use of these schemes can easily be detected by a passive warden. The algorithms used in the generation of some TCP/IP header fields are then looked at in detail, and an alternative method for embedding data, *Lathra*, is proposed. The chapter will show that a passive warden cannot detect the use of this method without knowledge of a secret key, subject to some realistic constraints. These results are also relevant to the field of operating system and physical device fingerprinting.

## 3.2 Threat model

In the passive warden threat model, steganography can only be countered by detection, not by attempting to remove any hidden information. An active warden can modify traffic without needing to first establish whether it contravenes rules. Fisk *et al.* [56] and Handley *et al.* [74] show that an active warden can remove most, if not all, TCP/IP level steganography, and lower layer steganography will already have been removed by routing. He will, however, have difficulty removing steganography at higher layers (e.g. in JPEG images) without damaging the carrier.

In many scenarios, it may be infeasible for a warden to be active: the kind of filtering necessary to remove TCP/IP steganography can increase network latency, and might require a filtering router that can store large amounts of state. The warden may also wish to avoid the users being aware that the use of steganography is suspected. Furthermore, passive wardens degrade more gracefully when overloaded or when only partial traffic is available.

In this chapter, we assume that Alice operates in an environment with a passive warden and an unreliable network (permitting packet loss, duplication and reordering) and requires a TCP/IP-based covert channel giving

- *indistinguishability*: Walter (a passive warden) should be unable to detect the presence of the data hidden in packets leaving Alice’s machine; and
- *reliability*: she desires some indication of whether her messages to Bob have indeed arrived, so she can retransmit them if necessary.

### 3.3 TCP/IP-based steganography

A common failing of previous steganography proposals is the production of fields with values drawn from a different probability distribution to that which would be generated by unmodified TCP/IP implementations. In some cases, it is even outside the relevant specifications. For this reason, to design steganographic techniques or to detect their use, it is necessary to be familiar with both the applicable standards and the details of their implementation. This section gives an overview of the TCP/IP standards and related work from a steganographic encoding perspective.

The basic TCP/IP protocol is specified in RFC 793 [125] and RFC 791 [124]. There are extensions to it (e.g., the TCP Extensions for High Performance, in RFC 1323 [82]) that specify additional header options; these also give some scope for steganographic coding.

IP itself does not aim to provide any stream reliability guarantees, but rather allows client protocols on a host to transport blocks of data (datagrams) from a source to a destination, both specified by fixed-length addresses. One noteworthy feature of IP, for our purposes, is that it allows the fragmentation and reassembly of long datagrams.

TCP, on the other hand, does aim to provide a reliable channel to its clients. It is connection-oriented, and keeps its reliability properties even over networks that exhibit packet loss, reordering and duplication. Its features for implementing reliability and flow control give scope for steganographic coding.

A protocol header can serve as a carrier for a steganographic covert channel if a header field can take one of a set of values, each of which appears plausible to our passive warden. The warden should not be able to distinguish whether the header was generated by an unmodified protocol stack or by a steganographic encoding mechanism. In this section we examine which TCP/IP header fields have more than one plausible value, and look at the bandwidth available in each of them for use by a steganographic coding scheme.

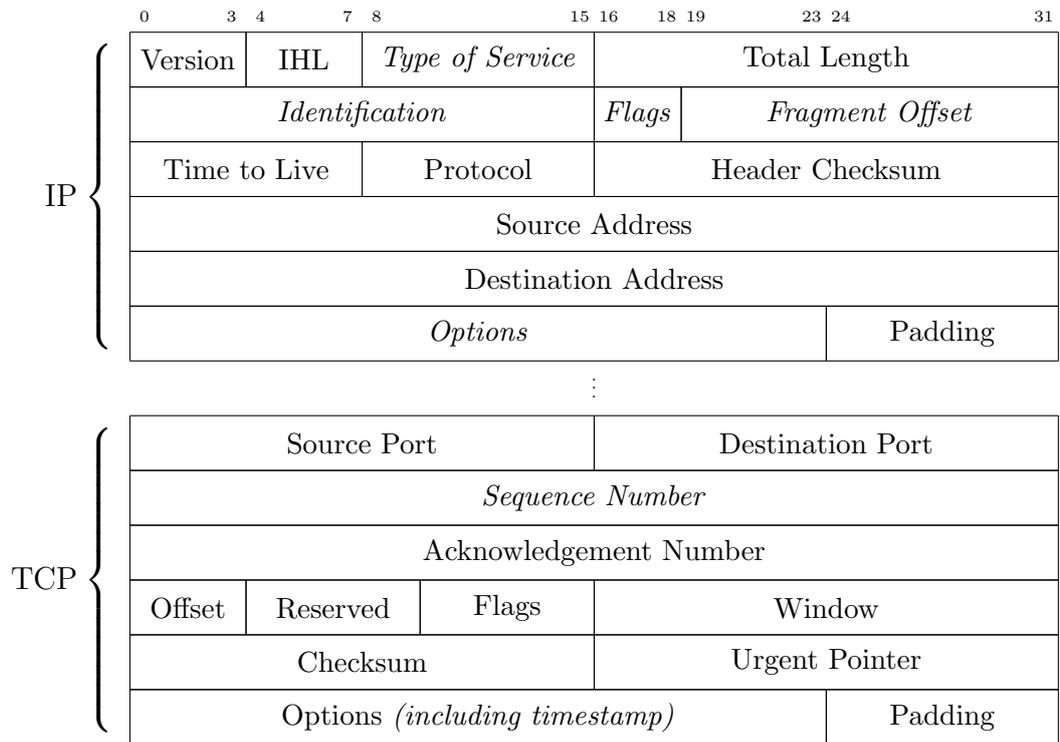


Figure 3.2: Basic TCP/IP header structure

TCP/IP steganography exploits the fact that few headers are altered in transit. As mentioned above, IP packets can be fragmented, but (unless we are hiding data in the fragmentation-related fields) no information is lost. The time-to-live field in the IP header is decremented each time the packet passes through a router, but the initial values used by IP stacks are well known, so this field gives little scope for detection resistant steganography.

Figure 3.2 illustrates the basic TCP/IP headers. The fields shown in italics are those that may be used to embed steganographic data. We now consider each of the fields that are of the most interest in turn, assessing their potential for use as steganographic carriers.

### 3.3.1 TYPE OF SERVICE

The eight Type of Service (ToS) bits in the IP header indicate quality of service parameters to routers on a packet’s path. They are now rarely used with their original semantics (as defined in RFC 791 [124]) but have been re-used in the implementation of DiffServ [115] in RFC 2474.

There is potential for using this field as a steganographic carrier, as described by Handel and Sandford [73], because many networks never use them.

However, this would be easily detected by the warden in our threat model, as the field is set to zero in almost all default operating-system configurations.

### 3.3.2 IP IDENTIFICATION

As described in RFC 791 [124], the IP Identification field (IP ID) is “an identifying value assigned by the sender to aid in assembling the fragments of a datagram”, and is allocated 16 bits of the IP header. Because the IP ID is used to distinguish fragments making up one packet from fragments making up another, it should be unique over the length of time that fragments of a packet might reasonably remain in a network, and unpredictable.

IP IDs that are unique within a given time window are necessary to ensure that fragments of different packets are not reassembled into one packet on the receiving host. Unpredictability prevents *idle scanning* [60], discussed in Section 1.4, whereby an attacker can portscan a host while spoofing the source address of all packets.

A scheme for embedding data in the IP ID is described by Ahsan and Kundur [3]. It uses a pseudorandom sequence to form a substitution cipher, generated by a “toral automorphism” [153], to ensure that the modified field is scrambled. Here, the sequence of  $(x_i, y_i)$  generated from a key  $(k, N, x_1, y_1)$  is defined by:

$$\begin{bmatrix} x_{n+1} \\ y_{n+1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ k & k+1 \end{bmatrix} \begin{bmatrix} x_n \\ y_n \end{bmatrix} \pmod{N}$$

However, this can be detected since IP ID fields are not random, as will be described in Section 3.5.1 below.

### 3.3.3 IP FLAGS

IP packets include two flags, *Do Not Fragment* (DF), indicating that the packet should be discarded if it cannot be sent without fragmentation, and *More Fragments* (MF) which is 0 if the packet contains the last fragment or has not been fragmented. Ahsan and Kundur propose the use of the DF bit for steganographic signalling [3]. If this is used on packets smaller than the maximum segment size, the DF flag has no effect on the packets’ behaviour. However, the normal state of DF can be predicted from the packet’s context, so the warden in our threat model can detect the use of this technique.

### 3.3.4 IP FRAGMENT OFFSET

When IP packets are fragmented, the individual fragments contain an offset field; this allows the receiving host to reconstruct the fragments in the correct positions in its receive buffers. Information can be transmitted covertly by modulating the size of the fragments originated by a host, and thus the fragment offsets. As with the IP identification and ToS fields, this method of steganographic encoding is easily detected. In environments where path MTU discovery [102] is routinely used, fragmented packets are unusual.

### 3.3.5 IP OPTIONS

IP packets very rarely contain “options”, so their potential for use in undetectable steganography is limited. Handel and Sandford [73] describe the use of the IP Timestamp option (not to be confused with the TCP Timestamp discussed below), but in addition to being easily detectable, packets with this option present can travel at most 20 hops, so it is of limited use on the Internet.

### 3.3.6 TCP SEQUENCE NUMBER

TCP sequence numbers support the reliability and flow control features provided by TCP. Each octet of data transmitted over a TCP stream is assigned a sequence number. In TCP, a connection, defined by a pair of (IP, port) tuples can be re-used, and hence the host must be able to detect whether a segment is from a current or previous incarnation of a connection.

When a connection is established, both hosts must choose an *initial sequence number* (ISN). Careful design of the algorithm for generating these initial sequence numbers ensures that an immediate overlap in sequence number space, between different incarnations of a connection, is prevented.

There are other properties required of the algorithm used for initial sequence number generation. To prevent spoofing, for a given connection, the ISNs used must be hard to guess for those not involved in the connection [18]. To allow a connection in the `TIME_WAIT` state to be restarted, the sequence numbers for a given tuple pair should also be monotonically increasing.

A prototype implementation of steganography using TCP ISNs (and also the IP ID), `Covert_TCP`, is described by Rowland [136]. It replaces the chosen field with the data to be sent, so can be detected by observing that fields do not meet the required overlap and uniqueness constraints, or by comparing the data observed with statistical patterns of suspected plaintext.

A passive warden using a Support Vector Machine (SVM) is presented by Sohn *et al.* [144]. It is designed to detect the use of `Covert_TCP` within the IP ID and TCP ISN. A SVM is a machine-learning technique for automatically identifying features which are not well understood. The algorithms for generating IP IDs and ISNs are well understood and precisely described in source code, so it is not necessary to resort to machine learning techniques. The SVM can only identify simple features, so it cannot detect the complex structure present in these fields, and their interdependencies.

The design and implementation of *Nushu*, an improvement to `Covert_TCP`, for Linux 2.4, is described by Rutkowska [137]. *Nushu* uses TCP ISNs for encoding information and encrypts outgoing ISNs to hide the use of steganography, however it still may be detected. Firstly, the output will not exhibit the structure of TCP ISNs expected from Linux. Secondly, a flaw in the use of DES for encryption allows the recovery of statistical information on the plaintext. Following the initial publication of the work in this chapter [109], neural networks were used to detect such patterns [150] but performance can be improved by understanding the precise nature of the ISN field, as will be shown in Section 3.5.3.

### 3.3.7 TCP TIMESTAMP

The TCP timestamp option allows a host to accurately measure the round trip time of a path, and also mitigates problems associated with sequence number wrap-around in links with large bitrate-delay products. For our purposes, it is only necessary to understand the constraints on the values of TCP timestamps; more details about the features based on them can be found in RFC 1323 [82].

The timestamp option consists of two 32 bit fields, TS Value and TS Echo Reply. The TS Value field is set based on the *timestamp clock* of the sender, and it is into this field that hidden data can be embedded. The only constraints on the timestamp clock are that its tick frequency be between 1 Hz and 1 kHz, and that it be strictly monotonic. The TS Echo field is filled with the TS Value from the packet being acknowledged.

A covert channel based on modulating the least significant bit of the TCP timestamps transmitted by a host, `devcc`, is proposed by Giffin *et al.* [63]. The scheme works by incrementing the timestamp associated with a packet (and delaying it accordingly) in order to transmit a “1” bit of ciphertext. The use of TCP timestamps is not universal, but it is deployed as standard on newer versions of Linux and other Unix-like operating systems, so the observation

of timestamps from an operating system which does not support them would be suspicious. As will be described in Section 3.5.3 below, the distribution of values in the timestamp field is modified from the expected one, in a detectable manner, by the use of this covert channel.

### 3.3.8 PACKET ORDER

In addition to the content of the packet, the ordering of packets can be used to carry information, as is described by Ahsan and Kundur [3]. This approach relies on being used on an IPSec network to recover the original order, limiting its applicability. Since packets are seldom reordered by the transmitting host, a warden who is close to Alice will undoubtedly notice the unusually large amount of re-ordering.

### 3.3.9 PACKET TIMING

This chapter will concentrate on packet content (so-called *storage* covert channels), but in a similar manner to the “timing of move” channel in Chapter 2, packet timing can carry information. We do not explore this avenue further, but proposals for such schemes, along with detection mechanisms, are discussed by Cabuk *et al.* [28].

## 3.4 IP ID and TCP ISN implementations

The passive warden considered in this chapter has knowledge of both the TCP/IP standards and particular implementations. He can check whether the values he observes could have been generated by an unmodified operating system, or even by the operating system he knows to be installed on the originating host.

Two fields which are commonly used to embed steganographic data are the IP ID and TCP ISN. It was not possible to find a sufficiently precise description of their generation within the public literature, so details of their implementation are included here. Due to their construction, these fields contain some structure, but as mentioned in Section 3.3, they must also be partially unpredictable. This is achieved by having randomly generated, per-host, secrets and by the use of cryptographic functions. We assume that the warden is aware of the implementation, but does not have access to these secrets and is not able to exploit vulnerabilities in the cryptographic primitives.

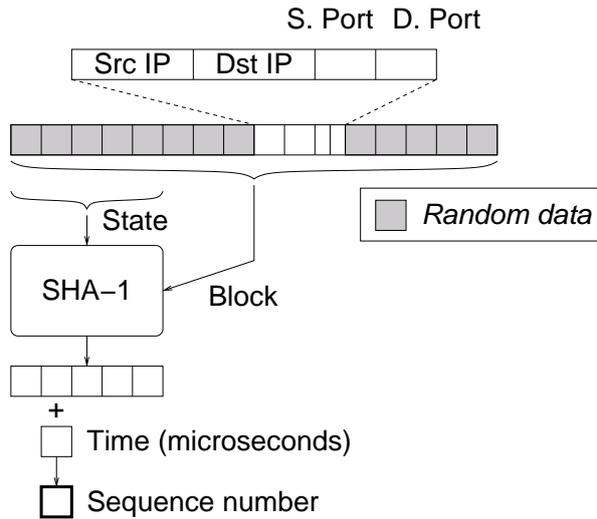


Figure 3.3: Linux 2.0 ISN generator

### 3.4.1 LINUX

The Linux 2.0 ISN generator (shown in Figure 3.3) is based on RFC 1948 [17]. It uses SHA-1 to hash a block of 16 32-bit words, with words 9–11 set to the source and destination IP address and port, and the remaining 13 words filled with a cryptographically secure secret, initialised on boot. Rather than using the values defined in the SHA-1 standard for the initial state, the first 5 words of the block are used. To obtain an ISN, the second word of the hash is selected and the current time (in microseconds) added. This technique achieves the goals of RFC 1948, but calculation of a SHA-1 hash is slow, and hence this algorithm causes a significant delay in connection establishment.

The algorithm used in Linux 2.2 (shown on the left in Figure 3.4) was modified to reduce the time needed to calculate each ISN. Rather than using SHA-1, a reduced block-size variant of MD4 was used, which reads 8 32-bit blocks per iteration, rather than the 16 in the original, and so it also reduces the steps per round from 16 to 8. This is used in a similar way to SHA-1 in Linux 2.0, except it limits the re-use of random data. Since even the security of the full size MD4 algorithm is suspect (it is now known not to be collision-resistant [51]), the random data is rekeyed every 300 seconds (5 minutes) to limit the impact of secret compromise. To avoid this resulting in repeated ISNs, after the hash is calculated, the most significant byte is replaced with a counter incremented on rekeying and initialised to the current time divided by 300 s. Finally, as with Linux 2.0, the time in microseconds is added.

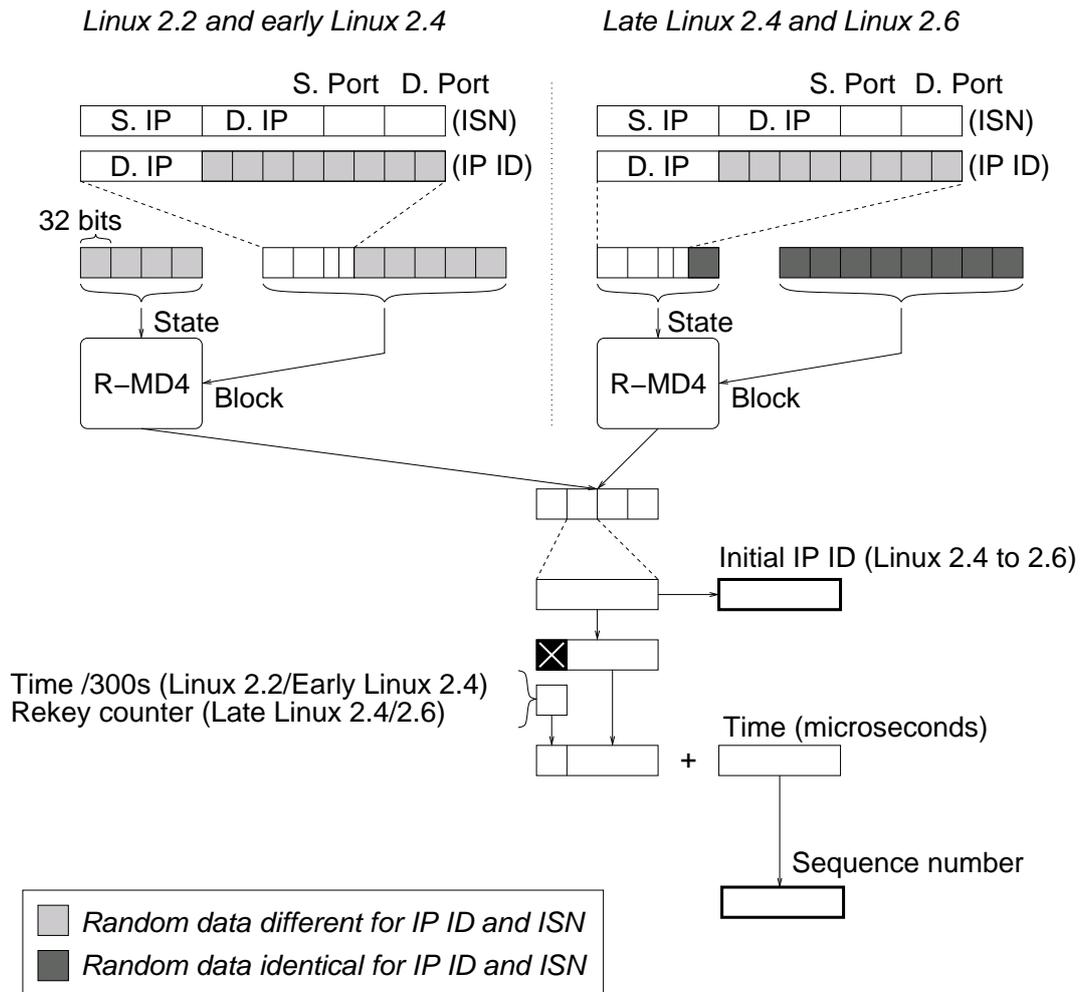


Figure 3.4: Linux 2.2–2.6 ISN generator and Linux 2.4–2.6 IP ID generator

Early versions of Linux 2.4 contained the same ISN generator as Linux 2.2. It was also used (up to the hashing step), with a different secret, to initialise the per-destination counters for IP IDs on packets which may be fragmented. A global counter was previously used, but this was vulnerable to idle scanning. In later versions of Linux 2.4 and Linux 2.6 the algorithm was changed slightly, as shown on the right of Figure 3.4, mainly to improve performance on multiprocessor systems. The difference from a detection perspective is that the rekey counter is initialised to zero on boot. The use of MD4 is changed, and the same secret is used for both ISN and IP ID generation. Exploiting this for detection would require finding a pre-image attack against MD4, or the weaker variant of being able to tell whether two hash outputs were generated

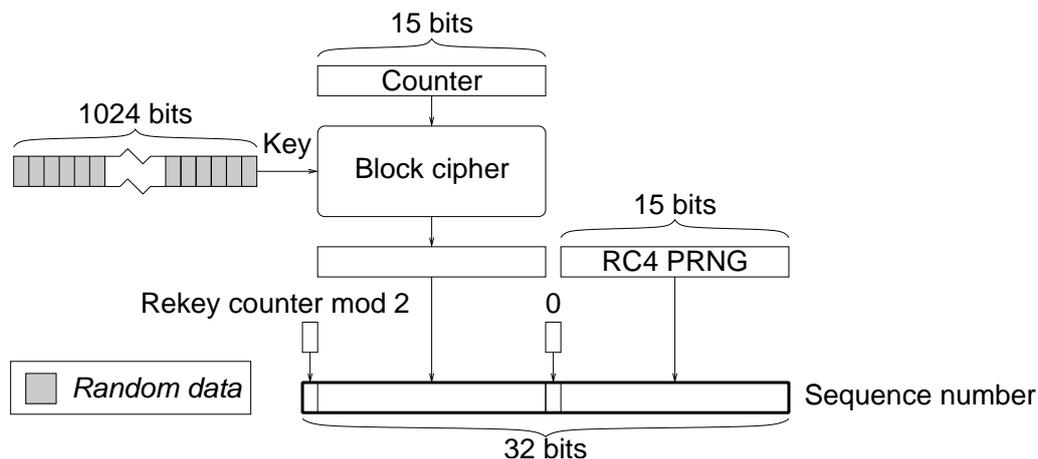


Figure 3.5: OpenBSD ISN generator

by similar, but not identical, inputs. Packets which will not be fragmented, due to the DF bit being set, are assigned a predictable IP ID. For TCP this is a per-socket counter initialised to the sequence number xor-ed with the *jiffy* timer; for UDP a per-socket counter initialised with a timer; while for other protocols, it is set to zero.

### 3.4.2 OPENBSD

The algorithm used for ISN generation in OpenBSD was introduced in December 2000; Figure 3.5 shows its operation. It is initialised by keying a block cipher with 1 024 bits of random data and setting the most significant bit of the generated ISNs to zero. It is rekeyed every 2 hours, or every 30 000 connections, whichever is sooner. On rekeying, the MSB of the generated ISNs is toggled: this prevents collisions between ISNs generated in adjacent rekey intervals. When a new TCP connection is made, the ISN is generated as follows:

- The MSB is set to either “1” or “0”, depending on whether the operating system is in an “odd” or “even” rekey interval.
- The next 15 bits are set to the output of a custom block cipher run in counter mode; the counter is incremented each time an ISN is generated.
- The next bit is *always* zero.
- The final 15 bits are generated by an RC4-based pseudorandom number generator (PRNG).

The result of running the block cipher in counter mode is that a different pseudorandom sequence is defined in each rekey interval. The 15-bit values in this sequence are then inserted into the ISNs, followed by a zero bit: this ensures that no two ISNs within a given rekey interval are closer together than  $2^{15}$  octets. The scheme thus satisfies all of the constraints described in Section 3.3 apart from per tuple pair monotonicity.

The IP ID algorithm in OpenBSD uses a linear congruential generator, described by de Raadt *et al.* [42], rekeyed every 3 minutes (or after 30 000 IDs have been generated, whichever is sooner). It uses the same MSB-toggling mechanism as the sequence number generator to prevent collisions between rekey intervals.

## 3.5 Detection of TCP/IP steganography

As described above, each operating system exhibits well defined characteristics in generated TCP/IP fields. These can be used to identify any anomalies that may indicate the use of steganography. I have therefore outlined a suite of tests which may be applied to network traces and used to identify whether the results are consistent with known operating systems (and in particular, with the operating system believed to be installed on the source host). However, these are not intended as acceptance tests for proposed steganographic schemes, because it is possible to design a steganography scheme which will pass these tests, but still be detectable.

### 3.5.1 IP ID CHARACTERISTICS

- T1. *Sequential Global IP ID.* Some operating systems, particularly older ones (e.g. Linux <2.4), use a global counter for the IP ID. If connections to different hosts have sequentially increasing IP IDs then it is likely that this strategy is in use.
- T2. *Sequential Per-host IP ID.* Others (e.g. Linux  $\geq$ 2.4) use a per-host counter for packets which may be fragmented. The warden can test whether connections to different hosts use apparently unrelated IP IDs, but connections to the same host have a sequentially increasing IP ID.
- T3. *IP ID MSB Toggle.* OpenBSD toggles the most significant bit of the IP ID every rekey interval (3 minutes or 30 000 IP IDs), so the MSB is examined to check if it matches this pattern.

T4. *IP ID Permutation.* Within a rekey interval, the OpenBSD IP ID is non-repeating; the presence of any duplicates eliminates the possibility that this strategy is in use.

### 3.5.2 TCP ISN CHARACTERISTICS

T5. *Rekey Timer.* In Linux 2.2 (and early 2.4) the most significant byte of the ISN is initialised to the system time since the epoch, divided by 300 s. The system time in microseconds is then added. The rekey timer can be recovered by subtracting the system time, in microseconds, from each ISN and verifying that the top byte increases by one every 5 minutes. This requires a clock synchronised to 8 seconds accuracy ( $2^{23}/1\,000\,000$ ), which seems a reasonable assumption, since many systems use NTP synchronisation. The system time can even be queried directly, for example by using the daytime service [126], or indirectly, by observing patterns in the ISNs.

T6. *Rekey Counter.* In Linux 2.6 (and late 2.4) the MSB of the ISN is set to the time since system startup divided by 300 s. The system time in microseconds is added, as before, and hence the rekey counter can be recovered using the same method as in Test 5.

T7. *Zero bit 15.* All ISNs generated by OpenBSD will have bit 15 cleared.

T8. *ISN MSB Toggle.* As with the IP ID, OpenBSD toggles the MSB of the generated ISN every rekey interval (2 hours or 30 000 IP IDs).

T9. *ISN Permutation.* Bits 16 to 30 within OpenBSD ISNs will not repeat within a rekey interval.

T10. *Full TCP Collisions.* In Linux 2.0–2.6, and other RFC 1948-inspired systems, the hash used for ISN generation is based on the address/port tuple pair, so collisions may be encountered. For Linux 2.0 there is no rekeying, so all 32 bits will be identical (subject to clock skew), after subtracting the time. This test and the following one can also be used to estimate clock skew between Alice and the warden and hence identify the physical device without the use of TCP timestamps [90].

T11. *Partial TCP Collisions.* For Linux 2.2–2.6 it would be expected that collisions within a rekey period will have the same least significant 24 bits (subject to clock skew), after subtracting the time.

### 3.5.3 EXPLICIT STEGANOGRAPHY DETECTION

T12. *Nushu Cryptography* As covered in Section 3.3.6, Nushu encrypts data before including it in the ISN field. This will result in a distribution unlike that normally generated by Linux and so will be detected by the other TCP tests. However, due to a flaw in the way that encryption is used, Nushu also exhibits characteristics of its own which may be exploited. The encryption operates by DES encrypting the initialisation vector (IV) (source port  $\oplus$  destination port  $\parallel$  source IP address  $\oplus$  destination IP address) with a shared key, then xor-ing the first 32 bits of the resulting keystream with the hidden data. When IV collisions occur, the ISNs can be xor-ed to remove the key-stream; the result is the xor of two plaintexts. If these plaintexts are the same, as is the case when data is not being sent, the result would be zero, and in other cases redundancy in encoding would be apparent.

T13. *TCP Timestamp* The scheme used in `devcc`, described by Giffin *et al.* [63], can be detected using the methods outlined by Hintz [78]. If a low bandwidth TCP connection is being used to leak information, a randomness test can be applied to the least significant bits of the timestamps in the TCP packets. If “too much” randomness is detected in the LSBs, it can be deduced that a steganographic covert channel is in use.

For a high bandwidth TCP connection (where segment transmission rate  $\gg$  timestamp update rate), a warden can merely calculate the ratio of the number of timestamp values seen to the difference between the start and end timestamp values. If the covert channel described by Giffin *et al.* [63] is in use, this ratio will be close to 2.

T14. *Other Anomalies* Features which would indicate the use of steganography include: unusual flags (e.g. DF when not expected, ToS set), excessive fragmentation, use of IP options, non-zero padding, unexpected TCP options (e.g. timestamps from operating systems which do not generate them) and excessive re-ordering.

### 3.5.4 ACCURACY

Table 3.1 shows which tests detect which operating systems or steganographic techniques. All of these tests (except Test 13) are based directly on the original

Software	Tests													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Linux 2.0	•									•				
Linux 2.2	•				•						•			
Early Linux 2.4		•			•						•			
Late Linux 2.4/2.6		•				•					•			
OpenBSD			•	•			•	•	•					
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Covert_TCP														
Nushu		•										•		
devcc		•			•						•		•	•
Lathra/Linux		•				•					•			
Lathra/OpenBSD			•	•			•	•	•					

Table 3.1: Expected results of tests on unmodified operating systems and TCP/IP steganography systems. A matching test is indicated by “•”. The last three columns are tests for the presence of steganography, the others test for the absence. Nushu and `devcc` were written for early Linux 2.4 and are assumed to share the characteristics of all fields which are not explicitly modified. `Covert_TCP` creates all fields itself. Our improved TCP/IP steganography scheme, Lathra, is described in Section 3.6

implementations, and make no assumptions about probabilistic effects. Hence, they will not suffer from false negatives. False positives are possible, so in this section we consider the number of packets required to avoid these.

*IP ID.* Test 1 will reach an error probability of  $1/2^{16}$  after only 2 packets, as will Test 2 for two fragmentable packets directed to the same host within a rekey interval. Due to the prevalence of path MTU discovery, fragmentable packets are rare, however this test will still be effective in the normal case, where sockets are used to send several packets, because of the per-socket IP ID counters used in TCP and UDP. The probability of error in Test 3 halves with every packet after the first one is observed. From the “birthday paradox”, after around 181 packets ( $2^{N/2}$ , where  $N$  is 15, the number of bits in the PRNG output) a collision would be expected which would match Test 4.

*TCP ISN.* Test 5 needs one packet to achieve a  $1/2^8$  error probability; Test 6 needs 2 packets to get the same. Test 7 halves the error probability with every SYN packet, as does Test 8 after the first packet. As with the equivalent IP ID check, Test 9 needs around 181 SYN packets within a rekey interval. Tests 10

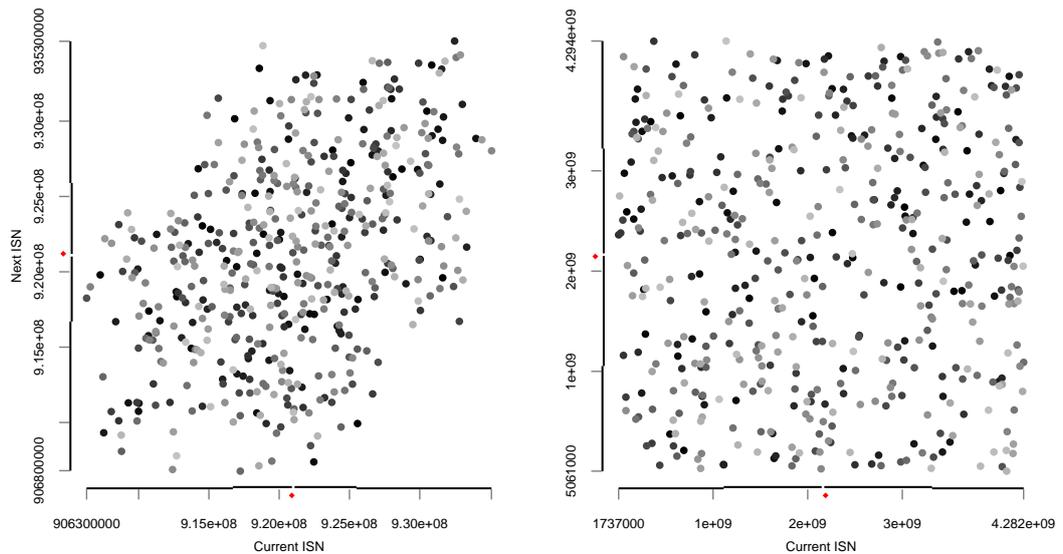


Figure 3.6: Correlation of consecutive ISNs. Packets generated by Linux are shown on the left and Nushu on the right. Note the differing scales and distribution. With Linux, large changes do not occur whereas with Nushu, ISNs are uniformly distributed over the full range. The colour of points indicates packet ordinal (dark to light), and as can be seen there is no obvious relationship. Marginal first quartile, median and third quartile are shown by breaks on the axes; the mean is indicated by a  $\blacklozenge$

and 11 depend on the randomness of the source port selection, but on a heavily loaded machine, my experiments show these collisions occur approximately every 1000 SYN packets for a fixed destination port.

*Steganography.* Test 12 also depends on port selection randomness, but my experiments show collisions every 1000 SYN packets (even with random destination ports). Test 13 depends on the expected communication speed; accuracy will improve the more the capacity exceeds two packets per timestamp. The accuracy of Test 14 depends on the steganography being used, but for naïve implementations only one packet is needed.

### 3.5.5 RESULTS

A few of the tests described above merit further explanation. Tests 5 and 6 show that in Linux, the most significant byte of ISNs in consecutive packets will differ by at most 1, assuming at least one packet is sent per rekey. In contrast, Nushu and random ISN generation should show no correlation between consecutive packets. This is illustrated in Figure 3.6.

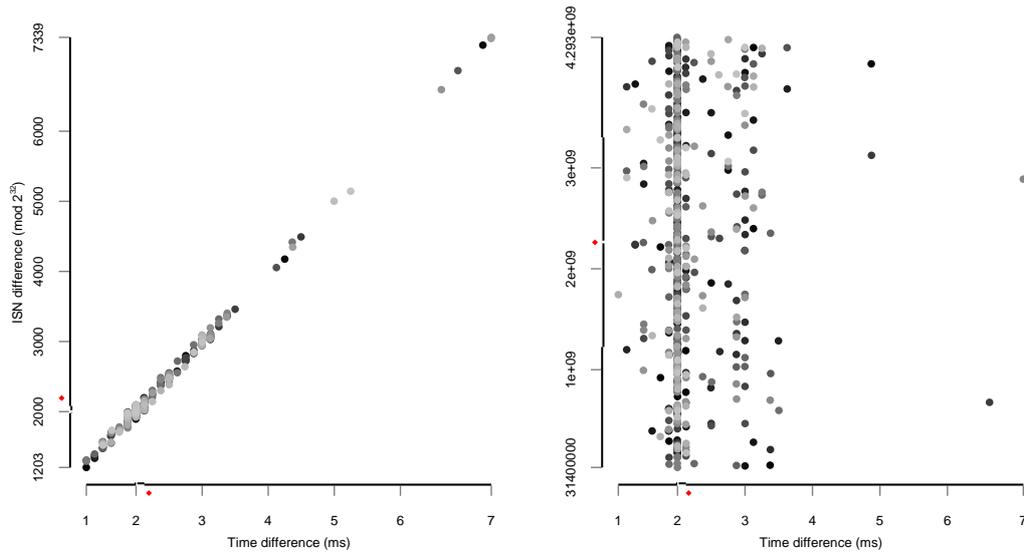


Figure 3.7: Time dependence of consecutive ISNs. Linux is shown on the left and random ISNs, with the same inter-packet timing, is on the right. The packets were sent approximately every 2 milliseconds, hence the clustering on the  $x$  axis around this point. The ISN difference is clearly correlated with inter-packet sending time for Linux, but uniformly distributed with random ISNs. Again point colour indicates packet ordinal

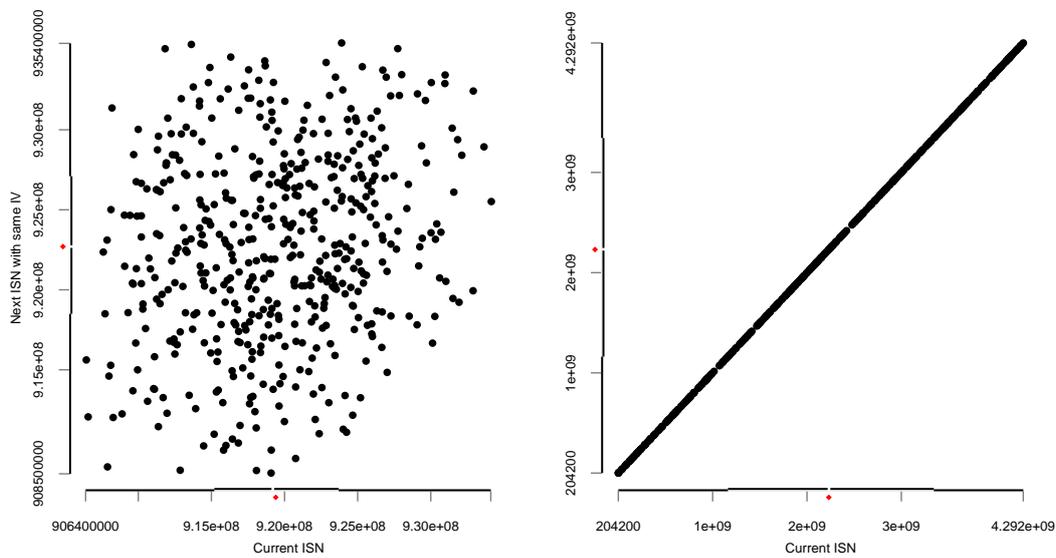


Figure 3.8: With Linux, shown on the left, there is no significant correlation between pairs of packets with the same Nushu ISN, other than that expected from the similar most significant byte. In contrast, for such packets, the Nushu generated ISN, shown on the right, is identical, assuming a constant payload

Test 11 implies that for Linux 2.2–2.6, during one rekey period, the difference between any pair of ISNs generated for the same source/destination port/IP address will be the sending time difference in microseconds. Figure 3.7 shows how Linux ISNs can be differentiated from random ISNs by the application of this technique.

Finally, Figure 3.8 shows the characteristic patterns of Nushu compared to unmodified Linux. It is similar to Figure 3.6, except rather than comparing consecutive packets, it compares pairs of packets for which the Nushu initialisation vector would be identical.

## 3.6 Detection-resistant TCP steganography

Our aim in this section is to develop a robust scheme, *Lathra*, using the TCP ISNs generated by OpenBSD and Linux as a steganographic carrier. We achieve this by following the implementation of the real ISN generators, so OpenBSD and Linux are discussed as separate cases. This discussion only considers the content of the steganographic channel, but to be protected from timing attacks, care must also be taken to ensure that the generation algorithms have the same response time as the original implementations.

### 3.6.1 LINUX

Simply encoding our data within the least significant 24 bits of the ISN could be detected by the warden, because he can detect the absence of a uniform distribution after subtraction of the time. Our generated ISN thus needs to have the carry into the top byte set correctly. We must add one to the top byte of our generated ISN if, and only if, the value that we place into the bottom 24 bits (our encrypted data) is strictly less than the least significant 24 bits of the time.

To ensure reliability, Alice can check if she receives an ACK for each ISN and resend if not, then use an internal sequence number to recover the original order. The data must be encrypted, so that it cannot be differentiated from random numbers. To prevent duplicated plaintexts being apparent, while allowing each packet to be decrypted independently, the plaintext is xor-ed with a hash of other header fields (not including the IP ID, which is derived from the ISN), then encrypted with a variable length block cipher. Half of the space of plaintexts is reserved specifically to avoid duplications in the space of ciphertext: if the same data is to be sent, and the hash of other header fields is

the same, one of these dummy values is encoded and transmitted. Due to the RFC 1948 based design, if Alice encounters a packet with the same source and destination IP address and port as one already used, within a rekey interval, it must be skipped.

### 3.6.2 OPENBSD

The most significant bit of our output must exactly mimic the output of the real OpenBSD TCP stack: it must toggle every 2 hours (or 30 000 connections). The next 15 bits, when extracted in turn from each ISN generated within a rekey interval, must resemble a pseudorandom sequence. An algorithm that conforms to these requirements was developed by Stephen Lewis and for completeness it is shown in Figure 3.9.

These functions encode (and decode) the integer  $n$  as a permutation of the sequence  $(0, 1, 2, \dots, m - 1)$ , with  $x$ -times redundancy (i.e., Bob only needs to receive one in  $x$  ISNs transmitted by Alice). In order to remove patterns in our permutation, Alice must choose a key,  $K$ , for each rekey interval, and transmit it to Bob in, for example, the least significant 15 bits of the first ISN.

By using these functions we can encode the 16 MSBs of the ISN. The 17th bit must be zero, and we encode data into the remaining 15 bits using a block cipher. The output of the cipher is analogous to the stream cipher in the genuine implementation. We xor a hash of other header fields with the input data to hide duplicated plaintexts. If the hash yields the same value more than once, we skip that packet.

## 3.7 Conclusion

This chapter has provided an overview of the opportunities for using TCP/IP header fields as a carrier for a steganographic covert channel. A detailed description of the ISN and IP ID generation schemes in Linux and OpenBSD were presented, and a number of previously proposed schemes for TCP/IP-based steganography were described.

We have shown that a passive warden can detect the use of these schemes because the modified headers that they produce can easily be distinguished from those generated by a genuine TCP/IP stack.

Finally, we have outlined two schemes for encoding data with ISNs generated by OpenBSD and Linux. Both schemes generate ISNs that are almost indistinguishable from those generated by a genuine TCP stack, except by war-

dens with knowledge of a shared secret key or who can exploit vulnerabilities in the underlying cryptography used in Lathra or the original ISN generation algorithms. In particular, for the Lathra/Linux case, we assume that the warden cannot tell that two adjacent sequence numbers could not have been generated by an instance of MD4 with the same partial input. In Lathra/OpenBSD, we make a similar assumption about the counter mode output of the block cipher and the use of RC4.

```

PERMUTATION-CODE( $m, n, x$ )
1   $base \leftarrow m$ 
2   $output\_symbols \leftarrow (0, 1, 2, \dots, m - 1)$ 
3  while  $n \neq 0$ 
4  do  $index \leftarrow n \bmod base$ 
5      $n \leftarrow \lfloor n/base \rfloor$ 
6     for  $i \leftarrow 0$  to  $x - 1$ 
7     do output ENCIPHER( $output\_symbols[index] + i \times m, K$ )
8      $output\_symbols \leftarrow output\_symbols \setminus output\_symbols[index]$ 
9      $base \leftarrow base - 1$ 

```

```

PERMUTATION-DECODE( $m, x$ )
1   $base \leftarrow m$ 
2   $multiplicand = 1$ 
3   $input\_symbols \leftarrow (0, 1, 2, \dots, m - 1)$ 
4   $n \leftarrow 0$ 
5  while input  $symbol$ 
6  do  $symbol \leftarrow$  DECIPHER( $symbol, K$ )
7      $symbol \leftarrow symbol \bmod m$ 
8     if seen  $symbol$ 
9     then skip
10     $n \leftarrow n +$  INDEX-OF( $symbol$  in  $input\_symbols$ )  $\times multiplicand$ 
11     $input\_symbols \leftarrow input\_symbols \setminus symbol$ 
12     $multiplicand \leftarrow multiplicand \times base$ 
13     $base \leftarrow base - 1$ 
14 return  $n$ 

```

Figure 3.9: OpenBSD permutation coding and decoding functions

# Chapter 4

## Low-cost traffic analysis of Tor

Tor [50] is the second generation Onion Router [130], supporting the anonymous transport of TCP streams over the Internet. Its low latency makes it appropriate for common tasks, such as web browsing, but vulnerable to traffic-analysis attacks by a global passive adversary. This chapter presents new traffic-analysis techniques, based on covert channels and side channels, that allow adversaries with only a partial view of the network to infer which nodes are being used to relay anonymous streams and therefore greatly reduce the anonymity provided by Tor. Furthermore, we show that otherwise unrelated streams through Tor can be linked back to the same initiator. The attack is feasible for the adversary anticipated by the Tor designers. The theoretical attacks are backed up by experiments performed on the deployed Tor network, and should also be applicable to any low-latency anonymisation network. These attacks highlight the relationship between the field of traffic analysis and more traditional computer-security issues, such as covert-channel analysis of multilevel secure systems. This research also highlights that the inability to directly observe network links does not prevent an attacker from performing traffic analysis: the adversary can use the anonymising network as an oracle to infer the traffic load on remote nodes.

### 4.1 Introduction

Since the development of Chaum's mix [31], mentioned in Chapter 1, several email anonymity systems have been based on this architecture. Most notably, these include Babel by Gülcü and Tsudik [72], Mixmaster by Möller *et al.* [116] and the newer Mixminion by Danezis *et al.* [41]. Their latency is tolerable

for email, but is unsuitable for interactive applications such as web browsing. Other systems, based on the idea of a mix, were developed to carry low-latency traffic. ISDN mixes [120] propose a design that allows phone conversations to be anonymised, and the Java Anon Proxy (JAP) [19] follows the same design pattern to anonymise web traffic.

The Onion Routing project [30] has been working on stream-level, low-latency, high-bandwidth anonymous communications. Their latest design and implementation, Tor, has many attractive features, including forward secrecy and support for anonymous servers. These features, and Tor's ease of use, have already made it popular, and a testing network, available for public use, has around 1 000 nodes acting as Onion Routers (as of July 2007).

While anonymous email systems are designed to resist attack despite all communication links being monitored, Tor aims to protect the anonymity of its users from non-global adversaries. This means that the adversary has the ability to observe and control some part of the network, but not its totality. Similarly, the adversary is assumed to be capable of controlling some fraction of Tor nodes. By making these assumptions, the designers of Tor believe it is safe to employ only minimal mixing of the stream cells that are relayed, thereby lowering the latency overhead of the communication.

This choice of threat model, with its limitation of the adversaries' powers, has been a subject of controversy in the anonymity community, yet most of the discussion has focused on assessing whether these restrictions on attackers' capabilities are realistic. We leave this discussion aside and instead show that traffic-analysis attacks can be successfully mounted against Tor even within this very restricted threat model.

Our attacks are based on the notion that the timing signature of an anonymised stream can be used to track it in the Tor network, since the low latency does not significantly distort it. As the adversary is not global, he cannot observe timing signatures directly in the network. Instead, the adversary sends his own traffic streams through a node and monitors the latency. He uses the fact that the traffic volume of one stream influences the latency of other streams carried by the same Tor node. To assess if a target stream is carried over a Tor node, the adversary routes a stream that he can observe, through the same node, and measures the changes in latency. Besides tracing the route of an anonymous communication, our traffic-analysis attacks can also link transactions together. Our techniques allow any user to perform traffic analysis on the whole network, and thereby approximate a global passive observer.

The results presented should be seriously considered by designers of anonymous communication systems. They concern a widely deployed, popular, and well used system that represents the state of the art in both research and implementation. Few systems of such a standard have been deployed (Freedom [13, 24, 66] and JAP [19] being the others), which has made practical experimentation, to verify the effectiveness of theoretical attacks, very difficult. These attacks highlight an architectural flaw that leads to information leakage, and this could affect other designs of anonymity systems. The parallels that this problem has with covert channels in multilevel security links the field of anonymous communications with more traditional computer-security disciplines. The approach of performing covert-channel analysis to assess the security of an anonymous communication system was pioneered by Moskowitz *et al.* [104, 105, 114]. This chapter, and the thesis as a whole, illustrates that theirs is not simply a theoretical model, but that techniques from the covert channel community can be effective, in practice, in degrading the security provided by real anonymous communication systems.

Despite the link to covert channels in MLS, there is a fundamental difference: anonymous communication relies on traffic from many different sources being mixed together. Therefore, the established solution to covert channels, of separating the different streams to avoid timing information leakage, would eliminate the anonymity properties of the system. For this reason, novel techniques must be devised to cope with our attacks.

## 4.2 Understanding Tor

The Onion Router (Tor) [50] is the culmination of many years of research by the Onion Routing project [67, 130, 147]. Not only is it a completely new design and implementation, but it reflects a shift from the traditional threat models anonymous communication systems have tried to protect against. We first describe the Tor architecture and then introduce the threat model considered in the Tor design.

### 4.2.1 ARCHITECTURE

The Tor network can be used to transport TCP streams anonymously. It is composed of a set of nodes that act as relays for a number of communication streams, hopefully from different users. Each Tor node tries to ensure that the correspondence between incoming data streams and outgoing data streams is

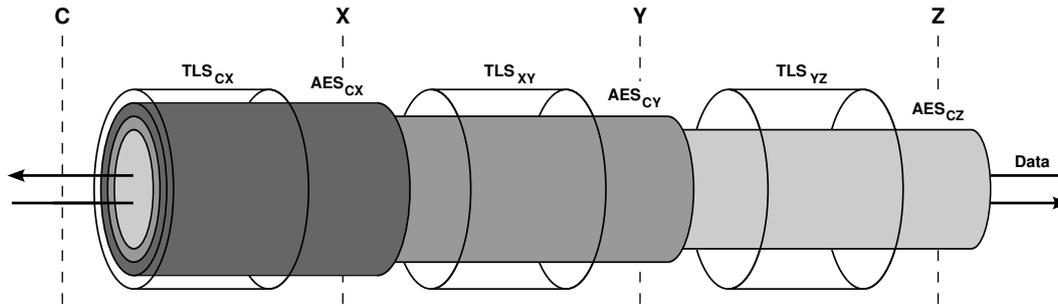


Figure 4.1: Encryption performed and key usage for a Tor circuit, going from the Onion Proxy C, through three Onion Routers: X, Y and Z.  $TLS_{ab}$  indicates the TLS tunnel secured by keys shared between  $a$  and  $b$ .  $AES_{ab}$  indicates the CTR mode encryption performed using a key shared between  $a$  and  $b$

obscured from observers. Therefore, an attacker cannot be sure about which of the originating user streams corresponds to an observed output of the network.

The Tor architecture is similar to circuit-switched networks. The connection establishment, shown in Figure 4.1, has been carefully crafted to preserve anonymity, by not allowing observers to cryptographically link or trace the route that the connection is using. The initiator of the stream creates a *circuit* by first connecting to a randomly selected Tor node, negotiating secret keys and establishing a secure channel with it. The key establishment uses self-signed ephemeral Diffie-Hellman key exchange [46], and Transport Layer Security (TLS) [45] is further used to protect the connections between nodes and provide forward secrecy.

All communications are then tunnelled through this circuit, and the initiator can connect to further Tor nodes, exchange keys and protect the communication through multiple layers of encryption. Each layer is decoded by a Tor node and the data is forwarded to the next Onion Router using standard route labelling techniques.

Finally, after a number of Tor nodes are relaying the circuit (by default three), the initiator can ask the last Tor node on the path, the *exit node*, to connect to a particular TCP port at a remote IP address or domain name. Application layer data, such as HTTP requests or SSH sessions, can then be passed along the circuit as usual. Since we are not attacking the cryptographic components of Tor, we will not go into any further details on this subject. Interested readers should consult the Tor specification [47].

TCP streams travelling through Tor are divided and packaged into cells. Each cell is 512 bytes long but, to cut down on latency, it can contain a shorter

useful payload. This is essential for supporting interactive protocols, such as SSH, that send very small keystroke messages.

Controversially, Tor does not perform any explicit mixing. Cells are stored in separate buffers for each stream, and are output in a round-robin fashion, going round the connection buffers. This ensures that all connections are relayed fairly, and is a common strategy for providing best effort service. Importantly, when a connection buffer is empty, it is skipped, and a cell from the next non-empty connection buffer is sent as expected.

Since one of the objectives of Tor is to provide low-latency communications, cells are not explicitly delayed, reordered, batched or dropped, beyond the strategy described above.

Tor has some provisions for fairness, rate limiting and traffic congestion avoidance at particular nodes. Tor implements a *token bucket* strategy to make sure that long-term traffic volumes are kept below a specified limit set by each Tor node operator. Since the current deployment model relies on volunteer operators, this was considered important. Yet this approach, on its own, would not prevent spikes of traffic from being sent, and propagating through a connection. These spikes of data are subject to the maximum bandwidth of each node, and can saturate the network connection of some Tor nodes.

To avoid such congestion, a second mechanism is implemented. Each stream has two windows associated with it, the first describes how many cells are to be received by the initiator, while the other describes how many are allowed to be sent out to the network. If too many cells are in transit through the network – and have not already been accepted by the final destination – the Tor node stops accepting any further cells until the congestion is eased.

It is important to note that this mechanism ensures that the sender does not send more than the receiver is ready to accept, thereby overfilling the buffers at intermediary Tor nodes. It also makes sure that each connection can only have a certain number of cells in flight without acknowledgement, thus preventing hosts from flooding the network. Tor does not, however, artificially limit the rate of cells flowing in any other way.

Finally, it is worth mentioning that each Tor circuit can be used to relay many TCP streams, all originating from the same initiator. This is a useful feature to limit the public-key cryptography overhead when using protocols such as HTTP, that might need many connections, even to different hosts, as part of a single transaction. Tor circuits that have been used, but which have become idle, are short-lived – replacements are set up every few minutes.

This involves picking a new route through the Tor network, performing the key exchanges and setting up the encrypted tunnels.

#### 4.2.2 THREAT MODEL

We consider an adversary whose principal objective in attacking an anonymous communication system is to link the initiators of connections with their respective communication partners and vice versa. For example, an adversary observing a web request coming out of the Tor network might be interested in determining its originator. Similarly, an attacker observing a connection into the Tor network would be interested in knowing which remote machine it is ultimately accessing. A secondary objective of the attacker is to link transactions, namely network connections, so as to establish that they are from the same initiator. This could allow an adversary to profile the initiator, by observing patterns in his communication habits.

Tor aims to protect against a threat model that is unusual within the anonymous communications community. It is conventional to attempt to guarantee the anonymity of users against a global passive adversary, who has the ability to observe all network links. It is also customary to assume that transiting network messages can be injected, deleted or modified and that the attacker controls a subset of the network nodes. This models a very powerful adversary, hence systems that protect against it can be assumed to be secure in a very wide range of real-world conditions.

In contrast, Tor, like some other designs, most notably MorphMix [133] and Tarzan [57, 58], assumes a much weaker threat model. It protects against a *non-global* adversary that can only observe a fraction of the network, modify the traffic only on this fraction and control only a fraction of the Tor nodes. Furthermore, Tor does not attempt to protect against *traffic confirmation attacks*, where an adversary observes two parties that he suspects to be communicating with each other, to either confirm or reject this suspicion. Instead, Tor aims to make it difficult for an adversary with a very poor *a priori* suspicion of who is communicating with whom, to gain more information.

It could be claimed that the weaker threat model makes Tor insecure and incapable of protecting the anonymity of users against powerful real-world adversaries. In particular, while real-world adversaries are not omnipotent they do have the ability to be *adaptive* and select where to monitor the network based on previous observations. This monitoring can be performed on deployed TCP/IP or telephone networks using the lawful interception capabilities inte-

grated into most modern routing equipment [160]. Access to these capabilities should be restricted only to authorised parties with legal permission (although recent revelations [26, 127, 143] have shown that this may not always be true). Prior work by Murdoch and Zieliński [111] demonstrated that even sampled traffic logs are sufficient to perform traffic analysis.

The importance of our attacks is that an adversary can extract information about the path of a Tor connection without stepping outside the threat model considered by Tor, and the methods used are accessible to any Tor user. Therefore, we show that even relatively weak adversaries can perform traffic analysis, and get vital information out of Tor. This means that even non-law-enforcement agencies can significantly degrade the quality of anonymity that Tor provides, to the level of protection provided by a collection of simple proxy servers, or even below.

## 4.3 Attacking Tor

An attacker aims to gain some information about who is communicating with whom through the Tor network. This section will present an overview of the techniques that an attacker can use to trace communications and the constraints introduced by the restrictive Tor threat model. These lead to the theoretical exposition of our attacks, the practical results of which are presented in Section 4.4.

### 4.3.1 TRADITIONAL TRAFFIC ANALYSIS

Traffic analysis is extracting and inferring information from network metadata, including the volumes and timing of network packets, as well as the visible network addresses they are originating from and destined for. In the case of anonymous communications, an adversary would use this data to perform traffic analysis with the aim of tracing who the originator or destination of a connection is – therefore violating the anonymity properties that the system is designed to provide. We assume that Tor intermediaries, through the use of encrypted tunnels, effectively hide the bit patterns of data travelling through a Tor connection. Therefore, an attacker cannot use any information from the content to trace the stream and must resort to traffic analysis.

Traffic analysis can be performed at different levels of granularity. The first class of attacks treats the anonymous network as a “black box” and only considers the times when users are initiating connections, and connections

are being relayed to network services outside the Tor network. Kesdogan *et al.* [88] were the first to show how repeated communications would eventually be revealed even if the anonymous channel was otherwise perfect. A statistical variant of these attacks, presented by Danezis [37], and validated through simulations by Mathewson and Dingledine [97], is more general and can be applied to a wider variety of anonymous communication channels.

Both these attack families are very powerful and would uncover repeated patterns of communication through Tor. For example, the disclosure and statistical disclosure attacks could, in the long run, reveal if a particular user connects, every day, to a set of web sites through Tor. An analysis of how long this would take can be found in Mathewson *et al.* [97] and Agrawal *et al.* [2]. Yet, to effectively mount such attacks, an adversary is required to observe a large fraction of the network in order to log who is accessing it and which outside services are used. This attacker is outside the threat model that Tor tries to protect against and therefore cannot be considered to break Tor<sup>1</sup>.

A second category of attacks work at a much finer granularity. They inspect the traffic within the anonymous communication network, and further, the actual shape (load) of the traffic on each network link. Earlier work by the Onion Routing project drew attention to the fact that overall traffic patterns in connections are not particularly distorted by each Onion Router that relays them [147]. Therefore, a global observer would be able to correlate the timing and volume of incoming and outgoing streams in order to trace the route an onion-routed connection is taking through the network.

Danezis [40] presents these finer granularity attacks in detail, and a theoretical framework is developed to assess their effectiveness. In practice, an attacker observes a stream of data that is to be traced, for example, the reply of a web server to the initiator of a request. This stream of data can be represented as a function of traffic volume over time. The function is convolved with an decay function that matches the delay characteristic of the mix (in the example given, exponential): the result is a *template* that predicts what the stream will like look in the anonymity network. All links of the network

---

<sup>1</sup>How realistic these attacks are is a completely different subject, that requires careful consideration of the size and topology of the anonymous communication network. In the case of Tor, a fully-connected network, an attacker would have to be able to know all the meta-data associated with the TCP connections to and from all Tor nodes. Given their small number (as of July 2007, approximately 1000) this might not be such a large effort. In the case of JAP [19], which arranges all relays in a cascade, only two nodes have to be under surveillance when applying disclosure or statistical disclosure attacks.

are then compared to assess if they match (more precisely, could have been generated by) the target template. Each network link will have a degree of similarity to the template that can be used to classify it as being after the first, second or third node on the path of the connection. Similar attacks have also been presented in Fu *et al.* [162] and Levine *et al.* [94].

The obvious problem of these attacks is that, as presented, the adversary observes all nodes, network links and is able to record traffic meta-data at a much finer granularity than required for the disclosure attacks above. As a result, these attacks assume a global passive adversary, which is not considered within the Tor threat model. At the same time, it is important to highlight that these attacks are robust [97]: when less, partial or lower resolution data is available they will take longer, and require more evidence until they provide the attacker with the same degree of certainty, but in the long run they will still work. Therefore, an attacker who controls part of the network, as Tor assumes, might still be able to trace some communications at random. However, this is not very appealing to an attacker because of the amount of interception effort and analysis required.

A further relevant attack has been presented by Serjantov and Sewell [141]. They notice that by doing simple packet counting on lone connections, they can follow the anonymised streams. Their attack is appealing, since packet counting can be performed easily and cheaply by most routing equipment. Others have also looked at detecting *stepping stones* (relays) for intrusion detection [23, 161], using similar techniques.

#### 4.3.2 TRAFFIC ANALYSIS OF TOR

As we have seen, traditional traffic-analysis techniques rely on privileged access to vast amounts of data. The conventional wisdom has been that such data can only be gathered by a global passive adversary, which lies outside the Tor threat model. The key contribution of this chapter is the realisation that such observation capabilities are not necessary to perform these attacks. The ability to route over the anonymous communication network, an unrestricted privilege, can be used to estimate the traffic load on specific Tor nodes accurately enough to perform traffic analysis. Therefore, adversaries with very modest capabilities can still detect the path that target connections are taking through the Tor network.

Mix systems rely on the fact that actions, be it relayed messages, stream cells or connection startups, from different users, are processed by one party,

the mix, in such a way that they become unlinkable to their initiator. In Tor, multiple connections from different users have to be relayed by the same node for any of them to be provided with any anonymity at all<sup>2</sup>. Since the relayed streams are processed and transported over a particular Tor node, they interfere with each other. This is because they consume shared resources on a single machine – such as processor time and network bandwidth.

Some mixing strategies try to entangle streams in order to make them indistinguishable. The best example is the threshold-mix batching strategy that waits until a particular number of messages have arrived and outputs them all at once. Tor does not use any particular batching strategy, since it would increase the latency of the communication. Instead, cells from different streams are sent out in a round robin fashion. When a stream has no cells available, it is skipped and the next connection with cells, waiting to be delivered, is serviced. This means that the load on the Tor node affects the latency of all connection streams that are routed through this node. A similar increase in latency is introduced at all layers of the protocol stack. As expected, the higher the load on the node, the higher the latency.

The simple observation that higher load, even due to one extra connection, on a Tor node will result in higher latency of all other connections routed through it can be used by an attacker. By routing a connection through specific Tor nodes, and measuring the latency of the messages, the adversary can get an estimate of the traffic load on the Tor node, that is, the superposition of the traffic load resulting from all relayed connections. This, in turn, can be compared with a known traffic pattern to assess if it is present, and therefore relayed through the node, using conventional traffic-analysis techniques [40].

Any Tor user can perform these measurements and try to estimate the load on a Tor server. On the other hand, a Tor node is not subject to some restrictions that apply to clients (e.g. bandwidth limits), therefore for generality we consider that the attacker controls a corrupt Tor node. This is in accordance with the Tor threat model, and allows us to ignore whether the node to be measured is also an exit node or not. This corrupt Tor node creates a connection that passes through another Tor node, whose traffic load is to be measured. This connection is then filled with *probe traffic*, that measures the latency of the connection and therefore estimates the load on the target Tor

---

<sup>2</sup>Acquisti *et al.* [1] go as far as claiming that a multitude of users that do not trust each other have incentives to share the same anonymous network since their traffic is then all mixed together.

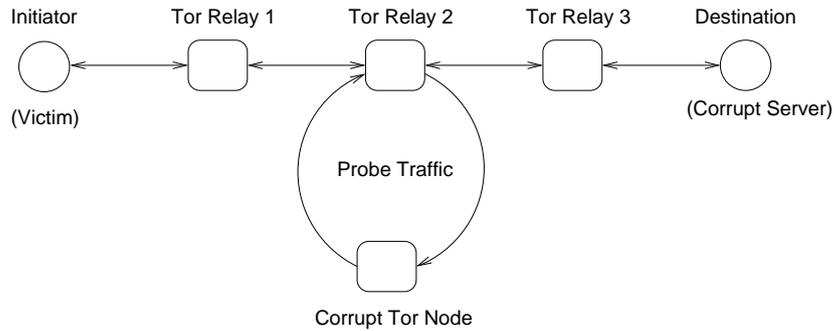


Figure 4.2: The attack setup

node. This should allow the detection of transient traffic signals propagating through the measured Tor node.

The adversary could observe a connection to or from the Tor network and use the technique outlined above to detect which nodes it is being relayed through. We propose a more powerful variant of this attack: we assume that the adversary controls a network server that the user to be traced is accessing. This falls within the Tor threat model, and to some extent is its *raison d'être*: users should be able to anonymously access network services that might be interested in identifying them. This corrupt server sends to the user, through the Tor connection, data modulated in a very specific traffic pattern. In my experiments, I have used a pattern that consists of sequences of short (a few seconds) bursts of data. Since the attacker knows the input pattern to the Tor network, he can construct a template, and use it to detect whether the traffic volume in Tor nodes is correlated with it.

Figure 4.2 illustrates the setup necessary for the attacks. In the next section we will present the results of setting up and performing such an attack against the operational Tor network.

#### 4.3.3 TRAFFIC-ANALYSIS METHODOLOGY

The goal of an attacker is, based on timing data from all nodes on the network, to identify which nodes are carrying the traffic with the pattern injected by the corrupt server. For each node, I performed a test where the stream from the corrupt server went through the target node, and one where the stream did not. For the test to be considered a success, the correlation between the traffic modulation and probe latency in the case where the victim stream did go through the target node should be higher than the case where it did not. Otherwise either the victim stream did not sufficiently affect the probe traffic

(causing false negatives), or “echos” of the victim stream propagated through the network and affected the probe stream (causing false positives).

The correlation performed was very simple: the template formed by the modulated traffic from the corrupt server was multiplied with the probe data and the sum of the result was evaluated. More specifically, the template function from the corrupt server  $S(t)$  is:

$$S(t) = \begin{cases} 1 & \text{if server is sending at sample time } t \\ 0 & \text{otherwise} \end{cases}$$

The data from the probe is expressed as  $L(t)$ , which is the measured latency of the target Tor node (in microseconds) at sample time  $t$ .  $L'(t)$  is the normalised version of the probe data, formed by dividing  $L(t)$  by the mean of all samples.

The correlation  $c$  is the sum of the product between  $S(t)$  and  $L'(t)$ , divided by the number of samples where the corrupt server was sending:

$$c = \frac{\sum S(t) \times L'(t)}{\sum S(t)}$$

A number of improvements could be made to this technique, by using better models of the effect of load on latency. One obvious addition is to shift the template in time by an estimate of the latency, or to convolve it with an exponential-decay function. Also, quantities other than simple latency could be used, such as a moving variance measure. I have had satisfactory results with the simple technique, and so I leave the design of efficient and optimal transient signal detectors for traffic analysis as future work.

## 4.4 Experimental setup and results

In order to validate the feasibility of the traffic-analysis attacks described in the previous section, I built and evaluated a simple version of the approach. The probe computer used was a standard 800 MHz PC running the Debian GNU/Linux 3.0 operating-system. Tor version 0.0.9 was set up as being a client only (in Tor terminology, an *Onion Proxy*) and modified to choose routes of length one (not including itself), rather than the default of three. In addition to the modified Tor software, the corrupt Tor node consisted of a TCP client and server, both written in C and carefully crafted to avoid the timing properties being interfered with by runtime services such as garbage collection. The interface between the TCP client and the Onion Proxy is achieved using

`socat` [135] to connect to the SOCKS interface of Tor. The targeted Tor node then connected back to the TCP server running on the same machine.

At regular intervals (in my experiment, every 0.2 seconds) the probe client sent data containing the current system time in microseconds (as reported by `gettimeofday()`) and optional padding. The TCP socket used was configured with the `TCP_NODELAY` option to disable the Nagle algorithm, ensuring that the data was sent immediately. Also, in the TCP stream establishment and in each segment sent, I added a nonce value, to distinguish it from port scans and prevent other Internet “background radiation” from interfering with the results. The probe server recorded the time the segment was sent, and also when the segment was received, then saved both to a file. While this approach limits us to only probing Tor nodes that allow outgoing TCP streams (*exit nodes*), it could be generalised to all nodes if the attacker controlled a Tor server, even one which had not been vetted by the Tor network operators.

The corrupt server was simulated by a TCP server which would send pseudorandomly generated data at as fast a rate as allowed by Tor, for a pseudorandom time period (in our experiment between 10 and 25 seconds), then stop sending for another period (between 30 and 75 seconds). The times at which it stopped and started sending were stored in a file for later analysis. The victim was simulated by a TCP client which would receive data and record the time at which each buffer of data was received. These records were used to evaluate how much the timing signature of the data was being distorted by Tor, however this data would not be available to an attacker and so was not used in correlation calculations.

The Tor Onion Proxy on the victim node was unmodified since it would not be controlled by the attacker. Again, `socat` was used for the interface between the victim client and Tor. The non time-critical parts of the experiment, such as the starting and stopping of programs and the collection of results from the remote machines, were written in Python. The probe server was hosted in the University of Cambridge Computer Laboratory. The victim and corrupt server were run on PlanetLab [32] nodes in two separate US institutions. The full layout of our system is shown in Figure 4.2.

In each experimental run, targeting nodes in turn, the procedure was as follows: the probe server would be set to monitor the target node, then after four minutes the victim stream would be created so that its first hop would be the node monitored (i.e., the furthest away from the corrupt server, so the timing pattern would be the most distorted). Monitoring by the probe server

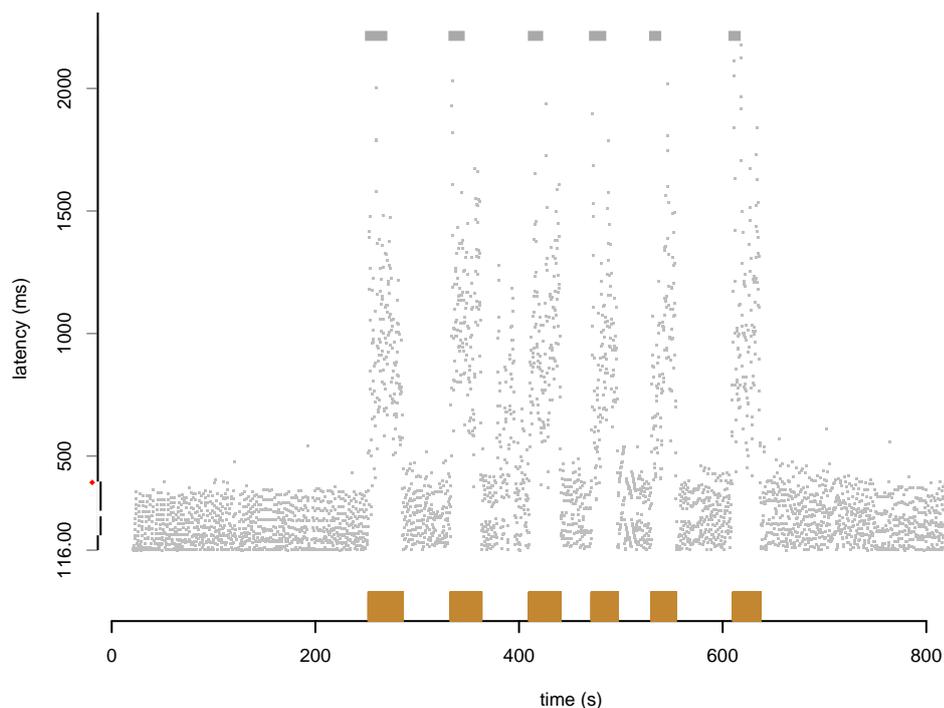


Figure 4.3: Probe results showing good correlation (Node K). The breaks in the  $x$  axis indicate quartiles and the mean is shown as a  $\blacklozenge$

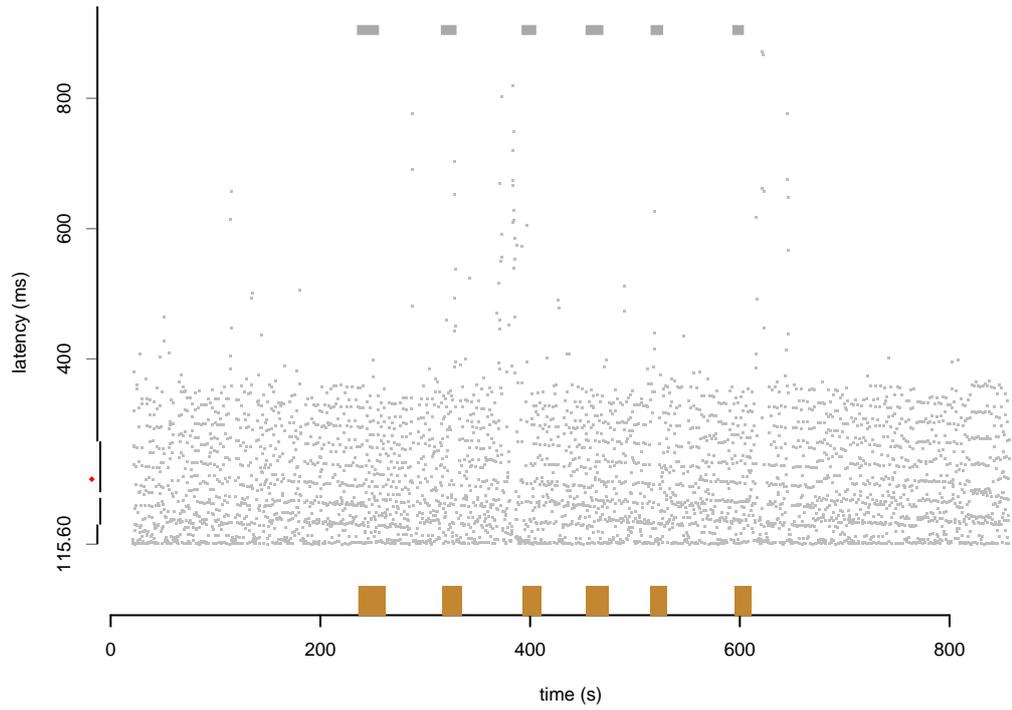
would continue for another four minutes after the victim stream was closed. In order to check for false positives, this test was then repeated, except the victim stream was sent on a path that excluded the target node.

#### 4.4.1 RESULTS

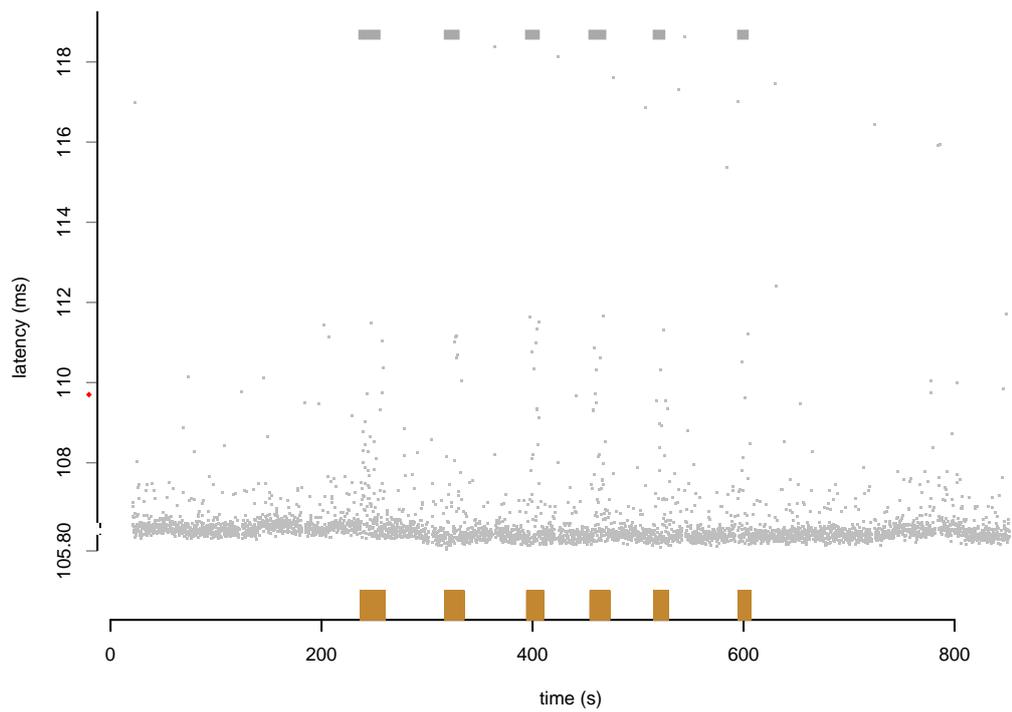
Data from probing 13 Tor nodes<sup>3</sup> was collected and processed as described in section 4.3.3 using GNU R [128]. The correlation quality varied, however for all but 2 nodes it correctly differentiated the case where the node was carrying the victim traffic and the case where it flowed through other nodes.

Figure 4.3 shows a good correlation between probe data and victim traffic. The dots indicate the latency of the probes and the pattern of the victim stream sent is shown at the bottom. The victim stream received is overlaid to show how the pattern is distorted by the network. In contrast, Figure 4.4(a) shows the same node being monitored when the victim stream was being routed elsewhere. Figure 4.5 shows a summary of the correlation over all nodes.

<sup>3</sup>Out of the 50 Tor nodes that made up the network at the time, five were not included so as to check for false positives, and the rest did not carry the probe or victim stream due to being down or because of exit-policy restrictions.



(a) Probe results without traffic pattern (Node K)



(b) False negative (Node E)

Figure 4.4: Results without positive correlation

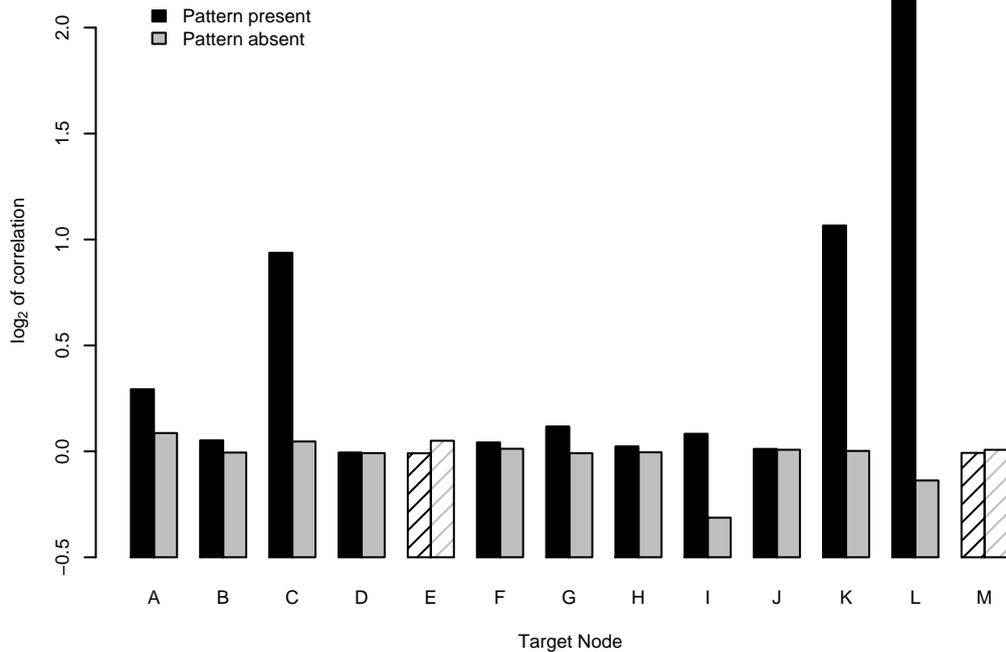


Figure 4.5: Summary of correlation. For each node, the left bar shows the correlation when the victim stream was travelling through the node (false negative test) and the right bar shows the correlation when it was not (false positive test). The two incorrect results (E and M), where the correlation was higher when the traffic was not being sent through the nodes, are highlighted with diagonal shading lines

None of the results from the false positive test show any obvious correlation to the traffic pattern, which suggests that “echos” of load are not significantly propagated through the network. This means that it should be possible to increase the accuracy of the attack simply by running the test for longer than the 6 minutes in our experiments. Other options would be to increase the sampling frequency or to improve the correlation function as suggested in Section 4.3.3. There appears to be significant room for improvement, as shown in Figure 4.4(b), which was not correctly identified as being correlated, despite showing visible similarity to the traffic pattern.

## 4.5 Discussion

The experiments clearly show that Tor streams retain their timing characteristics as they travel through the anonymising network. Furthermore, these characteristics affect other streams in such a way that it is possible to observe

them without direct access to the Tor nodes. This shows that, as a result, it is possible for an attacker to discover which Tor server is being used to inject the traffic stream, and degrade the anonymity provided into being equivalent to a collection of simple proxy servers.

The fact that the timing characteristics of streams are not substantially altered, and can be used to identify the Tor nodes carrying them, comes as no surprise. The low-latency requirement of Tor does not allow it to shape the traffic in any way, and it would require large amounts of cover traffic for these characteristics to be hidden. Since the attack relies on the indirect measurement of the stream’s traffic characteristics, a simple minded cover traffic strategy – that only filled the links with cover traffic – would not work.

The cover traffic should not only fill the links, to confuse a direct observer, but also make sure that it confuses indirect measurements as presented in this paper. When designing such a cover traffic strategy, it is also important to keep in mind Wei Dai’s attack [36, 14]: an adversary can fill the victim node with their own traffic, trying to eliminate all the cover traffic. This is very similar to the indirect measurement of traffic load that we have performed, and shows that Tor would have to use cover traffic all the time, and not simply when there is not enough genuine traffic to fill all the links.

The interference between the timing characteristics of different streams is both a benefit for anonymity and at the same time, a vehicle for attack. One would hope that streams on the same Tor node would interfere with each other to such a degree that it is impossible to differentiate them from each other, therefore making it difficult for an attacker to know which one to trace, but this is not the case. This perfect interference should create “echos” of the traced stream throughout the network and cause any traffic analysis to produce many false positives. Nevertheless, streams relayed by the same Tor node interfere with each other just enough to leak information to an adversary controlled stream and thus allow the measurement of the load of the node. In some sense, Tor exhibits the worst possible behaviour: not enough interference to destroy individual stream characteristics, yet enough to allow the remote measurement of the node’s load.

Two strategies could be employed to protect Tor: *perfect interference* and *non-interference*. Perfect interference amongst all streams relayed through the same node means that the output streams all have the same shape. Hence, the adversary will have a very difficult time determining which output stream corresponds to the input stream to be traced. Since Tor relies on a sequence

of relays, it would be interesting to study how long paths would need to be so that streams would interfere with each other in such a way that all the outputs of the network would have the same characteristic. Note, that since the vehicle of this entanglement is traffic streams, one needs to assess how many other streams have been “touched”, by being relayed through the same node, and therefore might become indistinguishable with. A second strategy for implementing perfect interference is to shape the stream traffic into another arbitrary shape, either the same for all streams or different for each of them, yet unlinkable to any particular input stream. Causality means that this shaping can only be done by delaying the packets (you cannot send a packet received at time  $t$  out into the network at time  $< t$ ). Therefore, any traffic-shaping strategy will inevitably increase the latency of the communication.

Non-interference between streams, through strict quality of service guarantees, can also be used to protect against our attacks. This would eliminate the side channel we use to remotely infer the timing of streams on Tor nodes. This property could be very difficult to implement in practice. All streams share many common resources: the Tor packet scheduler, the TCP/IP stack, the physical network and the CPU of the Tor node. There is an established methodology for discovering and eliminating covert channels [65], and it is recognised as a difficult problem. Even hardened systems exhibit covert channels of  $>1$  bit/s.

These might be tolerable for multilevel secure systems, but would be devastating for anonymous communication systems – in a few seconds an adversary could distinguish the victim’s communication amongst all of the streams. This is because there are inherently fewer actors to identify in an anonymous communication system than possible cryptographic keys or possible documents in a multilevel secure system. Even if eliminating this flaw was possible, Chapter 5 will describe an additional channel which will remain, despite the above safeguards being in place.

#### 4.5.1 LINKABILITY ATTACK

A variant of our attack can also determine whether two streams coming out of the same Tor node belong to the same initiator. Remember that Tor uses the same connection to route many streams from the same initiator – we can use this property to test whether two streams, coming out of the Tor network and accessing two corrupt servers, belong to the same user. We determine, using the main attack presented, the Tor nodes that route the two streams. While

the probability that two different initiators use the same exit node is  $1/N$  (where  $N$  is the network size), the probability that the full path of three nodes is the same, assuming each node was chosen randomly, is only about  $1/N^3$ . Therefore, the fact that two streams use the same path strongly indicates that they belong to the same initiator. Testing whether a second stream belongs to the same initiator as an already traced stream is cheaper than performing the analysis to start with. The attacker already knows the two nodes on the path of the first stream and can just test them to confirm that the second stream belongs to the same connection and initiator.

This attack is especially interesting since it shows that Tor makes it easier to link two events to the same initiator than a simple proxy. These events exhibit a particular signature, that a simple proxy does not have, namely their path through Tor, which can be uncovered. If the attacks presented here are not eliminated, augmenting the length of the Tor path, which one might assume to increase security, would make it even more vulnerable. The longer the common chain of Tor nodes two connections share, the less likely it is that they belong to different users. The same is true for the total number of nodes: it is conventionally believed that more nodes is better for anonymity, but a larger population of nodes makes common chains less common and allows for more precise identification. The fact that idle circuits are short lived, and that a stream can exit at any node in the path might make such attacks slightly less reliable, but does not solve the underlying problem.

#### 4.5.2 VARIANTS OF THE ATTACK

The attack we have presented so far relies on a probe stream being routed through a Tor node to detect the timing of a modulated communication stream from a corrupt server. Using the principle that timing information leaks from one stream to the other, we can conceive quite a few variants of this attack.

Firstly, we could modulate the probe traffic that is sent to the victim Tor node in a loop and try to detect the effects on requests sent by the initiator of the anonymous communication. In cases where the traffic is mainly from the victim to the server, the corrupt server does not have much opportunity to significantly modulate the traffic load, so this variant may be the only option. The difficulty with this approach is that the normal method of probing all Tor nodes in the network simultaneously is problematic, since the modulation of the victim stream will be the combination of the load induced on all three of the Tor nodes along the path.

An alternative would be to probe each Tor node in turn, but for a given stream lifetime, this would reduce the probe duration and thus accuracy. Instead, the attacker could probe all nodes, but using a different, “orthogonal” pattern for each node, so the resulting combination observed can be decomposed into the original components. An adaptive attack could be mounted by, for example, connecting to all nodes in the network briefly and observing the latency of the probe streams.

While this short test will have a poor accuracy, it could be used to eliminate some nodes known not to be on the path. The remaining nodes could be probed again (possibly with a longer pattern) further eliminating more nodes. This process is repeated until only three nodes remain. Another option is to probe some fraction of the nodes at one time; if the resulting stream is affected then at least one node on the path must be in that fraction, if not then all nodes in that group can be eliminated from consideration. The above techniques could be combined.

If the attacker does not have total control over the corrupt server and can only monitor the link but not modify the load, then there are still variants of our attack that can be used. One is to use the probe-modulation variant above. Another is to take advantage of a known traffic pattern observed on the corrupt server. Since this pattern cannot be optimised, the attack may take longer to reach a result, but the traffic may still be suitable for inducing an observable effect on the intermediate Tor nodes. One could mount attacks without any monitoring, if the traffic being sought has known characteristics, which can be observed on the Tor nodes it is being sent through.

If an attacker can neither directly observe nor change the traffic on the corrupt server, it may be possible to infer the load by the attacker accessing the server and observing the response time, in the same way as Tor nodes are monitored. An attacker could also alter the load of the destination server by modulating a denial of service (DoS) attack on it. When the DoS attack is running, the load of the target connection should be decreased and so decrease the load of the Tor nodes on the path it is taking.

Research on IP router flow-control has shown that by exploiting the TCP back-off algorithm, it is possible to mount an effective and difficult-to-trace denial of service attack without large resources [71]. Techniques similar to this could also be used in the probe-modulation variant and to design better patterns for the corrupt server to send, so that the influence on other Tor connections through each node is maximised.

The above attacks allow the nodes that relay a particular stream to be identified, which already severely degrades anonymity. In order to identify the initiator, the attacker must look at incoming connections to all three nodes. If resources are limited, then it would be desirable to identify the entry node, to target monitoring. This could be done by estimating how much the induced traffic pattern is shifted as it travels through the network. We did not perform this because our probe sampling frequency was too low (every 0.2 seconds) to show effects on the scale of typical Tor latency. However, once an attacker has found the three nodes on the connection path, he could probe these at higher frequency, to watch for the precise timing of the pattern. Another possibility is to look at the distortion of the induced pattern. As it progresses through the network, noise will be added, so it is likely the node showing the third-best correlation is the entry node.

#### 4.5.3 ATTACK COSTS

Our attack is feasible for the adversary anticipated by the Tor designers and can be mounted without direct access to the communication links of the Tor nodes. To reliably perform the attacks, each Tor node in the network should be observed all the time. Assuming there are  $N$  Tor nodes, we therefore require  $N$  probe streams going through them – a set of machines, or a single machine connected to the Internet with a low-latency connection suffices. This means that the cost of the attack is  $O(N)$  since it increases linearly with the number of nodes in the Tor network.

Note that higher volumes of traffic in the Tor network would make the quality of the observation poorer and degrade the performance of the attack. Therefore, there is a hidden cost that has yet to be estimated, which rises with the number of streams relayed by each node. At the same time, any increase in latency that might hinder the attacker, by making the remote measurements less precise, will inevitably also increase the latency of genuine Tor traffic. Therefore, we are again confronted with the choice of increasing the security of the system, versus keeping the latency as low as possible.

Aside from observing the nodes, an adversary is assumed to have the ability to modulate the replies of a dishonest accessed server. The simplest way of doing this is by deceiving anonymous users and making them access an attacker controlled server. This way, arbitrary data streams can be sent back and forth, and can be detected. Where Tor is used to access an HTTP [55] (web) service, the attacks can be mounted much more simply, by including *traffic-analysis*

*bugs* within the page, in the same way as web bugs [5, 34] are embedded today. These initiate a request for an invisible resource that, due to the HTTP architecture, can have an unconstrained traffic shape and characteristic. The attacker can then simply try to detect them, using our attack as described.

#### 4.5.4 UNDERSTANDING THE TRAFFIC ARTIFACTS

As described earlier, our attack is based on the fact that the traffic characteristics of streams are hardly affected by Tor, and that these characteristics leak into other streams sufficiently so that they can be remotely estimated. It is interesting to study how these processes are taking place in practice.

Streams interfere with each other at all levels. At the highest level, Tor routers relay a set of streams using the non-blocking polling strategy whose code is presented in Figure 4.6. Each of the relayed streams is polled to see if any data is available to be relayed. If data is available, it is processed, otherwise the next stream is considered. This strategy in itself ensures that a different stream being relayed will delay the probe stream, and leak information about the latency of the node.

Aside from the polling strategy, streams relayed by Tor share the operating-system resources, the TCP/IP stack, the network and the hardware of the Tor node. The operating-system scheduler could influence the timing of the streams by allocating more resources when the node relays more traffic, or less when the node is mostly waiting for more input. Memory management operations could also take more time if data is being routed. The TCP protocol would back-off if the link is congested. Finally, the network has a fixed capacity, and has to be shared amongst connections. All of these contribute to the latency of the probe data being influenced by the volume of data relayed by the Tor node. Figure 4.3 illustrates this. It is clear that the probe data (top) can be used to infer the volume of the actual traffic sent (bottom).

Other traffic patterns have been observed in the measurement data that are not yet fully explained. These could be artifacts of the measurement technique that, by its indirect nature, can only act as an estimate of the load, or genuine latency introduced by the remote Tor node. We present here two examples that could be used to perform traffic analysis, if they were linked with particular states of the Tor nodes.

Figure 4.7(a) shows the results of probes against an exit node in the Tor network. Again, the top graph represents the latency over time of probe traffic, while the bottom represents the times the corrupt server was sending data.

```

/* Tor main loop */
for(;;) {
    timeout = prepare_for_poll();
    ...
    /* poll until we have an event,
       or the second ends */
    poll_result = tor_poll(poll_array, nfds, timeout);
    ...
    /* do all the reads and errors first,
       so we can detect closed sockets */
    for(i=0;i<nfds;i++)
        /* this also marks broken connections */
        conn_read(i);

    /* then do the writes */
    for(i=0;i<nfds;i++)
        conn_write(i);

    /* any of the conns need to be closed now? */
    for(i=0;i<nfds;i++)
        conn_close_if_marked(i);
    ...
}

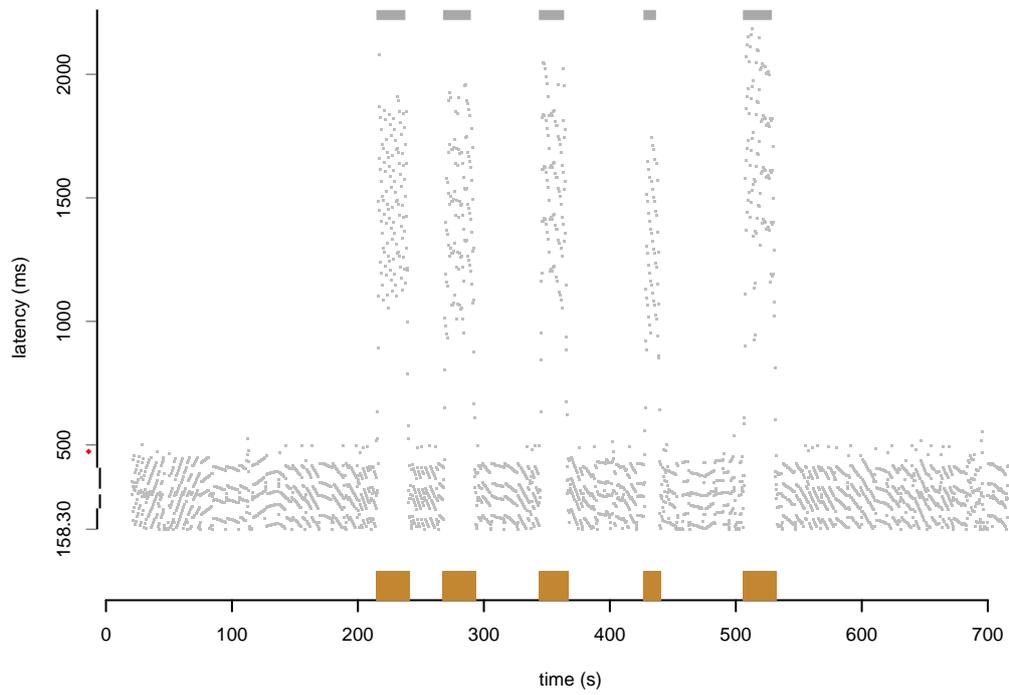
/* Read from connection */
static void conn_read(int i) {
    ...
    if(!(poll_array[i].revents & (POLLIN|POLLRDHUP|POLLERR)))
        if(!connection_is_reading(conn) ||
            !connection_has_pending_tls_data(conn))
            return; /* this conn should not read */
    ...
    connection_handle_read(conn) < 0) {
    ...
}

```

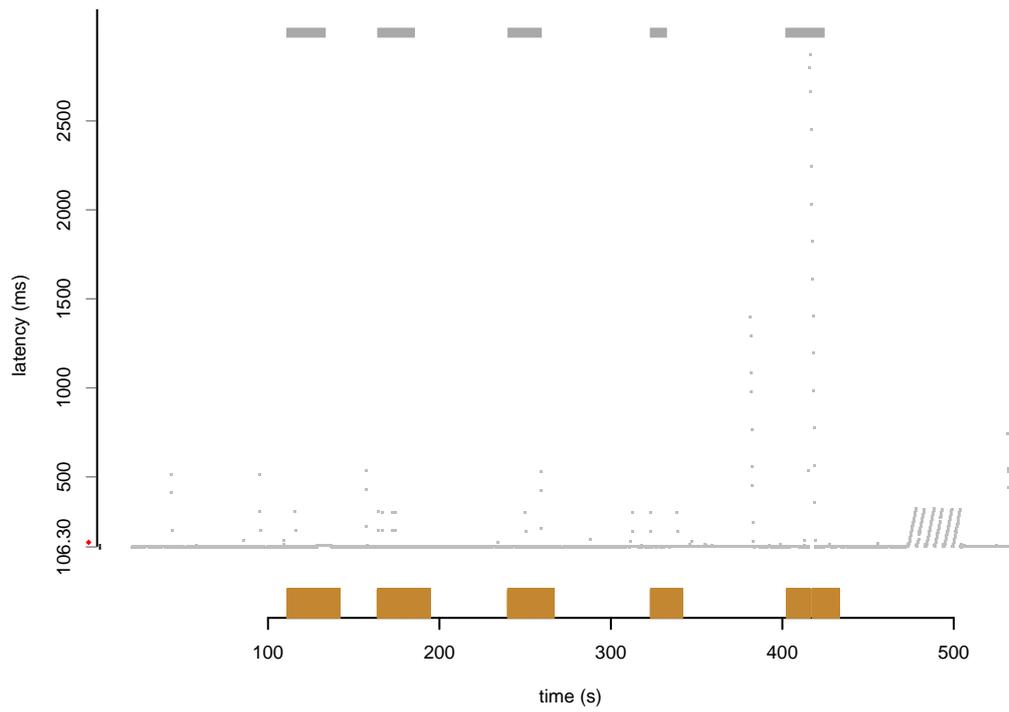
Figure 4.6: The Tor 0.0.9 polling code

Note that the latency of the probes seems to be quantised into four or five bands – even when a high volume of traffic is injected. This effect could be explained by a lack of precision in the measurement process. Another explanation is that the bands are formed by measuring the node when one, two, three or four other streams are being served at a time. This matches the experimental data: only four clusters are visible when the corrupt server is not relayed, and five when the stream is present. This technique could be developed to extract information about the number of streams relayed – and in turn used to infer the beginning and termination of a stream.

Figure 4.7(b) illustrates a completely different type of traffic pattern. After the last burst of traffic from the corrupt server (bottom) the latency of the



(a) Horizontal line artifacts



(b) End of session artifacts

Figure 4.7: Artifacts under investigation

probe traffic exhibits a very peculiar pattern, it rises six times by around 210 ms each time falling back into the average latency. This event has been observed many times in conjunction with the closure of a Tor connection, and could be due to time devoted in tearing down connections. If such events can be observed, the connection tear down could be tracked through the network to gain information about the route of a connection. I leave further investigation of these artifacts to future work.

Aside from the precise load information extracted from the probe traffic, these secondary traffic artifacts could also be used to perform traffic analysis and assess which Tor server is being used to relay the target traffic. Therefore, a strategy to eliminate information leakage into other streams should also try to eliminate these artifacts.

## 4.6 Conclusion

We have presented an attack against Tor, a deployed and popular, anonymising network. This attack can be performed by a modest adversary, using powers comfortably within the restricted Tor threat model. In fact, we show that the anonymising network itself can be used to route probe traffic and gather information otherwise available only to a global passive adversary.

In November 2004, I performed extensive experiments on current Tor nodes and found them to be susceptible to our attack. This does not give us the ability to trace the actual originator of the communication, since we do not have the ability to observe who is connected to a Tor node. Nevertheless, our attacks greatly degrade the anonymity provided by Tor, by allowing adversaries to discover the path of a Tor connection and thereby reduce the protection to the level provided by a collection of simple proxy servers. We expect the same attack to be applicable against other low-latency anonymising network designs, since none of them have been specially hardened against it.

Furthermore, since Tor re-uses the same path for multiple streams within a short time interval, our attacks allow different operations to be linked to the same initiator with greater certainty. The observable path of each stream can act as an identifier or identity that links streams amongst themselves and to the initiator – a property that makes Tor weaker than a simple proxy when it comes to protecting the unlinkability of actions.

We discussed some strategies that could be used to protect Tor against our attacks. They all, to some degree, involve an increase in the latency of

the communication. They also highlight the need for a full covert-channel and side-channel analysis of such anonymising networks, to assess whether any information that could be used for traffic analysis is leaked to other streams that are potentially observable to an adversary.

This attack brings the field of anonymous communications even closer to more traditional computer-security disciplines. On one hand we show that the literature on covert channel analysis and elimination is directly applicable and necessary to truly secure Tor. On the other hand, our attack relies on using Tor nodes as oracles that disclose their load – therefore not requiring a global observer. Similar techniques have been used in the past in breaking cryptographic protocols, by composing the services they provide. It is the first time that such techniques have been applied in performing traffic analysis of an anonymous communication system.

# Chapter 5

## Temperature-based channels

Location-hidden services, as offered by anonymity systems such as Tor, allow servers to be operated under a pseudonym. As Tor is an overlay network, servers hosting hidden services must be accessible both directly and over the anonymous channel. The previous chapter illustrated that traffic patterns through one channel have observable effects on the other, and that this fact could be exploited to track user behaviour. Defences to this attack were also discussed, such as Tor nodes providing fixed quality of service to each connection, regardless of other traffic, thus reducing network capacity but resisting such interference attacks.

However, even if no connection influenced any other, total throughput of a node would still affect the load on its CPU, and thus, on modern computers, the heat output. Unfortunately for anonymity, the result of temperature on clock skew can be remotely detected through observed timestamps. This attack works because existing abstract models of anonymity-network nodes do not take into account the inevitable imperfections of the hardware they run on. Furthermore, I suggest that the same technique could be exploited as a classical covert channel and may even provide geolocation.

### 5.1 Introduction

Hidden services allow access to resources without their operator's identity being revealed. Not only does this protect the owner, but also the resource, as observed by Needham [112, 113], because anonymity can help prevent selective denial of service attacks (DoS). Tor [50], has offered hidden services since 2004, allowing users to run a TCP server under a pseudonym. As of April

2007, there are around 80 publicly advertised hidden services, offering access to resources that include chat, low and high latency anonymous email, remote login (SSH and VNC), websites and even gopher [10]. The full list of hidden services is only known to the three Tor hidden-service directory servers.

Systems to allow anonymous and censorship-resistant content distribution have been desired for some time, but recently, anonymous publication has been brought to the fore by several cases of blogs being taken down and/or their authors being punished, whether imprisoned by the state [134] or being fired by their employers [15]. In addition to blogs, Tor hidden websites include dissident and anti-globalisation news, censored or otherwise controversial documents, and a PGP keyserver. It is clear that, given the political and legal situation in many countries, the need for anonymous publishing will remain for some time.

The credible threat faced by anonymous content-providers emphasises the importance of evaluating the security, not only of deployed anonymous publication systems, but also of proposed changes believed to enhance their security. As discussed in the previous chapter, guaranteed quality of service (QoS), ensuring non-interferences between streams, is one such defence, designed to protect against indirect traffic-analysis attacks that estimate the speed of one flow by observing the performance of other flows through the same machine.

QoS acts as a countermeasure by preventing flows on an anonymity-network node from interfering with each other. However, an inevitable result is that when a flow is running at less than its reserved capacity, CPU load on the node carrying it will fall. This reduces temperature, which affects the frequency of the crystal oscillator driving the system clock. I measure this effect remotely by requesting timestamps and deriving clock skew.

I have tested this vulnerability hypothesis using the current Tor implementation (0.1.1.16-rc), although – for reasons explained later – using a private instance of the network. Tor was selected due to its popularity, but also because it is well documented and amenable to study. However, the attacks I present here are applicable to other anonymity systems, particularly those based on overlay networks.

In Section 5.2 we review how hidden services are implemented in Tor, discuss the threat models used in their design and summarise existing attacks. Then, in Section 5.3 we provide some background on clock skew, the phenomenon we exploit to link a hidden service pseudonym to the server's real identity. In Section 5.4 we present the results of experiments on Tor and discuss the potential impact of the attack and defences against it. Finally, in

Section 5.5 we suggest how the general technique (of creating covert channels and side channels which cross between the digital and physical worlds) might be applied in other scenarios.

## 5.2 Hidden services

The attacks presented in this chapter are independent of the underlying anonymity system and hidden service architecture, and should apply to any overlay network. While there are differing proposals for anonymity systems supporting hidden services, e.g. the PIP Network [66], Tor is a popular, deployed system, suitable for experimentation, so initially we will focus on it. Section 5.5 will suggest other cases where these technique can be used.

Tor hidden services are built on the connection-anonymity primitive that Tor provides. This was discussed in the previous chapter and full details are given in [47, 48, 49, 50]. However, neither the Tor hidden service protocol nor our attack relies on the underlying implementation of connection anonymity – all that is required to understand the attack is that Tor can anonymously tunnel a TCP stream to a specified address and port number. It does this by relaying traffic through randomly selected nodes, wrapping data in multiple layers of encryption to maintain unlinkability. Unlike email mixes, it does not intentionally introduce any delay: typical latencies are in the 10–100 ms range.

There are three phases in accessing a hidden service. To maintain security, all links between the participants are anonymised through the Tor network. The full sequence is illustrated in Figure 5.1.

**Service publication:** When the hidden service is activated, the information needed by clients to connect must be published. This stage is also repeated periodically. First, the hidden service selects an *introduction point (IP)* and makes a persistent connection to it **(1)**. The address of the IP is sent to the Tor *directory servers* and stored, linked with the hidden service pseudonym **(2)**.

**Connection setup:** Once a client wishes to access the hidden service, knowing the hidden service pseudonym (a hash of the service’s public key), it first recovers the address of the IP from the directory server **(3)**. The client then selects a *rendezvous point (RP)* and connects to it **(4)**. The address of the RP is sent to IP **(5)** which then sends it to the hidden service over the connection established in (1) **(6)**.

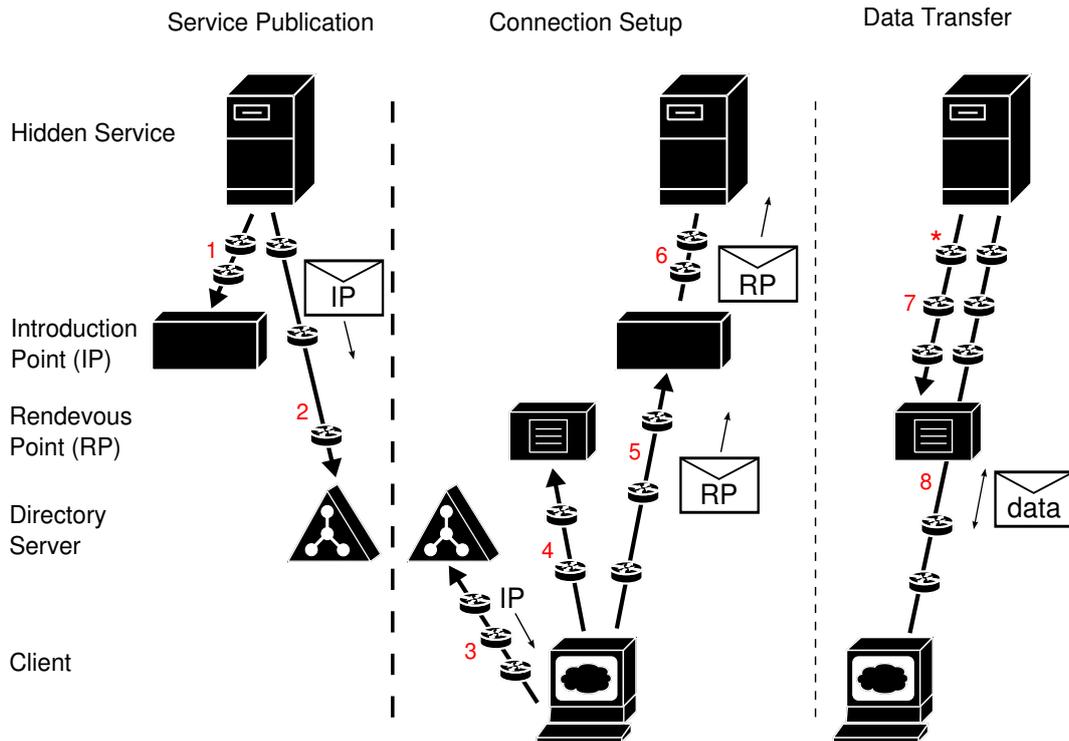


Figure 5.1: Tor Hidden Service setup, connection and data transfer. Thick lines indicate circuits with arrows pointing away from circuit initiator. Circles indicate Tor nodes relaying traffic. Thin arrows show the primary data flow direction. The node marked with \* knows the hidden server's IP address and is discussed in Section 5.2.2

**Data transfer:** Finally, the hidden service connects back to the RP (7), which then joins this link to the connection established in (4). Now a connection has been established between the client and hidden service so application data transfer (e.g. HTTP) can take place (8).

For clarity, some details have been omitted from this summary; a more complete description was written by Øverlier and Syverson [117] and the full details are covered by the Tor rendezvous specification [49]. For the remainder of the chapter, we will deal only with an established data connection (8), from the client to the rendezvous point and from there to the hidden server.

### 5.2.1 THREAT MODEL

The primary goal of our attacker is to link a pseudonym (under which a hidden service is being offered) to the operator's real identity, either directly or through some intermediate step (e.g. a physical location or Internet protocol address). For the moment, we will assume that identifying the Internet protocol address

is the goal, but Section 5.5.3 will discuss what else can be discovered, and some particular cases in which an Internet protocol address is hard to link to an identity.

Low-latency anonymity networks without dummy traffic, like Tor, cannot defend against a global passive adversary. Such an attacker simply observes inputs and outputs of the network and correlates their timing patterns, so called *traffic-analysis*. For the same reason, such networks cannot protect against traffic confirmation attacks, where an attacker has guessed who is communicating with whom and can snoop individual network links in order to validate this suspicion.

It is also common to assume that an attacker controls some of the anonymity network, but not all. In cases like Tor, which is run by volunteers subject to limited vetting, this is a valid concern, and other work has made use of this, including the previous chapter as well as Øverlier and Syverson’s attack [117]. However, the attacks presented here do not require control of any node, so will apply even to entirely uncompromised anonymity networks.

In summary, we do not assume that our attacker is part of the anonymity network, but can access hidden services exposed by it. We do assume that he has a limited number of candidate hosts for the hidden service (say, a few thousand). However, we differ from the traffic confirmation case excluded above in that our attacker cannot observe, inject, delete or modify any network traffic, other than that to or from his own computer.

### 5.2.2 EXISTING ATTACKS

The first documented attack on hidden servers was by Øverlier and Syverson [117]. They proposed and experimentally confirmed that a hidden service can be located within minutes or hours if the attacker controls one, or preferably two, Tor network nodes.

The attack relies on the fact that a Tor hidden server selects nodes at random to build connections. The attacker repeatedly connects to the hidden service, and eventually a node he controls will be the one closest to the hidden server (marked with \* in Figure 5.1). Now, by correlating input and output traffic, the attacker can confirm that this is the case, and so he has discovered the hidden server’s Internet protocol address.

As the paper exploits the routing choices of Tor, the authors propose modifications to resist such attacks. The main defence is *entry guards*, based on the concept of *helper nodes* [158]. Here, a small number (three is suggested)

of Tor nodes are semi-permanently selected as the first node in the vulnerable circuit. If one of these is controlled by an attacker, the hidden server is completely compromised, but if not, it should remain safe while the guard nodes are operational. A variety of enhancements are also proposed to enhance reliability and security against active attack.

The attack in the previous chapter could also be extended to hidden services if the hidden server was also a Tor node. By exercising the hidden service, the performance of all Tor nodes on the path will be affected. Many hidden servers are also publicly advertised Tor nodes, in order to mask hidden-server traffic with other Tor traffic, so this scenario is plausible. Even where the hidden server is not a Tor node, if the attacker has a limited number of candidates for the hidden service, and some way to remotely measure load, this attack could also reveal the hidden server's identity.

One of the defences proposed in the previous chapter is non-interference – where each stream going through a node is isolated from the others. Here, each Tor node has a given capacity, which is divided into several slots. Each circuit is assigned one slot and is given a guaranteed data rate, regardless of the others.

This chapter's observation, which underpins the attack presented, is that when circuits carried by a node become idle, its CPU will be less active, and so cool down. Temperature has a measurable effect on clock skew, and this can be observed remotely. We will see that an attacker can thus distinguish between a CPU in the idle state and one that is busy.

### 5.3 Clock skew and temperature

Kohno *et al.* [90] used timing information from a computer to fingerprint its physical identity. By examining timestamps from the machine, either through passive or active probing, they estimated its *clock skew*, the ratio between actual and nominal clock frequencies.

They found that a particular machine's clock skew deviates very little over time, around 1–2 parts per million (ppm), depending on operating system, but that there was a significant difference between the clock skews (up to 50 ppm) of different machines, even identical models. This allows a host's clock skew to act as a fingerprint, linking repeated observations of timing information. The paper estimates that, assuming a stability of 1 ppm, 4–6 bits of information on the host's identity can be extracted.

Two sources of timestamps were investigated by Kohno *et al.*: ICMP timestamp requests [123] and TCP timestamp options [82]. The former has the advantage of being of a fixed nominal frequency (1 kHz), but if a host is Network Time Protocol (NTP) [101] synchronised, the ICMP timestamp was found to be generated after skew adjustment, so defeating the fingerprinting attack. The nominal frequency of TCP timestamps depends on the operating system, and varies from 2 Hz (OpenBSD 3.5) to 1 kHz (Linux 2.6.11). However, it was found to be generated before NTP correction, so attacks relying on this source will work regardless of the NTP configuration. Additionally, in the special case of Linux, Chapter 3 of this thesis showed how the value of a host’s NTP-disciplined clock could be derived from TCP sequence numbers.

In this chapter, we will primarily use TCP timestamps, which are enabled by default on most modern operating systems. This feature was intended to improve performance by providing better estimates of round-trip times and protect against wrapped sequence numbers. Because of their utility, TCP timestamps are commonly passed by firewalls, unlike ICMP packets and IP options, so are widely applicable. These alternative measurement techniques will be revisited in Section 5.5.4.

### 5.3.1 BACKGROUND AND DEFINITIONS

Let  $T(t_s)$  be the timestamp sent at time  $t_s$ . Unless specified otherwise, all times are relative to the receiver clock. The skew  $s$  is  $(h_{\text{actual}} - h)/h$ , where  $h_{\text{actual}}$  is the sender clock’s frequency, relative to the receiver clock, and  $h$  is the sender clock’s nominal frequency. As we are interested in changes of clock frequency, we split the skew into two components, the constant  $s_c$  and the time-varying part  $s(t)$  with  $s = s_c + s(t)$ . Without loss of generality, we assume that the time-varying component is always negative.

Before a timestamp is sent, the internal value of time is converted to a number of *ticks* and rounded down. The nominal length of a tick is the clock’s resolution and the reciprocal of this is  $h$ . The relationship between the timestamp and input parameters is thus:

$$T(t_s) = \left\lfloor h \cdot \left( t_s + s_c t_s + \int_0^{t_s} s(t) dt \right) \right\rfloor \quad (5.1)$$

Now, we sample timestamps  $T_i$  sent at times  $t_{s_i}$  chosen uniformly at random between consecutive ticks, for all  $i$  in  $[1 \dots n]$ , with  $t_{s_1} = 0$ . The quantisation noise caused by the rounding can be modelled as subtracting a random variable

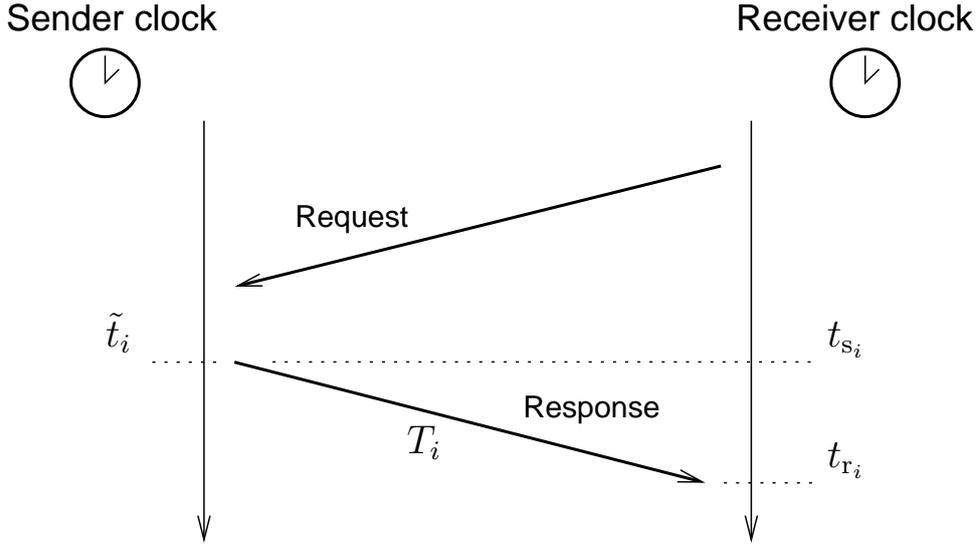


Figure 5.2: Time and timestamp quantities discussed.  $t_{s_i}$  and  $t_{r_i}$  are relative to the receiver clock;  $\tilde{t}_i$  is relative to the sender clock. Only  $t_{r_i}$  and  $\tilde{t}_i$  are directly accessible to the measurer so  $t_{s_i}$  must be inferred

$c$  with uniform distribution over the range  $[0, 1)$ . By dividing by  $h$ , we can recover the time according to the sender in sample  $i$ :

$$\tilde{t}_i = T_i/h = t_{s_i} + s_c t_{s_i} + \int_0^{t_{s_i}} s(t) dt - c_i/h \quad (5.2)$$

These quantities are summarised in Figure 5.2.

We cannot directly measure the absolute clock skew of a remote machine, but we can calculate the *offset*. This is the difference between a clock's notion of the time and that defined by the reference clock (receiver). The offset  $o_i$  is  $\tilde{t}_i - t_{s_i}$ . However, the receiver cannot directly observe  $t_{s_i}$ , only the time  $t_{r_i}$  when a packet was received.

Let  $d_i$  be the latency of a packet, from when it is timestamped to when it is received, then  $t_{s_i} = t_{r_i} - d_i$ . Skew is typically small ( $< 50$  ppm) so the effect of latency to these terms will be dominated by the direct appearance of  $d_i$  and is ignored otherwise. The measured offset is thus:

$$\tilde{o}_i = \tilde{t}_i - t_{r_i} = s_c t_{r_i} + \int_0^{t_{r_i}} s(t) dt - c_i/h - d_i \quad (5.3)$$

Figure 5.3 shows a plot of the measured offset against packet receipt time. Were the sampling noise  $c/h$ , latency introduced noise  $d$ , and variable skew

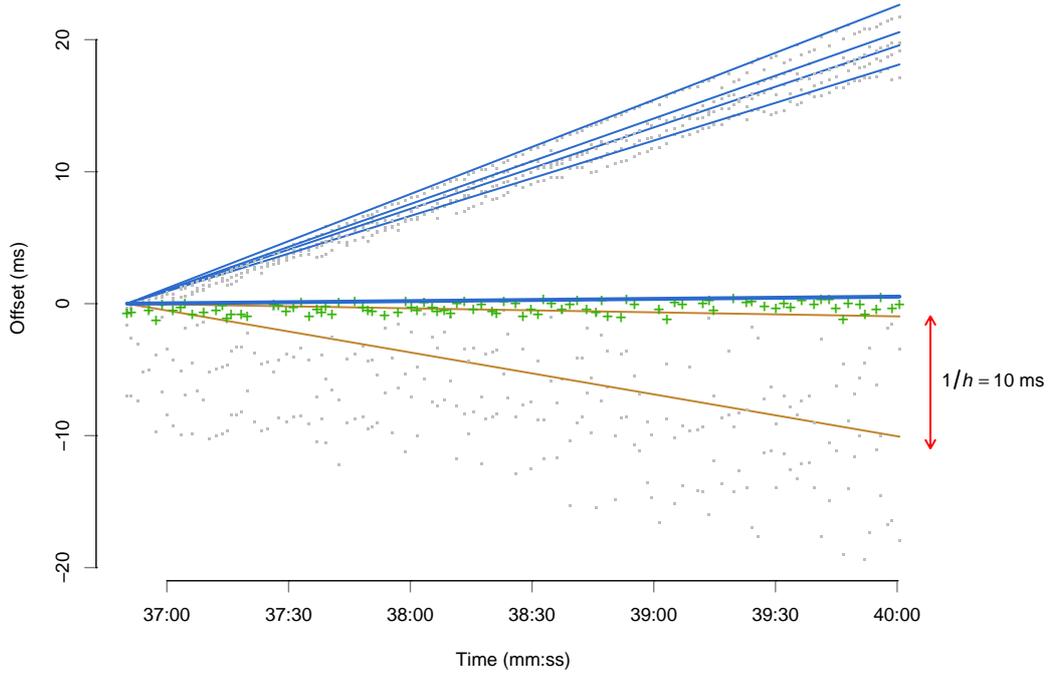


Figure 5.3: Offset between TCP timestamps of seven machines and the measurer’s clock over time. The bottom two lines (—) show clocks with 100 Hz resolution and the others are 1 kHz. The range of the quantisation noise is  $[0, 1/h)$ , as indicated for the  $h = 100$  Hz case. The time since the beginning of the experiment was subtracted from the measurer’s clock and the first timestamp received was subtracted from all timestamps. All machines were on the same LAN, except one (+), which was accessed over a transatlantic link, through 14 hops

$s(t)$  absent, the constant skew  $s_c$  would be the derivative of measured offset with respect to time. To form an estimate of the constant skew observed,  $\hat{s}_c$ , in the presence of noise, we would like to remove these terms. Note that in (5.3) the noise contributions, as well as  $s(t)$ , are both negative.

Following the approach of Kohno *et al.*, we remove the terms by fitting a line above all measurements while minimising the mean distance between each measurement and the point on the line above it. By applying the linear-programming based algorithm described by Moon *et al.* [103], we derive such a line. More formally this finds an estimate of the linear offset component  $\hat{o}(t) = \hat{s}_c \cdot t + \beta$  such that, for all samples,  $\hat{o}(t_{r_i}) > \tilde{o}_i$  and minimises the expression:

$$\frac{1}{n} \cdot \sum_{i=1}^n (\hat{o}(t_{r_i}) - \tilde{o}_i) \quad (5.4)$$

The offset  $\hat{o}(t)$  is also plotted on Figure 5.3. The band of offset samples below the line is due to the sampling noise  $c/h$ , as illustrated by the different width depending on  $h$ . Points are outside this band because of jitter in the network delay (any constant component will be eliminated), but latencies are tightly clustered below a minimum which remains fixed during the test. This is to be expected for an uncongested network where routes change rarely. The characteristics of these noise sources will be discussed further in Section 5.5.4, and also how techniques to limit their effect may be developed.

### 5.3.2 IMPACT OF TEMPERATURE

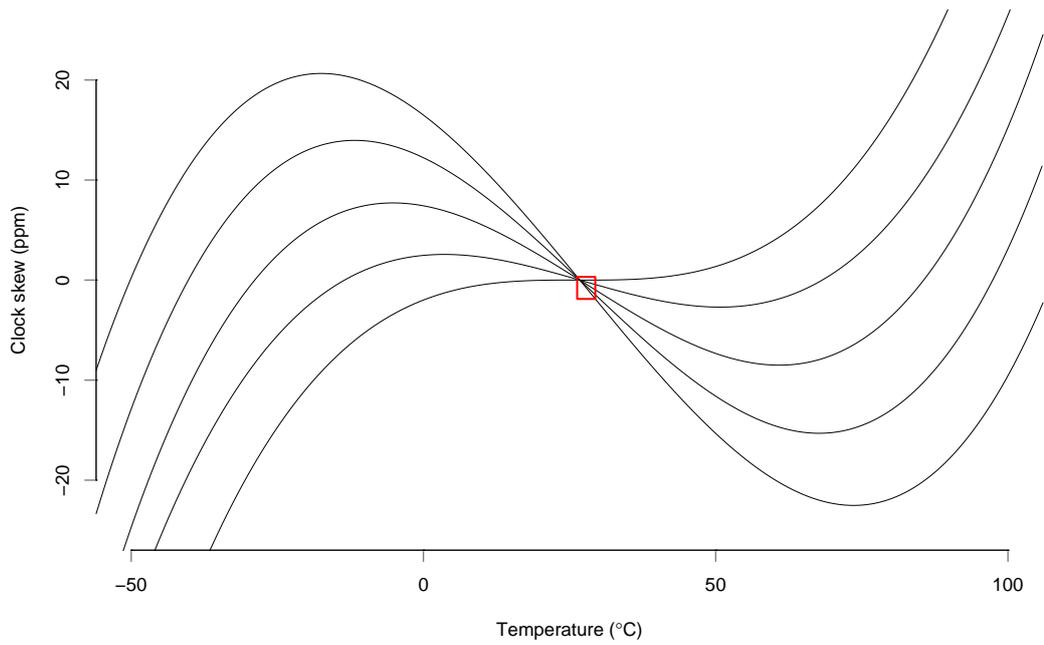
The effect of temperature on remote clock skew measurements has been well known to the NTP community since the early 1990s [92, 96] and was mentioned by Kohno *et al.* However, I believe that the paper on which this chapter was based [107] to be the first that proposes *inducing* temperature change and measuring the change in clock skew, in order to create a side channel and attack an anonymity system.

As shown in Figure 5.4, the frequency of a clock crystal varies with its temperature. Exactly how depends on tradeoffs made during manufacture. The figure shows an AT-cut crystal common for PCs, whose skew is defined by a cubic function of temperature. BT-cut is more common for sub-megahertz crystals and is defined by a quadratic.

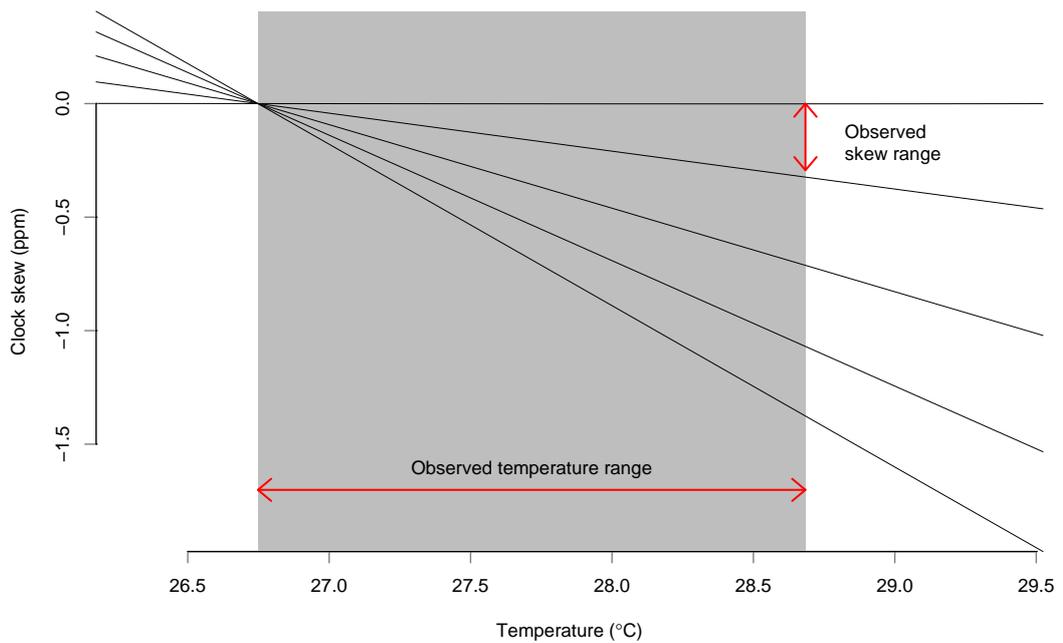
The angle of cut alters the temperature response and some options are shown. It can be seen that improving accuracy in the temperature range between the two turning points degrades performance outside these values. Manufacturers will thus select a crystal appropriate for its environment. Over the range of temperatures encountered in my experiments, skew response to temperature is almost linear, so for simplicity we will treat it as such.

The linear offset fit shown in Figure 5.3 matches the data almost perfectly, excluding noise. This indicates that although temperature varied during the sample period, the constant skew  $s_c$  dominates any temperature dependence  $s(t)$  present.

Nevertheless, the temperature dependent term  $s(t)$  is present and is shown in Figure 5.5. Here,  $\hat{o}(t_{r_i})$  has been subtracted from all  $\tilde{o}_i$ , removing our estimate of constant skew  $\hat{s}_c$ . To estimate the variable skew component  $\hat{s}(t)$ , the resulting offset is differentiated, after performing a sliding window line-fitting. We see that as the temperature in the room varied over the day, there is a correlated change in clock skew.

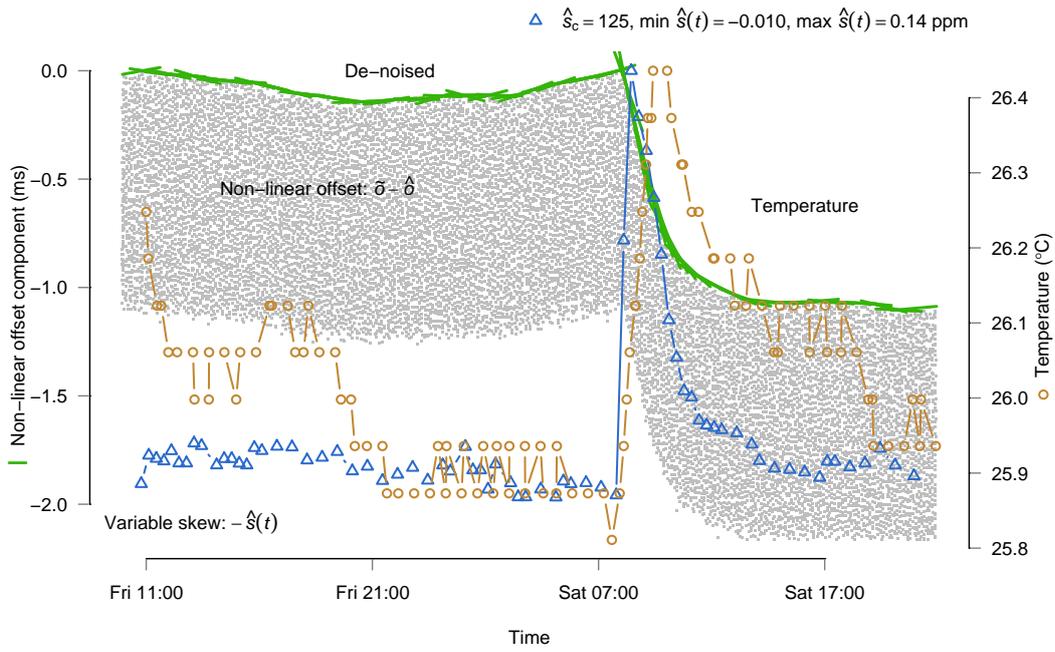


(a) Full operational range of the crystal

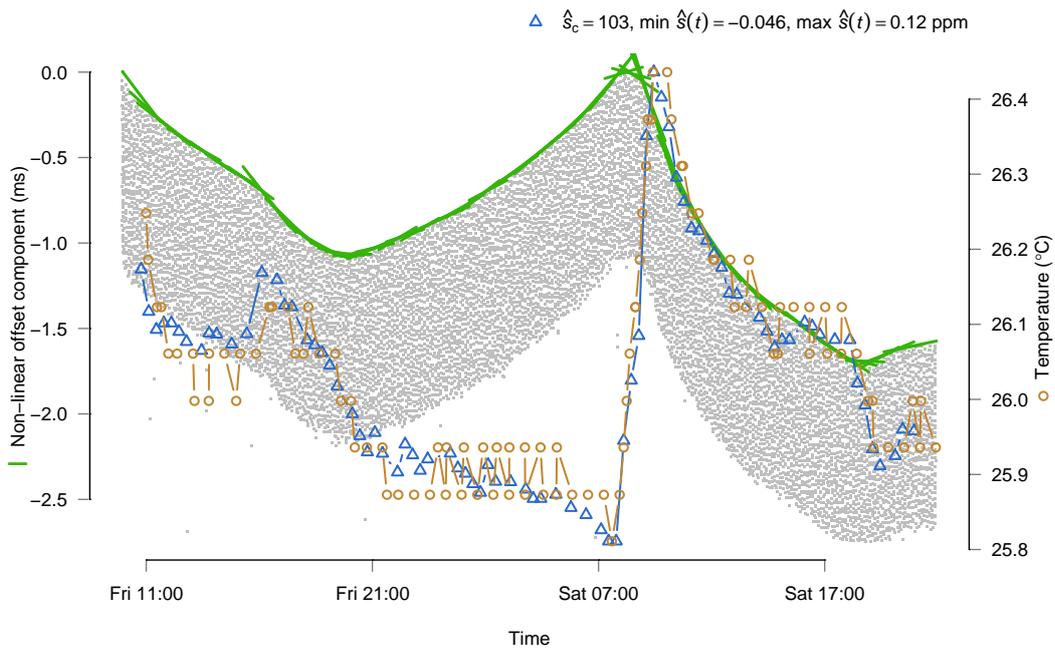


(b) Zoomed in area, as indicated in (a). The temperature and skew ranges found in Figure 5.7(a) are shown

Figure 5.4: AT-cut crystal clock skew over two temperature ranges [27]. As skews are relative, the curves have been shifted vertically so that skew is zero at the minimum observed temperature



(a) Mini-tower PC with ASUS A7M266 motherboard and 1.3 GHz AMD Athlon processor



(b) Mini-tower with 1.7 GHz Pentium 4

Figure 5.5: Offset after removing linear component (i.e.  $\tilde{\sigma} - \hat{\delta}$ ). The line ( $\text{—}$ ) above is the de-noised version. The  $\triangle$  show the negated slope of each piece ( $-\hat{s}(t)$ ) and  $\circ$  show the room temperature. The maximum and minimum values of  $\hat{s}(t)$  are shown, along with the constant skew  $\hat{s}_c$

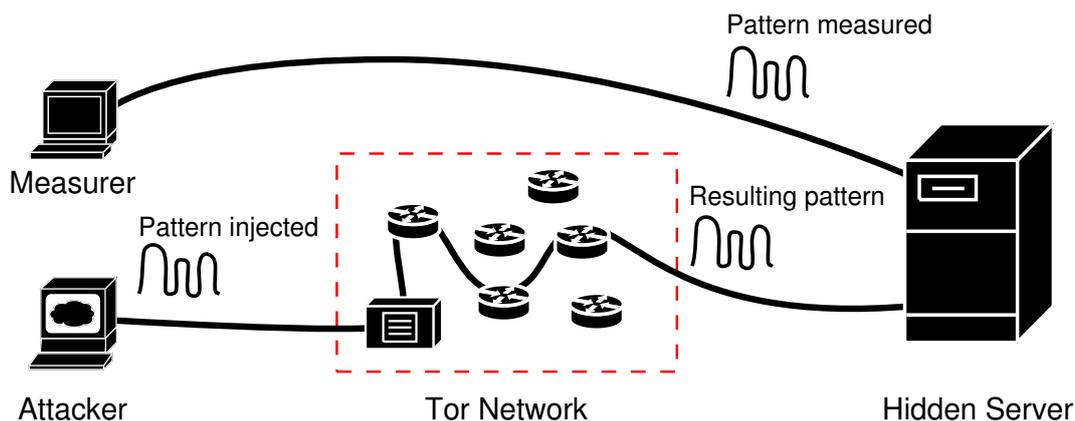


Figure 5.6: Experimental setup with four computers

## 5.4 Attacking Tor

We aim to show that a hidden server will exhibit measurably different behaviour when a particular connection is active as compared to when it is idle. In the previous chapter, I probed the latency of other connections going through the same node, but now I measure clock skew. Just as latency increases with load, we expect temperature to increase too.

This is because when a connection is idle, the host CPU will not be performing as many computations and so cool down. The CPU temperature change will affect that of the clock crystal, altering clock skew, and this can be observed remotely by requesting timestamps. The goal of my experiment is to verify this hypothesis.

Such an attack could be deployed in practice by an attacker using one machine to access the hidden service, varying traffic over time to cause the server to heat up or cool down. Simultaneously, he probes all candidate machines for timestamps. From these the attacker infers clock skew estimates and when a correlation between skew and the induced load pattern is found, the hidden service is de-anonymised.

The reliability and performance of the public Tor network for hidden servers is currently quite poor and long runs of data collection would be required. In order to simplify obtaining results, my experiments were run on a private Tor network. I see no reason why these results would not transfer to the real Tor network, even when it is made more reliable and resistant to attacks of the previous chapter.

The computers used in each test (shown in Figure 5.6) are:

**Hidden Server:** Tor client and webserver, hosting a 10 MB file; fitted with a temperature sensor.

**Tor Network:** Two Tor directory server processes and five normal servers, which can act as introduction and rendezvous points, all unmodified. While all processes are on the same machine and latency between processes has not be artificially increased, this does not invalidate our results as only the Hidden Server is being analysed.

**Attacker:** Runs the Tor client, repeatedly requesting the file hosted by the Hidden Server, through the Tor Network. For performance, this is modified to connect directly to the rendezvous point.

**Measurer:** Connects directly to the Hidden Server’s public Internet protocol address, requesting TCP timestamps, ICMP timestamps and TCP sequence numbers, although only the results for the first are shown.

For two hours the 10 MB file is repeatedly downloaded over the Tor network, with up to 10 requests proceeding in parallel. Then, for another two hours no requests are made. During both periods, timestamps are requested directly from the server hosting the hidden service at intervals of 1 s plus a random period between 0 s and 1 s. This is done to meet the assumption of Section 5.3.1, that samples are taken at random points during each tick. Otherwise, aliasing artifacts would be present in the results, perturbing the line-fitting algorithm.

Finally, the timestamps are processed as described in Section 5.3.2. That is, estimating the constant skew through the linear programming algorithm and removing it, then dividing the trace into pieces and applying the linear-programming algorithm a second time to estimate the varying skew.

Were an attacker to deploy this attack, the next step would be to compare the clock skew measurements of all candidate servers with the load pattern induced on the hidden service. To avoid false-positives, multiple measurements are needed. The approach taken in the previous chapter is to treat the transmission of the load pattern as a side channel and send a pseudorandom binary sequence. Thus, after  $n$  bits are received, the probability of a false-positive is  $2^{-n}$ . From inspection, we estimate the capacity of the side channel to be around 2–8 bits per hour. An alternative taken by Fu *et al.* [59], in a related

context, is to induce a periodic load pattern that can be identified in the power spectrum of the Fourier transformed clock skew measurements. With either approach, the confidence level could be increased arbitrarily by running the attack for longer.

#### 5.4.1 RESULTS

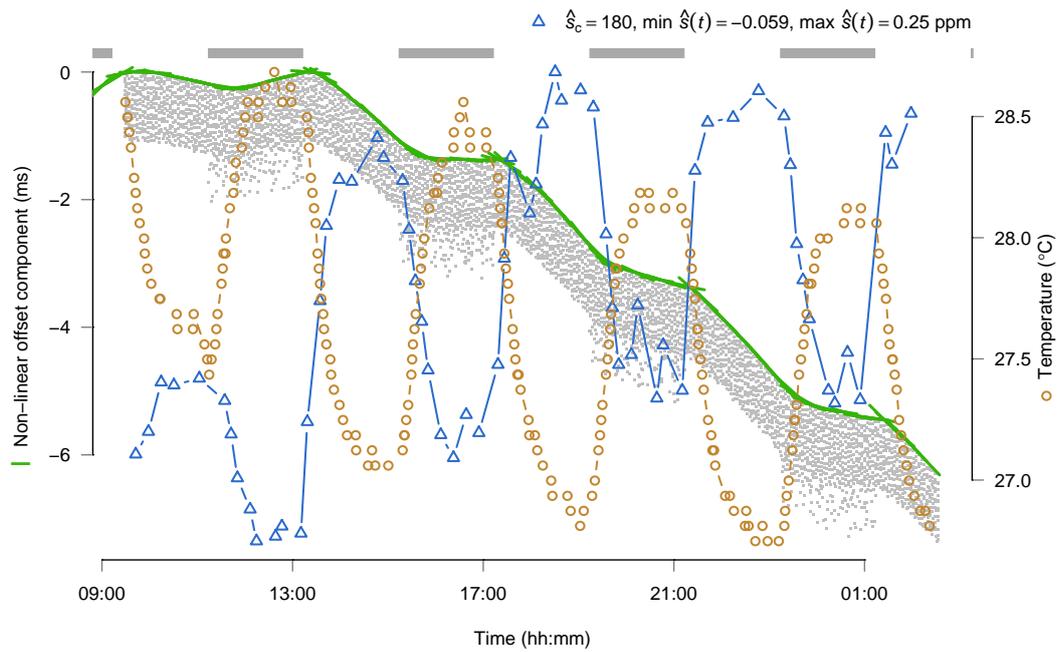
Overall throughput was limited by the CPU of the server hosting the private Tor network, so the fastest Hidden Server tested ran at around 70% CPU usage while requests were being made. CPU load on the Hidden Server was almost all due to the Tor process, I suspect as a direct result of the cryptographic operations it has to perform. A 1–1.5 °C temperature difference, as measured by the temperature probe, was induced by this load modulation.

Ideally, the measuring machine would have a very accurate clock, to allow comparison of results between different experiments over time and with different equipment. This was not available for these experiments, but as we are interested in relative skews, only a stable clock is needed, for which a normal PC sufficed.

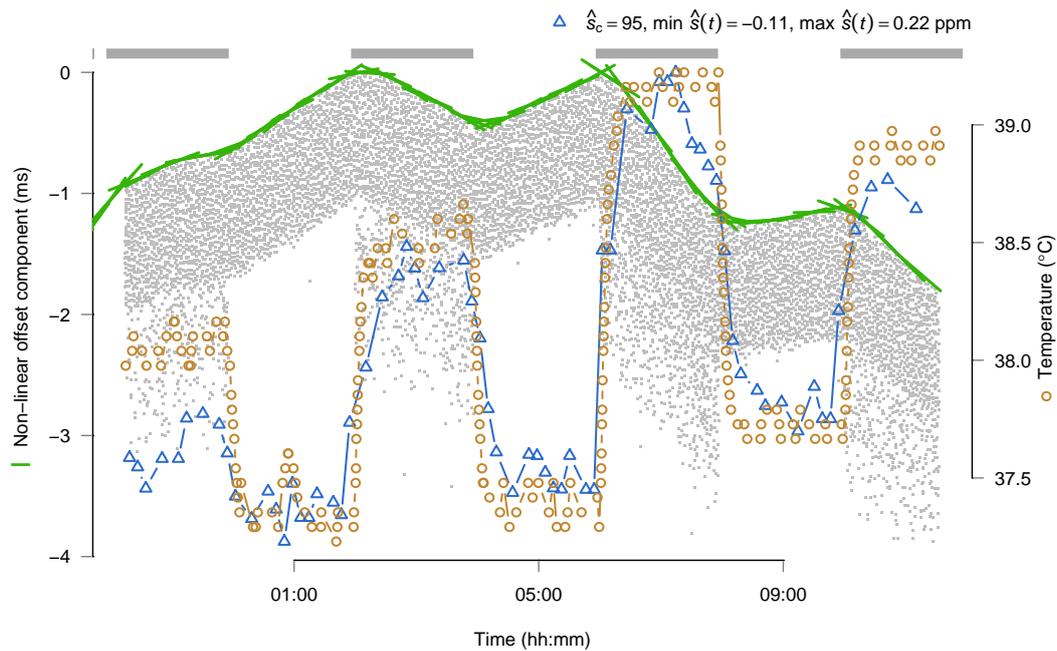
It would also be desirable to timestamp packets as near as possible to receipt, so while adding the timestamp at the network card would be preferable, the one inserted by the Linux kernel and exposed through the `pcap` [83] interface has proved to be adequate. Future work could make use of network cards with on-board timestamping.

Figure 5.7 shows the results of two experimental runs, in the same style as Figure 5.5. Note that Figure 5.7(a) shows a relationship between clock skew and temperature opposite to expectations; namely when temperature increases, the clock has sped up. One possible explanation is that the PC is using a temperature compensated crystal oscillator (TCXO), but is over-compensating; another is that the temperature curve for the crystal is different from Figure 5.4. In both cases, there is a clear correlation between temperature and skew, despite only a modest temperature change.

While the CPU is under load, there is increased noise present in the results. This could be due to increased latency on the network, or longer queues in between the steps of adding a timestamp to a packet and dispatching it. However, note that the minimum latency is unchanged (and is often reached) so the linear programming algorithm still performs well. Were the minimum to change, then a step in the graph would be expected, rather than the smooth curve apparent.



(a) Mini-tower PC with Dell GX1MT motherboard and Intel Pentium II 400 MHz processor



(b) Mini-tower PC with ASUS A7V133 motherboard and AMD Athlon 1.2 GHz processor

Figure 5.7: Clock skew measurements for two machines. The graphs are as Figure 5.5, but the grey bars at the top indicate when the hidden server was being exercised

#### 5.4.2 DISCUSSION

Implementing the non-interference solution described in the previous chapter is non-trivial because the QoS guarantees must not only be met by the host (e.g. CPU resources), but by its network too. Also, the impact on performance would likely be substantial, as many connections spend much of their time idle. Whereas currently the idle time would be given to other streams, now the host carrying such a stream cannot reallocate any resources. If the maximum number of connections is reached, further connections are refused, thus opening a DoS vulnerability. However, there may be some suitable compromise, for example dynamic limits which change sufficiently slowly to leak little information.

Even if such a defence were in place, our temperature attacks would still be effective. While changes in one network connection will not affect any other connections, clock skew is altered. This is because the CPU will remain idle during the slot allocated to a connection without pending data. Unless steps are taken to defend against our attacks, the reduced CPU load will lower temperature and hence affect clock skew. To stabilise temperature, computers could be modified to use expensive oven controlled crystal oscillators (OCXO), or always run at maximum CPU load. External access to timing information could be restricted or jittered, but unless all incoming connections were blocked, extensive changes would be required to hide low level information, such as packet emission triggered by timer interrupts.

While the above experiments were on Tor, I stress that the techniques apply to any system that hides load through maintaining QoS guarantees. Also, there is no need for the anonymity service to be the cause of the load. For example, Dean and Stubblefield [43] show that because SSL allows the client to force a server to perform a RSA operation before doing any substantial work itself, DoS attacks can be successful well before the connection is saturated. Such techniques could be used to attack hidden servers where the anonymity network cannot sustain high throughput.

The attack works because the normal abstraction of anonymous services does not take into account the imperfections inherent in physical devices. Previous such attacks have included Govindavajhala and Appel's demonstration of exploiting memory errors to violate access control in a Java VM [68] and the work by Kohno *et al.* [90] on fingerprinting particular hardware through its clock skew.

Inducing clock skew and remotely measuring it can be seen as a *thermal side channel* because attacking a hidden server could be modelled as violating an information flow control policy in a distributed system. The client accessing the hidden service over the anonymity network is making use of the link between between the server’s pseudonym and its public address, which is information at a “high” confidentiality level.

However, the client is prevented from leaking this information by the trusted computing base of the anonymity network. The user accessing the hidden server directly only has access to “low” information, the real address by itself, however if the “high” process can leak information to the “low” process, the server’s anonymity is violated.

## 5.5 Extensions and future work

The above experiments presented an example of how temperature induced clock skew can be a security risk, but I believe that this is a more general, and previously under-examined, technique which can be applied in other situations. In this section we shall explore some of these cases and propose some future directions for research.

### 5.5.1 CLASSICAL COVERT CHANNELS

The above section discussed an unconventional application of side channels, that is, within a distributed system where users can only send data but not execute arbitrary code. However, clock skew can also be used in conventional covert channels, where an operating system prevents two processes communicating which are on the same computer and are permitted, by the system security policy, to run arbitrary software.

CPU load channels have been extensively studied in the context of multi-level secure systems. Here, two processes share CPU time but the information flow control policy prohibits them from directly communicating. Each can still observe how much processing time it is getting, thus inferring the CPU usage of the other.

A process can thus signal to another by modulating load to encode information [93]. One defence against this attack is to distort the notion of time available to processes [79] but another is fixed scheduling and variations, ensuring that the CPU usage of one process cannot interfere with the resources of any other at a conflicting security rating [80]. Temperature induced clock skew

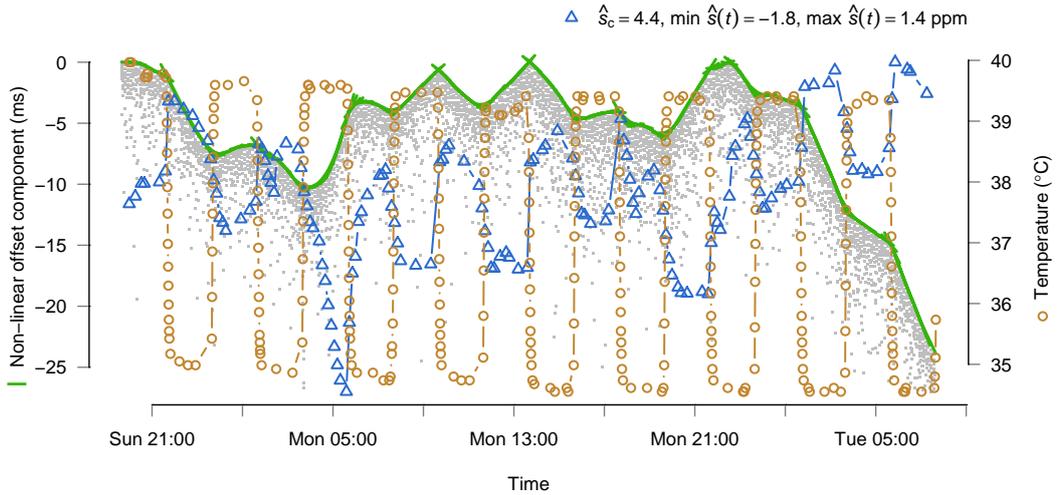


Figure 5.8: Clock skew measurements of a remote machine while modulating CPU load of the measurer (mini-tower, Intel D875 motherboard, Pentium 4 3.2 GHz CPU), for which temperature is also shown. The measurer and remote machine are separated by a transatlantic link, so the noise level is higher

can circumvent the latter countermeasure. Covert channels are also relevant to recently deployed separation kernels such as MILS [6, 151].

Figure 5.8 shows an example of how two processes on one host could communicate. In our previous experiments, the temperature in the measured machine has been modulated, but now we affect the clock skew of the measurer. This graph was plotted in the same way as before, but on the measurer machine, with NTP disabled, the `CPUBurn` program [129] was used to induce load modulation, affecting the temperature as shown. Timestamps are collected from a remote machine, to act as a time reference, and as we are calculating relative clock skew, we see the inverse of the measurer’s clock skew, assuming the remote clock is stable.

Note that the temperature difference is greater than before ( $5^\circ\text{C}$  vs.  $1\text{--}1.5^\circ\text{C}$ ). This is because we are no longer constrained by the capacity of the Tor network, and can optimise our procedure to induce the maximum temperature differential. While this attack is effective, it requires fairly free access to network resources, which may not always be the case in high-assurance systems where covert channels are a serious concern.

Where access to a remote timing source is blocked, the skew between multiple clock crystals within the same machine, due to their differing temperature responses and proximity to the heat source, could be used. For example, in a

typical PC, the sound card has a separate crystal from the system clock. A process could time how long it takes (according to the system clock), to record a given number of samples from the sound card, thus estimating the skew between the two crystals. USB controllers and Ethernet cards also commonly have separate crystals, but gaining access to the low-level timing results from these may be infeasible.

### 5.5.2 CROSS-COMPUTER COMMUNICATION

Physical properties of shared hardware have previously been proposed as a method of creating covert channels. For example, hard disk seek time can be used to infer the previous position of the disk arm, which could have been affected by “high” processes [86]. However, with temperature, such effects can extend across “air-gap” security boundaries.

My experiments so far have not shown evidence of one desktop computer being able to induce a significant temperature change in another which is in the same room, but the same may not be true of rack-mount machines. Here, a 3 °C temperature change in a rack-mount PC has been induced by increasing load on a neighbouring machine [70]. Blade servers, where multiple otherwise independent servers are mounted as cards in the same case, sharing ventilation and power, have even more potential for thermal coupling.

If two of these cards are processing data at different security levels, the tight environmental coupling could lead to a covert channel as above, even without the co-operation of the “low” card. For example, if a “low” webserver is hosted next to a “high” cryptographic co-processor which does not have Internet access, the latter could leak information to an external adversary by modulating temperature while the webserver clock-skew is measured. Side-channels are also conceivable, where someone probing one card could estimate the load of its siblings.

I simulated this case by periodically (2 hours on, 2 hours off) exposing a PC to an external heat source while a second computer measured the clock skew. The results showed that 3 °C temperature changes can be remotely received. Additionally, this experiment confirmed that it is indeed temperature causing the observed clock skew in the previous experiments, and not simply an OS scheduling artifact.

The resulting graph, shown in Figure 5.9, is similar to Figure 5.7, except there is no increased noise during heating, as would be expected from the hypothesised interference-attack resistant anonymity system.

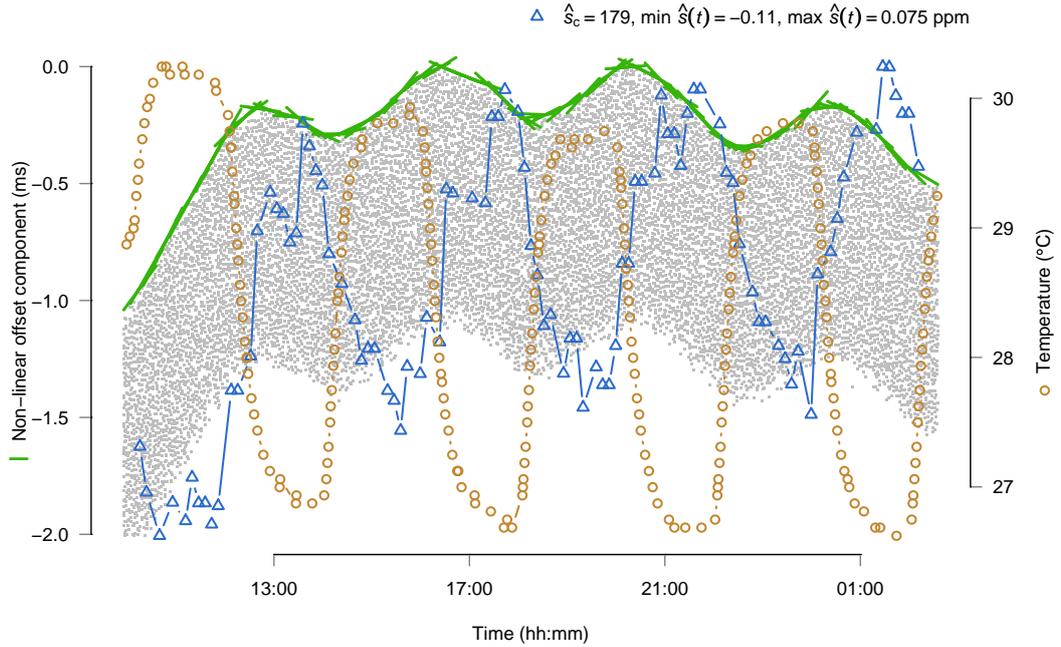


Figure 5.9: Clock skew measured while inducing temperature changes by an external heat source. Data collected from a mini-tower PC with Dell GX1MT motherboard and Intel Pentium II 400 MHz processor

### 5.5.3 GEOLOCATION

In the attacks on anonymity systems so far, we have been inducing load through the anonymity system and measuring clock skew directly. An alternative is to measure clock skew through the anonymity network and let the environment alter the clock skew. This allows an attacker to observe temperature changes of a hidden server, and so infer its location.

Clock skew does not allow the measurement of absolute temperature, only changes. Nevertheless, this still could be sufficient for geolocation. Longitude could simply be found by identifying sunrise or midday based on the temperature, to establish local time. To find latitude, the change in day length over a reasonably long period could be used. The relationship between these values and location is shown in Figure 5.10.

It was apparent in my experiments when a door to the cooler corridor was left open, so national holidays or when daylight saving time comes into effect might be evident. Distortion caused by air-conditioning could be removed by inferring the temperature from the duty cycle (time on vs. time off) of 2-point thermostatically controlled appliances.

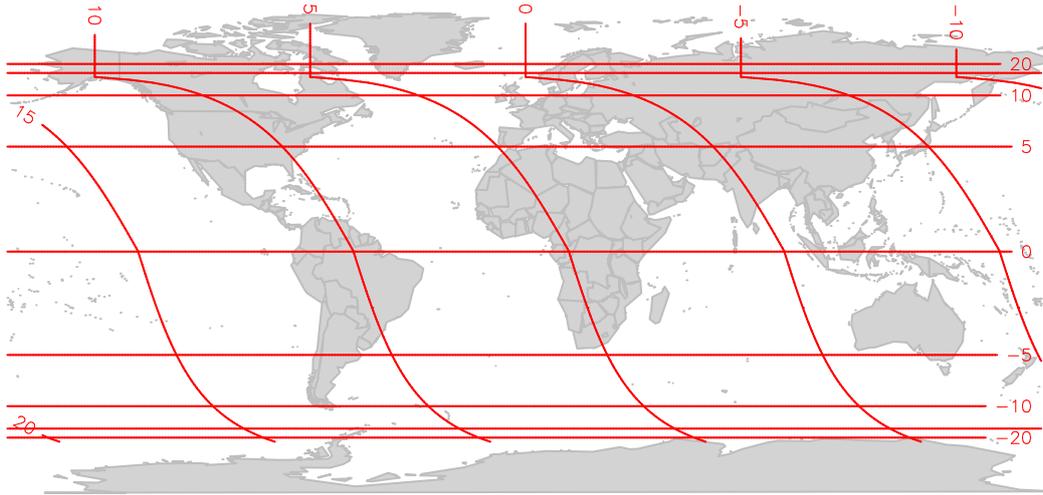


Figure 5.10: How temperature could be used to derive location. Vertical lines show time of sunrise (UTC) on 21 June 2006 (summer solstice). Horizontal lines show length of day (hours) on the summer solstice minus the length of day on 22 December 2006 (winter solstice)

In this section we have assumed that we probe through the anonymity network. In the case of Tor, this will introduce significant jitter, and it is unclear how badly this will affect timing measurements. Alternatively, the attacker could connect directly to the external address.

At first glance the utility of this attack appears questionable – often Internet protocol addresses can easily be mapped to locations [106]. However, this is not always the case. For example, anycast and satellite connections are hard to track to a location; as are users who seek to hide by using long-distance dialup. While latency in the last two cases is high, jitter can be very low, lending itself to the clock skew attacks.

#### 5.5.4 NOISE SOURCES AND MITIGATION

In Section 5.5.3, we proposed acquiring timing information from a hidden server through the anonymity network. Here, in addition to the problem of increased jitter, the timing sources we have used (ICMP/TCP timestamps and TCP sequence numbers) may not be available. For example, Tor operates at the TCP layer so these possibilities do not exist, unlike Freedom [13, 24] which allows the transmission of arbitrary IP packets.

One option proposed by Kohno *et al.* is to use a Fourier transform to extract the frequency of a periodic event, for example, packet emission caused by a

timer interrupt. Another possibility is to use application level timestamps. The most common Tor hidden service is a web server, typically using Apache, and by default this inserts a timestamp into HTTP headers. However, this only has a 1 Hz resolution, compared to the 1 kHz used in my experiments.

To improve performance in these adverse conditions by mitigating the effect of noise, we must first understand the source. The noise component of (5.3) is the sum of two independent parameters: quantisation noise  $c_i/h$  and latency  $d_i$ , although we only care about the variable component of the latter, *jitter*  $j_i$ . The quantisation noise is chosen uniformly at random from  $[0, 1/h)$ , and so is trivially modelled, but  $j_i$  can only be found experimentally.

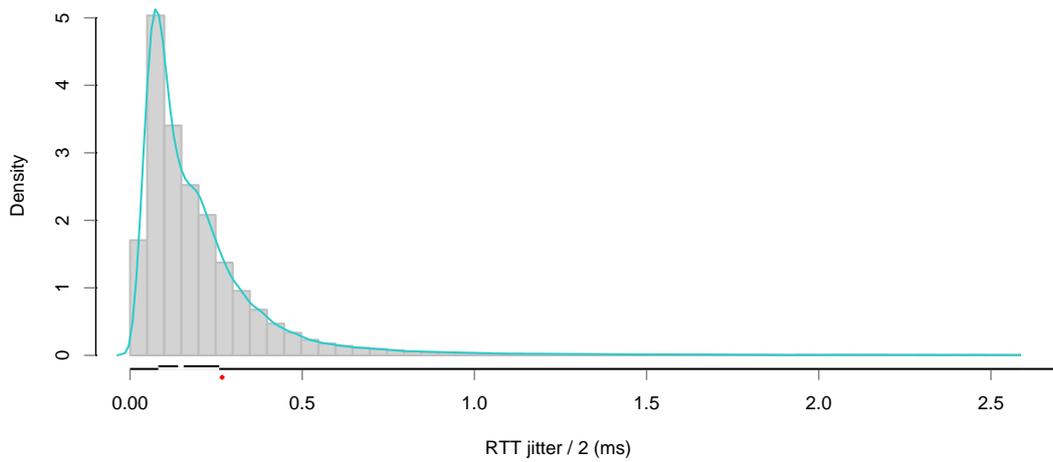
Figure 5.11(a) shows the smoothed probability density for round-trip jitter (divided by two), which can be measured directly. If we assume that forward and return paths have independent and similar jitter, then  $j_i$  would be the same distribution. By convolving the estimated densities of the two noise sources, we can show the probability density of the sum, which matches the noise measurements of clock offset shown in Figure 5.11(b).

The linear programming algorithm used for skew calculations is effective at removing  $j_i$ , because values are strongly skewed towards the minimum, but for  $c_i/h$ , it is possible to do better. One obvious technique is to increase  $h$  by selecting a higher resolution time source. We have used TCP timestamps in this chapter, primarily with Linux 2.6, which have a nominal frequency of 1 kHz. Linux 2.4 has a 100 Hz TCP timestamp clock, so for this, ICMP timestamps may be a better option, as they increment at a nominal 1 kHz.

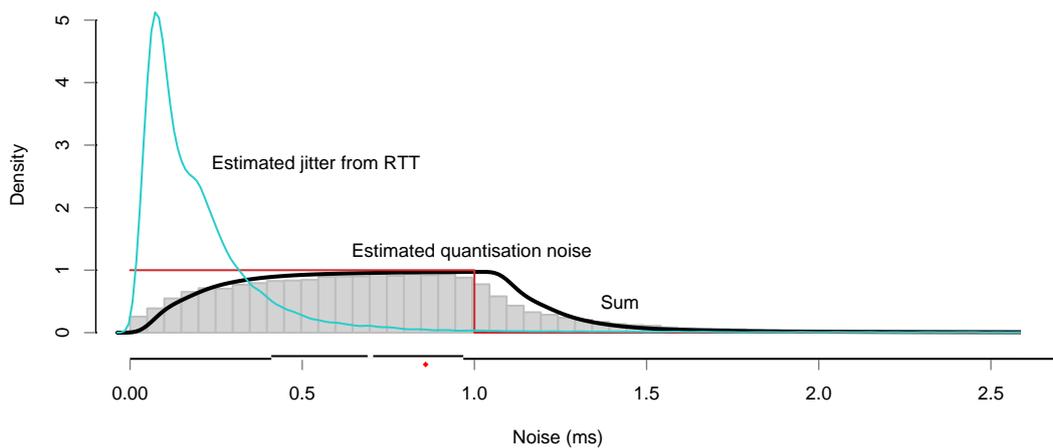
Unlike TCP timestamps, I found ICMP to be affected by NTP, but initial experiments show that while this is a problem for finding out absolute skew, the NTP controlled feedback loop [100] in Linux intentionally does not react quickly enough to hide the changes in skew this chapter considers. Another option with Linux is to use TCP sequence numbers, which are the sum of a cryptographic result and a 1 MHz clock. Over short periods, the high  $h$  gives good results, but as the cryptographic function is re-keyed every 5 minutes, maintaining long term clock skew figures is non-trivial.

Note that to derive (5.2) from (5.1) we assumed that samples are taken at random points between ticks. This allows the floor operation ( $\lfloor \cdot \rfloor$ ) to be modelled as uniformly distributed noise. Regular sampling introduces aliasing artifacts which interfere with the linear programming algorithm.

However, the points which contribute to the accuracy of the skew estimate, those near the top of the noise band, are from timestamps generated just after



(a) Probability density of measured round-trip time jitter (divided by two) with overlaid kernel density estimate (—)



(b) Density of measured offset noise, overlaid with the above density, uniform quantisation noise model (—) and the calculated sum of the two components (—)

Figure 5.11: Noise source comparison. The breaks in the  $x$  axis indicate quartiles and the mean is shown as  $\blacklozenge$ . Measurements were taken over a transatlantic link

a tick. Here, the value of  $c_i$  is close to zero, and just before the tick,  $c_i$  is close to one and the timestamp is one less. An attacker could use the previous estimate of skew to target this point and identify which side of the transition the sample was taken. From this, he can estimate when the tick occurred and so refine the skew estimate.

This approach effectively removes the quantisation error. Rather than  $1/h$  defining the noise band, it now only limits the sampling rate to  $h$ . Multiple measurements would still be needed to remove jitter, most likely by using the same linear programming algorithm as in the simple case, but perhaps also taking into consideration the round-trip time. Adequate results can be achieved using naïve random sampling, but the improved technique would be particularly valuable for low resolution clocks, such as the 1 Hz Apache HTTP timestamp mentioned in Section 5.5.4.

## 5.6 Conclusion

I have shown that changes in clock skew, resulting from only modest changes in temperature, can be remotely detected even over tens of router hops. My experiments show that environmental changes, as well as CPU load, can be inferred through these techniques. However, my primary contribution is to introduce an attack whereby CPU load induced through one communication channel, an anonymity network, affects clock skew measured through another. This technique can link a pseudonym to a real identity, even against a system that ensures perfect non-interference when considered in the abstract.

I have demonstrated how such attacks could be used against hidden services. I validated my expectations by testing them with the deployed Tor code, not simulations, although on a private network rather than the publicly accessible one. My results show that conjectured defences against interference attacks using quality of service guarantees are not as effective as conventional models might indicate. I suggest that when designing such systems, considering only the abstract operating system behaviour is inadequate as their implementation on real hardware can substantially decrease security.

I proposed future directions for security research using thermal covert channels. These include allowing two computers which share the same environment, but are otherwise isolated, to communicate. Also, processes on the same computer, under an information flow control policy, can send information through temperature modulation, despite fixed scheduling preventing CPU load based covert channels.

Finally, I discussed how localised temperature changes might aid geolocation and suggested methods to deal with low resolution clocks.

# Chapter 6

## Conclusions

Covert channels have historically been seen as being of exclusive relevance to multi-level secure systems (MLS), typically in military scenarios. As academic interest in MLS has waned, so has the perceived utility of covert channel research. At the same time, anonymity systems have moved from theoretical constructs, through research prototypes, into deployed systems with hundreds of thousands of users with diverse backgrounds and requirements.

Although both covert channel and anonymity research deal with blocking unauthorised flows of information, this link has not been well exploited. This thesis has examined the connection, particularly regarding how the wealth of practical experience in covert channel discovery can be applied to find and exploit weaknesses in real-world anonymity systems.

I hope that, in addition to their individual contributions, the cases studies given in this thesis demonstrate the usefulness of covert channel analysis in understanding, developing and defending against attacks on anonymity systems.

MLS operating system research originally considered the case of preventing communication between processes on the same hardware. Chapter 2 discusses such a case, but rather than leaking secret data, the attacker attempts to win a competition through collusion.

Information flow control is one solution to this weakness, a fact that is appreciated by some communities, such as Bridge players, but not all. Where an anonymity mechanism, one type of information flow control, is in place to inhibit cheating, or just where identity information is restricted by an accident of design, covert channels may be used for the identification of opponents.

Techniques for exploiting covert channels, in MLS operating systems, directly apply to this scenario. However, in the example of a Connect-4 pro-

gramming competition an application specific covert channel, of encoding data within the moves themselves, gave improved resistance to detection and elimination. This channel's reliability can be ensured through conventional error correction algorithms, but simpler authentication can be achieved with a coding similar to that used in spread spectrum, low probability of intercept, military radio communications.

Detection of covert channels in MLS systems has been well covered in the literature [65, 98] and these techniques apply just as well in the game scenario. For the new covert channel proposed in Chapter 2, defence is more difficult, but some techniques are discussed. This problem could be side-stepped entirely by negating the advantage of collusion so some options to achieve this are explored, in particular exploiting the parallels between deciding the winner in elections and games. However, results in this field are limited and suggest that totally collusion resistant competition mechanisms are impossible.

Where individual hosts process information at a single security level, local interprocess secrecy is of less concern, but networks become necessary for sharing information. The detailed mechanisms of operating-system covert-channel mitigation may no longer be applicable to network covert channels, but higher-level analysis will be. This fact is relevant to anonymity systems as they are typically distributed in order to give improved scalability and preserve user security under partial compromise.

Prior research in this area [63, 95, 136, 137] has found covert channels in network protocols which are not blocked by standard networking equipment and may comply with the respective protocol specification. In some cases, covert channel auditing may be an adequate, or even superior alternative to preventing unauthorised flows of information. Resistance to auditing requires indistinguishability, a property that previous proposals do not possess, since while their output might conform to the specification they diverge from the expected output of unmodified software.

Chapter 3 reviews existing network, in particular TCP/IP, covert channels, and compares them with the normal behaviour of Linux and OpenBSD. A series of tests are developed to detect the use of steganography and their effectiveness is compared. When designing schemes for detection-resistant steganography the chosen fields should not just contain redundancy, but the field content must also be unpredictable to the adversary. Only the TCP initial sequence number meets this requirement, because of its use in protecting against source-address spoofing.

Even though this field is unpredictable, for good network engineering reasons, it is not uniformly random. Prior use of this field in steganography systems [136, 137] has made this erroneous assumption and hence is detectable. Instead, I propose the generic principle of replicating the existing generation algorithm for encoding data to be carried by the covert channel, closely emulating the expected output. Encoding and decoding algorithms for Linux and OpenBSD are given as examples.

General purpose network covert channels such as these could be used by a Trojan horse program to “phone home”, even through an anonymity network which tunnels IP. However, scenarios where untrustworthy software is allowed such free network access are rare. Chapter 4 focuses on a more practical case where the side channel is the result of an unintended interaction between the activities of multiple users, on a node in the Tor anonymity system.

As a Tor node must transit multiple streams for any of them to receive anonymity, and the resources of the hardware hosting it are limited, high load on one stream will affect the throughput and latency of others. Just as two processes on a MLS operating system might communicate though signalling with increased CPU load, two Tor streams can signal through modulating network load. This phenomenon permits indirect traffic analysis attacks: while a weak attacker cannot directly monitor streams, their speed can be inferred by their effect on other streams flowing through the same node.

Where an attacker can modulate the load of a stream, but does not know the path it takes, by probing all candidate nodes, indirect traffic analysis can trace the stream back to the first Tor node connected to by the client. This reduces the anonymity provided to that of a single-hop proxy, and then mundane legal mechanisms might be used to discover the initiator. Furthermore, as circuits through Tor are used for multiple, unconnected streams, the discovery that two streams share the same path through the Tor network strongly suggests that they originate from the same client, in which case the anonymity provided is lower than a single-hop proxy.

The effectiveness of these attacks was evaluated by experiments on the deployed Tor network, which showed that, even with naïve signal processing, 11 out of the 13 Tor nodes tested could be de-anonymised, even after the signal passed through two Tor node hops. Variants of this attack are discussed, such as indirectly modulating traffic through carefully crafted denial-of-service attacks or not performing any modulation and detecting the effects of naturally occurring traffic patterns.

Such attacks could be prevented by ensuring that either all streams going through one node interfere perfectly, or not at all, but deploying these techniques would be difficult and have a substantial performance penalty. This should not be a surprise, given the experience of designers of covert channel resistant operating systems. In fact, Chapter 5 shows that by adding yet another level of indirection, an attacker can circumvent defences against indirect traffic analysis which insulate the performance of streams from CPU load.

Whereas the previous attack used stream latency to infer CPU load, the attack in Chapter 5 exploits the effect of CPU load on temperature and hence clock skew. The latter can be remotely measured over a network by requesting timestamps or as a side-effect of the TCP/IP covert-channel detection mechanisms of Chapter 3. This additional level of indirection inevitably adds extra noise, but with an adequate sample length this can be removed, even from results collected over a transatlantic Internet link.

This attack is evaluated on Tor hidden servers on a private network. The CPU load of the target is increased by making connections to its pseudonym over the Tor network. Simultaneously, clock skew measurements are collected, and from these the clock skew is derived. My results show this attack to be effective and while unlikely to be the fastest way to de-anonymise users of deployed anonymity networks, as systems become hardened against more conventional attacks, this attack could become a plausible threat.

Temperature as a side channel or covert channel has not previously been discussed in the literature and has wider applicability than attacking anonymity systems. Firstly, it could be used in MLS systems as a replacement to the CPU load covert channel, even when the latter is blocked by fixed CPU scheduling. The fact that heat can cross “air-gap” security boundaries can be exploited to create covert channels and side channels between computers that are electrically disconnected, but in physical proximity. Finally, even the geolocation of computers, albeit at low accuracy, might be accomplished by inferring the effect of weather through clock skew.

## 6.1 Future research directions

In each of the topics explored in this thesis there remain a number of possibilities for further research. With the continuing growth of the Internet gaming community and the pseudonymity it implies, preventing collusion is infeasible. Therefore, the construction of collusion-resistant competition structures could

be of substantial benefit, but they must be perceived as fair and moreover remain fun. Research on voting systems appears to be a fruitful source of ideas and although prior results show the impossibility of perfect elections, there may be adequate approximations which are better than the existing state of the art in games.

Although outside the remit of anonymity systems, covert channels for Trojan horse communication are a growing concern as the sophistication of malware authors increases. Covert channels also have more legitimate applications, such as bootstrapping censorship resistant publishing mechanisms. At higher levels of the protocol stack, the bandwidth of covert channels is larger, and the techniques presented in this thesis could be applied to exploit them. Also, a better understanding of the semantics of unmodified systems could improve covert channel detection. Timing characteristics remain largely unexplored in this respect.

Indirect traffic-analysis attacks could be extended by inducing load patterns through denial of service attacks, although legal concerns could constrain this being tested on the open Internet. Also, enhanced signal processing could give better recognition rates. This may be particularly relevant if the larger number of flows through the rapidly growing Tor network becomes sufficient to mask the induced signal. Equally, the increased capacity of the network could make the attacks more effective. Discovering how the attack scales with the size of the anonymity network would be a useful result in itself.

The potential of temperature covert channels remains unknown due to their early stage of investigation. Inter-computer communication and geolocation depend on the thermal conductivity and convection properties of computer installations, so comprehensive testing would be a substantial undertaking, but could be of interest to institutions with high security requirements. To explore the limits of geolocation, the measurement techniques would likely need to be refined to make better use of the low resolution clocks in this scenario. Here, there are additional complications of NTP synchronisation and interference from air-conditioning which may be removed through better signal processing.

There are a few common threads which run throughout this thesis that bear emphasising. One is finding security weaknesses by moving outside of conventional abstractions. The key observation of Chapter 2 was that the goal of the competition was not to play Connect-4 well, but to win the league, a fact not appreciated by other entrants. Chapter 3 exploited the fact that previous steganography schemes assume that if a field is permitted to be random, it

actually will be random. Chapter 4 and Chapter 5 rely on the Tor scheduling algorithm, and temperature dependent clock skew, respectively, which are normally not considered security critical components and hence are excluded from analysis.

In the process of breaking out of traditional models, there remains the risk that research, despite being internally consistent, will ultimately be inapplicable to real systems. For this reason, wherever feasible, I have tested my new techniques in situations as close to real-life as possible. Actually implementing these attacks, rather than relying on their theoretical properties, had significant beneficial side-effects.

For instance, it was only on testing the covert channel detection mechanisms in Chapter 3 that I realised that clock skew could be estimated from initial sequence numbers. Also, the idea behind Chapter 5 originated from the experience gained in Chapter 4 and from attempting to explain unusual patterns in noise measurements taken of clock skew estimates. These patterns turned out to be caused by temperature changes when the computer performed night-time maintenance tasks.

However, the phenomenon of temperature-induced clock skew is not an attack in itself. This is where techniques for covert channel discovery, such as shared resource matrices [87] may be applied to find flows between supposedly unlinkable constructs (streams through Tor nodes in this case). Although, in this example, extending the prior work on indirect traffic analysis was fairly straightforward, the rigour of systematic covert-channel analysis techniques was intended to detect all covert channels. These methodologies could be applied to give some assurance that no covert channels or side channel attacks remained, at least resisting casual lapses during analysis.

The case studies presented here have shown how covert channel analysis techniques have discovered hitherto completely unexpected practical attacks on anonymity systems and related infrastructure. Furthermore, the mechanisms for exploiting, preventing and detecting covert channels can be directly applied in situations other than the MLS scenario they were designed for. I hope this thesis will encourage future work on exploiting the vast literature of covert channel analysis for the advancement of anonymous communications research.

# Bibliography

- [1] A. Acquisti, R. Dingledine, and P. F. Syverson. On the economics of anonymity. In R. N. Wright, editor, *Proceedings of Financial Cryptography (FC '03)*, volume 2742 of *LNCS*, pages 84–102, Guadeloupe, French West Indies, January 2003. Springer.
- [2] D. Agrawal, D. Kesdogan, and S. Penz. Probabilistic treatment of mixes to hamper traffic analysis. In *IEEE Symposium on Security and Privacy*, pages 16–27, Berkeley, CA, US, May 2003. IEEE Computer Society.
- [3] K. Ahsan and D. Kundur. Practical data hiding in TCP/IP. In *ACM Workshop on Multimedia and Security*, Juan-les-Pins, France, December 2002. ACM Press. <http://ee.tamu.edu/~deepa/pdf/acm02.pdf> (checked 2007-06-14).
- [4] L. V. Allis. A knowledge-based approach of Connect-Four. Master's thesis, Vrije Universiteit, Amsterdam, The Netherlands, October 1988. [citeseer.ist.psu.edu/allis88knowledgebased.html](http://citeseer.ist.psu.edu/allis88knowledgebased.html) (checked 2007-06-14).
- [5] A. Alsaïd and D. Martin. Detecting web bugs with Bugnosis: Privacy advocacy through education. In R. Dingledine and P. Syverson, editors, *Privacy Enhancing Technologies (PET)*, volume 2482 of *LNCS*, pages 27–31, San Francisco, CA, US, April 2002. Springer.
- [6] J. Alves-Foss, C. Taylor, and P. Omanl. A multi-layered approach to security in high assurance systems. In *Proceedings of the 37th Hawaii International Conference on System Sciences*, HI, US, January 2004. IEEE Computer Society.
- [7] American Contract Bridge League. *Laws of Contract Bridge (Rubber Bridge Laws, American Edition)*, January 1993.

- [8] American Contract Bridge League. *Laws of Duplicate Contract Bridge (American Edition)*, May 1997. Law 20. Review and Explanation of Calls.
- [9] R. Anderson. *Security Engineering*. Wiley, 1st edition, 2001.
- [10] F. Anklesaria, M. McCahill, P. Lindner, D. Johnson, D. Torrey, and B. Alberti. The Internet gopher protocol (a distributed document search and retrieval protocol). RFC 1436, IETF, March 1993.
- [11] K. J. Arrow. *Social Choice and Individual Values*. Yale University Press, second edition, October 1970.
- [12] R. Axelrod. *The Evolution of Cooperation*. BASIC Books, NY, US, 1984.
- [13] A. Back, I. Goldberg, and A. Shostack. Freedom Systems 2.1 security issues and analysis. White paper, Zero Knowledge Systems, Inc., May 2001. [http://osiris.978.org/~brianr/crypto-research/anon/www.freedom.net/products/whitepapers/Freedom\\_Security2-1.pdf](http://osiris.978.org/~brianr/crypto-research/anon/www.freedom.net/products/whitepapers/Freedom_Security2-1.pdf) (checked 2007-06-14).
- [14] A. Back, U. Möller, and A. Stiglic. Traffic analysis attacks and trade-offs in anonymity providing systems. In I. S. Moskowitz, editor, *Information Hiding Workshop (IH 2001)*, volume 2137 of *LNCS*, pages 245–257, Pittsburgh, PA, US, April 2001. Springer.
- [15] BBC News. US blogger fired by her airline, November 2004. <http://news.bbc.co.uk/1/technology/3974081.stm> (checked 2007-06-14).
- [16] D. E. Bell and L. J. LaPadula. Secure computer systems: Mathematical foundations. Technical Report 2547, Volume I, MITRE Corporation, March 1973.
- [17] S. Bellovin. Defending against sequence number attacks. RFC 1948, IETF, May 1996.
- [18] S. M. Bellovin. Security problems in the TCP/IP protocol suite. *Computer Communication Review*, 19(2):32–48, April 1989.
- [19] O. Berthold, H. Federrath, and S. Köpsell. Web MIXes: A system for anonymous and unobservable Internet access. In H. Federrath, editor, *Designing Privacy Enhancing Technologies*, volume 2009 of *LNCS*, pages 115–129, Berkeley, CA, US, July 2000. Springer.

- [20] O. Berthold, A. Pfitzmann, and R. Standtke. The disadvantages of free MIX routes and how to overcome them. In H. Federrath, editor, *Designing Privacy Enhancing Technologies*, volume 2009 of *LNCS*, pages 30–45, Berkeley, CA, US, July 2000. Springer.
- [21] K. J. Biba. Integrity considerations for secure computer systems. Technical Report 3153, MITRE Corporation, April 1977.
- [22] G. D. Bissias, M. Liberatore, D. Jensen, and B. N. Levine. Privacy vulnerabilities in encrypted HTTP streams. In G. Danezis and D. Martin, editors, *Privacy Enhancing Technologies (PET)*, volume 3856 of *LNCS*, pages 1–11, Cavtat, Croatia, May 2005. Springer.
- [23] A. Blum, D. Song, and S. Venkataraman. Detection of interactive stepping stones: Algorithms and confidence bounds. In *Recent Advances in Intrusion Detection: 7th International Symposium, RAID 2004*, Sophia Antipolis, France, September 2004.
- [24] P. Boucher, A. Shostack, and I. Goldberg. Freedom Systems 2.0 architecture. White paper, Zero Knowledge Systems, Inc., December 2000. [http://osiris.978.org/~brianr/crypto-research/anon/www.freedom.net/products/whitepapers/Freedom\\_System\\_2\\_Architecture.pdf](http://osiris.978.org/~brianr/crypto-research/anon/www.freedom.net/products/whitepapers/Freedom_System_2_Architecture.pdf) (checked 2007-06-14).
- [25] S. L. Brand. *DoD 5200.28-STD Department of Defense Trusted Computer System Evaluation Criteria (Orange Book)*. National Computer Security Center, December 1985.
- [26] C. Bryan-Low. Vodafone, Ericsson get hung up in Greece’s phone-tap scandal. *Wall Street Journal*, 21 June 2006.
- [27] C-MAC MicroTechnology. HC49/4H SMX crystals datasheet, September 2004. <http://www.farnell.com/datasheets/64642.pdf> (checked 2007-06-14).
- [28] 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*, pages 178–187, Washington, DC, US, October 2004. ACM Press.

- [29] Cambridge University Computing Society. Winter programming competition, December 2002. <http://www.cucs.ucam.org/competition.html> (checked 2007-06-14).
- [30] Center for High Assurance Computer Systems, US Naval Research Lab. Onion routing program, 1995. <http://www.onion-router.net/> (checked 2007-06-14).
- [31] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–88, February 1981.
- [32] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. PlanetLab: An overlay testbed for broad-coverage services. *ACM SIGCOMM Computer Communication Review*, 33(3), July 2003.
- [33] D. D. Clark and D. R. Wilson. A comparison of commercial and military computer security policies. In *IEEE Symposium on Security and Privacy*, pages 184–195, Oakland, CA, US, April 1987. IEEE Computer Society.
- [34] R. Clayton, G. Danezis, and M. G. Kuhn. Real world patterns of failure in anonymity systems. In I. S. Moskowitz, editor, *Information Hiding Workshop (IH 2001)*, volume 2137 of *LNCS*, pages 230–245, Pittsburgh, PA, US, April 2001. Springer.
- [35] Common Criteria Project. *Common Criteria for Information Technology Security Evaluation, version 3.1*, September 2006. Part 1: Introduction and general model (revision 1).
- [36] W. Dai. Two attacks against Freedom, 2000. <http://www.eskimo.com/~weidai/freedom-attacks.txt> (checked 2007-06-14).
- [37] G. Danezis. Statistical disclosure attacks. In Gritzalis, Vimercati, Samarati, and Katsikas, editors, *Security and Privacy in the Age of Uncertainty, (SEC2003)*, pages 421–426, Athens, Greece, May 2003. IFIP TC11, Kluwer.
- [38] G. Danezis. Traffic analysis of the HTTP protocol over TLS, 2003. <http://homes.esat.kuleuven.be/~gdanezis/TLSanon.pdf> (checked 2007-06-14).

- [39] G. Danezis. Designing and attacking anonymous communication systems. Technical Report UCAM-CL-TR-594, University of Cambridge, Computer Laboratory, July 2004.
- [40] G. Danezis. The traffic analysis of continuous-time mixes. In D. Martin and A. Serjantov, editors, *Privacy Enhancing Technologies (PET)*, volume 3424 of *LNCS*, pages 35–50, Toronto, Canada, May 2004. Springer.
- [41] G. Danezis, R. Dingledine, and N. Mathewson. Mixminion: Design of a type III anonymous remailer protocol. In *IEEE Symposium on Security and Privacy*, pages 2–15, Berkeley, CA, US, May 2003. IEEE Computer Society.
- [42] T. de Raadt, N. Hallqvist, A. Grabowski, A. D. Keromytis, and N. Provos. Cryptography in OpenBSD: An overview. In *Proceedings of the USENIX Annual Technical Conference (FREENIX Track)*, pages 93–102, Monterey, CA, US, June 1999. USENIX.
- [43] D. Dean and A. Stubblefield. Using client puzzles to protect TLS. In *Proceedings of the 10th USENIX Security Symposium*, Washington, DC, US, August 2001. USENIX.
- [44] C. Díaz, S. Seys, J. Claessens, and B. Preneel. Towards measuring anonymity. In R. Dingledine and P. Syverson, editors, *Proceedings of Privacy Enhancing Technologies Workshop (PET)*, volume 2482 of *LNCS*, pages 184–188, San Francisco, CA, US, April 2002. Springer.
- [45] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) protocol version 1.1. RFC 4346, IETF, April 2006.
- [46] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.
- [47] R. Dingledine and N. Mathewson. Tor protocol specification. Technical report, The Free Haven Project, October 2004. <http://tor.eff.org/cvs/doc/tor-spec.txt> (checked 2007-06-14).
- [48] R. Dingledine and N. Mathewson. Tor path specification. Technical report, The Free Haven Project, April 2006. <http://tor.eff.org/cvs/doc/path-spec.txt> (checked 2007-06-14).

- [49] R. Dingledine and N. Mathewson. Tor rendezvous specification. Technical report, The Free Haven Project, February 2006. <http://tor.eff.org/cvs/doc/rend-spec.txt> (checked 2007-06-14).
- [50] R. Dingledine, N. Mathewson, and P. F. Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, San Diego, CA, US, August 2004. USENIX.
- [51] H. Dobbertin. Cryptanalysis of MD4. *Journal of Cryptology*, 11(4):253–271, November 1998.
- [52] B. Dutta, M. O. Jackson, and M. L. Breton. Strategic candidacy and voting procedures. *Econometrica*, 69(4):1013–1038, 2001.
- [53] B. Dutta, M. O. Jackson, and M. L. Breton. Voting by successive elimination and strategic candidacy. *Journal of Economic Theory*, 103:190–218, 2002.
- [54] E. W. Felten and M. A. Schneider. Timing attacks on web privacy. In *CCS '00: Proceedings of the 7th ACM Conference on Computer and Communications Security*, pages 25–32, Athens, Greece, November 2000. ACM Press.
- [55] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, IETF, June 1999.
- [56] G. Fisk, M. Fisk, C. Papadopoulos, and J. Neil. Eliminating steganography in Internet traffic with active wardens. In F. A. P. Petitcolas, editor, *Information Hiding Workshop (IH 2002)*, volume 2578 of *LNCS*, pages 18–35, Noordwijkerhout, The Netherlands, October 2002. Springer.
- [57] M. J. Freedman and R. Morris. Tarzan: A peer-to-peer anonymizing network layer. In *CCS '02: Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 193–206, Washington, DC, US, November 2002. ACM Press.
- [58] M. J. Freedman, E. Sit, J. Cates, and R. Morris. Introducing Tarzan, a peer-to-peer anonymizing network layer. In P. Druschel, M. F. Kaashoek, and A. I. T. Rowstron, editors, *International Workshop on Peer-to-Peer Systems (IPTPS)*, volume 2429 of *LNCS*, pages 121–129, Cambridge, MA, US, March 2002. Springer.

- [59] X. Fu, Y. Zhu, B. Graham, R. Bettati, and W. Zhao. On flow marking attacks in wireless anonymous communication networks. In *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems*, pages 493–503, Columbus, OH, US, June 2005. IEEE Computer Society.
- [60] Fyodor. Nmap: Idle scanning and related IPID games, July 2001. <http://www.insecure.org/nmap/idlescan.html> (checked 2007-06-14).
- [61] P. Gärdenfors. Manipulations of social choice functions. *Journal of Economic Theory*, 13:217–228, 1976.
- [62] A. Gibbard. Manipulation of voting schemes: A general result. *Econometrica*, 41(4):587–601, 1973.
- [63] J. Giffin, R. Greenstadt, P. Litwack, and R. Tibbetts. Covert messaging in TCP. In R. Dingledine and P. Syverson, editors, *Privacy Enhancing Technologies (PET)*, volume 2482 of *LNCS*, pages 194–208, San Francisco, CA, US, April 2002. Springer.
- [64] C. G. Girling. Covert channels in LAN's. *IEEE Transactions on Software Engineering*, SE-13(2):292–296, February 1987.
- [65] V. D. Gligor. *DoD NCSC-TG-030 A Guide to Understanding Covert Channel Analysis of Trusted Systems (Light-Pink Book)*. National Computer Security Center, November 1993.
- [66] I. Goldberg. *A Pseudonymous Communications Infrastructure for the Internet*. PhD thesis, UC Berkeley, December 2000.
- [67] D. M. Goldschlag, M. G. Reed, and P. F. Syverson. Onion routing. *Communications of the ACM*, 42(2):39–41, February 1999.
- [68] S. Govindavajhala and A. Appel. Using memory errors to attack a virtual machine. In *IEEE Symposium on Security and Privacy*, pages 154–165, Oakland, CA, US, May 2003. IEEE Computer Society.
- [69] W. M. Grossman. New tack wins prisoner's dilemma. Wired News, October 2004. <http://www.wired.com/culture/lifestyle/news/2004/10/65317> (checked 2007-06-14).
- [70] H. Grundy. Personal communication, August 2006.

- [71] Guirguis, Mina, Bestavros, Azer, and I. Matta. Exploiting the transients of adaptation for RoQ attacks on Internet resources. In *Proceedings of ICNP'04: The 12th IEEE International Conference on Network Protocols*, Berlin, Germany, October 2004.
- [72] C. Gülcü and G. Tsudik. Mixing E-mail with Babel. In *Proceedings of the Network and Distributed Security Symposium – NDSS '96*, pages 2–16, San Diego, CA, US, February 1996. The Internet Society.
- [73] T. Handel and M. Sandford. Hiding data in the OSI network model. In R. Anderson, editor, *Information Hiding Workshop (IH 1996)*, volume 1174 of *LNCS*, pages 23–38, Cambridge, UK, May/June 1996. Springer.
- [74] M. Handley, V. Paxson, and C. Kreibich. Network intrusion detection: Evasion, traffic normalization, and end-to-end protocol semantics. In *Proceedings of the 10th USENIX Security Symposium*, Washington, DC, US, August 2001. USENIX.
- [75] S. Helmers. A brief history of anon.penet.fi – the legendary anonymous remailer. *Computer-Mediated Communication Magazine*, September 1997. <http://www.december.com/cmc/mag/1997/sep/helmers.html> (checked 2007-06-14).
- [76] M. Herman. *Intelligence Power in Peace and War*. Cambridge University Press, October 1996.
- [77] J. C. Hernandez-Castro, I. Blasco-Lopez, J. M. Estevez-Tapiador, and A. Ribagorda-Garnacho. Steganography in games: A general methodology and its application to the game of Go. *Computers & Security*, 25(1):64–71, February 2006.
- [78] A. Hintz. Covert channels in TCP and IP headers. Presentation at DEFCON 10, August 2002. <http://guh.nu/projects/cc/> (checked 2007-06-14).
- [79] W.-M. Hu. Reducing timing channels with fuzzy time. In *IEEE Symposium on Security and Privacy*, pages 8–20, Oakland, CA, US, May 1991. IEEE Computer Society.
- [80] W.-M. Hu. Lattice scheduling and covert channels. In *IEEE Symposium on Security and Privacy*, pages 52–61, Oakland, CA, US, May 1992. IEEE Computer Society.

- [81] J. C. Huskamp. Covert communication channels in timesharing systems. Technical Report UCB-CS-78-02, University of California, Berkeley, CA, US, 1978.
- [82] V. Jacobson, R. Braden, and D. Borman. TCP extensions for high performance. RFC 1323, IETF, May 1992.
- [83] V. Jacobson, C. Leres, and S. McCanne. libpcap, March 2004. <http://www.tcpdump.org/> (checked 2007-06-14).
- [84] A. Juels, M. Jakobsson, and T. N. Jagatic. Cache cookies for browser authentication. In *IEEE Symposium on Security and Privacy*, pages 301–305, Oakland, CA, US, May 2006. IEEE Computer Society.
- [85] 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*, pages 119–129, Fairfax, VA, US, November 1993. ACM Press.
- [86] P. A. Karger and J. C. Wray. Storage channels in disk arm optimization. In *IEEE Symposium on Security and Privacy*, pages 52–63, Oakland, CA, US, May 1991. IEEE Computer Society.
- [87] R. A. Kemmerer. Shared resource matrix methodology: An approach to identifying storage and timing channels. *ACM Transactions on Computer Systems*, 1(3):256–277, 1983.
- [88] D. Kesdogan, D. Agrawal, and S. Penz. Limits of anonymity in open environments. In F. A. P. Petitcolas, editor, *Information Hiding Workshop (IH 2002)*, volume 2578 of *LNCS*, pages 53–69, Noordwijkerhout, The Netherlands, October 2002. Springer.
- [89] D. E. Knuth. *The Art of Computer Programming*, volume 2, Seminumerical Algorithms. Addison-Wesley, third edition, 1998.
- [90] T. Kohno, A. Broido, and k. claffy. Remote physical device fingerprinting. In *IEEE Symposium on Security and Privacy*, pages 211–225, Oakland, CA, US, May 2005. IEEE Computer Society.
- [91] M. G. Kuhn. Compromising emanations: eavesdropping risks of computer displays. Technical Report UCAM-CL-TR-577, University of Cambridge, Computer Laboratory, December 2003.

- [92] M. G. Kuhn. Personal communication, April 2006.
- [93] B. W. Lampson. A note on the confinement problem. *Communications of the ACM*, 16(10):613–615, October 1973.
- [94] B. N. Levine, M. K. Reiter, C. Wang, and M. K. Wright. Timing attacks in low-latency mix-based systems. In A. Juels, editor, *Proceedings of Financial Cryptography (FC '04)*, volume 3110 of *LNCS*, pages 251–265, Key West, FL, US, February 2004. Springer.
- [95] N. B. Lucena, G. Lewandowski, and S. J. Chapin. Covert channels in IPv6. In G. Danezis and D. Martin, editors, *Privacy Enhancing Technologies (PET)*, volume 3856 of *LNCS*, pages 147–166, Cavtat, Croatia, May 2005. Springer.
- [96] M. Martinec. Temperature dependency of a quartz oscillator. <http://www.ijs.si/time/#temp-dependency> (checked 2007-06-14).
- [97] N. Mathewson and R. Dingledine. Practical traffic analysis: Extending and resisting statistical disclosure. In D. Martin and A. Serjantov, editors, *Privacy Enhancing Technologies (PET)*, volume 3424 of *LNCS*, pages 17–34, Toronto, Canada, May 2004. Springer.
- [98] J. McHugh. Covert channel analysis. Technical Memo 5540:080A, Naval Research Laboratory, 1995. A Chapter of the Handbook for the Computer Security Certification of Trusted Systems.
- [99] J. K. Millen. Finite-state noiseless covert channels. In *Proceedings of the Computer Security Foundations Workshop*, pages 81–85, Franconia, NH, US, June 1989.
- [100] D. Mills. A kernel model for precision timekeeping. RFC 1589, IETF, March 1994.
- [101] D. L. Mills. Network time protocol (version 3) specification, implementation and analysis. RFC 1305, IETF, March 1992.
- [102] J. Mogul and S. Deering. Path MTU discovery. RFC 1191, IETF, November 1990.

- [103] S. B. Moon, P. Skelly, and D. Towsley. Estimation and removal of clock skew from network delay measurements. Technical Report 98-43, Department of Computer Science University of Massachusetts at Amherst, October 1998.
- [104] I. S. Moskowitz, R. E. Newman, D. P. Crepeau, and A. R. Miller. Covert channels and anonymizing networks. In *Workshop on Privacy in the Electronic Society (WPES 2003)*, pages 79–88, Washington, DC, US, October 2003. ACM Press.
- [105] I. S. Moskowitz, R. E. Newman, and P. F. Syverson. Quasi-anonymous channels. In M. Hamza, editor, *IASTED Communication, Network, and Information Security*, pages 126–131, New York, NY, US, December 2003. ACTAPress.
- [106] J. A. Muir and P. C. van Oorschot. Internet geolocation and evasion. Technical Report TR-06-05, Carleton University, School of Computer Science, April 2006.
- [107] S. J. Murdoch. Hot or not: Revealing hidden services by their clock skew. In *CCS '06: Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 27–36, Alexandria, VA, US, October 2006. ACM Press.
- [108] S. J. Murdoch and G. Danezis. Low-cost traffic analysis of Tor. In *IEEE Symposium on Security and Privacy*, pages 183–195, Oakland, CA, US, May 2005. IEEE Computer Society.
- [109] S. J. Murdoch and S. Lewis. Embedding covert channels into TCP/IP. In M. Barni, J. Herrera-Joancomartí, S. Katzenbeisser, and F. Pérez-González, editors, *Information Hiding Workshop (IH 2005)*, volume 3727 of *LNCS*, pages 247–261, Barcelona, Catalonia (Spain), June 2005. Springer.
- [110] S. J. Murdoch and P. Zieliński. Covert channels for collusion in online computer games. In J. Fridrich, editor, *Information Hiding Workshop (IH 2004)*, volume 3200 of *LNCS*, pages 355–369, Toronto, Canada, May 2004. Springer.

- [111] S. J. Murdoch and P. Zieliński. Sampled traffic analysis by Internet-exchange-level adversaries. In *Privacy Enhancing Technologies (PET)*, LNCS, Ottawa, Canada, June 2007. Springer. (to appear).
- [112] R. M. Needham. Denial of service. In *CCS '93: Proceedings of the 1st ACM Conference on Computer and Communications Security*, pages 151–153, Fairfax, VA, US, November 1993. ACM Press.
- [113] R. M. Needham. Denial of service: An example. *Communications of the ACM*, 37(11):42–46, November 1994.
- [114] R. E. Newman, V. R. Nalla, and I. S. Moskowitz. Anonymity and covert channels in simple timed mix-firewalls. In D. Martin and A. Serjantov, editors, *Privacy Enhancing Technologies (PET)*, volume 3424 of LNCS, pages 1–16, Toronto, Canada, May 2004. Springer.
- [115] K. Nichols, S. Blake, F. Baker, and D. Black. Definition of the differentiated services field (DS field) in the IPv4 and IPv6 headers. RFC 2474, IETF, December 1998.
- [116] A. Oram, editor. *Peer-to-peer: Harnessing the Benefits of a Disruptive Technology*, chapter 7, pages 89–93. O'Reilly & Associates, March 2001.
- [117] L. Øverlier and P. F. Syverson. Locating hidden servers. In *IEEE Symposium on Security and Privacy*, pages 100–114, Oakland, CA, US, May 2006. IEEE Computer Society.
- [118] S. Parekh. Prospects for remailers. *First Monday*, 1(2), August 1996. <http://www.firstmonday.org/issues/issue2/remailers/> (checked 2007-06-14).
- [119] A. Pfitzmann and M. Hansen. Anonymity, unlinkability, unobservability, pseudonymity, and identity management – a consolidated proposal for terminology. Draft, version 0.28, May 2006. Latest version available at [http://dud.inf.tu-dresden.de/Anon\\_Terminology.shtml](http://dud.inf.tu-dresden.de/Anon_Terminology.shtml) (checked 2007-06-14).
- [120] A. Pfitzmann, B. Pfitzmann, and M. Waidner. ISDN-mixes: Untraceable communication with very small bandwidth overhead. In W. Effelsberg, H. W. Meuer, and G. Müller, editors, *GI/ITG Conference on Communication in Distributed Systems*, volume 267 of *Informatik-Fachberichte*, pages 451–463. Springer, February 1991.

- [121] B. Pfitzmann and A. Pfitzmann. How to break the direct RSA-implementation of MIXes. In J.-J. Quisquater and J. Vandewalle, editors, *Advances in Cryptology – EUROCRYPT ’89: Workshop on the Theory and Application of Cryptographic Techniques*, volume 434 of *LNCS*, pages 373–381, Houthalen, Belgium, April 1989. Springer.
- [122] C. P. Pfleeger and S. L. Pfleeger. *Security in Computing*. Prentice Hall, 4th edition, 2006.
- [123] J. Postel. Internet Control Message Protocol. RFC 792, IETF, September 1981.
- [124] J. Postel. Internet Protocol. RFC 791, IETF, September 1981.
- [125] J. Postel. Transmission Control Protocol. RFC 793, IETF, September 1981.
- [126] J. Postel. Daytime protocol. RFC 867, IETF, May 1983.
- [127] V. Prevelakis and D. Spinellis. The Athens affair. *IEEE Spectrum*, 44(7):26–33, July 2007.
- [128] R Development Core Team. *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria, 2004. ISBN 3-900051-07-0 <http://www.R-project.org/> (checked 2007-06-14).
- [129] R. Redelmeier. CPUBurn, June 2001. <http://pages.sbcglobal.net/redelm/> (checked 2007-06-14).
- [130] M. G. Reed, P. F. Syverson, and D. M. Goldschlag. Anonymous connections and onion routing. *IEEE Journal on Selected Areas in Communications*, 16(4):482–494, May 1998.
- [131] M. Reiter and A. Rubin. Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security*, 1(1), June 1998.
- [132] M. Reiter and A. Rubin. Anonymity loves company: Anonymous web transactions with Crowds. *Communications of the ACM*, 42(2):32–38, February 1999.

- [133] M. Rennhard and B. Plattner. Introducing MorphMix: Peer-to-peer based anonymous Internet usage with collusion detection. In *Workshop on Privacy in the Electronic Society (WPES 2002)*, pages 91–102, Washington, DC, US, November 2002. ACM Press.
- [134] Reporters Without Borders. Blogger and documentary filmmaker held for the past month, March 2006. [http://www.rsf.org/article.php3?id\\_article=16810](http://www.rsf.org/article.php3?id_article=16810) (checked 2007-06-14).
- [135] G. Rieger et al. socat – multipurpose relay, October 2004. <http://www.dest-unreach.org/socat/> (checked 2007-06-14).
- [136] C. H. Rowland. Covert channels in the TCP/IP protocol suite. *First Monday*, 2(5), May 1997. [http://www.firstmonday.org/issues/issue2\\_5/rowland/](http://www.firstmonday.org/issues/issue2_5/rowland/) (checked 2007-06-14).
- [137] J. Rutkowska. The implementation of passive covert channels in the Linux kernel. In *21st Chaos Communication Congress*, Berlin, Germany, December 2004. Chaos Computer Club e.V. <http://www.ccc.de/congress/2004/fahrplan/event/176.en.html> (checked 2007-06-14).
- [138] M. Satterthwaite. Strategy-proofness and Arrow’s condition: Existence and correspondence theorems for voting procedures and social welfare functions. *Journal of Economic Theory*, 10:187–217, 1975.
- [139] A. Serjantov. On the anonymity of anonymity systems. Technical Report UCAM-CL-TR-604, University of Cambridge, Computer Laboratory, October 2004.
- [140] A. Serjantov and G. Danezis. Towards an information theoretic metric for anonymity. In R. Dingledine and P. Syverson, editors, *Privacy Enhancing Technologies (PET)*, volume 2482 of *LNCS*, pages 259–263, San Francisco, CA, US, April 2002. Springer.
- [141] A. Serjantov and P. Sewell. Passive attack analysis for connection-based anonymity systems. In E. Snekkenes and D. Gollmann, editors, *European Symposium on Research in Computer Security (ESORICS 2003)*, volume 2808 of *LNCS*, pages 116–131, Gjøvik, Norway, October 2003. Springer.
- [142] G. J. Simmons. The prisoners’ problem and the subliminal channel. In D. Chaum, editor, *Crypto ’83*, Advances in Cryptography, pages 51–67. Plenum Press, August 1983.

- [143] R. Singel. Judge halts NSA snooping. Wired News, August 2006. <http://www.wired.com/politics/law/news/2006/08/71610> (checked 2007-06-14).
- [144] T. Sohn, J. Seo, and J. Moon. A study on the covert channel detection of TCP/IP header using support vector machine. In P. Perner, S. Qing, D. Gollmann, and J. Zhou, editors, *Information and Communications Security*, volume 2836 of *LNCS*, pages 313–324. Springer, October 2003.
- [145] D. X. Song, D. Wagner, and X. Tian. Timing analysis of keystrokes and timing attacks on SSH. In *Proceedings of the 10th USENIX Security Symposium*, Washington, DC, US, August 2001. USENIX.
- [146] Q. Sun, D. R. Simon, Y.-M. Wang, W. Russell, V. N. Padmanabhan, and L. Qiu. Statistical identification of encrypted web browsing traffic. In *IEEE Symposium on Security and Privacy*, pages 19–30, Oakland, CA, US, May 2002. IEEE Computer Society.
- [147] P. F. Syverson, G. Tsudik, M. G. Reed, and C. E. Landwehr. Towards an analysis of onion routing security. In H. Federrath, editor, *Designing Privacy Enhancing Technologies*, volume 2009 of *LNCS*, pages 96–114, Berkeley, CA, US, 25–26 July 2000. Springer.
- [148] K. Szczypiorski. HICCUPS: Hidden communication system for corrupted networks. In *International Multi-Conference on Advanced Computer Systems*, pages 31–40, October 2003. <http://krzysiek.tele.pw.edu.pl/pdf/acs2003-hiccups.pdf> (checked 2007-06-14).
- [149] C.-R. Tsai, V. D. Gligor, and C. S. Chandrasekaran. A formal method for the identification of covert storage channels in source code. In *IEEE Symposium on Security and Privacy*, pages 74–87, Oakland, CA, US, April 1987. IEEE Computer Society.
- [150] E. Tumoian and M. Anikeev. Network based detection of passive covert channels in TCP/IP. In *30th IEEE Conference on Local Computer Networks*, pages 802–809, Sydney, Australia, November 2005. IEEE Computer Society.
- [151] G. Uchenick. MILS middleware for secure distributed systems. *RTC magazine*, 15, June 2006. <http://www.rtcmagazine.com/home/article.php?id=100685> (checked 2007-06-14).

- [152] U.S. Supreme Court. *McIntyre v. Ohio Elections Commission*. 514 U.S. 334 (1995).
- [153] G. Voyatzis and I. Pitas. Applications of toral automorphisms in image watermarking. In *International Conference on Image Processing*, Lausanne, Switzerland, September 1996. IEEE Computer Society.
- [154] D. Wagner. Re: Suggestions for the passing of passphrases. Usenet posting to sci.crypt and alt.privacy, June 2005. <d889mp\$2sah\$1@agate.berkeley.edu>.
- [155] X. Wang and D. S. Reeves. Robust correlation of encrypted attack traffic through stepping stones by manipulation of interpacket delays. In *CCS '03: Proceedings of the 10th ACM Conference on Computer and Communications Security*, pages 20–29, Washington, DC, US, October 2003. ACM Press.
- [156] J. Whitmore, A. Bensoussan, P. Green, D. Hunt, A. Kobziar, and J. Stern. Design for MULTICS security enhancements. Technical Report ESD-TR-74-176, Honeywell Information Systems Inc, Electronic Systems Division, Hanscom AFB, MA, US, December 1973.
- [157] P. Winkler. The advent of cryptology in the game of Bridge. *Cryptologia*, 7(4):327–332, October 1983.
- [158] M. Wright, M. Adler, B. N. Levine, and C. Shields. Defending anonymous communication against passive logging attacks. In *IEEE Symposium on Security and Privacy*, pages 28–41, Berkeley, CA, US, May 2003. IEEE Computer Society.
- [159] J. Yan. Security design in online games. In *19th Annual Computer Security Applications Conference*, Las Vegas, NV, US, December 2003. IEEE Computer Society.
- [160] J. Young and E. M. On obtaining “lawful interception” documents. <http://www.quintessenz.org/etsi> (checked 2007-06-14), March 2002.
- [161] Y. Zhang and V. Paxson. Detecting stepping stones. In *Proceedings of the 9th USENIX Security Symposium*, Denver, CO, US, August 2000. USENIX.

- [162] Y. Zhu, X. Fu, B. Graham, R. Bettati, and W. Zhao. On flow correlation attacks and countermeasures in mix networks. In D. Martin and A. Serrjantov, editors, *Privacy Enhancing Technologies (PET)*, volume 3424 of *LNCS*, pages 207–225, Toronto, Canada, May 2004. Springer.