DECLARATION BY BERNHARD PLATTNER

I, Bernhard Plattner, hereby declare as follows.

1. I am more than twenty-one (21) years of age and I am competent to make this declaration.
2. I was employed by ETH Zurich as a Full Professor of Computer Engineering in the Department of Electrical Engineering since April 1, 1994, and previously as an Associate Professor of Computer Engineering in the Department of Electrical Engineering since October 1, 1988.

   I am currently an Emeritus Professor of ETH Zurich and employed as a Confidant, serving as a contact person for all questions of research integrity.
3. I have been asked to certify the doctoral thesis titled, "MorphMix - A Peer-to-Peer-based System for Anonymous Internet Access", found on the ETH Zurich Research Collection at https://www.research-collection.ethz.ch/handle/20.500.11850/148022 (attached as Exhibit A).
4. I am familiar with "MorphMix - A Peer-to-Peer-based System for Anonymous Internet Access," Dr. Marc Rennhard's doctoral thesis, which I supervised at ETH Zurich. In accordance with the rules at the time, to complete his work, Dr. Marc Rennhard's thesis had to be deposited in the ETH Zurich Library, and I confirmed that Dr. Rennhard signed the agreement for the publication of his dissertation on April 16, 2004.
5. "MorphMix - A Peer-to-Peer-based System for Anonymous Internet Access" was published online and was available to the public on May 10 or 11, 2004.
6. "MorphMix - A Peer-to-Peer-based System for Anonymous Internet Access" was also published as a paperback book in April 2004 by Shaker Verlag, 52353 Düren, Germany

(https://www.shaker.de/de/content/catalogue/index.asp?lang=de&ID=8&ISBN=978-3-8322-2651-0, link accessed on June 9, 2020).

I declare under penalty of perjury under the laws of the United States of America that the foregoing is true and correct.

Executed on this 9th day of June 2020.

Bernhard Plattner

EXHIBIT A

MorphMix – A Peer-to-Peer-based System for Anonymous Internet Access

Author(s):
Rennhard, Marc

Publication Date:
2004

Permanent Link:
https://doi.org/10.3929/ethz-a-004709742 →

Diss. ETH No. 15420

MorphMix – A Peer-to-Peer-based System for Anonymous Internet Access

A dissertation submitted to the
SWISS FEDERAL INSTITUTE OF TECHNOLOGY
ZURICH

for the degree of
Doctor of Technical Sciences

presented by
MARC RENNHARD
Dipl. El.-Ing. ETH
born 10th February 1972
citizen of Böttstein (AG)

accepted on the recommendation of
Prof. Dr. Bernhard Plattner, examiner
Dr. Laurent Mathy, co-examiner

2004
(Examination date: 23rd January 2004)

TIK-SCHRIFTENREIHE NR. 61
Diss. ETH No. 15420

Marc Rennhard

MorphMix – A Peer-to-Peer-based System for Anonymous Internet Access

TIK-Schriftenreihe

Abstract

Contrary to popular belief, using the Internet is not anonymous at all. Since the Internet is a packet-switching network, every IP packet must carry the IP addresses of both communication endpoints. Consequently, anyone capable of observing at least one packet of a communication relationship can tell who is communicating with whom. The problem with this lack of anonymity is that it limits the privacy protection of Internet users. Today, privacy issues in the Internet are usually addressed by legislations that require operators of servers to clearly state their privacy practices and by encrypting the application data exchanged between two communicating parties. In general, privacy practices are difficult to enforce and encrypting the application data does not hide the IP addresses in the IP packets. However, learning the endpoints of communications relationships often reveals a lot of information about individual Internet users' preferences, habits, and problems; for instance when accessing web sites that provide medical information, religious sites, or the web site of a credit institution. These privacy issues can only be solved by enabling anonymous Internet communication.

In this thesis, we work on the problem of achieving anonymous Internet access for low-latency applications such as web browsing. With anonymous Internet access, we mean that a client can connect to and communicate with a server such that the server does not learn the client's IP address and an attacker interested in learning who is communicating with whom cannot find out the IP addresses of both client and server. Unlike encryption, anonymity cannot be "produced" by the communication endpoints themselves, but must be provided by a third party infrastructure. The concept of mix networks is widely considered to be the most promising approach for such an infrastructure, and consequently, we focus on mix networks in these thesis.

The main contribution of our work is *MorphMix*, which fulfils the princi-

pal goal of this thesis: to develop a practical mix network that enables anonymous low-latency Internet access for a large number of users. With practical, we mean that (1) everybody owning a state-of-the-art computer connected to the Internet can use the system, (2) the performance it offers is good enough such that users won't turn away from it, (3) it provides good protection from attacks by a realistic adversary, and (4) it scales well and can handle a large number of users.

We first analyse traditional mix networks that strictly separate between the mix network infrastructure and clients that access servers through the mix network. The conclusion is that traditional mix networks are not well suited to achieve our principal goal for various reasons. To overcome their limitations, we propose MorphMix, which presents a novel way of operating and organising a mix network. In contrast to traditional mix networks, MorphMix does no longer distinguish between clients and the mix network. Rather, the clients themselves build the mix network infrastructure in a peer-to-peer fashion. After describing the basic functionality of MorphMix, we give detailed analyses to show that MorphMix scales very well and provides good protection from a realistic adversary. To analyse the performance MorphMix offers to its users, we have implemented a simulator. The simulation results show that the expected performance of MorphMix is indeed good enough to attract users, and that the requirements to use MorphMix are modest. We have also specified the complete MorphMix protocol and have implemented a prototype. The main conclusion of our work is that with respect to our principal goal, MorphMix overcomes the limitations of traditional mix networks and is the first practical system that enables anonymous low-latency Internet access for a large number of users.

Zusammenfassung

Entgegen der weit verbreiteten Meinung ist die Benutzung des Internets nicht anonym. Weil das Internet ein paketvermittelndes Netwerk ist, muss jedes IP-Paket die IP-Adressen beider Kommunikationsendpunkte enthalten. Folglich kann jeder, der mindestens ein Paket einer Kommunikationsbeziehung beobachtet, sagen, wer mit wem kommuniziert. Dieses Problem der fehlenden Anonymität führt dazu, dass der erreichbare Schutz der Privatsphäre von Internetbenutzern limitiert wird. Derzeit wird die Privatsphäre im Internet üblicherweise so geschützt, dass Gesetze erlassen werden, welche die Betreiber von Servern verpflichten, ihre Praktiken im Umgang mit vertraulichen Benutzerdaten publik zu machen. Zusätzlich können die Anwendungsdaten, die zwischen den Kommunikationspartnern übertragen werden, mittels Verschlüsselung geschützt werden. Im Allgemeinen ist es jedoch schwierig zu überprüfen, ob die Betreiber ihre publizierten Praktiken einhalten, und trotz Verschlüsselung der Anwendungsdaten sind die IP-Adressen der Kommunikationspartner immer noch in den IP-Paketen sichtbar. Die Information, wer mit wem kommuniziert, liefert jedoch häufig bereits Erkenntnisse über die Vorlieben, Gewohnheiten und Probleme von individuellen Internetbenutzern, zum Beispiel wenn Daten von einem Webserver mit medizinischen oder religiösen Inhalten heruntergeladen werden oder wenn der Webserver eines Kreditinstituts kontaktiert wird. Solche Probleme betreffend des Schutzes der Privatsphäre können nur durch anonyme Internetkommunikation gelöst werden.

In dieser Arbeit beschäftigen wir uns mit dem Problem der Anonymisierung zeitkritischer Internetanwendungen wie Web-Browsing. Unter Anonymisierung verstehen wir, dass ein Client eine Verbindung zu einem Server aufbauen und mit diesem kommunizieren kann, ohne dass der Server die IP-Adresse des Clients erfährt. Darüber hinaus darf ein Angreifer, der er-

fahren möchte, wer mit wem kommuniziert, nicht zugleich beide IP-Adressen von Client und Server herausfinden. Im Gegensatz zu Verschlüsselung kann Anonymität nicht von den Kommunikationspartnern selbst "erzeugt" werden, sondern muss von einer Infrastruktur, welche von Dritten betrieben wird, gewährleistet werden. In der Forschungsgemeinde wird angenommen, dass das Konzept der Mix-Netzwerke am besten geeignet ist, eine solche Infrastruktur bereit zu stellen. Folglich beschränken wir uns in dieser Arbeit auf Mix-Netzwerke.

Der Hauptbeitrag dieser Arbeit ist das System *MorphMix*, welches unser Hauptziel erfüllt: ein praktikables Mix-Netwerk zu entwickeln, welches den anonymen Internetzugang für eine grosse Zahl von Benutzern ermöglicht. Unter praktikabel verstehen wir, dass (1) jeder, der einen zeitgemässen Computer besitzt, von dem System Gebrauch machen kann, dass (2) die Performanz des Systems für anwenderfreundliche Nutzung ausreicht, dass es (3) guten Schutz vor Attacken eines realistischen Angreifers bietet und dass es (4) gut skaliert und viele Benutzer gleichzeitig unterstützen kann.

Wir analysieren zuerst traditionelle Mix-Netzwerke welche strikt zwischen der Mix-Netzwerk Infrastruktur und den Clients, die mit Servern durch das Mix-Netzwerk kommunizieren, unterscheiden. Es wird gezeigt, dass sich traditionelle Mix-Netzwerke nicht gut eignen, um unser Hauptziel zu erreichen. Deshalb schlagen wir das System MorphMix vor, welches eine neue Art des Betriebs und der Organisation eines Mix-Netzwerks darstellt. Im Gegensatz zu traditionellen Mix-Netzwerken unterscheidet MorphMix nicht zwischen Clients und dem Mix-Netzwerk. Vielmehr bilden die Clients selbst die Mix-Netzwerk Infrastruktur auf Peer-to-Peer Basis. Nach der Beschreibung der grundlegenden Funktionalität von MorphMix liefern wir detaillierte Analysen, welche zeigen, dass MorphMix sehr gut skaliert und guten Schutz vor einem realistischen Angreifer bietet. Um die Performanz von MorphMix zu analysieren, haben wir einen MorphMix Simulator implementiert. Die Simulationsresultate zeigen, dass die erwartete Performanz die Benutzerzufriedenheit gewährleisten kann, und dass die Hardwareanforderungen von MorphMix von jedem zeitgemässen Computer erfüllt werden können. Des Weiteren haben wir das vollständige MorphMix-Protokoll spezifiziert und einen Prototypen implementiert. Insgesamt wird gezeigt, dass MorphMix unter Berücksichtigung unseres Hauptziels signifikante Vorteile im Vergleich zu traditionellen Mix-Netzwerken aufweist und das erste praktikable System darstellt, welches die Anonymisierung zeitkritischer Internetanwendungen für eine grosse Benutzerbasis ermöglicht.

Contents

1 Introduction **1**
- 1.1 Invading Privacy at the Application Level . . . 3
- 1.2 Invading Privacy at the Network Level . . . 7
- 1.3 Why do we need Anonymity . . . 8
- 1.4 Benefits versus Drawbacks . . . 11
- 1.5 Terminology and Definitions . . . 12
- 1.6 Problem Statement and Contributions of this Work . . . 14
- 1.7 Outline . . . 15

2 The Mix Network Approach **17**
- 2.1 The Mix Network Idea and Terminology . . . 17
- 2.2 Mix Networks based on Chaumian Mixes . . . 25
  - 2.2.1 Basic Functionality . . . 26
  - 2.2.2 Measures to Maintain the Sender's Anonymity . . . 28
  - 2.2.3 Basic Attacks on Mix Networks . . . 29
- 2.3 Circuit-based Mix Networks . . . 30
  - 2.3.1 Basic Functionality . . . 31
  - 2.3.2 Measures to Maintain the Client's Anonymity . . . 35
  - 2.3.3 Attacks on Circuit-based Mix Networks . . . 35
  - 2.3.4 Ways of Operating Circuit-based Mix Networks . . . 37
- 2.4 Summary . . . 40

3 Related Work **41**
- 3.1 Mix Networks Designs and Implementations . . . 41

3.1.1 Chaumian Mix Networks 42
3.1.2 Circuit-Based Mix Networks 44
3.2 Mix Networks Analysis and Attacks 49
3.3 Techniques beyond Mix Networks 54
3.3.1 Simple Remailers 55
3.3.2 Proxy Forwarders 55
3.3.3 Broadcast-Based Approaches 56
3.3.4 Anonymous Publishing 57
3.4 Other Applications 60
3.5 Economics of Anonymity and Reputation 61
3.6 Measuring Anonymity 62
3.7 Summary 62

4 A Detailed Analysis of Mix Networks 64
4.1 Why Anonymity is so Hard 64
4.1.1 Global Passive External Attackers 66
4.1.2 Partial Active Internal Attackers 70
4.1.3 Summary 71
4.2 A Quantitative Analysis of Mix Networks 72
4.2.1 No Dummy Traffic 73
4.2.2 Dummy Traffic between Clients and Mixes 75
4.2.3 End-to-End Dummy Traffic 77
4.2.4 Mix Cascades 78
4.2.5 Summary 80
4.3 A Realistic Threat Model 82
4.3.1 The Passive External Attacker 82
4.3.2 The Active Internal Attacker 85
4.3.3 Summary 86
4.4 Comparison of Mix Network Approaches 87
4.4.1 Static Mix Networks as Commercial Services 87
4.4.2 Static Mix Networks Operated by Volunteers 90
4.4.3 Dynamic, Peer-to-Peer-based Mix Networks 91
4.4.4 Summary 92
4.5 Conclusions 93

5 MorphMix **95**
5.1 Motivation and Goals 96
5.2 Basic Functionality of MorphMix 97
5.2.1 Overview 98
5.2.2 Anonymous Tunnels and Anonymous Connections . 99
5.2.3 Cells and Messages 101
5.2.4 Anonymous End-to-End Communication 102
5.3 Requirements to Break the Anonymity 106
5.4 Threat Model 108
5.4.1 The Passive External Attacker 109
5.4.2 The Active Internal Attacker 109
5.4.3 Summary 112
5.5 Establishing Anonymous Tunnels 113
5.5.1 Anonymous Tunnel Setup 113
5.5.2 Analysis of the Anonymous Tunnel Setup 119
5.5.3 Policy For Using the Virtual Links to Neighbours . . 122
5.5.4 Why Relaying Data for Other Nodes is Good 123
5.6 Collusion Detection Mechanism 123
5.6.1 Correlation and Correlation Distribution 124
5.6.2 Selection Size and Size of Extended Selections List . 128
5.6.3 Detecting Malicious Tunnels 129
5.7 Peer Discovery Mechanism 131
5.7.1 Initial Peer Discovery 131
5.7.2 Continuous Peer Discovery 133
5.7.3 Organising and Accessing Information about other Nodes 133
5.8 Scalability and Requirements to Run a Node 140
5.8.1 Scalability and General Requirements 140
5.8.2 NAT Gateways and Dynamic IP Addresses 142
5.9 An Outlook on IPv6 144
5.10 Summary 148

6 Attacks on MorphMix **151**
6.1 Basic Attack Model 151
6.1.1 The Node Simulator 153

6.1.2 Basic Scenario 154
6.2 Varying the Attack Level 155
6.2.1 The Adversary Attacks Always 155
6.2.2 The Adversary Attacks Selectively 160
6.2.3 Summary 161
6.3 Attacks Including Malicious Witnesses 164
6.4 Denial of Service Attacks 165
6.5 Exploiting the Peer Discovery Mechanism 168
6.6 Why Counting the Occurrences of Subnets does not Work . . 170
6.7 Summary 171

7 Analysis of the Collusion Detection Mechanism 174
7.1 Joining MorphMix for the first Time 174
7.2 Honest and Malicious Nodes in the same /16 Subnet 177
7.3 Large Realistic Systems 180
7.3.1 The Nodes have Abundant Capabilities and are Continuously Participating in MorphMix 180
7.3.2 The Nodes have Different Capabilities and Up-Times 182
7.4 Optimising the Quality of Anonymous Tunnels 188
7.5 The Subnets Contain Different Numbers of Honest Nodes . . 193
7.6 Varying the Tunnel Length 194
7.7 Summary 194

8 MorphMix Simulation and Results 198
8.1 The MorphMix Simulator 198
8.2 Basic Simulator Settings 200
8.2.1 Protocol Settings 200
8.2.2 Virtual Link Settings 201
8.2.3 Tunnel Settings 201
8.2.4 Node Settings 202
8.2.5 Web Browsing Scenario Settings 204
8.3 Simulation Results 205
8.3.1 Contacting the Web Server Directly 205
8.3.2 Contacting the Web Server through MorphMix . . . 207
8.3.3 Optimising the Throughput of Anonymous Tunnels . 208

8.3.4 Using Multiple Anonymous Tunnels in Parallel . . . 213
8.3.5 Bandwidth Usage and Overhead . . . 214
8.3.6 The Influence of Failed Tunnel Setups and Rejected Tunnels . . . 220
8.3.7 The Influence of the Tunnel Length . . . 221
8.3.8 The Influence of the Cell Length . . . 224
8.3.9 Crashing Nodes and Blocked Virtual Links . . . 225
8.4 Summary . . . 227

9 Conclusions **231**
9.1 Summary . . . 231
9.2 Achievement of Goals and Assessment . . . 234
9.3 Comparison with Other Systems . . . 237
9.3.1 Comparison with Crowds . . . 238
9.3.2 Comparison with Tarzan . . . 240
9.4 Further Work . . . 242

Bibliography **245**

Acknowledgements **257**

Biography **259**

A MorphMix Protocol and Prototype Implementation **260**
A.1 Notation . . . 260
A.2 Basic Protocol Properties . . . 261
A.2.1 Cryptographic Algorithms . . . 261
A.2.2 Cell Format . . . 262
A.2.3 Node Levels . . . 264
A.2.4 Encoding . . . 264
A.3 Messages between Neighbours . . . 267
A.3.1 Establishing a Virtual Link . . . 267
A.3.2 Appending a Node to a Tunnel . . . 268
A.3.3 Peer Discovery Messages . . . 269
A.3.4 Virtual Link Status Information Messages . . . 271
A.3.5 Terminating an Anonymous Tunnel . . . 271

A.3.6 Flow Control Messages 272
A.3.7 Virtual Link Data Messages 273
A.4 End-to-End Messages 274
A.4.1 Appending a Node to a Tunnel 274
A.4.2 Initiating and Terminating an Anonymous Connection 275
A.4.3 End-to-End Status Information Messages 277
A.4.4 End-to-end Data Messages 278
A.5 Virtual Link and Tunnel Usage 278
A.5.1 Virtual Links and Tunnel Lifetimes 279
A.5.2 Policy for Using Virtual Links 280
A.6 Quantitative Analysis of the Data Overhead 281
A.6.1 Tunnel Setup and Teardown Overhead 281
A.6.2 Virtual Link Setup Overhead 284
A.6.3 Virtual Link Status Information Overhead 285
A.6.4 End-to-End Status Information Overhead 285
A.6.5 Other Protocol Overhead 285
A.6.6 Protocol Overhead Summary 286
A.7 MorphMix Prototype Implementation 286

List of Figures

2.1 The mix overlay network and the underlying physical network. 19
2.2 Sending application data directly from h_c to h_s. 20
2.3 Sending application data via the mix network from h_c to h_s. 22
2.4 Sending application data AD through a Chaumian mix network. 26
2.5 A circuit-based mix network. 32
2.6 Different ways of operating circuit-based mix networks. . . . 37

4.1 Traffic analysis at a mix 66
4.2 End-to-end Traffic analysis 68
4.3 End-to-end Traffic analysis by an internal attacker 71

5.1 Basic idea of MorphMix. 98
5.2 Multiple anonymous connections within one anonymous tunnel. 100
5.3 Virtual links and layers of encryption along an anonymous tunnel. 101
5.4 Anonymous connections and cell forwarding. 103
5.5 Appending a node to a tunnel and establishing the layer of encryption. 117
5.6 Correlation distribution with 10000 nodes. 127
5.7 Node Lookup list. 135

6.1 f_{a_m} if the adversary attacks always with the same attack level. 156
6.2 Correlation distribution when varying the attack level from 0–14. 157
6.3 f_{a_m} if the adversary attacks with different attack levels. . . . 159

6.4 f_{a_m} if the adversary attacks always with the same attack level but only if he controls the first intermediate node. 161
6.5 f_{a_m} if the adversary attacks with different attack levels but only if he controls the first intermediate node. 162
6.6 f_{a_m} if the adversary attacks with different attack levels but only if he controls the first intermediate node, assuming he does not always guess correctly. 163
6.7 f_{a_m} if the adversary hopes for a malicious witness when the final node is appended. 165
6.8 f_{a_m} if the adversary attacks always with the same attack level and refuses to forward data along any tunnel where he controls at least one node but not both the first intermediate and the final node. 166
6.9 Occurrences of /16 subnets in the extended selections list. . . 171
6.10 f_{a_m} without and with collusion detection, and if the adversary plays fair. 173

7.1 Performance with 10000 nodes. 175
7.2 Performance with 50000 nodes. 176
7.3 f_{a_m} depending on the fraction controlled by the adversary in subnets with malicious nodes. 179
7.4 100000 honest, 10000 malicious nodes (abundant capabilities, always participating). 181
7.5 1000000 honest, 10000 malicious nodes (abundant capabilities, always participating). 182
7.6 100000 honest, 10000 malicious nodes (different capabilities and participation probabilities). 185
7.7 1000000 honest, 10000 malicious nodes (different capabilities and participation probabilities). 188
7.8 100000 honest, 10000 malicious nodes (with tunnel optimisation). 190
7.9 1000000 honest, 10000 malicious nodes (with tunnel optimisation). 192
7.10 1000000 honest, 10000 malicious nodes (different numbers of honest nodes in the /16 subnets). 193
7.11 1000000 honest, 10000 malicious nodes (diff. tunnel lengths). 195

8.1 Download times, accessing the web server directly. 206
8.2 Download times, accessing the web server through MorphMix. 207
8.3 Download times with optimised tunnel throughput. 208
8.4 Download times with even more optimised tunnel throughput. 209
8.5 Ratio between the download times when accessing the web server through MorphMix and directly using HTTP 1.1. . . . 210
8.6 Download times for a single file. 211
8.7 Download times for a single file (more detailed illustration). 212
8.8 Download times using multiple tunnels in parallel. 213
8.9 Bandwidth usage. 216
8.10 Data sent and received by the nodes. 217
8.11 Download times bandwidth usage with reading time = 0. . . 219
8.12 Data sent and received by the nodes with reading time = 0. . 220
8.13 Download times with failed and rejected tunnels. 221
8.14 Bandwidth usage with failed and rejected tunnels. 221
8.15 Download times depending on the tunnel length. 222
8.16 Bandwidth usage depending on the tunnel length. 223
8.17 Download times using different cell lengths. 225
8.18 Download times when nodes crash or are temporarily blocked from their neighbours. 227

A.1 Cell format. 262

List of Tables

4.1 Minimum capacities to support 100000 users (web browsing, 5 MB per day and user). 81

7.1 Realistic bandwidth distribution of MorphMix nodes. 183
7.2 Acceptable intermediate and final nodes 189

8.1 Data volume depending on the cell length (all lengths in bytes) 224
8.2 Percentage of pages that failed during their first download. . 228

A.1 Node levels in MorphMix 264
A.2 Encoding of fields in the payload. 265
A.3 Encoding of message types. 266
A.4 Fields and cells to establish a virtual link. 268
A.5 Fields and cells (corresponding to messages 4–9) to append a node to an anonymous tunnel. 270
A.6 Fields and cells to learn about other nodes. 271
A.7 Fields and cells to exchange status information between neighbours. 272
A.8 Cell to terminate an anonymous tunnel. 272
A.9 Cell to reset the credit of a tunnel on a virtual link. 273
A.10 Cell to transport end-to-end messages. 273
A.11 Fields and cell payloads (corresponding to messages 1–3 and 10) to append a node to an anonymous tunnel. 276
A.12 Fields and cell payloads to initiate and terminate anonymous connections. 277

A.13 Fields and cell payloads to exchange status information between endpoints of a tunnel. 278
A.14 cell payloads to transport end-to-end data. 278
A.15 Overhead for the initiator to set up a tunnel. 281
A.16 Overhead for the i^{th} intermediate node to set up a tunnel. . . 282
A.17 Overhead for the final node to set up a tunnel. 283
A.18 Overhead for a witness to set up a tunnel. 283
A.19 Overhead summary to set up and tear down a tunnel with length five. 284

Chapter 1

Introduction

Until the early 1990s, the Internet was mainly an academic research network where security and privacy issues were of little importance. However, driven by the huge popularity of the World Wide Web (WWW) due to graphical web browsers, the Internet has become a platform used by hundreds of millions of people everyday for activities that often have been shifted from the physical to the online world.

Soon, it was recognised that especially the growth of e-commerce calls for some security mechanisms. There is no universal definition of computer or network security because it always depends on what must be protected, but in the Internet context, security often means secure communication, which can be defined as follows[1]:

Definition 1 Secure Communication *between two parties A and B is defined as a communication relationship with the following three properties:*

- **Confidentiality**: *data exchanged between A and B cannot be read by an eavesdropper*
- **Integrity**: *data exchanged between A and B cannot be altered (accidentally or intentionally) in transit in a way that is not detectable by the recipient*
- **Authentication**: *A (or B) can be sure she is indeed communicating with B (or A)*

[1] Other security properties include availability and non-repudiation, which are less important for secure communication.

Applied to an e-commerce scenario, a customer should be sure she is indeed communicating with the e-store she intends to and transmitting the payment information from the user to the e-store should be protected from modification or observation by third parties. Using the secure socket layer (SSL) protocol [51] or its successor, the transport layer security (TLS) protocol [33], together with X.509 digital certificates [60] solves these problems and brings confidentiality, authenticity, and integrity to the Internet. Another protocol for secure communication is IPSec [66], which "patches" the IP protocol [40] with security mechanisms to enable, for example, virtual private networks (VPNs) [108]. To put it briefly, regarding e-commerce and other business transactions in the Internet and leaving out mobile and ad-hoc networking scenarios, the basic security problems are well understood, solved, and widely accepted and deployed.

Beyond security, there is privacy. Applied to the Internet context, privacy can be defined as follows [54]:

Definition 2 Privacy *refers to the ability of an individual to control the information about herself. This does not necessarily mean that no information is revealed to anyone. Rather, a system that respects the privacy of its users allows them to select what information about them is revealed, and to whom.*

Unlike security, mechanisms to bring more privacy to the Internet are not yet widely deployed. One proposal is the World Wide Web Consortium (W3C)'s Platform for Privacy Preferences (P3P) project [93]. Its goal is to clarify a web site's privacy practices to its users. When a web site or e-shop is contacted, the user is informed of the privacy practices of that site. The user can accept these practices, reject them and ask for an alternative proposal, or send another proposal herself. If an agreement between user and web site is reached, the communication continues, otherwise it is terminated. However, P3P only specifies the protocol for exchanging structured data to reach an agreement, but it cannot do anything to enforce the privacy practices a web site has proposed: even if an e-shop has promised not to give away information about the user to third parties, there is no way for the user to check if the e-shop complies with the rules.

This thesis is primarily about anonymity, which is closely related to privacy. We introduce the term anonymity more formally in Section 1.5, but for now, anonymity can be defined as follows [54]:

Definition 3 Anonymity *means privacy of identity. A system that offers*

anonymity is one where the user gets control who learns her identity. In the Internet context, identity not only means the true name of the user, but also her e-mail or IP address.

For completeness, we also introduce the term pseudonymity, which is a special case of anonymity:

Definition 4 Pseudonymity *enhances anonymity with pseudonymous identities. Such identities are also called nyms [13] and make it possible for an Internet user to have an identity while her true name is kept secret. There are nyms that are completely unrelated to the individual's real identity (for instance a self-chosen alias) and other nyms that make it possible to unambiguously uncover its owner's identity if certain conditions are met (for instance if a court order has been issued).*

In the remainder of this chapter, we first discuss common practices in use today, which invade the privacy of Internet users. We also point out possibilities and limitations for a user to protect herself from such invasions and show that anonymous Internet access is one way to overcome these limitations. We discuss benefits and drawbacks of anonymity and finally, we state the main goal of our work and describe our contributions.

1.1 Invading Privacy at the Application Level

One way to invade the privacy of Internet users is to do so at the application level. A prominent example is tracking users as they navigate through the WWW, which is possible by combining several mechanisms of how web browsers access web pages using the hypertext transfer protocol (HTTP) [8, 45] and its secure version HTTPS [105]. We briefly describe these mechanisms below:

- The **HTTP referer**[2] field in the header of HTTP requests field tells a web server or an eavesdropper the uniform resource locator (URL) of the web object the user has downloaded before. For instance, if a web site is accessed from the results page of a search engine, it contains the entire search string the user entered.

[2]Note that *referer* (for *referrer*) was spelt incorrectly in the original standard and made it into the first implementations of the HTTP protocol. For backward compatibility, the misspelled word is still used in newer implementations of the protocol as of today.

- **Cookies** [69] were originally invented to create a session over several HTTP request/reply pairs, thereby allowing a web server to track a user as she navigates through different web pages at the same site. A cookie is a small piece of information that a web server sends to the browser within an HTTP reply. If a page at the same site is requested later using the same browser, the cookie is sent to the web server as part of the HTTP request, which allows the web server to recognise subsequent visits by the same user. If a user ever registers at the web site, the server can associate her identity with the pages she has visited and form a profile of her browsing interests.
- **Embedded objects** of a web page are automatically downloaded by the web browser from their respective servers. Embedded objects must not reside on the same server as the page containing them, but can be locates on any server. One type of embedded objects are **banners** for advertisement, and many institutions allow third parties to place banners on their web pages in return for monetary compensation.

Combining HTTP referers, cookies, and embedded objects (usually in the form of banners) make it possible for a third party to track users across different web sites and accumulate detailed profiles. To do so, a company C interested in collecting data about users places banners on several web pages at different sites. If a user downloads a page containing a banner from C, the browser automatically requests the banner form C's web server. Since the browser also includes the HTTP referer in the request, C learns the URL of the page the user is downloading. When sending the HTTP reply containing the banner, C's web server includes a cookie, which is stored in the user's browser. If the user later visits the same or another web site that contains a banner from C, the browser includes this cookie in the HTTP requests to fetch the banner, which allows C to recognise the user.

While this does not seem to be a significant loss of privacy at first glance, it gets much more serious when looking at a real example. We visit Health-Central.com, a site providing medical information, and enter "cancer" in the search form on their entry page, which results in an HTTP request sent to the web server that contains the following fields:

```
GET /search.asp?query=cancer HTTP/1.1
Host: search.healthcentral.com
Referer: http://www.healthcentral.com/home/home.cfm
```

It tells the server identified with search.healthcentral.com (Host field) to execute the script search.asp with parameter "cancer" (GET field). Note the referer, i.e. the URL of the page that was just downloaded is also passed to the server (Referer field). The server replies by sending an HTML page containing the search results. The page contains several embedded objects that are fetched automatically by the browser. One of the embedded objects happens to be a banner, which is downloaded by the browser using an HTTP request that contains the following fields:

```
GET /adi/N2552.healthcentral/B1106194.4 HTTP/1.1
Host: ad.doubleclick.net
Referer: http://search.healthcentral.com/search.asp?query=cancer
```

Apparently, the object is requested from the server ad.doubleclick.net (Host field), which belongs to the company DoubleClick. Since the object is requested from the result page of the search for "cancer", the Referer filed corresponds to the URL of the result page and therefore contains the search string entered by the user. This means that DoubleClick knows a user has requested information about "cancer" at HealthCentral.com. The HTTP reply to the request above contains the following field:

```
Set-Cookie: id=800000255cc5216; path=/; domain=.doubleclick.net; expires=Fri, 02
Jun 2006 11:18:21 GMT
```

This sets a cookie in the user's browser that only expires after three years. Now we assume that later, the user uses her favourite search engine HotBot[3] to get information about "chemotherapy". The result page again contains a banner from DoubleClick, and the request issued by the browser contains the following fields:

```
GET /adi/hb.ln/r;kw=chemotherapy HTTP/1.1
Host: ln.doubleclick.net
Referer: http://www.hotbot.com/default.asp?prov=Inktomi&query=chemotherapy
Cookie: id=800000255cc5216
```

Since the request goes to doubleclick.net (Host field), the browser automatically includes the cookie (Cookie field) it received earlier. In addition, the Referer field contains again the search string and in this case, the keyword is even included in the GET field. As a result, DoubleClick knows that searching for "chemotherapy" with HotBot has been performed by the same user (or

[3] http://www.hotbot.com

at least the same web browser) that requested information about "cancer" at HealthCentral.com before.

So combining HTTP referers, cookies, and banners allows a third party to track users across different web sites. Regarding the enormous number of pages containing DoubleClick banners, it can be expected that the company has accumulated extensive profiles about vast amounts of web users. Note that if a user ever enters personal information such as her name into a form, clicks on the submit button, and the resulting page contains a DoubleClick banner, then DoubleClick can associate all activities of that user with her identity. Note also that not even a visible embedded object in the form of a banner is needed because so called web bugs, which are usually 1 by 1 pixel gif images, are often embedded in pages for the same purpose. If the web bug is requested from DoubleClick, the effect is the same as when using a banner. Since web bugs have the same colour as the background and are downloaded very quickly, they can hardly be noticed by the user without analysing the page source.

We do not accuse DoubleClick of any unlawful activities, but their practices show what's possible in today's Internet. There is not much to say against personalised advertisements embedded in web pages as long as no efforts are made to associate user profiles with their identity. However, in the case of DoubleClick, it has never been entirely clear what their practices are[4]. In the USA, state and federal lawsuits that charged the company with violating the privacy of Internet surfers were raised and finally settled in May 2002. The settlement requires the company to purge certain data files of personally identifiably information, including names, addresses, telephone numbers, and e-mail addresses. Among other provisions, the settlement requires DoubleClick to obtain permission from Internet surfers before it can tie personally identifiable information with their web surfing history[5].

It is quite simple to defeat being tracked by DoubleClick or others employing similar methods: all popular browsers allow to completely disable cookies, but this implies some web pages cannot be accessed anymore because they don't work without cookies. Another simple method is to limit the lifetime of cookies to the current session, which means all cookies are deleted when the browser is closed. It is also possible to prompt the user for every cookie that is received, but since so many sites try to store a cookie in

[4] `http://www.junkbusters.com/new.html#DCLK` provides more details about the history of DoubleClick

[5] `http://news.com.com/2100-1023-919895.html`

the browser, this is not very practical. Junkbuster[6] or its successor Privoxy[7] are freely available tools that allow cookies being received from and sent to explicitly specified sites, but block all others. In addition, they can be used to block certain requests. For instance blocking every request that contains "doubleclick" should effectively protect from being tracked by DoubleClick.

1.2 Invading Privacy at the Network Level

Besides invading the privacy of Internet users at the application level as described in the previous section, it can also be done at the network level. Every IP packet exchanged between a user's computer and another host contains the IP addresses of both communication endpoints. Although an IP address does often not directly identify a person, the knowledge to associate an IP address used at a certain time with a particular individual is virtually always available. If a user accesses a server through the Internet, there are various cases how the communication relationship between the user and the server can be uncovered:

- The system administrator of a company or of a department of a university can find out who has been assigned what IP address at what time. This is possible by checking the appropriate logs and derive what user has been logged onto what computer. If the computers do not use static but dynamic IP addresses using the dynamic host configuration protocol (DHCP) [38], the logs of the DHCP server inform about what computer was assigned what IP address at what time. As a result, the system administrator can easily learn who has been communicating with what server.
- A home user connected to the Internet using a dial-up connection often gets assigned a dynamic IP address from a pool of addresses available to the user's Internet Service Provider (ISP). The ISP knows what user was using what IP address at what time and therefore knows all communication relationships of all its subscribers.
- The server or any entity that gets access to packets exchanged between the user and the server sees the IP addresses of the communicating parties. Accessing packets is for instance possible for every ISP be-

[6] http://www.junkbuster.com
[7] http://www.privoxy.org

tween user and server, the FBI using their Carnivore diagnostic tool[8], or any other eavesdropper. Assuming cooperation of the user's system administrator or her access ISP, it is possible to learn what user is communicating with what server. With the growing popularity of access technologies such as (A)DSL or Cable that allow home users being connected to the Internet permanently, more and more people have an own public static IP address, which means identifying the user is much easier and possible without cooperation of the system administrator or the access ISP.

Defending against this kind of invasion of privacy is much more difficult than in the previous section where the user could protect herself by controlling the usage of cookies. The main problem is that IP addresses are one fundamental component of the network layer to transport data in the Internet and not just an application layer feature like cookies. In particular, the user cannot simply choose not to include her IP address in the packets she sends and receives.

One can argue that encrypting the application data carried in IP packets helps to increase the privacy of web users. This is certainly true, but only while a packet is in transit between user and server. The server still sees both the data and the user's IP address. But more importantly, there are many situations where even knowing only the identities of the communicating parties is too much and this is where anonymity comes into play. For instance, when sitting at your work place and browsing a site offering jobs, you do not want your employer to know what site you are accessing. Connecting to medical sites (e.g. www.healthcentral.com), pornographic sites, your favourite religious site, or the web site of a credit institution already may reveal significant information about your personal preferences and problems. Anonymity can be considered as the ultimate form of privacy and is extremely difficult to achieve in the Internet.

1.3 Why do we need Anonymity

It is a legitimate question to ask why we need anonymity in the Internet at all. At least, the Internet has become very popular during the second half of the 1990s although it was never built with privacy or anonymity in mind and did not offer such measures at all. Don't Internet users care about their

[8] http://www.fbi.gov/hq/lab/carnivore/carnivore.htm

privacy? According to a Forrester Research survey of online users in 1999, 67% said they were "extremely" or "very" concerned about releasing personal information over the Internet[9]. An Arthur Andersen survey in 2000 found that 94% of 365 Internet users expressed some level of concern for their privacy[10]. So it seems Internet users definitely care about privacy. But then, why are they giving away so much personal information?

An explanation could be that most people are simply not aware of how easy it is to accumulate information about them in the Internet. Indeed, comparing Internet applications with their counterparts in the real world shows that their online versions offer usually much less privacy, as the following examples show:

- A notice-board as used at schools, work places, or public buildings can be considered as the physical world equivalent of a newsgroup. Although much less powerful than their counterparts in the Internet because far fewer people will ever read a notice, they have the advantage that the lifetime of notices is usually limited. When a notice is removed from the board, it will soon be forgotten. With newsgroups, every single posting will be stored for a very long time and can easily be found using search engines.
- Sending an e-mail message discloses the identities of both sender and recipient. In addition, the content is easily accessible to any mail server or eavesdropper on the path from sender to recipient unless it is encrypted. Although tools for encrypting are readily available, they are often cumbersome to use and require both communicating parties to have them installed on their computers. In addition, encrypting e-mail messages only hides the content but not the addresses of sender and recipient. Letters, on the other hand, usually hide the content from outsiders. Of course it is possible to open an envelope, read the content, close the envelope again, and send it to the recipient without anybody noticing this. But this is a time-consuming process and can hardly be done with every single letter. With e-mail messages, this is much simpler because they are processed electronically. In addition, one can easily send anonymous letters simply by omitting the sender address. Note that with e-mail messages, this is also possible, for instance by sending e-mail messages only using anonymous e-mail accounts as of-

[9] http://www.fdic.gov/news/conferences/transcript.html
[10] http://www.privacydigest.com/2000/09/30

fered by Yahoo[11] and others. However, this only protects the sender's identity from the recipient, but not from Yahoo or other eavesdroppers because when accessing the mailbox at Yahoo from her home or work computer, a user discloses her IP address. We conclude that e-mail messages are more similar to postcards that contain the full sender's address than to letters.

- Traditionally, one had to go to the library to look up information. This is a very private and in fact anonymous process because nobody has to disclose her identity when simply consulting some books. Today, people start their favourite search engine and type in a search string, or they visit a web portal, both of which is not private at all. In either case, they will end up browsing through the web, accessing several web servers and leave extensive traces about their personal preferences, habits, and dislikes.
- Browsing through a physical store and looking at goods is anonymous. When buying a product, the customer can choose to pay with cash, which is difficult to trace. Internet-based e-commerce is different. First of all, anonymously buying products is not possible because payment is usually done by credit cards. But anonymously looking at the goods is also not possible because with every user sending her IP address, the e-store may know a customer's identity right when accessing the entry page of the store. Consequently, e-commerce is like a customer would have to provide her identity when entering a physical shop.

It could well be this resemblance of many Internet applications to the real world which causes users to have a wrong sense of privacy. In any case, if people start shifting more and more of their activities to the Internet, the probability that information about them is logged and stored significantly increases. Storage has become so cheap that even vast amounts of information can be stored for a long time. Using powerful search engines allows to learn virtually everything stored in public databases about a particular individual. For example, using Google[12] and searching within "Groups" makes it possible to access every single post ever made to newsgroups. It's virtually impossible to remove a message from a newsgroup. Once it has been posted, it is likely to remain forever available to every Internet user[13]. The same is true for every piece of information stored about a certain person in the Internet: if the

[11] `http://www.yahoo.com`
[12] `http://www.google.com`
[13] Considering the time the Internet has been around, "forever" means "a few decades".

server holding the information is not under control of that person, removing the information is very difficult.

Today, only a fraction of the traces one leaves in the Internet are available in public databases. However, other parties may monitor and store parts of the traffic being sent over the Internet. As mentioned in Section 1.2, various parties may see what an individual is doing. Just imagine the personal information an individual discloses over a course of ten years of using search engines, shopping online, visiting web sites in general, sending e-mail messages, posting to newsgroups, chatting, using file-sharing systems, and more. Large dossiers about Internet users could be accumulated and sold, and it is reasonable to assume that there is a market for such a business. For instance, employers could extensively evaluate potential employees before hiring them, or politicians could dig for information they could use against their opponents.

One – and probably the only – way to mitigate these problems and to significantly enhance the privacy of Internet users is by enabling anonymous Internet access. Anonymous Internet access makes the task of accumulating large amounts of information about a particular individual much more complicated because Internet activities can no longer easily be associated with a particular IP address or identity.

1.4 Benefits versus Drawbacks

Like every technology, anonymity in the Internet can be used for good and bad purposes. As the examples given in Section 1.3 show, the physical world often is anonymous. However, the availability of anonymity in real life is often abused because anonymous letters or phone calls from public telephones are used to threaten people all the time. Nevertheless, the benefits seem to outweigh the drawbacks and consequently, anonymity in the physical world is widely accepted as beneficial for society.

In the Internet, the situation is similar. We have given many arguments for anonymity in this chapter, but there are several ways to abuse anonymous Internet access. Distributing junk e-mail messages (spam) without being detected would get even easier than today. Similarly, harassment of people through e-mail messages or posts to newsgroups are likely to increase. In addition, anonymity-providing technologies may make it even more difficult to derive the origin of denial of service attacks than today. But the greatest

fear of opponents of anonymity are that it may provide terrorists, drug dealers, and other criminals with a platform that facilitates their communications. Basically, their arguments are the same we already heard during the discussions in the 1990s about whether strong cryptography should be made available to the broad public or not, and we do not deny that some of these arguments are at least partially true. Like encryption, anonymity will make it more difficult for intelligence agencies to spy on their enemies.

One thing to remember is that we can have very good anonymity in the Internet even today. To browse the Web anonymously, we simply go to a public Internet terminal as found at various places (Internet cafes, airports, train stations, libraries, ...). To exchange e-mail messages anonymously, we can establish an anonymous e-mail account at Yahoo and only access it through a public terminal. Furthermore, more and more public wireless access points are installed and many of them do not require any form of authentication to be used. To summarise, there are various possibilities to access the Internet anonymously, but regularly using the Internet in such a way is cumbersome. However, smart criminals take their time, are careful, and may in fact take great care to access the Internet only in such a way. Simply spoken, we can say that criminals already have anonymity, but normal people don't.

Ultimately, society will balance the benefits against the drawbacks and either make use and thereby boost anonymity-providing techniques or not. We believe that anonymity in the Internet is valuable for society, but it is not the goal of this thesis to educate Internet users or impose our opinion on others. Rather, we want to bring research on anonymity-providing systems one step further by exploring the possibilities and limits of anonymous communication in the Internet.

1.5 Terminology and Definitions

The terminology and definitions we use throughout this thesis are based on a proposal for the terminology in anonymous communication systems [83].

The basic setting is that *senders* send *messages* to *recipients*. This terminology works well for scenarios such as e-mail communication where a message correspond to an e-mail message, but for applications that make use of request/reply pairs, we prefer the notation that a *client* exchanges *messages* with a *server*. In this case, the message correspond to either a web request or a web reply.

Anonymity is the state of being not identifiable within a set of subjects, the *anonymity set*. The anonymity set is the set of subjects (for instance users) that may have caused an action (for instance having sent an e-mail message or having accessed a web server). The concept of the anonymity set is fundamental to research on anonymity. Less formal, it means that when using an anonymity-providing service, one is only anonymous among the set of all those using the same service at the same time, and not among all Internet users. In general, larger anonymity sets imply better anonymity.

Unlinkability of two or more items (e.g. subjects or messages) means that within an anonymity-providing system, these items are no more and no less related than they are related concerning the a-priori knowledge. This means the probability of those items being related stays the same before (a-priori knowledge) and after the run within the system (a-posteriori knowledge of the attacker).

Using the definition of unlinkability, anonymity can be defined as unlinkability of subjects and messages. In particular, *sender* (or *client*) *anonymity* means that for the recipient (or server), messages it receives are not linkable to a sender (or client). In the client/server case, it also means that the server can send back messages to the client without this data being linkable to a particular client. *Recipient* (or *server*) *anonymity* means that for the sender (or client), messages it sends are not linkable to a recipient (or server). *Relationship anonymity* means that except for the communicating parties, no other party can learn who communicates with whom. In other words, sender (or client) and recipient (or server) are unlinkable.

It should be noted that the definition of anonymity using unlinkability is absolute. Either there is unlinkability and therefore anonymity or not. For instance, if an attacker manages to exclude 10% of all current users of a system of having been the sender of a particular data, unlinkability and therefore also anonymity are no longer given according to the definitions above, even if every one of the remaining 90% of all senders could have sent the message with equal probability. While this makes sense for researchers exploring possibilities to achieve information-theoretic (or perfect) anonymity, it is less well suited for those working on practical anonymity-providing systems because in the latter case, attackers are usually capable to reduce the set of potential senders of a certain message. We therefore define *unambiguous linkability* of two or more items, which means these items can be unambiguously related.

Throughout this thesis, we equal IP addresses and identities, which means that if the IP address of an item (for instance the sender) is known, we assume

its identity (for instance the true name of the sender of an e-mail message or the person acting as a client when downloading a web page) is also known. Consequently and using our definition of unambiguous linkability, we say the sender (or client) anonymity is *broken* if the recipient (or server) can unambiguously link a message it receives to the sender's (or client's) IP address. Likewise, the recipient (server) anonymity is broken if the sender (or client) can unambiguously link a message it sends to the recipient's (or server's) IP address. Finally, the relationship anonymity is broken if any party except communicating parties themselves can unambiguously link the IP addresses of both sender (or client) and recipient (or server).

It should be noted that this terminology is well suited for anonymous communication systems in general and is independent of a particular technology that aims at providing anonymous communication. However, since this thesis focuses primarily on systems based on mix network to enable anonymous communication, we will have to adapt and extend this terminology when we discuss mix networks in more detail in the next chapter.

1.6 Problem Statement and Contributions of this Work

In this thesis, we focus on the problem of achieving anonymous Internet access for low-latency applications such as web browsing. As we will see in Chapters 2 and 3, anonymity-providing systems have often been separated into those supporting non-time-critical applications such as electronic mail and those aiming at low-latency applications[14], although the systems often have many similarities independent of the type of application they support. However, experience has shown that supporting the former type of application is a much more challenging problem than supporting the latter (see Chapter 2) and as a result, no sophisticated system that provides practical low-latency anonymity for a large number of users is available today.

Throughout this thesis, we focus on the concept of mix networks (see Chapter 2), which is considered as the most promising approach to enable anonymous Internet access. There are also alternative concepts (see Section 3.3), but they either provide only little protection from attacks or are

[14] Of course, systems supporting low-latency applications also support non-time-critical applications, but not vice versa.

not practical for a large number of users in the Internet context. Mix networks provide client and relationship anonymity at the network level (see Section 1.2), which means they hide the client's IP address from the server and prevent an adversary from unambiguously linking the IP addresses of the client and the server of a communication relationship.

The principal goal of this work is to provide a practical system that enables anonymous low-latency Internet access for a large number of users. Although this will be specified in more detail in Chapter 5, with a practical system we mean that (1) everybody owning a state-of-the-art computer connected to the Internet can use the system, (2) the performance it offers is good enough such that users won't turn away from it, (3) it provides good protection from attacks by a realistic adversary, and (4) it scales well and can handle a large number of users. To do so, we first analyse traditional mix networks to demonstrate that they are not well suited to achieve this goal (see Chapter 4). To overcome the limitations of traditional mix networks and to achieve our principal goal, we have developed *MorphMix*, which proposes a novel way of operating and organising a mix network. We have carried out detailed analyses to show that MorphMix scales very well and provides good protection from a realistic adversary. To analyse the performance MorphMix can offer to its users, we have implemented a simulator. The simulation results show that the expected performance of MorphMix is indeed good enough to attract users, and that the requirements to use MorphMix are modest. Finally, we have specified the complete MorphMix protocol and have implemented a prototype. The main conclusion of our work is that with respect to our principal goal, MorphMix overcomes the limitations of traditional mix networks and is the first practical system that enables anonymous low-latency Internet access for a large number of users.

MorphMix will be presented and analysed in Chapters 5 to 8, and the precise goals MorphMix should achieve are stated in Section 5.1.

1.7 Outline

In Chapter 2, we start with a description of the concept of mix networks that will provide the basis for our work. In Chapter 3, we look at other work that has been conducted in the field of anonymity. We do not restrict ourselves to mix networks, but also present other techniques to tackle the problem of anonymous Internet communication. In Chapter 4, we examine mix networks

in great detail. We especially focus on mix networks for low-latency applications and analyse their resistance to attacks.

In Chapters 5–8, we present MorphMix, which is the major contribution of our work. Chapter 5 describes the basic design and functionality and introduces the core components of MorphMix. In Chapter 6, we examine different attack strategies that can be employed by an adversary to analyse how well MorphMix protects its users from the corresponding attacks. In Chapter 7, we analyse the performance of the collusion detection mechanism, which is one of the core components of MorphMix. We focus on large, realistic scenarios where participants have different capabilities and are not online all the time. In Chapter 8, we describe the MorphMix simulator and present the simulation results.

Finally, we summarise our work, draw the conclusions, compare MorphMix with similar systems, and provide an outlook on possible further work in Chapter 9.

Appendix A contains a detailed description of the MorphMix protocol and the MorphMix prototype implementation.

Chapter 2

The Mix Network Approach

Mix networks are the basis for our our work. In this chapter, we first describe the basic idea of mix networks and the terminology in general. Then we look at mix networks in more detail, starting with the original approach that was designed to support non-time-critical application such as electronic mail. Afterwards, we describe how the original approach has been modified to support near-real-time applications such as web browsing. In this chapter, we only discuss the basic concepts; systems implemented on these concepts are presented in Chapter 3. Similarly, we only talk about basic attacks in this chapter. We will present more sophisticated attacks in Section 3.2 and provide a more thorough analysis in Section 4.1.

2.1 The Mix Network Idea and Terminology

In 1981, the concept of a *mix network* was introduced by David Chaum [16]. Although the basic idea is independent of the underlying communication infrastructure, nearly all work on actual systems to provide a mix network (see Section 3.1) has been conducted in the Internet context. In this thesis, we also focus on mix networks operated in the Internet context and, unless noted otherwise, assume that the underlying communication infrastructure is the Internet. In this section, we provide the basic idea of mix networks and the terminology we will use throughout this thesis. The purpose of this section is to show the relation between a mix network and the underlying physical net-

work and how a client and a server application can communicate with each other such that an eavesdropper observing the Internet traffic cannot learn the IP addresses of both communication endpoints by inspecting the network or transport protocol headers of the corresponding IP packets. The purpose of this section is *not* to explain measures employed by mix networks to defend against more sophisticated attacks. In particular, the important concept of *layered encryption* will be left out for now and introduced in Section 2.2.1. It should also be noted that there is no such thing as a generic mix network because although all mix networks have fundamental similarities, every proposal for a specific design of a mix network has its own typical properties. Consequently, the model we use in this section to explain the basic mix network idea is also not generic, but has many similarities to MorphMix, which is our own proposal for a circuit-based mix network (see Chapters 5–8). Nevertheless, it serves well to explain the fundamental ideas of mix networks and many of the components we identify in this section can also be found in the following two sections when we describe two specific mix networks.

A mix network is an *overlay network* that aims at providing sender (or client) and relationship anonymity at the IP level. According to our definitions in Section 1.5, this means the recipient (or server) cannot learn the IP address of the sender (or client) and an adversary observing the data being exchanged cannot learn the IP addresses of both communication endpoints. Mix networks do not offer recipient (or server) anonymity because the sender (or client) must know how to contact the other party.

A mix network is composed of multiple *mixes* m_i. Basically, mixes are proxies that relay data, but they provide additional functionality as we will see in Sections 2.2 and 2.3. Mixes can be accessed using their *mix address* that unambiguously identifies a mix. Up to now, all designs and implementations of proposed mix networks have modelled a mix as a service running at the application layer on an Internet host, and we will make use of this concept throughout this thesis. Consequently, mixes are accessed by specifying the protocol to be used (either the transmission control protocol (TCP) [41] or the user datagram protocol (UDP) [89]) together with an IP address and a port, and therefore, the mix address is the combination of an IP address and a port, i.e. mix m_i is the mix address ip_{m_i}:p_{m_i}. Note that one could also imagine a mix network operating below the application layer, for instance by running the mixes directly on a subset of all routers using an extensible router platform [65], but we will not pursue this approach further in this thesis. Figure 2.1 illustrates the basic idea of a mix network and the relationship of the

mix overlay network and the underlying physical network.

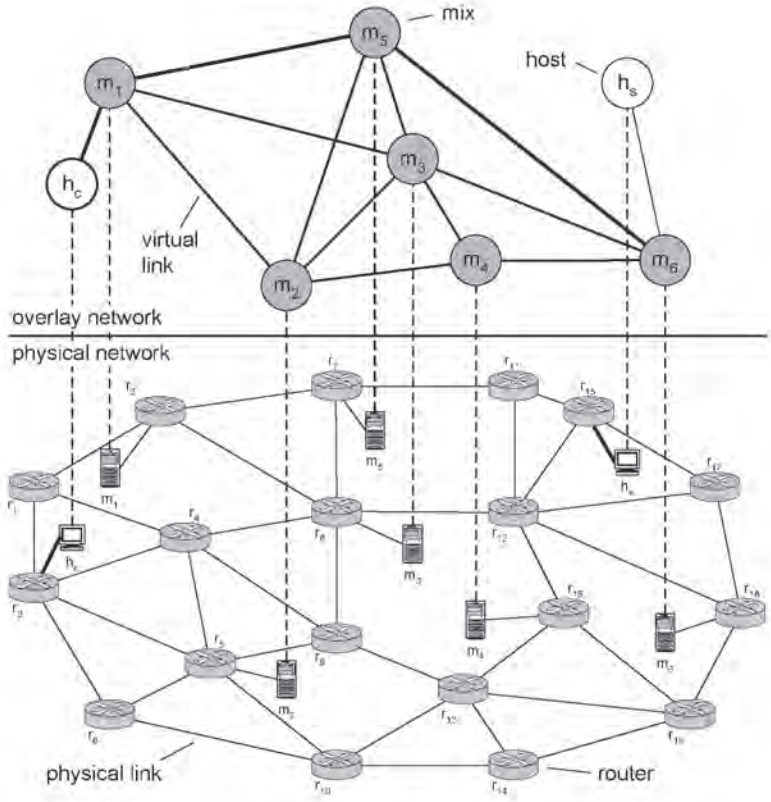


Figure 2.1: *The mix overlay network and the underlying physical network.*

The lower half of Figure 2.1 shows a simplified IP network where routers r_i, hosts h_c and h_s, and mixes m_i are interconnected by *physical links*. We first look at what happens if a client application on h_c communicates with a server application on h_s directly, which is illustrated in Figure 2.2.

Figure 2.2 shows the two hosts h_c and h_s and five routers r_2, r_4, r_8, r_{12}, and r_{15} that specify the likely route IP packets follow when being sent from h_c to h_s according to Figure 2.1. In addition, Figure 2.2 illustrates the ap-

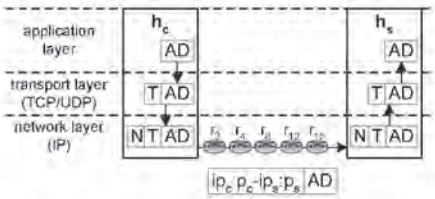


Figure 2.2: *Sending application data directly from h_c to h_s*

plication, transport, and network layer. We do not display layers below the network layer as they are not relevant for the following discussion. We assume the client application wants to send *application data* AD to the server application. The server application can be accessed using the appropriate *address* consisting of h_s's IP address ip_s and the appropriate port p_s. Using this addressing information, the client application sends the application data via the socket interface on h_c. This results in the generation of one or more IP packets that contain the addressing information in the transport (T) and network (N) protocol headers. The headers also contain the address to identify the client application consisting of h_c's IP address ip_c and a port p_c. The resulting socket pair ip_c:p_c–ip_s:p_s unambiguously identifies the communication relationship between the client and server applications and therefore also between h_c and h_s. For simplicity, we assume the application data fits into the payload of a single IP packet, but in reality, sending application data often results in generating as many IP packets. The resulting IP packet is sent to h_s, and leaving out any kind of Network Address Translation (NAT) [43], IP tunnelling [119], or application layer proxies, the addressing information in the IP packet is left unchanged on the route between h_c and h_s. As discussed in Section 1.2 this simultaneous presence of the identifiers of both communication endpoints provides the fundamental prerequisite to easily invade the privacy of the user at h_c at the network level. Note that although not depicted in Figure 2.2, sending application data back to h_c works vice versa. In addition, and also not depicted in Figure 2.2, the IP packets to establish the end-to-end connection if TCP is used already contain the socket pair before the actual application data transfer happens.

The upper half of Figure 2.1 illustrates the mix overlay network. Here, we do no longer look at the routers and physical links of the underlying communication infrastructure, but only at hosts, mixes, and the communication

relationships between them. The mixes build the core of the mix network. At any time, a mix can have a communication relationship with a subset of all other mixes, but not necessarily with with all of them. We identify such a communication relationship with *virtual link* to distinguish it from physical links. To establish a virtual link to m_j, m_i uses m_j's mix address. Virtual links can make use of TCP or UDP (see Section 2.3.4) and can be short-lived or long-standing. The mixes with which a mix currently has established virtual links are its *neighbours*. The basic idea of a mix network is that application data are not directly exchanged between h_c and h_s as in Figure 2.2, but are relayed by a subset of the mixes.

To access the mix network, h_c first contacts any one of the mixes of which it knows the mix address by establishing a virtual link to it. Once a host has established a virtual link to a mix, we say that the host is part of the mix network and we collectively identify the mixes and the hosts that have established a virtual link to a mix as *nodes*. To function properly, a mix network uses its own *mix network protocol*. This protocol is used to exchange *protocol messages*, or simply *messages*[1] between two nodes. Since we assume the mix network to operate on the application level, this protocol is an application level protocol. Messages are exchanged between two neighbouring nodes within fixed-length *cells*[2]. The length of cells and their precise format is part of the mix network protocol. A cell has a header and a payload. Among other information such as a checksum to protect the integrity of the payload or information to forward the payload of the cell correctly along its path, the cell header contains the type of the message that is carried in the payload to distinguish between different types protocol message.

[1] The proposal for terminology in anonymous communication systems (see Section 1.5) uses *message* where we use *application data*, which makes sense if anonymity is considered in general and not bound to a particular system that provides anonymity. However, since we focus on systems based on mix networks, we will use *messages* to identify the protocol messages used in mix networks. In addition, the term *application data* is a reasonable choice because we focus on mix network that enable the anonymous communication between a client and a server application located on two different Internet hosts.

[2] It should be noted that there is no terminology that is commonly used by all mix networks. For instance, the term *cell* was introduced in the context of Onion Routing (see Section 3.1.2) and is more often used in circuit-based mix networks (see Section 2.3) than in Chaumian mix networks (see Section 2.2). Chaum used *item* and systems that were implemented based on Chaum's idea (see Section 3.1.1) used *packet* or *message*. Neither of these terms is well suited because *item* was only used by Chaum, *packet* may easily be confused with IP packets, and *message* are usually used to identify protocol messages. To avoid confusion, we clearly distinguish between *application data*, *message*, *cell*, and *(IP) packet*.

Before h_c can send application data anonymously to h_s, h_c selects a *path* in the mix overlay network. Whether this path is actually selected by h_c or the mixes depends on the implementation of the particular mix network. Likewise, some systems allow paths to be selected only between mixes that have already established a virtual link while others create virtual links on demand. The path consists of h_c and a sequence of mixes. The sequence of mixes used by h_c is also named h_c's *chain of mixes*. As an example in Figure 2.1, we assume h_c has built a path via three mixes, m_1, m_5, and m_6. This path can now be used by the client application to communicate anonymously with the server application, as illustrated in Figure 2.3.

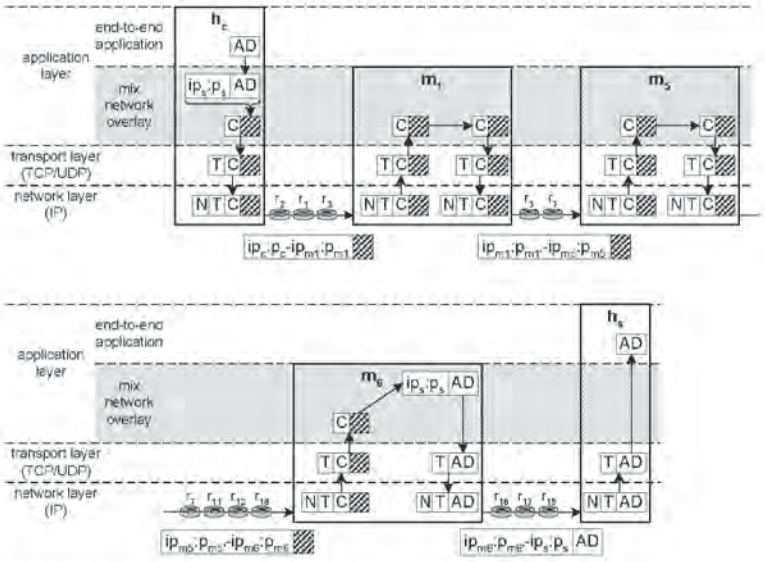


Figure 2.3: *Sending application data via the mix network from h_c to h_s.*

Looking at Figure 2.3, we can see that mix networks introduce a new layer between the traditional end-to-end application and transport layers. The client application again uses h_s's IP address ip_s and the appropriate port p_s to identify the server application on h_s but this time, this information is not directly put into the transport and network protocol headers. Rather, a mix network protocol message is built that contains the information about the server appli-

cation to contact (ip_s:p_s) and the application data (AD) itself. This message is then sent via m_1 and m_5 to m_6. To do so, the message is transported within cells over the virtual links between neighbouring nodes. A cell has a header (C) and the payload of a cell is encrypted (see Section 2.2.1) such that only the last mix in the path can decrypt it, which implies that only the last mix learns ip_s and p_s. The cell itself is then sent over the virtual link to the first mix m_1 using the underlying physical network via three routers r_2, r_1, and r_3. Since the mix network runs on the application layer, a cell is nothing else than application level data, and it is consequently sent within an IP packet by prepending transport (T) and network (N) protocol headers that contain ip_c, p_c, and m_1's mix address ip_{m_1}:p_{m_1}. This means that IP packets exchanged between h_c and m_1 contain the socket pair ip_c:p_c–ip_{m_1}:p_{m_1}, which does not identify both h_c and h_s. Again, we have assumed for simplicity that the message containing AD fits into a single cell and that the resulting cell fits into the payload of a single IP packet. In general, a message can result in multiple cells and a cell may be spread across the payloads of several IP packets. The opposite is also possible, i.e. cells may be so short that multiple cells fit into the payload of a single IP packet. When receiving the cell, m_1 inspects the cell header, generates a new header, and forwards the cell over the virtual link to m_2, which results in an IP packet containing the socket pair ip_{m_1}:$p_{m'_1}$–ip_{m_5}:p_{m_5}. Note that we have silently assumed that m_1 was the initiator of the virtual link to m_5. Consequently, p_{m_5} corresponds to the port specified in m_5's mix address (m_5:p_{m_5}), but $p_{m'_1}$ is completely unrelated to p_{m_1} specified in m_1's mix address (m_1:p_{m_1}). Mix m_5 essentially does the same as m_1 and when the cell finally arrives at the last mix, m_6 decrypts the payload of the cell to extract the message, which reveals ip_s:p_s and AD. This allows m_6 to establish a communication relationship with h_s and forward AD. The resulting IP packets between m_6 and h_s contain the socket pair ip_{m_6}:$p_{m'_6}$–ip_s:p_s, which again does not identify both h_c and h_s. Here again, although not depicted in Figure 2.3, sending a message back to h_c works vice versa. Also, any IP packets to establish, maintain, or close TCP connections between two nodes or between the last mix and h_s have been omitted.

Following this discussion and looking at the mix network layer in Figure 2.3 more closely, we can say that the mix network layer itself can again be separated into two layers: the message layer and the cell layer. Application data sent by the client typically results in a mix network protocol message exchanged between the client and the last mix in the path, and the message is transported hop-by-hop through the mix network within one or more fixed-

length cells.

The discussion in this section has left out many details about mix network that will be explained in the following two sections, but it has already identified several important fundamental properties:

1. No IP packet on the route between h_c and h_s contains the IP addresses of both communication endpoints. In contrast to the case where h_c communicates with h_s directly, this prevents an adversary that observes the IP packets anywhere on the physical route between h_c and h_s to break the relationship anonymity by simply inspecting the IP and transport protocol headers.
2. If at least two mixes are used in a path, no single mix learns the end-to-end communication relationship because a mix knows the IP address of at most one communication endpoint.
3. In contrast to the case when h_c contacts h_s directly, the IP packets arriving at h_s carry the IP address of the last mix in the path and not h_c's IP address. Consequently, h_s cannot easily identify h_c by inspecting the packets it receives, which is an essential property to achieve sender (or client) anonymity.
4. Comparing Figures 2.2 and Figures 2.3, one can see that virtually nothing has changed for h_s. In fact, communication between h_c and h_s in the non-anonymous case works in exactly the same way as communication between m_6 and h_s in the anonymous case. The only difference is that h_s sees m_6's instead of h_c's IP address in the latter case. This is a very important property of mix networks which states that accessing a host anonymously is possible without changing that host in any way, in particular without having to install additional software on that host. Consequently, a host that is contacted via a mix network is not considered to be part of the mix network itself. On the other hand, as mentioned above, it makes sense to consider a client host being part of the mix network once it has established a virtual link to a mix because it communicates with this mix using the mix network protocol by sending and receiving cells. This also implies that additional software must be installed to access mix networks.

If the user sitting at the client host can freely determine the mixes she uses in a path, the mix network is also called a *free-route mix networks*. The extreme opposite are *mix cascades* where disjoint subsets of the mixes form long-standing chains. In this case, all users using the same cascade use exactly the same mixes in the same order. In this case, users using different mix

cascades are in different anonymity sets. In between there are *restricted route mix networks*, where users can still choose among different paths, but with certain restrictions. For instance, one restriction could be to build paths only along virtual links that are already established.

From now on, we focus on the mix overlay network and only consider the underlying physical network when this is required. We are concerned with *application data* that are put into mix network protocol *messages*. The messages are transported within *cells* over *virtual links* between *nodes*. We also do not care if the underlying topology changes as long as nodes in the overlay network can still reach each other. If we say that an adversary has access to cells sent over a virtual link, this means that he has access to the IP packets carrying these cells somewhere on the physical route between the two nodes. Similarly, if we say an adversary can access application data on the route between the last mix and the host that is contacted anonymously, we mean he can access the corresponding IP packets somewhere on the physical route between the mix and the host.

In this section, we have shown that mix networks are overlay networks where data are exchanged between client and server application via a subset of mixes. Consequently, an adversary cannot learn anything by inspecting the network or transport protocol headers of the corresponding IP packets. In the following two sections, we will describe to specific mix network systems that basically work very similar to the mix network described in this section. However, as already mentioned above, every proposal for a specific design of a mix network has its own typical properties. Consequently, it is also not always possible to identify and separate application data, protocol messages, and cells as clearly as we have done it in this section. In the next two sections, we will also introduce more advanced features employed by mix networks to defeat more sophisticated attacks than simply inspecting protocol headers.

2.2 Mix Networks based on Chaumian Mixes

Chaum's original idea of a mix network was targeted at enabling anonymous e-mail communication between a sender s and a recipient r. To distinguish the basic mixes proposed by Chaum from more recent variations, they are often referred to as *Chaumian mixes* and the corresponding systems as *Chaumian mix networks*.

2.2.1 Basic Functionality

Every mix m_i can be identified with its mix address a_i and has a public-key pair consisting of a public key PK_i and a secret (or private) key SK_i. Similarly, the recipient, or strictly speaking the application running on the recipient's computer, can be identified with address a_r (which, in the case of e-mail communication, is an e-mail address) and has a public key PK_r and a secret key SK_r. To use the mix network, the sender must know the addresses and public keys of at least some of the mixes and of the recipient. To send application data AD anonymously to a recipient, the sender picks a subset of all mixes. Assuming that s picks mixes m_1, m_2, and m_3 in this order and $E_{PK_i}(d)$ denotes the encryption of data d with the public key PK_i, s generates the following cell:

$$E_{PK_{m_1}}(E_{PK_{m_2}}(E_{PK_{m_3}}(E_{PK_r}(AD), a_r), a_{m_3}), a_{m_2})$$

The idea is that this cell can be sent to r via the three selected mixes in a way such that each mix can remove a *layer of encryption* to learn the next hop to which the cell must be forwarded to, but nothing more. The mixes used by a sender s is also named s's *chain of mixes*. Figure 2.4 illustrates how the cell containing the application data AD is sent to the recipient along s's chain of mixes.

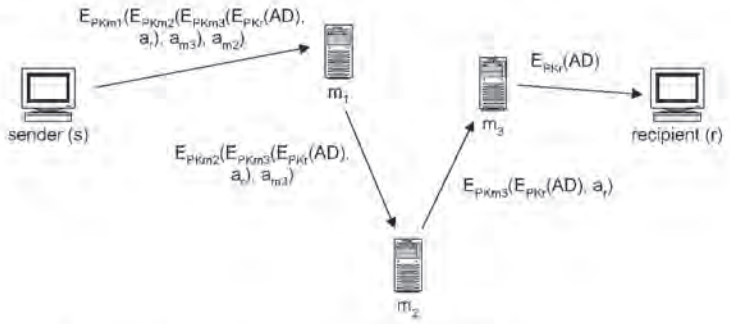


Figure 2.4: *Sending application data AD through a Chaumian mix network.*

The sender establishes a virtual link to m_1, sends the cell, and terminates virtual link again. Having received the cell from s and assuming $D_{SK_i}(d)$

denotes the decryption of the encrypted data d with the secret key SK_i, m_1 performs the following operation:

$$D_{SK_{m_1}}(E_{PK_{m_1}}(E_{PK_{m_2}}(E_{PK_{m_3}}(E_{PK_r}(AD), a_r), a_{m_3}), a_{m_2})) \rightarrow E_{PK_{m_2}}(E_{PK_{m_3}}(E_{PK_r}(AD), a_r), a_{m_3}), a_{m_2}$$

This tells m_1 to forward the resulting cell to m_2 identified with address a_{m_2}. To do so, m_1 establishes a virtual link with m_2, sends the cell, and tears down the virtual link. The same is done by m_2 and m_3 until the application data AD finally arrives at the recipient. In the case of e-mail communication, this usually means a mailbox, which is eventually accessed by the intended recipient who can decrypt the e-mail message. Note that since Chaum's proposal was inspired by sending e-mail messages anonymously, he made the reasonable assumption that users interested in sending e-mail messages anonymously would also want to encrypt them. Therefore, he include the encryption of the application data (corresponding to an unencrypted e-mail message) for the recipient in the mix protocol, although the protocol would also work without encrypting the application data for the recipient. We will see in Section 2.3 when discussing circuit-based mix networks that this encryption for the recipient (or server) is not present in those mix network protocol. Rather, it is left to the application using the mix network, so AD corresponds to either encrypted or unencrypted application data.

To allow the recipient to send a reply, the sender can include a *reply block* into the cell, which is constructed as follows[3]:

$$E_{PK_{m_3}}(E_{PK_{m_2}}(E_{PK_{m_1}}(a_s, k_1), a_1, k_2), a_2, k_3), PK'_s, a_3$$

PK'_s is the public key of a key pair generated by the sender exclusively for this reply block, k_1, k_2, and k_3 are symmetric keys randomly generated by the sender, and a_s is the address of the sender. The reply block tells the recipient to encrypt a reply with PK'_s and send it to m_3 identified with address a_3. Assuming AD′ is the recipient's reply, the following cell is sent to m_3:

$$E_{PK_{m_3}}(E_{PK_{m_2}}(E_{PK_{m_1}}(a_s, k_1), a_1, k_2), a_2, k_3), E_{PK'_s}(AD')$$

[3] We assume that replies take the same path in opposite direction, but in fact, the sender can choose any set of mixes for the return path.

Subsequently, every mix m_i decrypts the first half of the cell using its secret key SK_{m_i} to get the next hop along the return path, encrypts the second half of the cell with the symmetric key k_i, and forwards the resulting cell. Finally, the following cell arrives at the sender:

$$E_{k_3}(E_{k_2}(E_{k_1}(E_{PK'_s}(AD'))))$$

Since the sender knows all k_i and SK'_s, the application data AD′ can be uncovered.

A noteworthy property of Chaumian mix networks is that the mixes are *stateless*: a mix receives a cell, gets all information to process it correctly with the cell, and forgets it after the data have been forwarded. This statelessness of the mixes is one fundamental difference to circuit-based mix networks (see Section 2.3). Consequently, Chaumian mixes are also called *store and forward mixes* and the systems *store and forward mix networks*.

2.2.2 Measures to Maintain the Sender's Anonymity

We have already discussed fundamental properties to protect the sender's identity towards the end of Section 2.1. In particular, we have shown that inspecting only the network and transport protocol headers does not help an adversary. However, Chaumian mix networks employ additional measures to protect against more sophisticated attacks that compare cells at various places in the mix network. As explained in Section 2.1, all cells exchanged between two nodes have the same length. This means that the sender must pad the application data with random bits such that the resulting cell has the desired length. On the other hand, if the application data are too large to fit into a single cell, they are split up into several pieces and sent as multiple cells. Systems that were implemented based on Chaumian mix networks use relatively long cells such that breaking up a application data into multiple cells occurs rarely. For instance, Mixminion (see Section 3.3.1) uses 32 KB long cells. Since every mix removes the address of the next hop, a cell would get smaller on its way to the recipient. To guarantee a cell keeps its length, additional random bits are appended to the cell by every mix before it is forwarded to the next hop. Together with the layered encryptions, this results in all cells exchanged between two nodes in the mix network having exactly the same length and appearing to be composed of random bits.

When arriving at a mix, cells are not forwarded right away but stored until several cells from different senders have been accumulated and forwarded in batches to the next hop. Cells in a batch are reordered such that the incoming and outgoing sequences of cells are not related. Note that a cell can potentially be delayed in a mix for a long time because it does not necessarily have to be included in the next batch that is processed, but this is usually not critical with applications such as e-mail. When a batch is forwarded by a mix, the mix establishes virtual links to all mixes it sends at least one cell, sends the cells in the batch over the virtual links, and tears them down. Consequently, a virtual link is short-lived and exists only as long as it takes for one or more cells to be sent from one node to another. Finally, *dummy cells* (or *cover traffic*) that look like real cells for an eavesdropper can be included in batches if there are not enough real cells or simply to confuse an attacker further.

2.2.3 Basic Attacks on Mix Networks

With attacks on mix networks, we usually mean different kinds of *traffic analysis* attacks. Traffic analysis means observing and correlating the data exchanged at various places in the mix network to get information about who is communicating with whom. One prominent attacker is the eavesdropper that is able to observe all or parts of the traffic sent and received the mixes. This includes cells sent across virtual links and the data exchanged between mixes and the recipients. Such an attacker could try to detect the mapping of incoming and outgoing data at a mix. If he manages to successfully carry out this attack at all mixes, he has broken the entire system and knows all communication relationships between senders and recipients. However, recalling that all cells entering a mix have exactly the same length, this *cell volume attack* is defeated because (1) cells are either forwarded to another mix which results again in cells having the same length or (2) a mix forwards the content of a cell to the recipient, which results in a data volume that is completely unrelated to the fixed size of the corresponding incoming cell. Similarly, using of the *cell coding attack* to compare the patterns of incoming and outgoing data does not help because cells are decrypted (or encrypted if replies to the sender are enabled using reply blocks) and therefore completely change their encoding when traversing a mix. Another option for the attacker is to employ a *timing attack* to correlate data based on the time at which they enter and leave a mix. But since cells are reordered, delayed, and processed in batches, this attack will only reveal little information. It is therefore unlikely that at-

tacking every single mix will be successful. Another strategy is to attack at the edges of a mix network, which means cells on the virtual links between senders and first mix in their path and data on the route between last mix and recipients are compared. But again, this attack is unlikely to succeed for the same reasons as attacking a single mix. The conclusion is therefore that Chaumian mix networks are very resistant to external observers.

Another threat are *collusion attacks* by mixes that share their knowledge. In general, colluding mixes have a significant advantage over the eavesdropper because they know the mapping of incoming and outgoing data at their mixes. If all mixes in a chain are colluding, it is trivial for them to correlate the sender and recipient. If not all but a subset of the mixes in a chain are colluding, the anonymity may decrease depending on the specific design of the mix network. In the case of synchronous mix cascades (see Section 3.2), one honest mix in the cascade is enough to protect the relationship anonymity between sender and recipient as well as if all mixes in the cascade were honest. In the case of free-route mix networks, the relationship anonymity is usually less well protected and depends on the number and positions of the colluding mixes in a chain.

There is an additional fundamental external attack to consider, the *cell replay attack*. Since a mix merely removes or, if replies based on reply blocks are allowed, adds a layer of encryption, processing a cell again produces the same output (although padding bits may change). An attacker can therefore resend a cell to a mix and wait until the same output is produced again. To defend against this attack, a mix must process every cell only once. To do so, all cells that have been processed before must be stored and a newly arriving cell is compared with all of them. To reduce the complexity, cells can include a time stamp that limits the time during which they are processed and the public keys of mixes can be changed periodically. After the time stamp has expired or the public key is no longer valid, a cell must no longer be remembered by a mix.

2.3 Circuit-based Mix Networks

In particular with the advent of graphical web browsers in the 1990's and the popularity of the WWW, researchers became interested in applying Chaum's original idea to near-real-time applications. With near-real-time, we mean applications that benefit from getting an answer quickly but that do not require

hard real-time guarantees. Web browsing is a good example for a near-real-time application because users prefer receiving pages quickly, but the service still works if there is a delay of ten seconds from time to time.

The problem is that Chaumian mix networks were designed for applications that are not time-critical and don't work well to support near-real-time applications for various reasons. One is that a cell must be completely received by a mix before it is forwarded to the next hop. Assuming a large web page and three mixes in the chain, it may take quite a while until the first byte of the page arrives at the client's computer. Another reason is that public-key operations are computationally expensive and not well suited for large amounts of data[4]. Furthermore, delaying cells a long time in a mix until enough cells are accumulated to forward them in a batch is completely out of the question when near-real-time applications should be supported.

The basic solution has been presented in the context of Onion Routing [94], all mix networks that support near-real-time applications are based on this approach, and they are commonly referred to as *circuit-based mix networks*. In this section, we look at the basic properties of circuit-based mix networks and compare them with Chaumian mix networks. Traditionally, circuit-based networks have been used by a client c to access a server s anonymously[5]. We also name mix networks for near-real-time applications *low-latency mix networks*.

2.3.1 Basic Functionality

There are again several mixes that are distributed in the Internet. Every mix m_i is identified with its mix address a_i and has a public-key pair consisting of PK_i and SK_i. The application on the server to be contacted anonymously is identified with address a_s. A mix has established a virtual link to some other mixes but not necessarily with all of them. Two mixes m_i and m_j that have established a virtual link share a secret key, which we identify with k_{ij}. This key is established when the virtual link is set up using the public keys of the two mixes for key-exchange. In contrast to Chaumian mix networks where a application data can be sent to a recipient right away by picking some mixes

[4] Although not proposed by Chaum, this problem could be reduced by encrypting the bulk data with a ephemeral symmetric key and only encrypt the symmetric key with the public key.

[5] In general, circuit-based mix networks can be used for any anonymous near-real-time communication between two peers. But even in peer-to-peer system, one peer is always the initiator (the client) of a communication relationship with another peer (the server).

and generating one or more cells, accessing a server via a circuit-based mix network is a three-step process.

First, a *circuit* is established via a subset of the of the mixes, then data are exchanged anonymously with a server via the circuit, and finally the circuit is torn down. Figure 2.5 illustrates the basic idea of a circuit-based mix network.

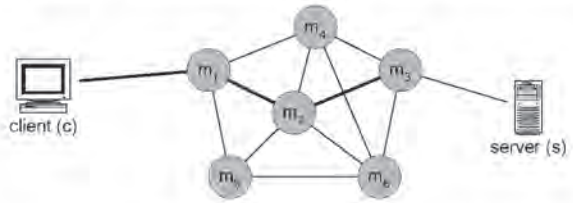


Figure 2.5: *A circuit-based mix network.*

In Figure 2.5, c has established a circuit via three mixes m_1, m_2, and m_3. During circuit setup, the client exchanges a key with each of the mixes it wants to use in the circuit in a way such that only the first mix in the circuit knows who the client is. These keys are then used to add and remove the layers of encryption. There are different ways to establish such a circuit. The one we present here is similar to the method introduced by Onion Routing. First, the client established a virtual link to the first mix it wants to use in its circuit. Then, the client prepares a data structure that contains the keys for each mix. Assuming k_{ci} is the key prepared by c that will be shared between c and m_i, the data structure for our example in Figure 2.5 looks as follows:

$$E_{PK_{m_1}}(E_{PK_{m_2}}(E_{PK_{m_3}}(k_{c3}, a_s), k_{c2}, k_{m_3}), k_{c1}, a_{m_2})$$

The client sends this data structure within one or more cells over the virtual link it just has established with m_1. When receiving the data, m_1 decrypts them using its secret key SK_{m_1}, which reveals both the key k_{c1} and the address a_{m_2} of the next mix m_2. If m_1 and m_2 have not yet established a virtual link, they set it up now. Then, the remainder of the data structure is forwarded within one or more cells to m_2, but m_1 also includes a *circuit identifier* CID_{12} that has only local significance on the virtual link between m_1 and m_2. The reason for using the circuit identifier is that multiple circuits established by different users may include the virtual link between m_1 and m_2. The circuit

identifier is put into the cell header and serves a mix to separate the cells belonging to different circuits and to correctly process an incoming cell. To avoid that different circuits can be identified by an eavesdropper, the circuit identifier is always encrypted with the key that belong to the virtual link in which is used. Consequently, CID_{12} is encrypted using the key k_{12} shared between m_1 and m_2. If $E_{k_{ij}}\langle d\rangle$ denotes the encryption of data d with the symmetric key k_{ij}, m_1 sends the following cell to m_2:

$$E_{k_{12}}(CID_{12}), E_{PK_{m_2}}(E_{PK_{m_3}}(k_{c3}), k_{c2}, a_{m_3})$$

Similarly, m_2 establishes a virtual link to m_3 if they do not yet share a virtual link, picks CID_{23} to unambiguously identify this circuit on the virtual link between m_2 and m_3 and sends the encrypted circuit identifier and the remainder of the data structure within one or more cells to m_3. Finally, m_3 learns that the server with address a_s should be contacted and established a communication relationship with s.

In contrast to Chaumian mixes, virtual links are long-standing and can potentially remain established while several circuits are established and torn down on top of them. In addition, mixes in circuit-based mix networks are *stateful* because they must store some state to send cells back and forth correctly along a circuit once it has been set up. In particular, m_1, m_2, and m_3 must remember the following:

$$\begin{array}{lcll} m_1 & : & c \leftrightarrow m_2, CID_{12} & \text{with } k_{c1} \\ m_2 & : & m_1 : CID_{12} \leftrightarrow m_3 : CID_{23} & \text{with } k_{c2} \\ m_3 & : & m_2 : CID_{23} \leftrightarrow s & \text{with } k_{c3} \end{array}$$

This means m_1 knows everything arriving from c must be forwarded to m_2 using circuit identifier CID_{12} and the corresponding layer of encryption can be removed with key k_{c1}. Similarly, m_2 knows cells arriving from m_1 with CID_{12} are forwarded to m_3 with CID_{23} and the key it shares with the client of this circuit is k_{c2}. Finally, m_3 forwards everything it gets from m_2 with CID_{23} to s and removes the layer of encryption with key k_{c3}.

Once a circuit has been completely set up, the actual data transport takes place where application data are exchanged between client and server. Just like in Chaumian mix networks, application data are transported within cells and all cells exchanged over virtual links have the same length. However,

these cells are much shorter than the typical amount of data exchanged between client and server. For instance, cells in Onion Routing have a length of 128 bytes. Since typical web objects are usually several KB long, a web reply often results in many cells sent back to the client. Although omitted for simplicity, the data structures described above to set up a circuit are also transported within one or multiple fixed-length cells.

To send a application data AD to the server, the client splits them into one or more parts such that the resulting cells have the appropriate length. If necessary, the payload of the last cell is padded with random bits. Assuming the application data AD is split into n pieces AD_i, $1 \leq i \leq n$, the client generates n cells as follows:

$$E_{k_{c3}}(E_{k_{c2}}(E_{k_{c1}}(AD_i)))$$

All n cells are sent over the virtual link to the first mix m_1. Upon receiving such a cell, m_1 knows it's from c, which means k_{c1} must be applied according to the state it has stored about the circuit to remove a layer of encryption. It also knows that the resulting cell must be forwarded to m_2 using the circuit identifier CID_{12} in the cell header. To hide the circuit identifier from an eavesdropper, the entire cell header (including the circuit identifier) is encrypted using the key k_{12} that belong to the virtual link between m_1 and m_2. The cell sent to m_2 therefore includes:

$$E_{k_{12}}(CID_{12}), E_{k_{c2}}(E_{k_{c1}}(AD_i))$$

Decrypting the circuit identifier and consulting the state it has stored about this circuit, m_2 knows that the cell must be decrypted with k_{c2} and forwarded to m_3 using CID_{23} as the circuit identifier:

$$E_{k_{23}}(CID_{23}), E_{k_{c1}}(AD_i)$$

Receiving this cell, m_3 used the appropriate key k_{c3} and forwards AD_i to the server. When all cells have been processed, the application data AD have arrived at s. To send data back to the client, it works vice versa: m_3 gets the application data from the server, splits them such that they fit into the payloads of one or more cells, encrypts the payloads of the cells with the key it shares

with the client, puts the circuit identifier into the cell header, and encrypts the cell header. The cells are then sent back to the client along the circuit in opposite direction, where every mix in the circuit adds a layer of encryption. Eventually, the client receives the cells, removes all encryptions and gets the entire application data. To terminate the communication relationship with the server, the client tears down the circuit by sending a special control cell along the circuit all the way to the last mix, which causes the mixes to remove the state they have stored about the circuit.

Compared to Chaumian mix networks, this design has several advantages to support near-real-time applications. Since the cell size is small, a mix only needs to receive a small fraction of a potentially large amount of application data until data can be forwarded to the next hop in the circuit. In addition, only the setup of the circuit involves public-key operations and the actual data cells use symmetric cryptography, which means processing a data cell is computationally significantly less expensive.

2.3.2 Measures to Maintain the Client's Anonymity

Like Chaumian mix networks, circuit-based mix networks make use of fixed-length cells and layered encryption. In addition, dummy cells can be employed to further complicate traffic analysis and a mix makes sure no cell is processed more than once. However, delaying cells for a long time and processing them in large batches is impossible because the end-to-end delay of cells should be at most a few seconds. It is a fundamental difference between low-latency mix networks and Chaumian mix networks that in the former, a cell is forwarded within some fractions of a second after it has been received by a mix while in the latter, a cell may be delayed in a mix for hours. This has a major impact on traffic analysis attacks and we provide a first insight below. In Section 4.1, we will analyse traffic analysis attacks on low-latency mix networks in more detail.

2.3.3 Attacks on Circuit-based Mix Networks

For the same reasons as in Chaumian mix networks, an attacker that performs a cell volume or cell coding attack by observing the data entering and exiting a mix can be defeated. Timing attacks, however, are much more likely to succeed because data are forwarded quickly. If a mix network is heavily loaded and a mix processes a thousand cells per second, the timing attack

is more difficult but if only a few cells are relayed by a mix, incoming and outgoing data can be related with high probability. Note that what matters when discussing timing attacks is not so much the time a cell remains in a mix, but the total number of cells that are processed by a mix compared to the time a cell remains in a mix. If a thousand cells are processed per hour by a Chaumian mix and they are delayed for one hour on average, then this is comparable with a thousand cells that are processed per second in a mix in a low-latency mix network where cells are delayed for one second on average.

Another attack on a single mix is the *application data volume attack*. The amount of application data are usually larger than what fits into a single cell (a long web reply can easily result in hundreds of cells) and as a result, streams of cells that carry parts of the same application data travel along a circuit. So instead of trying to correlate single cells, an attacker counts the number of cells that may belong to the same application data entering and leaving a mix and tries to correlate the entire application data. Combined with the timing attack, the application data volume attack is a very powerful attack and difficult to defend against. One possibility to increase the resistance is to employ dummy traffic between neighbouring mixes to disguise the patterns of real data cells. This mechanism is also known as *virtual link padding* because "missing" real cells on a virtual link are padded with dummies.

Observing the edges of a low-latency mix network is even more promising. Due to the low-latency property of the mixes, the (part of the) application data enclosed in a cell entering the first mix will leave the last mix towards the server at most a few seconds afterwards. Consequently, combining the application data volume and timing attacks at the edges reveals a lot of information to an eavesdropper. Making this attack more difficult is possible by employing dummy traffic also on the virtual links between clients and their first mix.

Collusion attacks by mixes are also a more significant threat than in Chaumian mix networks. It can be expected that by using timing and application data volume attacks, any two mixes along the same circuit are likely to be able to correlate cells flowing through them if enough cells are sent back and forth along the circuit. In particular, if these two mixes happen to be the first and last mix in that circuit, they can unambiguously link the client and the server and have broken the relationship anonymity. In this case, dummy traffic between mixes does not help at all because the mixes can distinguish dummies from real data cells. Rather, end-to-end dummies sent from the client all the way to the last mix in a circuit and back are used to increase the resistance to

this attack. This is also known as *end-to-end padding* or *circuit padding*.

2.3.4 Ways of Operating Circuit-based Mix Networks

With Chaumian mix networks, virtual links are usually based on short-lived TCP connections that are only established to send one or a few cells. This makes sense because Chaumian mixes may delay cells for a long time and it is therefore reasonable to break the "natural" single end-to-end connection that is used when hosts communicate with each other directly as in Figure 2.2 into multiple ones. In circuit-based low-latency mix networks, however, cells are forwarded quickly and it is not necessarily needed to break the end-to-end connection between client and server. Consequently, two different approaches have found their way into implemented systems and they are illustrated in Figure 2.6 assuming a web browser communicates with a web server.

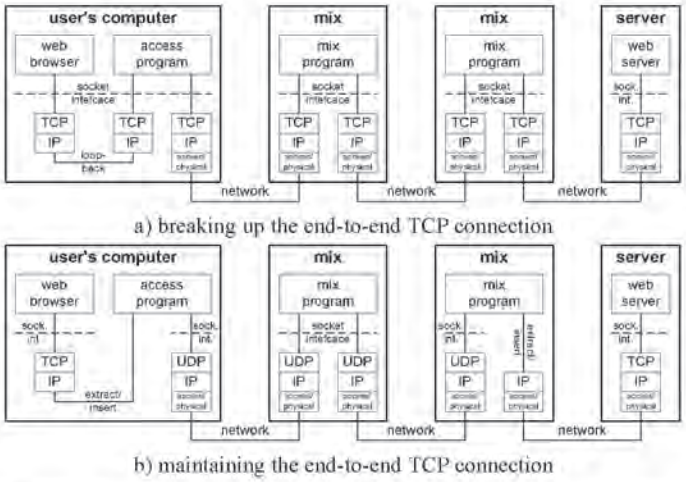


Figure 2.6: *Different ways of operating circuit-based mix networks.*

The first approach shown in Figure 2.6(a) breaks up the end-to-end TCP connections such that a TCP connection is used as the basis for the virtual link between any two nodes along the path from client to the last mix in the circuit. To access the mix network, software that is usually provided by the

developers of the mix network is installed on the client computer. We name this software *access program*. The client application only interacts with this access program, in exactly the same way a web browser accesses a web proxy: a TCP connection is set up to the access program and instead of communicating directly with the web server, requests are sent to and replies are received from the access program. All communication with the mix network and with the web server through the mix network is handled by the access program, transparently to the client application. This includes circuit setup and teardown, and generating and unpacking the cells. The access program can also remove information from the data stream that could possibly give hints to identify the user. In the case of web browsing, for instance, the access program can modify or remove the user-agent field to not reveal information about the user's operating system or her web browser. Similarly, the access program can block every cookie included in HTTP requests. Breaking up the end-to-end TCP connection implies that its properties – flow control and correct delivery of all data in the right order – are guaranteed between two adjacent nodes and between the last mix and the web server, and not end-to-end between the web browser and the web server. Consequently, a mix must not lose any data of an end-to-end connection or the application will usually fail. While breaking up the end-to-end connection seems unnatural, operating a mix network according to Figure 2.6(a) has some practical advantages. First of all, using TCP for the virtual links between two nodes that have very different bandwidth connections makes communication quite easy because the transport layer takes care that all data are delivered correctly over a virtual link. In addition, installing the required software on the client computer and supporting different platforms is also easy because all that is needed is the access program, which runs in the user space and accesses the standard socket interface without requiring special privileges. A disadvantage of this approach is that it requires the client application to be proxy-aware, which is usually the case web browsers and file transfer protocol (FTP) [88] clients. But in general, virtually any application can be made proxy-aware and for many of them there exist implementations that can be downloaded for free (for example PuTTY, which is a proxy-aware telnet [87] and secure shell (SSH)[6] client[7]). Other applications such as e-mail can be supported by specifying the access program as the simple mail transfer protocol (SMTP) [90] server.

[6] http://www.ietf.org/html.charters/secsh-charter.html
[7] http://www.chiark.greenend.org.uk/~sgtatham/putty

The second approach is illustrated in Figure 2.6(b) and does not break up the end-to-end TCP connections. To do so, virtual links are implemented on top of UDP datagrams that are exchanged between two nodes. Data are extracted after the IP layer on the client's computer and passed to an access program. After removal of information that could possibly identify the user's computer such as its IP address, the IP packets are transported within cells to the first mix. The cells are forwarded across the virtual links until the last mix in the circuit is reached. The last mix extracts the IP packets from the cells, sets its own IP address in the source IP address field, and sends them to the web server by inserting them into its network stack. To send data back to the client, it works vice versa. Note that since the data are extracted and inserted at the IP layer, all TCP control data are also exchanged and interpreted end-to-end, which includes TCP segments to establish and tear down the connection and TCP segments that are only sent to acknowledge the reception of data. While this approach seems cleaner because it does not break up the end-to-end connection, it has some disadvantages. Bypassing the socket interface and extracting data from and inserting them into the IP stack is not easily possible without support of the operating system. In addition, this is usually requires special user privileges. Note that with virtual links based on UDP, cells can be lost anywhere in the mix network. But unlike in the approach above, it is not necessary for the mix network to deliver all data in correct order because this is handled by the end-to-end TCP connection used by the application.

Both approaches have advantages and disadvantages. In general, operating the mix network as shown in Figure 2.6(a) is easier because (1) no special privileges are needed on the client's computer to access the mix network because the additional software that must be installed runs entirely in the user space and (2) especially if the capabilities of the mixes are heterogeneous, employing UDP between them could result in losing and re-sending so many cells that the end-to-end performance suffers significantly.

Assuming anonymity-providing techniques become very popular in the future, one could consider providing a special socket type that handles the communication with the mix network, completely transparent to the application. An application would then simply make use of this new socket type instead of traditional TCP (or UDP) sockets.

2.4 Summary

Mix networks are overlay networks and are the most promising technique to anonymise Internet communication. The basic idea of mix networks is to break up the end-to-end communication relationship and relay all data exchanged between the communication endpoints via some mixes to provide sender (or client) and relationship anonymity at the IP level. This means the recipient (or server) cannot learn the IP address of the sender (or client) and an adversary observing the data being exchanged cannot learn the IP addresses of both communication endpoints.

Basically, there are two types of mix networks. Chaumian mix networks are well suited for non-time-critical application such as e-mail and circuit-based mix networks help to anonymise near-real-time applications such as web browsing. Although similar in design, there are some differences between the two types regarding cell length, use of public- or symmetric-key cryptography, and allocation of state in the mixes.

The most significant difference in terms of operation is that in Chaumian mix networks, cells can be delayed for a long time in a mix. This removes virtually all correlation between incoming and outgoing cells. In low-latency mix networks, this is not the case and as a result, they are certainly not expected to offer better protection from attacks than Chaumian mix networks. We will analyse the protection circuit-based mix networks offer in much more detail in Chapter 4.

Chapter 3

Related Work

In this chapter, we look at other work in the field of anonymity. We first present designs and implementations on Chaumian and circuit-based mix networks. Then we look at several papers that cover analysis of mix networks and attacks on them. Afterwards, we examine other approaches than mix networks to achieve anonymity in the Internet. We also look at anonymous and pseudonymous applications that operate on top of an anonymising infrastructure. Finally, we briefly look at recent work on the economics of anonymity, how reputation systems may help to increase the performance of mix networks, and proposals on how to measure anonymity.

3.1 Mix Networks Designs and Implementations

Several mix networks have been proposed and some of them have been operational. Not all of these systems employ all measures used in mix networks (see Sections 2.2.2 and 2.3.2) to protect from attacks. In this section, we list those approaches that make at least use of layered encryption and multiple intermediate hops (mixes) between the communicating endpoints. There have been practical systems based on Chaumian and circuit-based mix networks and accordingly, we distinguish between the two approaches.

3.1.1 Chaumian Mix Networks

Different types of remailers have been implemented and several of them make use of Chaum's original ideas. Today, the different types are categorised as type 0, I, II, and III anonymous remailers. A higher category corresponds to a more sophisticated design that provides better protection from attacks than a lower category. Note that type 0 remailers do not make use of encryption and only one intermediate node is used between sender and recipient. Consequently, type 0 remailers will not be discussed in this section, but in Section 3.3.1.

Type I anonymous remailers, also known as *Cypherpunk remailers* [73] were the first significant implementation of Chaumian mixes. They became available in 1994 and were the result of discussions within the Cypherpunks mailing list. One main motivation was to overcome the problems of Type 0 remailers (see Section 3.3.1). Type I remailers use PGP [129] for the layered encryptions and make use of reply blocks (see Section 2.2.1) for the recipient of an e-mail message be able to reply to the sender. On the other hand, they do not employ fixed-length cells, batching, or replay protection. As of November 2003, there were about 40 Cypherpunk remailers available [42].

Type II anonymous remailers, also known as *Mixmasters* [20, 79] go beyond type I remailers by adding fixed-length cells, protection from replay attacks, and processing of cells in batches. Mixmasters does no longer allow using reply blocks because reply blocks provide a way to point back to the sender of an e-mail message by means of "rubber hose cryptanalysis" [110]. In this attack, the recipient or anyone possessing the reply block may ask or force (by means of threatening, blackmailing, torturing, ...) all operators of mixes used in the path back to the sender to process the reply block for them and reveal the next mix to use. Eventually – if all mixes comply – this reveals the sender of the original e-mail message. It should be noted that even without reply blocks, it would be possible to derive the true sender of an e-mail message. To do so, operators of mixes would have to be required by law to keep logs of the mapping of incoming and outgoing data they process and deliver this information to law enforcement agencies if requested. To our knowledge, no operator of any type of remailer has ever been forced to keep such logs in a large scale (see Section 3.2) and consequently, reply blocks can indeed be considered as the only pointer to the sender. Beyond this potential problem, reply blocks as used by Cypherpunk remailers can be used multiple times, which enables an attack where multiple e-mail messages using the

same reply block are sent to the same mix. Since all resulting cells must be forwarded to the same next hop mix, an attacker can use this knowledge to trace the e-mail messages to the original sender. Unlike Cypherpunk remailers, Mixmasters employ a sophisticated batching strategy, which is known as *timed dynamic pool* batching. A mix keeps a pool of cells and as new cells arrive, they are decrypted and enter the pool. Every t seconds, the mix fires, i.e. it sends a certain number of cells, but only if the pool contains more cells than a certain threshold. In addition, not all cells in the pool are forwarded, but only a constant fraction of them. This means a cell entering a mix can be forwarded during the next time the mix fires, or only after several rounds. As of November 2003, there were about 40 Mixmasters available [42]; most of them actually supporting both the Mixmaster and Cypherpunk remailer protocol. *Babel* [58] is similar to Mixmaster but allows for reply blocks.

Recently, *Mixminion* [26] has been proposed as a standard implementation for a *Type III anonymous remailer* to overcome the flaws of previous remailers. Since reply blocks are convenient, Mixminion allows them again, but every reply block can only be used once. In addition, cells corresponding to replies can no longer be distinguished from forward cells, not even by the mixes themselves. To provide forward anonymity, Mixminion uses both ephemeral keys between each pair of communicating mixes and every mix rotates its public-key pair from time to time. Once keys have been changed, the old versions are forgotten, which means that a mix cannot comply with demands for decryption of a cell that was previously intercepted by an adversary. Changing the public-key pair regularly also makes it more efficient to protect from replay attacks because once the keys have been changed, the cells processed with the old key no longer need to be remembered. To keep its users informed about keys and availability and performance of mixes, Mixminion employs a directory service. Basically, Mixminion employs the same batching strategy as Mixmaster remailers, but when a mix forwards a batch of cells, it always adds a few dummy cells to the batch. This increases the protection from attacks because an adversary no longer knows which of the cells are real and which are dummies. Finally, Mixminion allows for exit policies that allows a user to specify not to receive anonymous e-mail messages at all. To summarise, Mixminion is not a completely new or innovative design, but brings many state-of-the-art techniques together to provide a remailer system that is efficient, practical, and protects from a variety of attacks.

Besides the systems that were actually implemented, several proposals

on how to operate Chaumian mix networks have been made. *Stop-and-Go Mixes* [68] do not batch cells, but have each cell wait a random time in a mix before it is forwarded. Researchers have also worked on improving the robustness of Chaumian mix networks by making sure that it can be verified if a mix has processed all cells correctly, resulting in *Flash Mixes* [61, 75], *Hybrid Mixes* [62, 82], *Provable Shuffles* [52, 80], and other proposals [1, 30]. Although these schemes have very strong and provable properties, they are often not practical because they assume strong coordination and synchronisation between the mixes and result in a significant computational and communication overhead. Since this thesis focuses on practical methods for providing anonymity, we do no longer consider such theoretical approaches.

3.1.2 Circuit-Based Mix Networks

ISDN-mixes, a system to anonymise ISDN-telephony [84, 63] via a mix cascade is based on the idea that the subscribers connected to the same end-office build an anonymity set. The approach makes heavily use of the synchronised telephony system in the sense that all subscribers are always sending data to the end-office so that real phone calls cannot be distinguished from the dummy data. Although the idea could be realised very efficiently in the telephony world, it is not well suited for the highly asynchronous Internet.

Onion Routing [57, 94] was the first circuit-based mix network that became operational in the Internet. The system employs uniform cell length of 128 bytes and layered encryption to complicate traffic analysis. A prototype network was online for about two years until January 2000. The prototype consisted of five mixes (which are called onion routers) that were actually all hosted on a single computer. During the final months of operation, about 50000 connections were established through the prototype network every day. Onion Routing supported remote login (rlogin) [64], HTTP, and SMTP. The Onion Routing analysis[1] and visualisations[2] pages provide some interesting quantitative results that were collected during the operation of the prototype. To put it briefly, Onion Routing was a proof of concept that Chaumian mix networks can indeed be modified to support low latency applications. A successor of Onion Routing named *TOR* was proposed as early as June 2000 [122] and was being tested as a limited public user trial

[1] `http://www.onion-router.net/Analysis.html`
[2] `http://www.onion-router.net/Vis.html`

as of November 2003[3]. It should be noted that the U.S. government, or more specifically the Naval Research Laboratory, was awarded a patent for Onion Routing on 24th July 2001. In particular, the patent covers the method how circuits are established in Onion Routing. Since MorphMix, the system we will propose later in this thesis, employs a different method to establish the circuits, it does not fall under this patent.

The *Freedom Network* [55, 13, 54] was a commercial mix network provided by Zero-Knowledge Systems. Subscribing to the service cost about US$ 50 per year and during its peak, it had about 15000 subscribers[4]. Besides anonymising Internet connections, it also provides pseudonymous e-mail addresses (or nyms). The Freedom Network consisted of about 150 mixes named anonymous Internet proxies (AIPs) operated by various ISPs in Europe, North America, and Japan. Every AIP was connected to the Internet at least at T1 speed (1.544 Mb/s) [117]. The Freedom Network makes use of layered encryption, but – although included in the original design – neither fixed-length cells nor dummy traffic was employed for efficiency reasons. The system designers' argument is that the increased resistance to traffic analysis is not worth the data overhead. In addition, the AIPs do not really mix the traffic but forward the cells in a first in, first out (FIFO) manner. As a result, the Freedom Network offers a slightly smaller level of anonymity than Onion Routing. The Freedom network was shut down in October 2001. An interesting discussion took place on Slashdot[5] about the reasons for the termination of the service because it followed shortly after the terrorist attacks against the USA on 11th September 2001. But apparently, the Freedom network was shut down due to economic reasons[6].

Web MIXes [9, 10] is a very ambitious project that wants to provide anonymous access to near-real-time services in the Internet assuming a very strong attacker model. Instead of a mix network, a mix cascade that employs layered encryption and fixed-length cells is used as the basis. In addition, the mix cascade is operated *synchronously*, which means the continuous time line is split up into slices of the same length and at the end of every slice, all cells in a mix are forwarded. A mix operating in this way is also called a *timed mix*. As a result, all cells the first mix in a cascade receives from the clients during the same time slice are processed and forwarded through the cascade

[3] http://www.freehaven.net/tor/
[4] http://www.politechbot.com/p-03619.html
[5] http://slashdot.org/articles/01/10/04/1526256.shtml
[6] http://slashdot.org/comments.pl?sid=22261&cid=2388977

together. Similarly, all data the last mix in a cascade receives from the servers during the same time slice result in cells being sent back through the cascade together. Based on this concept of a synchronous mix cascade, Web MIXes introduces some novel concepts to beat sophisticated attacks. For instance, in the *flooding attack*, an attacker tries to flood the first mix in the cascade with many cells to make sure that only cells of one other user are processed during the time slice. This would allow the attacker to break the anonymity of the single user as the data in her cells leaves the last mix towards the contacted host. To counter this attack, Web MIXes proposes a ticket-based authentication system where every user gets tickets that allow to send a limited number of cells per time slice. Users must possess tickets for every mix along the cascade and to protect the user's identity, tickets are issued using blind signatures [17]. Web MIXes also proposes a dummy traffic scheme where every user exchanges dummies with the last mix all the way through the cascade and back. The idea is to send dummies whenever the client does not have real data to send to maximise protection from traffic analysis attacks. If Web MIXes could be operated according to its design, it would probably offer the best protection that can be imagined when aiming at supporting low-latency applications. However, it remains to be shown if such a system is practical because end-to-end padding introduce a tremendous overhead (see Section 4.2) and the ticketing mechanism produces a significant management burden as well. A prototype of their system is known as *JAP* (Java Anon Proxy) and has been up and running[7] since 2000. Our trials of JAP provided acceptable performance for web browsing but the system does not yet provide the kind of resistance to attacks Web MIXes is aiming at. In particular, the ticketing and dummy traffic mechanism are not used. Consequently, the level of protection JAP offers in its current state is comparable with Onion Routing. Recently, there has been a controversial discussion surrounding JAP. On 21st August 2003, The Register[8] informed about a back-door that was included into the access program by the JAP developers without informing the users. As a result, the service was logging access attempts to a particular (unnamed) web site and reporting the IP addresses of those who attempted to contact it to the German police. The JAP developers were especially criticised for not having informed the users in first place. A few days later, the back-door was removed again[9] because the JAP operators managed to successfully appeal against the

[7] http://anon.inf.tu-dresden.de/index.html
[8] http://theregister.co.uk/content/55/32450.html
[9] http://www.theregister.co.uk/content/6/32533.html

court order and the issue was finally settled in favour of the JAP developers[10].

PipeNet [23] proposes a synchronously operated mix network for low-latency applications. Like Web MIXes, its design aims at providing resistance to a powerful adversary that is able to observe all traffic sent and received by all mixes and that can selectively block the flow of data entering or exiting the mixes. Besides the basic measures of circuit-based mix networks (see Section 2.3.2), PipeNet proposes to use end-to-end padding, which means that during the time a circuit is established, cells are continuously exchanged all the way through the circuit between the client and the last mix in the circuit. Whenever the client must send a cell to maintain a constant flow of cells, it either sends a real data cell if one is available, or a dummy cell otherwise. The network is synchronous in the sense that during each round, exactly one cell is sent over every virtual link. Between any pair of mixes, there may be more than one virtual link and the number of virtual links between two mixes corresponds on the number of circuits that are currently established between them. The synchronous way of operation makes PipeNet very resistant to several attacks because the systems begins the next round only after a cell has been received over every virtual link. If one virtual link fails, the whole system is brought to a temporary halt until the missing cell is received. On the downside, such a system is totally vulnerable to Denial of Service (DoS) attacks: any user can shut down the entire system by creating a circuit but never sending cells through it. One can argue that such a user should simply be ignored because he would be mainly hurting himself and all the others only a little bit. But then, PipeNet would no longer operate truly synchronously and in addition, a malicious mix can perform the same attack by stopping forwarding cells. Even in the absence of this attack, the whole system adapts its performance to the slowest virtual link and since it must be expected that virtual links happen to be temporarily blocked from time to time due to congestion or failure of the underlying physical network, the system would probably be more often stalled than forwarding cells. PipeNet is illustrative to show that there are theoretical ways of operating mix networks such that they resist very powerful adversaries, but is of little practical value.

The *Anonymity Network* [102, 103, 104] is another proposal for a circuit-based low-latency mix network. Its designers focused on finding a good balance between usability, protection from attacks, and overhead. Unlike in Onion Routing, circuits in the Anonymity Network are not established and

[10] `http://www.datenschutzzentrum.de/material/themen/presse/anonip3.htm`

torn down for a single web request/reply pair. Rather, they are set up, used to communicate for a while with potentially multiple servers in parallel, and eventually torn down. The advantage of this approach is that the load induced on the mixes by public key operations to set up a circuit (see Section 2.3.1) is significantly reduced. On the other hand, contacting different servers through the same circuit could leak information about the user to the last mix in the circuit because the combination of the severs accessed by the user could reveal hints at her identity. The Anonymity Network also introduced a novel cover traffic scheme that results in fewer dummy data than employing constant flows of cells between two neighbouring mixes (see Section 4.1.1), especially during times when the amount of real data is relatively low and if the load is approximately equally distributed among all mixes. Using this cover traffic scheme, every mix forwards cells in rounds and at the end of each round, one cell is sent to each neighbouring mix. If there is a real data cell waiting to be sent over a virtual link, the cell is sent, otherwise a dummy cell is sent. The duration of a round is not fixed but is determined individually by a mix for itself and depends on the amount of incoming data: if many cells are arriving, the rounds get shorter, if fewer cells are incoming, they get longer without getting so long that the end-to-end performance suffers too much. Compared to a system that uses a fixed duration of a round, this dynamic adaptation has the advantage that it decreases the amount of dummy cells during low load situations by increasing the duration of a round and that it can cope with high load situations by making the duration of a round small to process as many cells as the computational power or Internet connectivity of the mix allows. The performance of the Anonymity Network has been analysed in great detail using a testbed consisting of six mixes. One main conclusion of the performance analysis was that the number of dummy cells used between two neighbouring mixes is indeed low using the cover traffic mechanism described above. However, this mechanism can only be employed between mixes but not between clients and mixes and to protect the virtual links between clients and mixes, constant bidirectional flows of cells must be employed on all these virtual links (see Section 4.1.1), which results in much more overhead and a significantly worse end-to-end performance.

Tarzan [50] is an effort to provide a peer-to-peer anonymising network layer. There is no distinction between clients and mixes, every client is also a mix at the same time and called a node in the overlay mix network. Tarzan provides anonymous best-effort IP service and works similar as illustrated in Figure 2.6(b). The system makes use of layered encryption, fixed-length

cells, and cover traffic between any two nodes that have established a virtual link to achieve high protection from traffic analysis attacks. The cover traffic mechanism is especially worth mentioning: each node maintains a bidirectional cell stream with a fixed number of other nodes (its *mimics*). Circuits through a node are only relayed via the node's mimics, which implies that real data are always hidden in the cell streams between the node and its mimics. While this approach limits the possible paths that can be selected for a circuit, it has the advantage that cover traffic is exchanged only between a few of all potential pairs of nodes. The data rates of the bidirectional cell streams between two neighbours can vary within an upper and a lower bound. This seems to be a good idea because different nodes have different capabilities but it is not entirely clear how much protection such a scheme really offers. Mimics are not selected at will by each node, but are assigned in a pseudo-random, but universally verifiable way from the pool of all present nodes. Consequently, the probability that a malicious node has only other malicious nodes as its mimics is very small, which implies it is difficult for an adversary that operates several nodes himself to control all nodes along a circuit. To select the own and verify another node's mimics, a node needs to know about all nodes in the system. Additionally, a node validates each other node upon learning from its presence by contacting it. It is reasonable to assume that Tarzan works quite well if the set of participating nodes is relatively static and does not change too frequently. On the other hand, especially the requirement to know about all other nodes leaves open the question how well Tarzan can cope with a large dynamic environment where nodes come and go. Basic source code of Tarzan has been made available, but no further development of Tarzan was planned as of January 2003 [49]. It is therefore unlikely to see a public user-trial to really evaluate the system.

3.2 Mix Networks Analysis and Attacks

Mazières et al. [74] report about their experiences operating an e-mail pseudonym server. Users could get a pseudonymous e-mail address at the server and deposit a reply block such that e-mail messages sent to their nym could eventually find their way to the intended recipient via Cypherpunk remailers. An important argument of the authors is that one way to attack an anonymity system is to abuse it and stir enough trouble that it must shut down. The operators encountered all sorts of problems typically happening with e-mail

such as bulk mail, mail bombs, and spam. Often, simple mechanisms such as daily limits or not accepting blind carbon copies helped to reduce the problems. Once, somebody posted a message to a newsgroup to exploit a bug in the Unix news server that causes the server to send its password file to a specified e-mail address. The sender of the message specified a nym as the address to send the password file to in order to receive the file anonymously. As the message was replicated across multiple news servers, the number of e-mail messages received by the nym exceeded its daily limit, which caused the e-mail messages to bounce back to the administrators of the attacked news servers. In another case, someone posted child pornography from a nym. The operators were contacted by the FBI and handed out the reply block. This does not directly disclose the identity of the owner of a nym, but helps the FBI to issue more subpoenas. Still, the FBI did not request the operators to keep logs or shut down the service. The paper teaches a valuable lesson by showing that operators of anonymity-providing service must be prepared to cope with abuse. Apparently, the operators of the pseudonym server managed to do so during the two years the service was operational.

Raymond [92] provides an introduction to the traffic analysis problem. The paper covers both Chaumian and circuit-based mix networks and gives a thorough overview of different attacks. Among others, it mentions the *intersection attack*, which exploits the fact that users tend to communicate with the same communication partners (e.g. web servers or e-mail recipients) whenever they are online. By performing an operation similar to an intersection of the sets of active users at different times it is likely that the adversary can gain some information about communication relationships. Raymond also mentions the *tagging attack* where an attacker slightly modifies a cell such that it can be recognised later in the chain of mixes. Assuming the adversary owns the first and last mix in a chain, this attack may make it possible to link sender and recipient of a cell.

Zero-Knowledge Systems have provided documents with a security analysis of their Freedom Network [118, 5]. They point out several attacks their system is vulnerable to. In particular, they state that since there is no cover traffic or traffic shaping, an adversary capable of observing most of the Freedom Network should be able to learn quite a lot about communication relationships by performing statistical analysis. Dai [24] has pointed out that the Freedom Network is vulnerable to a tagging attack where any two colluding AIPs can easily learn whether they are the first and last AIP in a circuit. The attack exploits the fact that a message authentication code (MAC) to check

the integrity of a cell sent through the network is only used between the client and last AIP. To execute the attack, the first AIP simply modifies the payload of a cell it receives from the client and the last AIP waits until it detects a cell that fails the MAC check.

Berthold et al. [12] discuss different methods of how to choose the route through a mix network. In particular, they compare free-route mix networks where every user picks the mixes she likes and synchronously operated mix cascades where all users use the same mixes in the same order. The authors assume a strong adversary that controls most mixes in the system and demonstrate that under these circumstances, the mix cascade has advantages compared to free-route mix networks. The main reason is that in a mix cascade, cells arriving at the first mix are processed together in a batch and are only forwarded when the complete batch has been received. The anonymity set remains the same because the batches always contain the same cells from the same set of users. A mix networks, on the other hand, works asynchronously and every batch processed by a mix contains cells from different sets of users. Assuming an attacker observes or owns most mixes and a sender sends multiple cells to a recipient, the adversary should be able to reduce the set of potential senders of a cell by calculating intersections of incoming and outgoing cells at the mixes. One can also say that although mix networks can support many more users and therefore potentially have much larger anonymity sets, the effective anonymity set may be smaller than in mix cascades assuming a very powerful adversary. The paper also briefly discusses dummies and concludes that dummies exchanged only between neighbouring mixes are of no help against mixes controlled by an adversary.

An analysis of Onion Routing is given by Syverson et al. [122]. The authors point out that an adversary controlling or observing both the first and last mix along a circuit should be able link the communication endpoints. The paper also proposes to use at least partial end-to-end padding in addition to virtual link padding to significantly complicate the task for an adversary that controls some mixes.

Back et al. [6] talk about traffic analysis and trade-offs in anonymity-providing systems. They look closely at the Freedom Network and PipeNet and come to the conclusion that designing such a system means finding a balance between traffic analysis resistance, performance, resistance to DoS attacks, and bandwidth cost. The paper contains one very important statement that should always be kept in mind when aiming at designing a practical anonymity-providing system: in anonymity systems, usability, efficiency, re-

liability, and cost become security objectives because they affect the size of the user base which in turn affects the degree of anonymity that is possible to achieve. Simply spoken, this means that anonymity-providing systems should aim at supporting as many users as possible because this means potentially larger anonymity sets and therefore better anonymity.

Wright et al. [127] examine the vulnerability of anonymity-providing systems to the *predecessor attack*. The attack bases on a scenario where there is no distinction between sender and mixes like in the Onion Routing local-COR configuration [121], Tarzan (see Section 3.1.2), and Crowds (see Section 3.3, although we do not consider Crowds as a mix network variation because of the lack of layered encryption). The attacks requires that a subset of all mixes is controlled by an adversary and that there are recurring sessions between the sender and recipient, for instance a client that frequently connects to the same web server or a sender that often sends e-mail messages to the same recipient. In addition, the attack requires that there is some information available in the cells that allows the last mix in a chain to link the different sessions of the same sender. According to the authors, this includes cookies, user IDs, or e-mail addresses. The attack exploits the fact that assuming every mix along a chain is picked randomly from the set of all mixes and the last and one other mix along this chain is malicious, then the predecessor of the first malicious mix in the chain is more likely to be the actual sender than an intermediate mix in the chain. Carrying out this analysis over multiple rounds allows to identify the sender with increasing probability. It should be noted that the attack works always no matter how sophisticated the methods employed by the mix network to resist traffic analysis are, but takes much longer (more sessions are needed) in the latter case. In addition, the attack is more difficult if there are more mixes in the system and if the percentage of compromised mixes is smaller. Follow-up work by the same authors [128] contains simulation results of the predecessor attack, which show the attack times are significantly lower in practice than the upper bounds given in their theoretical analysis. The paper also shows that choosing the mixes in the chain non-randomly increases the resistance to the attack because choosing always the same first and last mixes and not both are malicious does not allow the adversary to learn anything. Finally, and not directly related to the predecessor attack, the paper discusses the intersection attack in a dynamic environment where mixes join and leave. Again assuming that different sessions can be linked by malicious last mixes in different chains, exploiting the fact that different nodes are present at different times should eventually iden-

tify the sender. To do so, the adversary also needs a complete list of all nodes available at any time, which is exactly what is offered to every single node in Tarzan and Crowds. Although very interesting and insightful, the practicability of these attacks are questionable because linking different sessions to the same sender is often impossible, for instance by carefully filtering HTTP headers and cookies in the web browsing scenario. However, in certain situations or when only one or very few senders communicate with a particular recipient or server, the attack is definitely of practical value.

A method to prevent *long-term intersection attacks* by using dummy traffic is presented by Berthold et al. [11]. The authors propose that pre-generated dummies are being sent to the communication partner (e.g. a web server) during the user's offline periods. The simple approach would be a dummy server that just generates traffic during the offline hours, but this requires too much trust in the single service and the user would have to tell the server when to send traffic and when not, since no dummies should be send during the user's online hours to not expose statistical significant variations in total traffic generated. Therefore, the authors propose a distributed database that contain prefabricated dummies by all users. The distributed database publishes regularly dummies and users pick them randomly. The databases themselves are accessed via an anonymised channel. A user that wants to prevent her own dummies from being processed simply picks her own dummies and does not send them. Tickets similar as in Web MIXes (see Section 3.1.2) are proposed to prevent an adversary from draining or flooding the distributed database. The solution is very costly in terms of data and computational overhead and the proposed design principles should mainly be considered as inputs for further work to make such a scheme practical.

Serjantov et al. [112] analyse the different batching strategy of mixes. The authors focus on the question if an attacker can manipulate cells entering a mix such that the produced batch contains only one cell unknown to the attacker. This may involve delaying or dropping incoming cells (a *trickle attack*) or flooding the mix with attacker cells (a flooding attack). The analysis shows that adding a "pool" to the mix and forwarding only a fraction of all cells when the number of cells in the pool is above a certain threshold can significantly improve anonymity. Díaz et al. [31] also examine the batching strategies of mixes and present a generalised framework for expressing them. The authors also propose a new batching strategy and name a mix employing this strategy *binomial mix*. It is basically a timed pool mix that adapts the number of cells to flush in each round to the traffic load. An advantage of this

new mix is that it resists well to the flooding attack because it makes guessing the actual number of cells in the mix difficult. In another paper [113], Serjantov et al. present an analysis of the anonymity of a timed pool mix and compare it with threshold pool mixes.

Kesdogan et al. [67] analyse the impact of the intersection attack. They assume an anonymity-providing system that itself provides perfect untraceability between incoming and outgoing cells (e.g. a mix or a mix cascade that reorders and changes the encoding of the cells). There are a total of n system users and every user has m communication partners. During every round, b senders send b cells to $n \leq b$ recipients. Note that always all b cells are processed by the system, i.e. there is no pool in the mix. The analysis shows the number of rounds it takes for an attacker able to observe all incoming and outgoing cells to identify the m communication partners of a particular sender with high probability. Not surprisingly, it takes longer if b or m get larger. Although not mentioned in the paper, employing a more advanced batching strategy that allows introducing long delays should significantly reduce the practical impact of the attack.

Danezis [25] examines mix networks with restricted routes. The idea is that every mix in a large mix network is only connected to relatively few other mixes, e.g. to 10% of all mixes. The author shows that the number of mixes that should be included in a chain such that traffic is "mixed" as well as in fully connected mix networks is logarithmically dependant on the total number of mixes. In addition, an interesting result is that mix networks with restricted routes are less vulnerable to intersection attacks because when processing a batch, the probability that there is a cell sent out on any virtual link is larger than in fully connected mix networks. Although the paper does not investigate how the virtual links between mixes should be chosen in practice, it is a valuable addition to the field because it shows there is a middle ground between free-route mix networks and extremely restrictive mix cascades.

3.3 Techniques beyond Mix Networks

There are a few noteworthy techniques to get a certain degree of anonymity that do not base on mix networks.

3.3.1 Simple Remailers

Simple *Remailers* (also known as Type 0 anonymous remailer) are proxies between sender and recipient that strip of headers from e-mail messages and replace the original sender address with a pseudonym. One popular remailer, `anon.penet.fi`, has been operational from 1993 to 1996. For Alice with address `alice@home.org` to send an e-mail message to Bob with address `bob@work.org`, she sends it to `anon.penet.fi`, which replaces Alice's address with, for instance, `an7184@anon.penet.fi`. This even allows Bob to reply to Alice because `anon.penet.fi` remembers the mapping between Alice's real address and her pseudonym. Type 0 remailers do not make use of encryption and the protection they offer from traffic analysis attacks is quite low. In particular, the remailer offers a single point of attack and in the case of `anon.penet.fi`, this was exploited in the *Church of Scientology vs. anon.penet.fi* case [81]: in early 1995, somebody posted a message to `alt.religion.scientology` via the anonymous remailer. Scientology representatives claimed the information of the message was stolen from an internal Scientology computer and used Interpol and the Finnish police to demand the true name of the poster of the message. Johan Helsingius, the owner of the remailer, reluctantly complied, fearing that if he resisted, he might be forced to give up his entire database that matched anonymous IDs to true names. In 1996, Scientology once again demanded the names of two anon.penet.fi users; as a result, Johan Helsingius shut the remailer down on 30st August 1996 [59]. At its peak, the remailer had 500000 registered users and processed 10000 messages per day.

3.3.2 Proxy Forwarders

The Anonymizer [22] is a simple proxy-based service that offers anonymous web browsing. The system works similar as a web proxy in the sense that all data exchanged between the user's browser and the web server are relayed by the Anonymizer. The advantages of the system are that it is simple and that the delay it introduces is relatively low compared to more sophisticated systems. The disadvantages are that the level of anonymity it offers is quite low and that the end-to-end relationship is not anonymous with regard to the Anonymizer itself.

Crowds [95] collects users in a group (the "crowd") to browse the Web anonymously. Crowds can be considered as a peer-to-peer system because

every user issues web requests and forwards data for others. However, it relies on a centralised lookup service to inform all users about all other current users in the crowd. A user is represented in a crowd by a process on her computer called "jondo". To join a crowd, the jondo contacts the lookup server to learn about the other jondos. Similarly, the lookup server informs the other jondos about the new participant. A user selects her own jondo as a web proxy. If she wants to request a web page, the jondo forwards the request randomly to another jondo in the crowd. Similarly, when a jondo receives a request from another jondo, it makes a random choice to either forward the request to another jondo or submit it to the server the request is intended for. The reply from the server uses the same path in opposite order to find its way to the requester. To the outside, the system provides anonymity in the sense that any crowd user could have requested the web page. Crowds does not make use of layered encryption but uses pairwise keys to encrypt the data between two jondos. Crowds does also not employ fixed-length cells. As a result, an eavesdropper can easily trace data and any two jondos along the same path should easily be able to correlate data flowing through them. One important feature of Crowds is *plausible deniability* because a user can always claim she merely relayed the data for someone else. Likewise, a jondo never really knows if its predecessor in a path is the original requester or not. A weakness of the system is the lookup service that provides a single point of attack and a potential bottleneck if a crowd gets large and its users change frequently.

3.3.3 Broadcast-Based Approaches

Chaum's solution to the *Dining Cryptographers Problem* [18] provides information theoretic sender and relationship anonymity. The basic idea is that all participants continuously send random-looking data as broadcasts to the entire group and only one of them is really transmitting a meaningful message. The message is encrypted using the intended recipient's public-key such that no other recipient can read it. Although appealing because of its information theoretic guarantees, it is of little practical value because participants are arranged on a logical ring and each participants must pre-share long random bit-sequences with each of its two neighbours. In addition, only one participant can send a message at any time, and a malicious participant can easily disrupt communication by sending data all the time.

P5 [116] is a peer-to-peer-based approach that aims at providing sender, recipient, and relationship anonymity between nodes. P5 organises the nodes

in a logical binary tree. Each node represents a broadcast group, which is defined as the sub-tree of which the node is the root, and every group includes all groups below it. Similarly, a user is not only member of the group that she actually represents, but also of all other groups on the path to the root of the tree. Recipients of messages are addressed by (one of) the broadcast groups in which they reside. When a message is sent to a broadcast group, it is propagated to all child-groups of that group. If the load gets too high, nodes drop message. The rule is that messages sent to large broadcast groups (closer to the root of the tree) are dropped with higher probability than messages sent to smaller broadcast groups. This allows each individual node to make a trade-off between communication latency, bandwidth usage, and anonymity by sending messages to larger or smaller broadcast groups in which the recipient resides: choosing a smaller group means less anonymity, but higher probability (and therefore lower latency) that the message is not dropped. P5 produces significant data overhead because every node in P5 is always broadcasting data to conceal her real sending of data and it is assumed that the traffic is always at the maximum of what nodes can sustain.

3.3.4 Anonymous Publishing

Work on anonymous publishing has been strongly influenced by Anderson's work on the *Eternity Service* [3]. The main goal of Eternity was not so much anonymous publishing, but censorship-resistance in the sense that it should be difficult for anyone to delete a document once it has been published. It assumes there are several Eternity servers available where publishers upload documents together with the requested storage duration onto multiple servers. Having done so, the publisher forgets about the servers where she has placed the documents, which removes the capability for the publisher to easily delete all available copies of the document. Anderson argues that forgetting about the locations where the documents have been stored is required for censorship-resistance, because if the capability to revoke a document exists, an adversary has incentive to find who controls this capability and threaten or torture that person until revocation takes place. To make it difficult for an adversary to identify the servers on which a publisher publishes the documents, upload is done via an anonymous channel and payment for the service is done with anonymous e-cash [15]. To download a document, queries are done via broadcast and document delivery is achieved through anonymous remailers. Anderson's design leaves open many practical questions such as

updating documents and fluctuations in the server population, but shows that a censorship-resistant anonymous publishing system is possible.

The *Rewebber Network* [56] aims at anonymous publishing. Rewebbers are proxies that relay web requests via multiple hops until they arrive at the server containing the requested document. A locator is used to request documents and is similar to a reply block in Chaumian mix networks. Using public-key cryptography and layered encryption, it guarantees that only the identity of the next rewebber in the chain to use is revealed. The actual web reply is sent back to the requester using the same chain in opposite direction. On the way back, each rewebber removes one layer of encryption from the document itself (the keys to do so are included in the locater and are remembered when the locater is sent through the chain of rewebbers) until the document is sent in plaintext from the first rewebber in the chain to the requester. Except for the identity of the first rewebber, a locator only contains random-looking data and is totally unrelated to the document it points to. To solve this naming problem, the authors propose *TAZ (Temporary Autonomous Zone*[11]*) servers* that provide the mapping of intelligible document names to locators. Note that if the server storing the document deletes it, it is lost until it is published again. However, since the document is encrypted multiple times, it is hard for the server to decide why to delete a document at all if not for storage restrictions. *JANUS* [29] is a proposal similar to the Rewebber Network.

In *Publius* [124], the publisher encrypts a document with a secret key. Using Shamir's secret-sharing algorithm [114], the key is then split into n shares such that any k of these shares are sufficient to reconstruct the key. Then the publisher picks n Publius servers and anonymously sends the encrypted document plus one share of the key to each of these servers. The name of the document is also published together with the addresses of the n servers, which forms an URL that can be used by potential readers to access the document. To do so, a local web proxy recognises such an URL and fetches all shares and the encrypted document, reconstruct the key, and decrypts the document. The main idea to encrypt the document but split the key is for a server not being able to easily read the document and delete because its operator does not like the content. In this sense, Publius does not only provide publisher anonymity, but also resistance to censorship: as long as at least k shares remain available, the document can be retrieved. A drawback is

[11] `http://www.t0.or.at/hakimbey/taz/taz.htm`

that a server can find out what documents it is storing by searching for URLs containing its own address.

Freenet [19] is a peer-to-peer file-sharing system. It has the property that popular files are replicated on several nodes while infrequently requested files eventually vanish. Both searching files and downloading them is anonymous in the sense that the requester does not know where it downloads the file from and the one offering the file does not know who the requester is. Nodes also do not know what files they are storing because the files are identified with the hash of the file name and the file itself is encrypted using the file name as the key. To search for a document, the requester must know the precise file name, builds its hash, and sends the request to the one of its current neighbours that "most likely" stores it. If that node indeed stores the file, it sends it back to the requester, otherwise it sends itself the request to the neighbour that is most likely to store it. This continues until the file is found or a time-to-live (TTL) counter reaches zero. One drawback is that the choice to drop a file is a purely local decision, which implies the system cannot guarantee a certain lifetime of a file.

The *Free Haven Project* [36] aims at deploying a system for distributed, anonymous, persistent data storage which is robust against attempts by powerful adversaries to find and destroy any stored data. The specific goals are (1) anonymity for all parties, (2) accountability using reputation and micropayment schemes without sacrificing anonymity, (3) persistence in the sense the publisher of a document determines its lifetime, and (4) flexibility, which means the system functions smoothly as peers dynamically join or leave. Although the project was not finished as of November 2003, it was put on hold because fundamental problems must be addresses first[12]. In particular, a working reputation system seems to be very difficult to establish and document retrieval based on broadcast is too inefficient.

Tangler [123] is another distributed document storage system that provides reader and publisher anonymity and censorship-resistance. The main idea is to entangle different documents by transforming them into fixed-sized blocks in such a way that many blocks belong to multiple documents. This not only diffuses responsibility from particular servers for particular documents, but also makes replicating parts of other documents an inherent part of publishing, and even gives a plausible cause for replicating other blocks in the system.

[12] `http://www.freehaven.net`

The goal of *GNUnet* [70, 7] is to provide practical anonymous and censorship-resistant file-sharing. GNUnet is a peer-to-peer system and data are basically forwarded similar as in mix networks. But unlike in traditional mix networks, the servers are also part of the system itself. This implies there are no easily identifiable "edges" and attacking the system at the edges (see Sections 2.2.3 and 2.3.3) is more difficult. The intended practicability of GNUnet has caused the designers to find a reasonable trade-off between anonymity and efficiency. For instance when a reply is sent back to the requester, it is possible to short-circuit two nodes to circumvent a highly loaded node along the path. This makes GNUnet more vulnerable to attacks, but the designers believe this is worth the improved performance. GNUnet also makes trade-offs when looking at censorship-resistance, where some resistance is sacrificed for improvements to search for and find documents.

3.4 Other Applications

Besides techniques to exchange data anonymously, there have been proposals for other applications that often rely on an underlying anonymous communication infrastructure.

Chaum et al. [15] proposed *untraceable electronic cash*. Alice can go to a bank, withdraw some digital coins from her account, and spend the money. Later, when the merchant takes the money to the bank, the bank checks if it has issued the coin earlier and if the coin has not been spent before. The trick is that the bank cannot link the coin it receives from the merchant to Alice, since the coin was blinded by Alice when the bank issued it, using a technique called blind signatures [17]. Untraceable electronic cash was never widely accepted, mainly because of its reliance on the cooperation of banks and software wallets that were difficult to use.

Low et al. [71] propose *Anonymous Credit Cards*. The idea is that a customer has two accounts in different banks. The first bank knows the person's identity whereas the second does not (e.g. a numbered Swiss bank account). Since the first bank knows the person, it is willing to grant her credit. The second bank is not willing to grant the person credit, as it does not know her. However, on the person's request, the first bank agrees to put credit into the anonymous account at the second bank. When the person pays, she uses a credit card for her account at the second bank. The bank checks the person's credit and – if she is credit-worthy – authorises the payment. Eventually, the

second bank sends a bill to the first bank, which sends a bill to the person. This separates the information such that no party knows the identities of both customer and merchant.

The *Lucent Personalized Web Assistant (LPWA)* [53] provides its users with aliases where each alias consists of an alias user name, alias password and alias e-mail address. The LPWA acts as a proxy and whenever the user has to submit user name, password, or the e-mail address, he uses predefined two character escape sequences (\u, \p or \@), and LPWA replaces them with the appropriate alias. The LPWA provides a simple and effective way to generate and use consistent pseudonyms.

Rennhard et al. [91, 104] propose a system to enable *Pseudonymous E-Commerce*. Using different components such as pseudonymous certificates and pseudonymous credit cards (using a protocol called *Pseudonymous Secure Electronic Transaction* [101]), the system allows a customer to browse through an e-shop, select goods, and pay the goods with her credit card such that neither the e-shop operator nor the credit card issuer nor an eavesdropper is able to get any information about the customer's identity. The system also guarantees that during the credit card payment process, none of the involved parties can act dishonestly without being detected.

The *Secure ANonymous GRoup InfrAstructure (SANGRIA)* [125, 126] proposed by Weiler enables secure and anonymous group communication. Her work combines traditional unicast-based approaches for privacy with authenticated and encrypted group communication. Thereby, only users who fulfil certain conditions are allowed to join the secure anonymous group, non-members of the group cannot understand the data, and the identity of a member cannot be uncovered by outsiders of the group. Additionally, a member may hide its identity from other group members.

3.5 Economics of Anonymity and Reputation

Two parties interested in keeping the content of their communications secret can easily do so by employing appropriate cryptographic measures. With anonymity, it is different because anonymity cannot be created by the sender or recipient. Anonymity must be provided be some third party, and the question about who could be interested to do so arises. One possibility is to pay mixes for the service they provide, and Franz et al. [48] have introduced and analysed different protocols for payment of an anonymity service.

Acquisti et al. [2] have looked at the incentives for participants (as senders or mixes) in mix-like anonymity services. There are some noteworthy remarks in that paper. For instance, before high-sensitivity users (those that really want good anonymity) join a system, there must already be several low-sensitivity users in the system to provide the necessary noise for good anonymity. Also, weak security parameters (small batches, lower latency) may actually provide stronger anonymity by attracting more users. The authors also states that high-sensitivity users have incentive to run mixes themselves to be certain the first mix in their chain is honest.

Other work targets at increasing the reliability of mix networks. Using reputation systems [35, 37] should enable users to use chains of mixes that will succeed in delivering a message with high probability. Although still in their infancy, such ideas may eventually help to increase the reliability of practical mix networks because they result in far less overhead than proposals aiming at giving provable guarantees about a system's robustness (see Section 3.1.1).

3.6 Measuring Anonymity

Two independent works by Díaz et al. [32] and Serjantov et al. [111] propose metrics to measure anonymity. The main argument is that the traditionally used anonymity set does not take into account potentially different probabilities of different members of the anonymity set actually having sent or received a cell. Both proposed measures are similar and base on Shannon's definition of entropy [115]. The metrics are well suited to analyse the anonymity of a single Chaumian mix or simple systems based on strict assumptions, but it is likely that they provide only a starting point to develop a more sophisticated anonymity metric that is able to take into account the changing state of practical systems over time, for instance users or mixes that are joining and leaving or links that are temporarily blocked. Due to their early development stage, we do not make use of these metrics in this thesis.

3.7 Summary

In particular since the early 1990s, a lot of work has been conducted in the field of anonymity, including work on systems to enable anonymous com-

munication and publishing, analysis of and attacks against proposed systems, anonymous payment methods, studies of the economics of anonymity, and how to measure anonymity.

With the notable exception of Cypherpunk remailers, Mixmasters, and the Anonymizer, none of the proposed systems has been available to the broad public for a long period. In particular, there is no sophisticated system based on mix networks to enable anonymous low-latency Internet access in widespread use as of today; either because they failed for economical reasons (Freedom), never made it beyond a limited user trial phase (Onion Routing, JAP, Anonymity Network), or were never implemented or deployed (PipeNet, Tarzan). We therefore conclude that none of these systems was really well suited or optimised to provide practical anonymity for a large number of users. Considering all the attacks against and analysis of systems in Section 3.2, we argue that finding the optimal design for such a system is closely associated with finding a reasonable trade-off between usability and protection from attacks. We will analyse this more closely in the next chapter.

Chapter 4

A Detailed Analysis of Mix Networks

In this chapter[1], we perform an analysis of mix networks. We focus on circuit-based mix networks that aim at supporting low-latency applications, although several of our findings apply to Chaumian mix networks as well. We first discuss why anonymity is so difficult to achieve. Afterwards, we provide a quantitative analysis of mix networks to estimate how big a mix network must be to support a certain number of users and to analyse the costs of dummy traffic overhead. Then we give arguments for what we believe is a realistic threat model for different mix networks. We also analyse different mix network approaches in light of our realistic threat model and derive conclusions which approaches are better suited than others to provide practical anonymity for a large number of users.

4.1 Why Anonymity is so Hard

We base our analysis on a client/server scenario where the client contacts a server anonymously. We assume the goal of an attacker is to learn who communicates with whom by means of traffic analysis. With mix networks, there are different kinds of attackers that can be described with the following

[1] The work in this chapter has been published in a refereed paper [99]

three attributes:

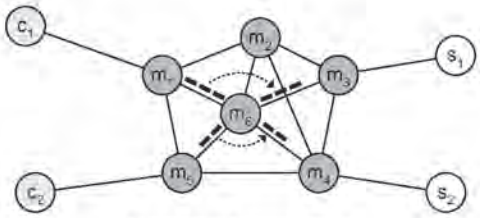
- *Passive vs. Active.* Passive attackers monitor the cells exchanged between nodes over virtual links and the data exchanged between the last mix in the circuits and servers. Active attackers have all capabilities of passive attacker, but in addition, they can insert, modify, duplicate, or delete the data exchanged between two nodes or the last mix in a circuit and the server.
- *External vs. Internal.* External attackers do not operate mixes themselves. Internal attackers control one or more mixes, for instance by running them themselves, which means an internal attacker knows the mapping of incoming to outgoing data at the mixes he controls and also which of the cells originating or ending at these mixes are real data cells and which are dummies.
- *Partial vs. Global.* A global attacker can attack the entire system, while a partial attacker can attack only parts of the system.

Basically, any combination of these attributes is possible. In addition, an attacker can be both active and passive and both internal and external at the same time. For instance, an adversary may operate a few mixes himself and modify the data flowing through them, which makes him an internal active attacker. At the same time it could be possible that he manages to passively observe a few other mixes, which makes him a passive external observer. In addition, some combination of attributes enable the adversary to break any mix network. For instance, the global internal attacker that operates all mixes can trivially relate all communicating endpoints. Note also that every user of a mix network is an active attacker because she can observe and manipulate all cells she exchanges with the the first mix the circuits she establishes. However, this alone does not tell her anything about the anonymous communications of other users.

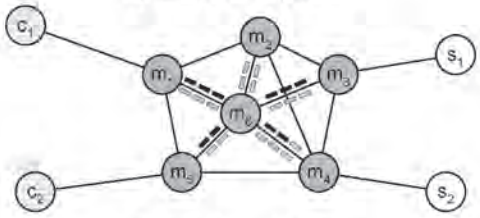
In the remainder of this section, we analyse two prominent attackers, the global passive external attacker and the partial active internal attacker. We examine the measures needed to defeat these two adversaries. Our methodology is as follows: we start with a basic circuit-based mix network (see Section 2.3) that does not make use of any cover traffic and show why this does not provide protection from certain traffic analysis attacks. Then we increase the resistance of the mix network step-by-step by employing different dummy traffic schemes, while demonstrating that even complex cover traffic schemes are not enough to protect from sophisticated traffic analysis attacks.

4.1.1 Global Passive External Attackers

The global passive external attackers (sometimes also called global eavesdropper or simply global passive attacker) can observe all data exchanged over every virtual link between two nodes and between mixes and the servers. As already mentioned in Section 2.1, observing the data on a virtual link means observing them somewhere on the physical route between two nodes. Similarly, monitoring the application data on the route between the last mix and the server means monitoring them somewhere on that physical route. Therefore, an adversary capable of observing all data entering and exiting all mixes is a global eavesdropper because it enables him to monitor all data exchanged between clients and mixes, all data exchanged between any two neighbouring mixes, and all data exchanged between mixes and servers. We briefly analyse the possibilities of such an attacker and the measures mix networks can employ to beat him. First, we look at attacks on a single mix and Figure 4.1 illustrates the basic scenario.



a) without dummy traffic



b) with dummy traffic

Figure 4.1: *Traffic analysis at a mix*

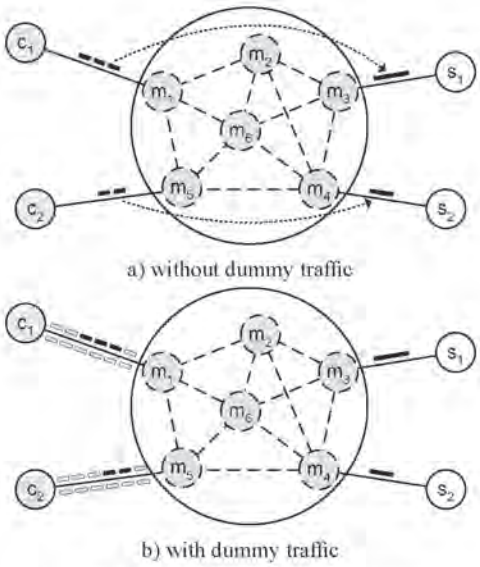
We use an example with six mixes (m_1–m_6). User u_1 uses client c_1 and is is connected to server s_1 via mixes m_1, m_6, and m_3. User u_2 uses client

c_2 and is connected to s_2 via m_5, m_6, and m_4. Although there should be many more users in a real mix network, the case with two users serves well to explain the basic attacks and defences. Our basic mix network makes use of fixed-length cells and layered encryption. The mixes may delay and reorder incoming cells or data they receive from the servers for a short time. However, as we are looking at near-real-time applications, cells must be sent out quickly (i.e. within a few tenths of a second after the corresponding data have been received) and if there is only little traffic, there may be no data from other users available to reorder anything. A first attack is trying to follow the data as they traverse a mix, as depicted in Figure 4.1(a). In our example, m_6 is used by both users, and let's assume c_1 is sending three cells to s_1, c_2 is sending two cells to s_2, and the cells happen to arrive at m_6 at nearly the same time. Although an attacker cannot correlate the cells entering and exiting m_6 based on their length or encoding, he can still easily deduce that the data from m_1 is forwarded to m_3 and the data from m_5 is forwarded to m_4 because of their different data volumes (three cells versus two cells). This corresponds to a combined application data volume and timing attack (see Section 2.3.3).

Using cover traffic that is indistinguishable from the real cells, this attack can be defeated. In Figure 4.1(b), m_6 employs constant flows of cells with all its neighbours in both directions. An observer at m_6 has no way to tell which of the cells entering and exiting the mix are real ones (the black ones) and which are just dummies (the gray ones). As a result, there is nothing to correlate as the application data volumes are hidden in the constant flows of cells.

Since the adversary can no longer trace cells as they traverse a mix, he can try to correlate the data at the endpoints. Knowing that each mix will delay the data for at most a fraction of a second, cells sent from a client to the first mix must result in data exiting the mix network towards the server at some other mix within at most a few seconds. Figure 4.2(a) illustrates the attack.

The attacker no longer looks at any traffic exchanged between mixes, but only sees three cells entering the mix network from c_1 and two from c_2. Within some seconds, he sees data exiting at m_3 that have a length corresponding to about three cells and data exiting at m_4 with a length of about two cells. Note that the fixed-length cells used on the virtual link between the clients and the mixes are not visible on the route between the last mix and the server, because the last mix just forwards the contents of the cells, i.e. the application data to the server. Nevertheless, this combined end-to-end application data volume and timing attack works well because the amount



a) without dummy traffic

b) with dummy traffic

Figure 4.2: *End-to-end Traffic analysis*

of data entering the first mix and exiting at the last mix are closely related. Figure 4.2(b) shows the countermeasure to this attack. The route between the last mix and the server is not part of the mix network, and there is nothing we can do there without requiring the servers to participate in the mix network protocol. But we can introduce cover traffic on the virtual links between the clients and mixes. Like between mixes, we use constant flows of cells on all virtual links between clients and mixes in both directions. This removes any correlation between the data entering the first mix and leaving the last mix.

However, there are still attacks possible. One is the long-term intersection attack (see Section 3.2), which also correlates events at the endpoints but over a long period of time. It makes use of the fact that every user has a certain behaviour when being online, e.g. most Internet users download similar web pages whenever they are hooked up to the Internet. As an example, assume u_1 sitting at c_1 regularly downloads data from s_1, and s_1 happens to be a web server that is visited only by a few Internet users. So even if the combined end-to-end application data volume and timing attack described above does

not work when observing just one Internet session of a user, it could work when correlating the patterns observed during 100 sessions, because c_1 contacts s_1 in 90% of her sessions. This attack is of course much more difficult to carry out when the server is visited by a huge number of people such as www.cnn.com. But who is interested in learning who visits www.cnn.com? An adversary can learn much more about an individual by knowing what "exotic" web servers she visits. But even this attack could be beaten, at least in theory: by making sure that users are always connected to the mix network and always exchange dummy traffic with their first mix. But this is an unrealistic assumption even if users want to be online all the time: computers and programs crash from time to time and Internet connections are not working all the time due to congestion or ISP failure. To be really resistant against the long-term intersection attack, mix networks would have to be brought to a temporary halt whenever the cell stream between any client and its first mix stalls to not leak any information to the global observer, as proposed in PipeNet (see Section 3.1.2), a mix network operated synchronously. Assuming a mix network with 100 mixes, it would be extremely difficult to distribute the information about a stalled virtual link between a client and a mix quickly enough to all other mixes. In addition, if 100 users were connected to each of the 100 mixes, the probability that all 10000 virtual links between clients and mixes are working at any time would be virtually zero in today's Internet. So even if the mix network could be brought to a full stop, it would be of little practical use because it would be halted most of the time. Additional problems of PipeNet with DoS attacks have already been discussed in Section 3.1.2.

There is a special case of mix networks that makes it easier to defend against the long-term intersection attack: the mix cascade operated in the way as proposed by the developers of the Web MIXes project (see Section 3.1.2). Their mix cascade is operated synchronously and in a mix cascade, every user uses the same set of mixes in the same order. For instance, we could use 25 mix cascades with four mixes each instead of a mix network with 100 mixes. This would also have the advantage that the different mix cascades could have different rates for the constant traffic flows on the virtual links between clients and the first mix in the cascade they use. Slow dial-up users could connect to a 32 Kb/s cascade and users with fast DSL connections could use one of the 512 Kb/s cascades. Bringing a synchronous mix cascade to a halt if any virtual link between a client and the first mix in its cascade is quite simple because all users of a cascade are connected to the same first mix

in the cascade and cells are only forwarded when all clients have sent one or a certain number of cells. Using the same example as above with 10000 users and 100 mixes, we can arrange the 100 mixes in 25 mix cascades of four mixes each. Consequently, each cascade must support 400 users. Note that this also means an anonymity set that is 25 times smaller than above because if a particular server is contacted by the last mix in a cascade, all users of the other cascades cannot have been the initiator of this communication relationship. But even with 400 clients per cascade, the probability that none of these 400 virtual links to the first mix has connection problems at any time is small. In addition, some problems remain: as with PipeNet-like mix networks, any user or mix can bring the cascade down by stopping sending cells.

But most of all, the huge amount of dummy traffic exchanged between clients and mixes in both directions consumes a lot of bandwidth at the mixes, as we will point out in Section 4.2. In general, the concept of dummy traffic to increase anonymity is still not really understood. One question is if there are "cheaper" ways than employing fixed streams of traffic between clients and mixes to defeat the combined end-to-end application data volume and timing attack. Probably not, because anything that is adapting to the user's behaviour in any way leaks some information [11].

4.1.2 Partial Active Internal Attackers

The partial active internal attacker controls a subset of the mixes, which opens a new spectrum of attacks. In particular, dummy traffic can no longer be generated on a per-virtual link basis between two nodes because unlike an external attacker observing a mix, the internal attacker knows which of the cells the mixes he controls send and receive are dummies. Figure 4.3 illustrates how this can be exploited by an adversary controlling two mixes, m_1 and m_3, which happen to be the first and last mix used by c_1.

In Figure 4.3(a), m_1 knows which cells on the virtual link to c_1 are real data, which means m_1 and m_3 can carry out a combined end-to-end application data volume and timing attack to break u_1's anonymity, just like in Figure 4.2(a). To resist this attack, dummies have to be sent from the client through the whole chain of mixes and back, as illustrated in Figure 4.3(b). As a result, m_1 is no longer able to distinguish between c_1's real data and its dummies and the application data volume attack does no longer work.

Unfortunately, even this cover traffic scheme is not enough to defeat the

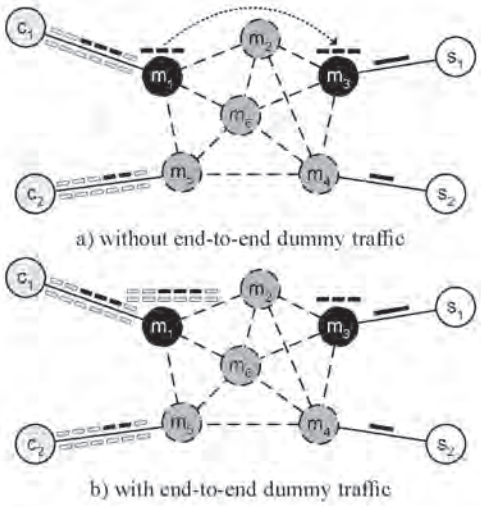


Figure 4.3: *End-to-end Traffic analysis by an internal attacker*

active internal attacker. If m_1 briefly blocks the constant cell stream from c_1 several times and checks with m_3 if it has noted a corresponding brief interruption of an incoming cell stream shortly afterwards, they can conclude with high probability that c_1 communicates with s_1. In fact, introducing such timing signatures in the cell streams is always possible for an adversary controlling some mixes, independent of the actual cover traffic scheme that is employed. Since dummy cells would only give the adversary more possibilities to introduce timing signatures, using dummies at all could even increase the adversary's chances to break the anonymity of a user. Indeed, synchronous designs could even cope with this attack, but practical issues refrain us from making use of them in low-latency systems.

4.1.3 Summary

Providing protection against very powerful attackers is extremely difficult. Cover traffic certainly helps to increase the protection from external observers that cannot continuously monitor the entire mix network or that are only capable of observing a subset of the system. However, a global observer using the

long-term intersection attack can most probably beat every system because there are always periods where clients cannot keep up a constant flow of traffic with the first mix in their chain. A global observer with additional capabilities of an active external attacker will be even more successful because he can block virtual links without having to wait for such failures to happen naturally. Stalling the whole system when any virtual link fails is simply not an option in a practical system for low-latency applications. Internal attackers could theoretically be beaten with end-to-end dummies, but since internal attackers controlling mixes can always be assumed to also have active capabilities, the adversary should again succeed by briefly blocking cell streams at the first mix and recognise this at the last mix (or vice versa) if he controls both the first and last mix in a chain. Introducing such timing signatures in the cell streams is always possible independent of the cover traffic scheme, which implies that dummy cells are in general of little value against internal active attackers. One also must remember that if there are internal attackers present in a system, perfect anonymity is not possible even in theory, because even in a perfectly balanced and synchronous system, it can always happen that all mixes along a chain are compromised.

4.2 A Quantitative Analysis of Mix Networks

In this section, we analyse how many data the mixes in a mix networks must handle to serve a certain number of web users. The reason for this is to learn more about the costs in terms of bandwidth of different cover traffic schemes compared to the case where no dummies are used at all. Note that we are only looking at the data handled by the mixes because from the point of view of the operator(s) of a mix overlay network, this directly determines the bandwidth costs they have to pay. We are not comparing the costs of anonymous communication with direct client/server interaction because besides the costs to operate the mixes, mix networks produce additional load on the whole Internet infrastructure due to the longer paths the data travel between the endpoints of a communication relationship. We use web browsing as the example application and for simplicity, we only take web requests and replies into account, leaving out any overheads resulting from underlying protocols or the mix network protocol itself.

We assume a mix network consists of m mixes m_i, $1 \leq i \leq m$ that are connected to the Internet with bidirectional bandwidths b_i b/s. We define the

capacity c of a mix network as the total number of bits all mixes together can send or receive in a second:

$$c = \sum_{i=1}^{m} m_i \cdot b_i \tag{4.1}$$

Note that it is reasonable not to distinguish between sending and receiving capacities because mixes always send and receive approximately the same amount of data: they get data from clients and forward the same amount of data to the next hop, they receive data from a mix and forward them to another mix, or they get data from a mix and forward them to a server. Consequently, if a mix has an asymmetric Internet connection, then the lower bandwidth determines the amount of data it can send and receive.

4.2.1 No Dummy Traffic

If no dummy traffic is used, then the whole capacity is devoted to transport real data. We analyse how many users a mix network with a given capacity can support. We assume that on average, each user sends d_s bits and receives d_r bits per day through l mixes. If the application is e-mail, data are only sent and $d_r = 0$. If the application is web browsing, d_r is about ten times a big as d_s. In any case, l mixes receive d_s bits in one direction, and l mixes receive d_r bits in the other direction during 24 hours (86400 seconds). Similarly, l mixes send d_s bits in one direction and l mixes send d_r bits on the way back. On average, each user is responsible that $l \cdot (d_s + d_r)$ bits must be sent and received by the mix network. This means that without dummy traffic, the load on the mix network is symmetric, i.e. the total amount of data sent or received by all mixes is the same. We denote $d = d_s + d_r$ as the total amount of data produced by each user during 24 hours. The minimum capacity a mix network must offer to support n users, each of them producing d bits during a day can therefore be computed as

$$c_{min} = \frac{n \cdot l \cdot d}{86400}\ [b/s]. \tag{4.2}$$

Transforming this equation, we get the maximum number of users a mix network can handle given its capacity:

$$n_{max} = \frac{86400 \cdot c}{d \cdot l} \tag{4.3}$$

As an illustrative example, we assume we want to support 100000 web users. To estimate the amount of data generated by the users, we use results from Internet traffic pattern studies. According to Nielsen//NetRatings[2] and Cyberatlas[3], the average web user had about 25 web sessions per month during September 2003. A web session is defined as a continuous series of user activity via URL requests. A session is considered ended if no requests for URLs have been made and if no corresponding applications (for instance a web browser) have been running for one minute[4]. The average duration of a web session is about 33 minutes and during a single session, about 50 web pages are completely downloaded. To determine the amount of data that is generated to download a single web page, we use appropriate values from scientific literature. We assume web requests to be 300 bytes with a probability of 0.8 and 1100 bytes with a probability of 0.2 [72]. This results in an average web request length of 460 bytes. The lengths of web replies follow a ParetoII distribution with parameters $k = 800$ and $\alpha = 1.2$ [44], which results in an average size of 12 KB. The number of embedded objects also follow a ParetoII distribution, this time with parameters $k = 2.4$ and $\alpha = 1.2$ [44], resulting in an average of four embedded objects per page. Consequently, requesting a web page results in sending 2300 ($= 5 \cdot 460$) bytes, and the size of a web page is 60 ($= 5 \cdot 12$) KB on average.

Summarising these data, the average user sends out 115 KB and receives 3 MB during each session. With 25 session during September 2003, the average data sent and received per day are 96 KB and 2.5 MB, respectively. For ease of the further analysis in this section, we slightly modify the usage pattern of the average user and assume every user has one session of 30 minutes per day. During a session, 115 KB are uploaded and 3 MB are downloaded. Note that these data are also similar to the outcome of WhiteCross/NARUS study[5] that the average web user generates about 2.5 MB of data per day. We further assume the average number of mixes in a circuit to three, which is a

[2] http://www.nielsen-netratings.com

[3] http://cyberatlas.internet.com/big_picture/traffic_patterns

[4] http://ereportsacn.netratings.com/help/Glossary%20of%20NetView%20Terms.pdf.

[5] http://www.whitecross.com/white-papers/wnfwp1102.htm

reasonable compromise between protection from attacks and end-to-end delay. Based on these assumptions and realising that every user generates 3115 KB per day, which is equivalent to 24920 Kb, the minimum capacity of the mix network according to (4.2) must be

$$c_{min} = \frac{100000 \cdot 3 \cdot 24920000}{86400} \approx 87\ Mb/s.$$

According to (4.1), this mix network could be built with 87 mixes capable of handling 1 Mb of data per second in both directions, or 9 mixes with a 10 Mb/s connection to the Internet. Note that there is no requirement for homogeneous mixes, i.e. the mixes can have different capabilities. However, the figures are based on the assumption that all traffic is equally distributed over time and that the circuits are chosen in a way that optimally distributes the traffic according to the capabilities of the mixes. In practice, this is never the case and the effective capacity needed to support 100000 users is probably several times bigger than the minimum capacities we computed according to (4.2). Nevertheless, the minimum capacity provides a good measure for the absolute minimum that is needed to support a certain number of users.

4.2.2 Dummy Traffic between Clients and Mixes

With dummy traffic, the amount of data certainly increases, but by how much? We have seen in Section 4.1.1 that dummy traffic employed only between mixes does not help much because of end-to-end traffic analysis. So let's look at the case with constant bidirectional cell flows on the virtual links between clients and mixes. The capacity c is no longer exclusively available for real data, but some of it must be devoted to handle the dummy data. We therefore divide the capacity into a part c_r to handle the real data and a part c_d to handle the dummy data.

Similar as in the case without dummy traffic, each user is responsible that $l \cdot d$ bits of real data must be sent and received by the mix network. But now we also have dummy traffic that is exchanged with the first mix. If t_{up} is the average uptime of a client during 24 hours and r_d is the rate at which data are exchanged between the users and their first mix, then the number of dummy bits received by the first mix is $r_d \cdot t_{up} - d_s$. The reason for subtracting d_s is that the real data is sent within the constant cell stream and does not account for the dummy data overhead. On the way back, the first mix sends

$r_d \cdot t_{up} - d_r$ bits of dummy data to the client in addition to the real data. So the dummy traffic sent and received by all mixes together is not symmetric. Since the load stemming from the real data is unchanged compared to Section 4.2.1, it follows that the total traffic sent and received by the whole mix network is also not symmetric. If $d_s < d_r$, then the whole mix network receives more data than it sends and vice versa. As a result, the minimum capacity and the maximum number of users given the capacity are defined as:

$$c_{min} = \frac{n}{86400}\left(l \cdot d + r_d \cdot t_{up} - \min(d_s, d_r)\right) \; [b/s] \tag{4.4}$$

$$n_{max} = \frac{86400 \cdot c}{l \cdot d + r_d \cdot t_{up} - \min(d_s, d_r)} \tag{4.5}$$

We can easily decompose (4.5) into the minimum capacity for the real and the dummy data:

$$c_{min,r} = \frac{n}{86400} l \cdot \bar{d} = \frac{n \cdot l \cdot d}{86400} \; [b/s] \tag{4.6}$$

$$c_{min,d} = \frac{n}{86400}\left(r_d \cdot t_{up} - \min(d_s, d_r)\right) \; [b/s] \tag{4.7}$$

Unsurprisingly, the part for the real data equals the minimum capacity in (4.2) where no dummy traffic is used. We use the same example with web users as above, assuming every user is online during 30 minutes (1800 seconds) a day and exchanges data with the first mix at 64 Kb/s. According to (4.5), the minimum capacity of the mix network is

$$\begin{aligned} c_{min,30m} &= \frac{100000}{86400}\left(3 \cdot 24920000 + 64000 \cdot 1800 - 920000\right) \\ &\approx 219 \; Mb/s. \end{aligned}$$

The capacity for the real data remains the same as in the example without any dummy traffic, i.e. $c_{min,r} \approx 87 \; Mb/s$.

As discussed in Section 4.1.1, users must be online all the time to beat long-term intersection attacks. So cover traffic should not only be generated during one but 24 hours a day. In this case, the minimum capacity increases to

$$c_{min,24h} = \frac{100000}{86400}(3 \cdot 24920000 + 64000 \cdot 86400 - 920000) \approx 6.49\,Gb/s.$$

According to (4.1), this could be provided by 649 mixes with a 10 Mb/s connection or 66 mixes with a 100 Mb/s connection each. Note that since we did not take dummies between mixes into account in (4.5), the actually needed capacity would be even higher.

4.2.3 End-to-End Dummy Traffic

Going even further, we can defeat internal attacks where the adversary controls the first and last mix of a chain by introducing end-to-end dummies (see Section 4.1.2). In this case, the constant cell streams go all the way through the whole chain of mixes and back. Each user is responsible that $r_d \cdot t_{up}$ bits are sent to l mixes in the forward direction and to $l-1$ mixes on the way back. In addition, d_r bits are sent from the contacted server (e.g. the web server) to the last mix in the chain. Similarly, $r_d \cdot u$ are sent by $2l-1$ mixes and d_s is sent by the last mix to the contacted server. Here again, the total amount of data sent and received by all mixes together is not symmetric. If $d_s < d_r$, then the whole mix network receives more data than it sends and vice versa. The minimum capacity and the maximum number of users given the capacity are defined as:

$$c_{min} = \frac{n}{86400}\left((2l-1) \cdot r_d \cdot t_{up} + \max(d_s, d_r)\right)\ [b/s] \quad (4.8)$$

$$n_{max} = \frac{86400 \cdot c}{(2l-1) \cdot r_d \cdot t_{up} + \max(d_s, d_r)} \quad (4.9)$$

Taking into account that the minimum capacity for the real data is the same as in (4.7), we can compute these capacities as follows:

$$c_{min,r} = \frac{n \cdot l \cdot d}{86400}\ [b/s] \quad (4.10)$$

$$c_{min,d} = \frac{((2l-1) \cdot r_d \cdot t_{up} + \max(d_s, d_r) - l \cdot d)}{86400}\ [b/s] \quad (4.11)$$

Using again the same example with web users as above, the minimum capacities assuming every user is online for 30 minutes or 24 hours a day are

$$c_{min,30m} = \frac{100000}{86400}(5 \cdot 64000 \cdot 1800 + 24000000) \approx 695\ Mb/s$$
$$c_{min,24h} = \frac{100000}{86400}(5 \cdot 64000 \cdot 86400 + 24000000) \approx 32.03\ Gb/s.$$

If every user is online all time, we need a mix network consisting of at least 321 mixes with a 100 Mb/s connection each.

4.2.4 Mix Cascades

For completeness, we also briefly analyse mix cascades. We assume that the mixes build k fixed cascades of length l, and each cascade handles n/k of all users. If no dummy traffic is used, then each user is responsible for sending d_s bits to the first mix in a cascade and d_r bits on the way back. The minimum capacity of the first mix in a cascade to support n/k users is

$$c_{1^{st}\ mix,min} = \frac{\frac{n}{k} \cdot d}{86400}\ [b/s].$$

Every mix in the cascade handles the same amount of data, so the capacity of all cascades together to support n users and the maximum number of user that can be handled are defined by

$$c_{min} = \frac{n \cdot l \cdot d}{86400}\ [b/s] \tag{4.12}$$
$$n_{max} = \frac{86400 \cdot c}{d \cdot l}. \tag{4.13}$$

Comparing these equations with (4.2) and (4.3), we can see that without dummy traffic, mix networks and mix cascades are equally efficient.

Introducing cover traffic on the virtual links between the clients and the first mix in a cascade, the first mix receives d bits of real data and $r_d \cdot t_{up} - d_s$ bits of dummy data. Similarly, it sends d bits of real data and $r_d \cdot t_{up} - d_r$

dummy bits. The needed capacity of a single first mix and the number of users it can serve given the capacity are

$$c_{1^{st}\ mix,min} = \frac{\frac{n}{k}}{86400}(d + r_d \cdot t_{up} - \min(d_s, d_r))\ [b/s] \quad (4.14)$$

$$n_{1^{st}\ mix,max} = \frac{86400 \cdot c}{d + r_d \cdot t_{up} - \min(d_s, d_r)}. \quad (4.15)$$

For the other mixes in every chain, the capacity and the users given the capacity can be computed as follows:

$$c_{2^{nd}\_l^{th}\ mix,min} = \frac{\frac{n}{k} \cdot d}{86400}\ [b/s] \quad (4.16)$$

$$n_{2^{nd}\_l^{th}\ mix,max} = \frac{86400 \cdot c}{d} \quad (4.17)$$

So the total minimum capacity of all mixes is $k \cdot (c_{1^{st}\ mix,min} + (l-1) \cdot c_{2^{nd}\_l^{th}\ mix,min})$, which is exactly the same as the capacity in the case of a mix network with dummy traffic on the virtual links between clients and mixes (4.5). The difference to mix networks is that the capacity of the first mix in each chain must be much bigger than the capacity of the others. Using our web browsing example and assuming that the $2^{nd} \ldots l^{th}$ mixes all have a capacity of 1 Mb/s, each of them can handle

$$n_{2^{nd}\_l^{th}\ mix,max} = \frac{86400 \cdot 1000000}{24920000} \approx 3467$$

users according to (4.17). This also means that we need $k = 100000/3467 \approx 29$ chains to serve 100000 users. Each user is online during 30 minutes a day, which means that using (4.15), the capacity of the first mix in every chain must be

$$\begin{aligned} c_{1^{st}\ mix,min} &= \frac{3467}{86400}(24920000 + 64000 \cdot 1800 - 920000) \\ &\approx 5.59\ Mb/s. \end{aligned}$$

So a system consisting of 29 mix cascades where 29 mixes have a capacity of 5.59 Mb/s and 58 mixes have a capacity of 1 Mb/s is one possible minimum configuration to support 100000 users.

With end-to-end dummy traffic, the capacities of the first $l-1$ mixes in each chain is the same, as is the number of users they can handle if the capacity is given:

$$c_{1^{st}...(l-1)^{th}\ mix,min} = \frac{\frac{n}{k}}{86400} \cdot 2 \cdot r_d \cdot t_{up}\ [b/s] \quad (4.18)$$

$$n_{1^{st}...(l-1)^{th}\ mix,max} = \frac{86400 \cdot c}{2 \cdot r_d \cdot t_{up}} \quad (4.19)$$

The capacity of the last mix and the number of users it can handle at most are as follows:

$$c_{l^{th}\ mix,min} = \frac{\frac{n}{k}}{86400} \left(r_d \cdot t_{up} + max(d_s, d_r)\right)\ [b/s] \quad (4.20)$$

$$n_{l^{th}\ mix,max} = \frac{86400 \cdot c}{r_d \cdot t_{up} + max(d_s, d_r)} \quad (4.21)$$

Adding up the capacities of all mixes in all cascades, we can see again that mix cascades have the same minimal capacity requirements as mix networks when end-to-end dummies are used.

4.2.5 Summary

Table 4.1 summarises the various cases we discussed above and also gives example configurations and the dummy traffic overhead for a system to support 100000 users.

To summarise, dummy traffic significantly increases the minimum capacity of mix networks. While accepting being vulnerable to the long-term intersection attack introduces a data overhead of a "only" a few times the real data, the measures to resist this attack are extremely costly in terms of data overhead. The minimum capacities of mix cascades are the same as those of mix networks but the capacities of either the first mix (if dummy traffic is used only on the virtual links between clients and mixes) or the last mix (if end-to-end dummies are employed) differ from the others. Especially in the

Table 4.1: *Minimum capacities to support 100000 users (web browsing, 5 MB per day and user).*

dummy data rate (b/s)	online time per user (hours/day)	capacity needed (Mb/s)	example configurations	dummy traffic overhead
		87	**mix network:** 87 mixes with 1 Mb/s **29 mix cascades:** 87 mixes with 1 Mb/s	
64000 (between client and first mix)	0.5	219	**mix network:** 22 mixes with 10 Mb/s **29 mix cascades:** 29 mixes with 5.59 Mb/s 58 mixes with 1 Mb/s	152%
64000 (end-to-end)	0.5	695	**mix network:** 70 mixes with 10 Mb/s **27 mix cascades:** 54 mixes with 10 Mb/s 27 mixes with 6.05 Mb/s	699%
64000 (between client and first mix)	24	6486	**mix network:** 649 mixes with 10 Mb/s **29 mix cascades:** 29 mixes with 222.9 Mb/s 58 mixes with 1 Mb/s	7355%
64000 (end-to-end)	24	32028	**mix network** 321 mixes with 100 Mb/s **128 mix cascades:** 256 mixes with 100 Mb/s 128 mixes with 50.27 Mb/s	36714%

case of dummy traffic only between the clients and the mixes, the first mix in each chain must handle many more data than the others.

Recalling that none of these cover traffic schemes provides full protection from powerful adversaries and considering the very significant bandwidth data, any use of cover traffic is questionable. In particular, given any mix network with a certain capacity, and considering that the bandwidth that is spent on handling dummies is not available to handle real data, an important question to consider is whether it is better to have a certain level of anonymity

among 10000 users without dummy traffic or to have "a bit more" anonymity among 1000 users when using dummies.

4.3 A Realistic Threat Model

We have seen in Section 4.2 that employing dummy traffic to protect from powerful adversaries is extremely expensive in terms of data overhead. We have also discussed in Section 4.1 that achieving perfect protection against a global observer or a partial internal attacker in a practical system is simply not possible. But how realistic are these powerful attackers? The community has been arguing for years about what a realistic threat model could be like. In this section, we give arguments for what we call a realistic threat model.

4.3.1 The Passive External Attacker

We start with the global passive attacker. The long-term intersection attack is the most powerful attack that can be executed by such an adversary. To resist this attack, users must be connected to the mix network all the time and all users must continually exchange constant, bidirectional cell streams with the first mix in their circuits. To not leak any information, the mix network must be operated completely synchronously, but we have already shown in Section 4.1.1 that this is impossible to achieve if the mix network is reasonably large and if an acceptable quality of service should be offered to the users.

Protection against a global passive attacker is only realistic using synchronous mix cascades with a small number (e.g. 100) of users per cascade. The synchronous operation of these mix cascades and the assumption that the probability that no virtual link between the clients and the first mix in a cascade is stalled is reasonably close to one imply that such a system would be usable in practice. Such a small number of users could also make it possible that the huge dummy traffic overhead (about 74 times the amount of real data traffic according to Table 4.1) can be absorbed by a powerful first mix in the cascade. But what does it help to be anonymous within such a small anonymity set?

Following this discussion, we state that perfect anonymity for very many users within large anonymity sets is impossible in the Internet. In fact, we go even further and say that if there is a global eavesdropper, then practical anonymous low-latency communication is not possible, at least not until

the implications of cover traffic schemes are better understood and efficient mechanisms that significantly increase protection from traffic analysis attacks will be developed.

But how realistic is such a global passive attacker? As discussed in Section 4.1.1, this attacker would have to be able to read every cell exchanged between two nodes and all application data between mixes and servers. In addition, he would need the capability to store this information together with precise timing information to correlate them at different places in the mix network based on their timing. If a cell is observed, he must store at least the IP addresses and ports of the two nodes between which it was sent. In the case of application data that are observed between a mix and a server, he must additionally store the length of the data. ISPs are a potential threat because their task of transporting data through the Internet implies they have also access to the data handled by a mix network. If a cell is sent from one node to another or application data are exchanged between a mix and a server, the corresponding IP packets usually travel across different ISPs. In general, this means they are sent from one access ISP via zero or more backbone ISPs to another access ISP, but it can also mean the involvement of only a single access ISP if both nodes or the mix and the server happen to be served by that single access ISP. Assuming a large mix network with mixes that are spread across the world, every ISP only gets a small part of the full picture. Some ISPs are larger than others and can therefore monitor a more significant portion of the Internet than others, but in general, the capabilities of a single ISP are limited. To act as a global passive attacker, several backbone ISPs must collect data and bring together the relevant information. Even when leaving out technical issues to collect and store the data, the threat from such a collusion of ISPs is minimal because of the large number of ISPs. As an example, there are about 40 backbone ISPs in the USA according to ISP Planet[6]. To get all data sent and received by mixes in the USA, nearly all of these 40 independent organisations would have to collaborate, which is an unlikely threat. Similar arguments can be made for other major Internet regions in the world such as Europe or Japan. We therefore conclude that if the number of mixes is sufficiently large and they are spread across several countries and use a variety of different ISPs, then the global observer is a very unlikely threat.

Another potential threat are federal agencies that are interested in getting the full picture about what is going on in a mix network. Using FBI's Carni-

[6] http://www.isp-planet.com/resources/backbones/index.html

vore diagnostic tool, this is possible by installing Carnivore at all backbone ISPs. Carnivore can theoretically capture all data flowing through an ISP, but by specifying filter rules, Carnivore only delivers those data that match these rules (e.g. only packets that contain a specific IP address). At least officially, Carnivore can only be used for a limited time after a court order has been issued, and even then only to read data "authorised for capture" by the court order, which directly affects the filter settings. In addition, a court order is only issued to gather hard evidence and not intelligence. Since a court order is needed for every single temporary installation of Carnivore at an ISP, getting continual access to all backbone ISPs using the legal way is not likely to be possible for federal agencies.

Another option for federal agencies is to circumvent the legal way and to convince the backbone ISPs to provide them with all data. This might even work with a few of them, but making deals to collaborate with every single backbone ISP is extremely unlikely to be successful – in particular without anyone leaking information about this criminal act.

To summarise, if a mix network contains only 10 mixes that are in a geographically small area such as a single country, then the global attacker may be a threat because only a few ISPs must combine their data to get the full picture. But with 100s or 1000s of mixes that are distributed over the whole planet, it is very unlikely an attacker can observe more than a small subset of them for the reasons given above. Note that we cannot provide a precise number for the maximum percentage of all traffic in a mix network an adversary may be able to observe in a large distributed mix network but following this discussion, anything significantly larger than 10% of all traffic is unlikely. Still, one must bear in mind that a partial observer capable of observing 10% of all traffic may be enough to break the anonymity of users from time to time. In particular, if no cover traffic is used, it is likely that an adversary eavesdropping on the virtual link between client and first mix and on the route between the last mix and the server can break that circuit using the combined end-to-end application data volume and timing attack, as illustrated in Figure 4.2. The more data that are sent along a virtual circuit, the higher the probability the attacker can indeed link the two communication endpoints but since this is difficult to quantify, we simply assume the adversary can always link the endpoints if he observes any data on both the virtual link between client and mix and on the route between last mix and server. The probability of success of this attack depends on the fraction of all Internet traffic the adversary can observe. Assuming the adversary observes a fraction of f_o of all Internet

traffic and assuming the traffic exchanged between nodes in the mix network and between mixes and servers is similarly distributed across the Internet as all traffic, the probability p_b the adversary can observe both endpoints in a random circuit and thereby break the relationship anonymity is given by:

$$p_b = f_c^2 \tag{4.22}$$

Consequently, if an adversary manages to observe 10% of the entire Internet traffic, he can expect to break 1% of all circuits and therefore also 1% of all anonymous communications, because 10% corresponds to a fraction 0.1 and therefore $p_b = 0.1^2 = 0.01$ according to (4.22).

4.3.2 The Active Internal Attacker

The active internal attacker controls a subset of the mixes, probably by running them himself. If he controls the first and last mixes in a circuit, he can observe both the virtual link to the client and the data on the route to the server, which implies he has broken the relationship anonymity according to our discussion above. Assuming the attacker controls n_c of n mixes, the probability p_b the adversary can observe a random circuit is given by [122]:

$$p_b = \left(\frac{n_c}{n}\right)^2 \tag{4.23}$$

We assume the government or any other powerful institution is the adversary and interested in breaking the anonymous communications. While this institution would probably not run mixes under its own name, it could provide private persons with the necessary equipment to operate mixes at their homes and pay th em 1000 US$ a year in addition. Assuming the infrastructure (a decent Internet connection and a personal computer) costs 4000 US$ a year per mix, there are yearly costs of 5000 US$ per mix. As an example, we take quite a big mix network consisting of 100 mixes that are operated by volunteers such as companies, universities, and private persons. If the adversary manages to convince 300 people to run a mix, he controls 75% of all mixes and the yearly costs are 1.5 million US$. According to (4.23), this would mean the adversary can break any random circuit with probability $p_b = (0.75)^2 \approx 0.56$. Note that as we have seen earlier in this chapter that

no dummy traffic scheme helps against an internal attacker if he makes use of his active capabilities to introduce timing signatures in the cell streams at one endpoint to recognise this later in the chain.

If a mix network is operated by volunteers, then this attack is a very real threat. Operating any significant fraction of all nodes is certainly much simpler than observing the same percentage passively. One possible defence against the adversary controlling a significant subset is to make sure that only "honest" people and institutions are allowed to operate a mix. But how could one guarantee this in practice? With 10 mixes, this is possible, maybe even with 100. But it gets more and more difficult as the number of mixes increases.

We conclude that it is quite possible for an adversary to operate a significant portion (e.g. 50%) of all mixes in a mix network operated by volunteers as described as above. The larger the number of honest mixes, the more difficult and expensive the attack gets.

4.3.3 Summary

Based on the assumption of a distributed mix network of reasonable size that aims at supporting at least several 1000 users, we conclude that there is no known method that can be applied in practice to really provide protection against a global observer. However, we have also given reasonable arguments that the likelihood of such an attacker depends on the size of the mix network. The global observer may be a threat in mix networks with no more than a few mixes. But if the number of mixes grows and the mixes are spread over the world, it is very unlikely any adversary can observe more than a small subset of all mixes. It is of course difficult to prove that global eavesdroppers are no threat to a large mix network if its mixes are spread over the whole planet, but we have given strong arguments to support this. We can only repeat that if there is a global observer, practical anonymity for low-latency applications is not possible, at least not with the current knowledge we have about cover traffic schemes. Internal attackers are always a threat if there is no strict access control about who is allowed to run a mix. Even assuming a mix network operated by volunteers that consists of 100 mixes that are spread across the world, the threat from an internal observer is significant, and we have given arguments that it is likely to be bigger than the threat from an external observer. The only way to defend against an adversary controlling a significant subset of all mixes is either by allowing only "trustworthy" institutions or per-

sons to run a mix, which makes it difficult to acquire a large number of mixes at all, or by making the attack more expensive by increasing the number of honest mixes.

It is important to remember that there is no practical defence against an internal attacker operating some mixes because even end-to-end cover traffic would not prevent any two colluding mixes from learning whether they are part of the same circuit. If the adversary controlled both endpoints of a circuit, he would succeed in breaking it, in particular if many data are exchanged between client and server. Similarly, when leaving out any cover traffic mechanism, a partial external attacker observing the data on the virtual link between client and first mix and the route between last mix and server is frequently able to break the relationship anonymity between the endpoints. In this case, however, dummy cells exchanged on the virtual link between clients and mixes make it more difficult for the attacker to correlate the endpoints of a communication, but again at the cost of supporting fewer simultaneous users.

4.4 Comparison of Mix Network Approaches

In Section 4.2, we have seen that very many mixes are needed to support 100000 users that want to browse the Web anonymously. In this section, we analyse how well different mix network approaches are suited to provide practical anonymity for such a large user base. We distinguish between three basic approaches: commercially operated static mix networks, static mix networks composed of volunteers that operate a mix, and dynamic, peer-to-peer-based mix networks where every client is also a mix at the same time. With static mix network, we mean an infrastructure where the set of mixes is highly stable over time. We focus especially on how well the different approaches are suited to acquire enough mixes to support many users and how well they are suited with respect to the realistic threat model we stated in Section 4.3.

4.4.1 Static Mix Networks as Commercial Services

The only really big mix network for near-real-time applications that has been operational so far was Zero-Knowledge Systems' commercial Freedom Network (see Section 3.1.2). Since the Freedom Network was shut down due to economical reasons, we briefly look at the major cost factors when offering such a service:

- **Bandwidth costs** associated with the data handled by the mixes. These costs are directly dependent on the data volume and therefore on the number of users.
- **Hardware costs** to provide the platforms to operate the mixes. Like bandwidth costs, these costs depend about proportionally on the user base.
- **Software costs**, especially to develop and maintain the mix network software itself. These costs are nearly independent of the number of users and can be expected to be relatively high in the beginning until the software has reached a certain stability. Once the software has entered its maintenance phase, these costs usually get smaller.
- **Network operations costs**, which includes human efforts to guarantee smooth operation of the system. These costs depend on the size of the mix network and the number of users.

According to Adam Shostack [117], Zero-Knowledge Systems' former director of technology, bandwidth, software, and networks operations costs were the dominating costs factors during the time the Freedom Network was operational, while he expected software costs to get smaller over time because the Freedom Network was basically always work in progress during the period it was operational. In general, Shostack argues that while software costs have to be considered as an investment to improve the product which eventually results in profits, bandwidth and network operations costs are fundamental prerequisites to guarantee the operation of such a system, which cannot easily be reduced. Since bandwidth costs are therefore indeed one dominating cost factor when operating commercially a mix network, we carry out the same analysis as in Section 4.2.1 using the figures we know about the Freedom Network (see Section 3.1.2) and assume again that every user generates 3115 KB of data per day. Recalling that no dummy traffic was employed, that two mixes were used per default in a circuit, and assuming the mixes were connected to the Internet with double T1 speed on average, the Freedom Network was theoretically able to support about 800000 users according to (4.3). Taking overhead and peak times into account, however, the Freedom Network was more likely to support 100000 users with reasonable service, which is about 660 per mix on average. Recalling that the Freedom Network had only about 15000 subscribers, we conclude that it simply did not manage to attract enough users, which is confirmed by Shostack.

Following our discussion in Section 4.3, the Freedom Network was certainly large and distributed enough to make the global observer extremely

unlikely and the probability that a partial eavesdropper can observe data on both the virtual circuits between client and first mix and the route between last mix and server in a random circuit is small. In addition, operating several mixes was difficult for a possible internal attacker because Zero-Knowledge Systems made contracts only with ISPs and it was not possible for volunteers to simply run a mix. But a problem with commercial mix networks is that to sell anonymity, it may not be enough to say "anonymity in 99% of all cases for 50 US$ a year". To offer better anonymity, cover traffic must be used on the virtual links between clients and mixes (see Section 4.1.2). Using constant streams of cells with a rate of 64 Kb/s between clients and mixes during the 30 minutes an average user was online, the theoretical maximum number of users would have dropped to about 330000 according to (4.5). Considering overhead and peak times, something like 40000 users (about 266 per mix) is more realistic. So introducing cover traffic between clients and mixes increases the bandwidth cost per user by about 150%. Note also that exchanging the data between clients and mixes at 64 Kb/s means a significant performance penalty for all users that have faster Internet connections, which has been analysed in the context of the Anonymity Network (see Section 3.1.2). But increasing this fixed data rate implies significantly higher bandwidth costs per user. For example, if we increase the data rate to 256 Kb/s, the number of users according to (4.5) drops to 85000 in theory and about 10000 in practice (about 66 per mix). Going even further, and assuming all users had been online all the time and end-to-end dummies with a rate of 64 Kb/s had been used to defeat long-term intersection attacks, the Freedom Network could have supported 2412 user in theory according to (4.9), which would have been about 16 per mix.

We state that it is certainly possible to commercially operate a mix network for a large number of users. The question is whether this can be done profitably, and Zero-Knowledge Systems did not manage to attract enough users to do so. One problem could be that users are not willing to pay for anonymity as long as the system is vulnerable to the external observer. However, as we have seen above, employing any cover traffic mechanism to increase the protection from attacks multiplies the bandwidth that is consumed by a user. Recalling that bandwidth costs were one of the dominating cost factors of the Freedom Network, the cost per user would have significantly increased. We therefore conclude that running profitably a commercial mix network with or without dummy traffic is very difficult today. However, it may well be the case to operate such a service profitably in the future, espe-

cially if operational costs drop significantly and if users become more aware of the lack of privacy in the Internet and recognise the value of anonymous Internet access.

4.4.2 Static Mix Networks Operated by Volunteers

A trustworthy, static mix network capable of supporting a large number of users must consist of very many mixes operated by independent institutions. Leaving out any dummy traffic, Table 4.1 tells that at least 87 mixes with a 1Mb/s connection each are needed at minimum to support 100000 users in theory, which grows to several 100 mixes in practice But how difficult is it to convince so many institutions to operate a mix?

There is no easy answer for this question. Not many mix networks consisting of really independent nodes have been around. The Cypherpunk and Mixmaster remailers (see Section 3.1.1) have been operational for years and each of them consists of about 40 remailers in total, where several of them support both protocols. Looking at mix networks for low-latency applications, no free mix network has grown beyond a limited user trial with more than five mixes, and the mixes were not really operated by independent institutions.

What does it cost to run a mix? First, one must dedicate a reasonably powerful computer and accept that significant amounts of traffic are entering and exiting one's network. The first one is not the main problem because a powerful enough computer can be bought for about 1000 US$. Bandwidth is a problem, though. In Switzerland, one can get a bidirectional 512 Kb/s DSL-connection for about 130 US$ a month to your home as of November 2003. Not many people are willing to spend this amount of money voluntarily just to run a mix, but again, the development of bandwidth costs is difficult to predict. That leaves universities and large companies, which both have the possibilities to easily spare a computer and "a few Mb/s" of their bandwidth. So are they willing to help building a large mix network?

We do not believe the main problem to achieve a critical mass lies in the potential availability of the resources but in the political field. The governments of several countries do not like the idea of anonymity in the Internet. For instance, academic institutions could be threatened to receive less research funding from the government if they operated a mix. Likewise, companies could risk news articles where they are accused of supporting terrorists and drug dealers and as a consequence, could lose customers. Recalling

the *Church of Scientology vs. anon.penet.fi* case we have discussed in Section 3.3.1 shows that threats on operators of anonymity service have already happened and are not only theoretical. For such reasons, we believe it will be very difficult to run by volunteers a static mix network that provides a similar capacity as the Freedom Network. Of course one can argue that bandwidth will get cheaper, and in 10 years we may have 10 Mb/s connections into our homes for 10 US$ a month. But with more bandwidth available, people send more and larger objects across the Internet. Yesterday, people were downloading small web objects, today, they are exchanging mp3-files, and in 10 years, they may be regularly sending whole movies around.

The fact that collecting a large number of mixes that are operated by volunteers is very difficult means that the active internal attacker controlling a large number of mixes becomes a very real threat (see Section 4.3.2), and any cover traffic scheme is of little value against this attacker (see Section 4.1.2). Since collecting a large number of honest mixes is difficult, the only other possible defence against this attack is to be very restrictive about who is allowed to operate a mix. But this will make it even more difficult to collect enough mixes to support a large user base.

4.4.3 Dynamic, Peer-to-Peer-based Mix Networks

The third option are dynamic mix networks that operate in a peer-to-peer fashion. The main idea is that there is no distinction between mixes and clients that access the mixes. Rather, every client that uses other mixes to access the Internet anonymously offers itself the service of a mix to other users. Instead of paying for a commercial service or hoping that there are enough volunteers to provide enough capacity for a static mix network, each user pays for the anonymity by dedicating some of her bandwidth and computing power to others, very much like in peer-to-peer networks for file-sharing. Since every user brings her own mix, the capacity of such a mix network grows with the number of users and as a result, it should be able to support a very large user base. In addition, as it can be expected that users may join and leave such a system at any time, the set of mixes in the system fluctuates over time and is no longer stable. Consequently, we also name this type of systems dynamic mix networks, in contrast to static mix networks.

One problem of static mix networks is that there are political and legal barriers that may hinder an institution willing to operate a mix from doing so. In a dynamic mix network, the barrier to join is quite low as in all peer-to-

peer system. Participating in the system knowing that 100000 other users are already doing so is a much smaller step than operating one of a small number of static mixes.

Dynamic, peer-to-peer-based mix networks provide good protection from the realistic threats we identified in Section 4.3. With a huge number of mixes distributed all over the world, the probability that any adversary is able to observe a significant portion of the mixes is small. In addition, dynamic mix networks protect much better from internal attacks than static mix networks, because one possibility to reduce the probability an adversary controls a significant portion of all mixes in a mix network is to make sure there are very many honest mixes. With every user bringing her own mix, this is the best we can do.

However, dynamic, peer-to-peer-based mix networks are also the least explored approach and there are also some potential drawbacks. Since mixes may leave at any time, circuits are more likely to break than in static mix networks, which makes dynamic mix networks less well suited for long-standing connections such as remote logins. Note that with Crowds (see Section 3.3.2 and Tarzan (see Section 3.1.2), two peer-to-peer-based anonymity providing systems have been proposed, but neither of them copes well with significant membership fluctuations. Another potential problem is exit abuse. It remains to be seen if users really want to send web request for others. Static mix networks may be in a better position here because a participating mix processes lots of traffic and its operator can plausibly argue he didn't send it himself.

4.4.4 Summary

Of the three approaches we identified, static mix networks operated by volunteers seem not to be the right choice when aiming at a low-latency anonymity service for a large number of users. They suffer from the problem of acquiring enough mixes and from the very real threat of an internal attacker controlling a significant portion of all mixes. Static mix networks operated commercially are more likely to prevent this attack because volunteers cannot easily run mixes themselves. However, commercial mix networks have yet to show whether they can be operated profitably. In particular, it is unclear if and how much potential users are willing to pay for good but not perfect anonymity. Dynamic, peer-to-peer-based mix networks seem to be the best option to handle a large number of users because they do not suffer from capacity problems and provide good resistance against our realistic

threat model. Still, dynamic mix networks are not well explored at this time and they have yet to demonstrate their usefulness in practice. In particular, guaranteeing good end-to-end performance if mixes may leave at any time and with some mixes offering poor performance (e.g. dial-up users) is more difficult than in static mix networks. Nevertheless, we identify dynamic, peer-to-peer-based mix networks as the potentially most promising way to provide practical anonymity for a very large number of users. Consequently, we will explore this approach in the remainder of this thesis to demonstrate that dynamic mix networks are feasible and can be operated in a way such that they provide good anonymity, acceptable end-to-end performance, and scale up to a very large number of users.

4.5 Conclusions

In this chapter, we have performed a detailed analysis of mix networks. Focusing on mix networks to support low-latency applications, we have seen designing and operating a practical system that protects from powerful adversaries is very difficult. Even employing cover traffic is of little value against certain adversaries, in particular against the internal attacker that controls a subset of all nodes.

In general, the concept of dummy traffic to increase anonymity is still not really understood, and especially on the virtual links between the clients and the mixes, there may be more efficient ways than employing fixed streams of cells to provide good protection from an external observer that cannot continuously monitor the entire system. However, based on the current knowledge, any use of cover traffic is questionable, in particular as its significant data overhead reduced the number of users that can be supported with any given system.

We have defined a realistic threat model. Assuming a mix network that consists of many mixes that are spread over the world, the passive external attacker that is able to observe a significant portion of the system is a small threat. However, it must be remembered that even a partial external attacker may succeed in breaking the anonymity of a user from time to time if he manages to observe the data on the virtual link between the client and the first mix and on the route between the last mix and the server. Another realistic threat in mix networks with no strict access control about who is allowed to run a mix is the internal attackers that controls a subset of all mixes. Even

in a mix network operated by volunteers that consists of 100 mixes that are spread across the world, the threat from an internal observer is significant.

Finally, we have compared different mix network approaches regarding their suitability to provide practical anonymity for a large number of users with respect to our realistic threat model. We concluded that dynamic, peer-to-peer-based mix network have advantages over static mix networks, in particular because their capacity increases as more and more users join the system. Furthermore, dynamic mix networks can potentially attract very many users, which implies it gets more difficult for an adversary to control a significant subset of all mixes. Nevertheless, dynamic mix networks have yet to show their usefulness in practice.

Chapter 5

MorphMix

In this chapter, we introduce MorphMix, a peer-to-peer-based dynamic mix network for low-latency applications. In Section 5.1, we first state our motivation for developing MorphMix and the goals we want to achieve. Then we describe the basic functionality of MorphMix by looking at its properties and illustrating how a client application can communicate anonymously with a server application in Section 5.2. Based on this functionality, we analyse the requirements an adversary must fulfil to break the relationship anonymity in Section 5.3 and elaborate on the threat model in Section 5.4. In Sections 5.5–5.7, we introduce and explain the three core components of MorphMix, which include:

1. The **anonymous tunnel setup** protocol to establish circuits to access servers anonymously (see Section 5.5)
2. The **collusion detection mechanism** to detect circuits that contain several nodes operated by an adversary with high probability (see Section 5.6)
3. The **peer discovery mechanism** to make sure that nodes can pick other nodes from a wide variety of all available nodes (see Section 5.7)

Afterwards, we analyse scalability issues and the requirements to run a node in Section 5.8 and take a look at what changes for MorphMix if IP version 6 gets widely deployed in Section 5.9.

This chapter is the first of four chapters on MorphMix. After having introduced MorphMix in this chapter, we examine different attack strategies that can be employed by an adversary in Chapter 6. In Chapter 7, we analyse

the performance of the collusion detection mechanism to assess the protection MorphMix offers assuming realistic scenarios. Finally, we describe the MorphMix simulator and present the simulation results in Chapter 8.

The basic idea of MorphMix has been published as a refereed paper [98] and in larger technical report [97] that contains more details and early analyses than the paper version. The detailed analysis of MorphMix has been published in another refereed paper [100].

5.1 Motivation and Goals

We have seen in Chapter 4 that dynamic mix networks have advantages over static mix network, especially if the number of users that should be supported gets large. However, it remains to be shown whether it is feasible to design a dynamic mix networks in a way such that it fulfils our principal goal we have stated in Section 1.6, which is to develop a practical system that enables anonymous low-latency Internet access for a large number of users. This is exactly what we want to achieve with MorphMix, and the more detailed goals are as follows:

1. **Requirements to Participate**: Anybody who has access to a computer that is connected to the Internet should be able to join and use MorphMix after having installed the MorphMix software. Participating should be possible with a computer that has a public (static or dynamic) IP address as well as with a computer that has a private IP address that accesses the Internet through a NAT gateway. The bandwidth or computing power requirements should be modest in the sense that any computer with a dial-up connection capable of running a modern graphical web browser should be sufficient.
2. **Scalability**: MorphMix should be able to efficiently cope with a very large number of users. This means that even if there are millions of participating users, the overhead to establish and tear down circuits and to manage the MorphMix overlay network can easily be handled by every participating computer that fulfils the requirements above. In addition, the performance as perceived by a user should not decrease as the number of participants increases.
3. **Protection from Attacks**: MorphMix should provide good protection from the realistic attackers we have identified in Section 4.3. With good protection, we mean that MorphMix does not guarantee the anonymity

of every single transaction, but provides very good protection from long-term profiling. In particular, MorphMix should protect well from an internal attacker operating parts of the system himself, as this is easily possible due to the openness of the system.

4. **Performance**: MorphMix should be able to deliver adequate end-to-end performance to its users despite the dynamic and heterogeneous environment where users that are part of MorphMix may turn off their computers at any time. Although adequate performance is difficult to quantify, it should be good enough such that MorphMix users are not turning away from the system because of its poor performance. In particular, the performance as perceived by a user with a good Internet connection (e.g. DSL with 512 Kb/s) should not be degraded to the performance of a dial-up connection.

It is important to realise that perfect anonymity is not a design goal of MorphMix, at least not at this time. In fact, as discussed in Section 4.1, considering the asynchronous nature of the Internet and powerful attacks on mix networks, it is very unlikely that operating a practical mix network that offers perfect anonymity is possible at all – especially when the mix network aims at supporting low-latency applications. We have seen that there are cover traffic schemes that help against certain kinds of external adversaries, but in particular when aiming at defeating end-to-end traffic analysis, they introduce significant data overhead. In addition, cover traffic is of little value against an internal attacker, which we consider as the most serious threat to MorphMix (see Section 5.4). Consequently, we do not employ any cover traffic mechanism in MorphMix because we believe it is not worth its costs. However, since MorphMix is essentially a mix network, we state that if more efficient cover mechanisms that significantly increase the protection mix networks offer from external observers will be ever developed, they should be easily applicable to MorphMix.

5.2 Basic Functionality of MorphMix

MorphMix is basically a circuit-based mix overlay network with many similarities to static circuit-based mix networks as described in Chapter 2. However, it has a few special properties and we therefore give a detailed description of its basic functionality, following the terminology introduced in Chapter 2 whenever possible. The MorphMix protocol itself is provided in Ap-

pendix A.

5.2.1 Overview

MorphMix is a peer-to-peer-based mix network and consequently, we no longer distinguish between clients and mixes and simply refer to them as *nodes*. Every node joining the system can itself establish circuits via other nodes to access a server anonymously, but can also be part of circuits established by other nodes and relay data for them at the same time. A node i is identified with its public IP address ip_i. For now, we assume all nodes have public static IP addresses; we will discuss other cases in Section 5.8.2. To be contacted by other nodes, a node listens on TCP port p_{mm_i}, which is 28080 per default, but which can be changed by the node operator. In addition, each node has a key pair consisting of a secret (or private) key SK_i and a public key PK_i. This key pair is generated locally when a node is started for the first time. At any time, MorphMix consists of a set of participating nodes. Nodes can join and leave the system at any time and must therefore not necessarily participate in the MorphMix protocol all the time. From now on and unless specified otherwise, we mean a currently participating node when we are talking about a MorphMix node. We assume that at any time, a node knows about some other nodes, i.e. their IP addresses, the ports on which the MorphMix application is listening for incoming connection requests, and their public keys. Learning about other nodes requires a peer discovery mechanism, which we will describe in Section 5.7. Figure 5.1 depicts the basic idea of MorphMix.

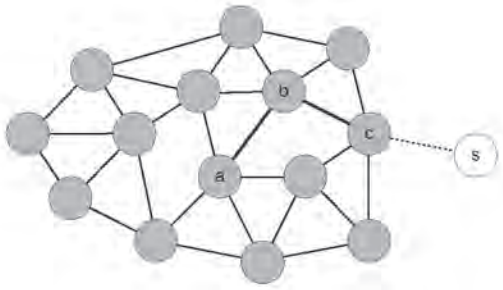


Figure 5.1: *Basic idea of MorphMix.*

A node that is participating in MorphMix has established a *virtual link* to one or more other MorphMix nodes at any time. In MorphMix, a virtual link means that (1) there is a TCP connection between the two nodes and (2) they share a symmetric key that is only known to these two nodes. To establish a virtual link to a node b, node a first establishes a TCP connection with b by connecting to ip_b:p_{mm_b}. Node a then selects a random bit-string that serves as the symmetric key for the virtual link. The key is encrypted with b's public key and sent to b. Using TCP connections between two nodes implies that MorphMix is basically operated as illustrated in Figure 2.6(b), which is reasonable because of the heterogeneity of the nodes (see Section 2.3.4). The set of nodes to which a node a has currently established virtual links are a's *neighbours*. In Figure 5.1, a has five neighbours because it has established virtual links to five other nodes.

5.2.2 Anonymous Tunnels and Anonymous Connections

Since MorphMix is basically a circuit-based mix network, a node establishes a circuit via some other nodes to access servers in the Internet anonymously. In Section 2.3, we have described how a circuit is set up in Onion Routing to connect to a server. This circuit can then be used to communicate anonymously with the server but if another connection must be established with the same server or if a different server is contacted, a new circuit must be established. While this is not a problem with long-standing communication relationships as used in remote login sessions, it is less well suited for applications such as web browsing that frequently establish multiple short-lived connections in parallel to the same web server. Since setting up circuits in MorphMix is a relatively expensive operation (see Section 5.5.1), we make use of a concept we have introduced in the context of the Anonymity Network (see Section 3.1.2) that allows using the same circuit to have several communication relationships with a single or different servers. To do so, we distinguish between *anonymous tunnels* (often referred to simply as *tunnels*) and *anonymous connections*. Anonymous tunnels correspond to the circuits as introduced in Section 2.3.

In Figure 5.1, we assume node a has established an anonymous tunnel via b and c. The first node in a tunnel (a) is the *initiator*, the last node (c) is the *final node*, and the nodes in between (b) are *intermediate nodes*. The initiator and the final node are also called the *endpoints* of a tunnel. The total number of nodes in a tunnel is the *tunnel length* and equals three in the

example above. If the tunnel length is more than three, there are multiple intermediate nodes. Note that like in any mix network, the servers that are contacted anonymously are *not* part of MorphMix and the tunnel ends at the final node, not at the server.

Within an anonymous tunnel, anonymous connections can be set up to anonymously communicate with a server. Assuming a client application running on a MorphMix node wants to communicate anonymously with a server, and assuming the node has set up at least one tunnel that is ready to be used, each TCP connection request issued by the application results in exactly one anonymous connection being established within an anonymous tunnel, which results in exactly one TCP connection being established between the final node in the tunnel and the server. If the client or the server terminates the communication relationship, the corresponding anonymous connection is also terminated, but the anonymous tunnel remains established and new anonymous connections can be set up within this tunnel. The idea of multiple anonymous connection within a single anonymous tunnel is illustrated in Figure 5.2:

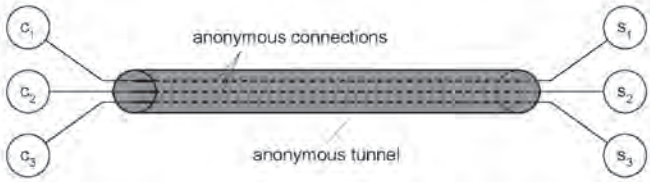


Figure 5.2: *Multiple anonymous connections within one anonymous tunnel.*

Three client applications c_1–c_3 (e.g. web browser windows) on the initiator's computer communicate anonymously with three server applications s_1–s_3 through the same anonymous tunnel. For each of these three communication relationships (corresponding to a direct TCP connection in the non-anonymous case), there is one anonymous connection. Note that it does not matter whether the server applications are located on the same physical server or not because despite using the same anonymous tunnels, anonymous connections within the same tunnel are independent from one another. As we will see below, anonymous connections are only visible at the endpoints of an anonymous tunnel, i.e. at the initiator and final node. Intermediate nodes only see the anonymous tunnel and cannot tell how many anonymous connections there are currently established.

Like static mix networks, MorphMix employs multiple *layers of encryptions* (see Section 2.3) to complicate traffic analysis attacks. Consequently, once the anonymous tunnel from a via b to c has been established, it looks as illustrated in Figure 5.3.

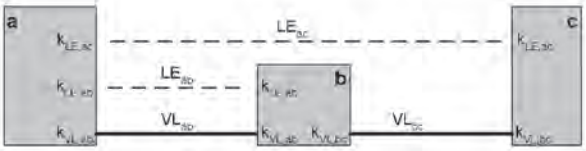


Figure 5.3: *Virtual links and layers of encryption along an anonymous tunnel.*

There are virtual links between nodes a and b (VL_{ab}) and b and c (VL_{bc}) that are communicating directly with each other across TCP connections. The corresponding symmetric keys shared by the two endpoints of each virtual link are $\mathrm{k}_{VL,ab}$ and $\mathrm{k}_{VL,bc}$, respectively. In addition, there is one layer of encryption between the initiator a and each other node (b and c) along the tunnel, identified with LE_{ab} and LE_{ac}. A layer of encryption between two nodes means that the two nodes share a symmetric key ($\mathrm{k}_{LE,ab}$ and $\mathrm{k}_{LE,ac}$, respectively) that is only known to them. At first glance, it seems pointless to have two different shared secret $\mathrm{k}_{LE,ab}$ and $\mathrm{k}_{VL,ab}$ between a and b. However, this has to do with the property of MorphMix that b cannot easily tell that a is the initiator of the tunnel, as will be discussed in more detail in Section 5.3. Note also that like in static mix networks, a virtual link is used to transport the data of potentially multiple anonymous tunnels that are making use of this virtual link. A layer of encryption, on the other hand is associated with exactly one anonymous tunnel.

5.2.3 Cells and Messages

All data exchanged between two neighbouring nodes are transported within the payload of *fixed-length cells*. A cell consists of a 16-byte header and a 496-byte payload, resulting in a cell length of 512 bytes, and we give arguments and show simulation results that support this choice in Section 8.3.8. If the length of the data is longer than what fits into the payload of a single cell, the data are split such that they fit into the payloads of multiple cells. The payload of the last cell is padded with random bits to its fixed length.

The header of a cell contains an identifier that has only local significance on a virtual link between two nodes to determine what cells belong to which anonymous tunnel and to correctly forward the data along its tunnel. If the cell does not belong to a specific anonymous tunnel and is merely used to exchange control information between two neighbours, a special identifier is used (0). The header also contains a type to distinguish different types of data transported within the cells and a checksum to check the integrity of the data and to counter replay attacks (see Section A.2.2). To prevent an external observer from easily identifying what cells exchanged between neighbours belong to which tunnel, the header is encrypted using the symmetric keys of the virtual link.

We refer to the data that are exchanged between nodes and transported within cells as *MorphMix protocol messages*, or simply *messages*. Messages are either exchanged between two neighbours or between the endpoints of an anonymous tunnel. In the first case, the messages are directly transported within the payloads of the cells the message must not be forwarded to another node. In the second case, we refer to them as *end-to-end messages* and the messages are not directly put into the payloads of the cells. Rather, end-to-end messages are always associated with a specific anonymous connection and to multiplex the end-to-end messages of multiple anonymous connections within the anonymous tunnel, a second 16-byte header is used at the beginning of the cell payload. This second header looks exactly like a cell header and we also refer to it as *anonymous connection header*. The identifier in the anonymous connection header is used to distinguish the data of different anonymous connections and similar as in the cell header, the identifier (0) is used to exchange control information between the endpoints of an anonymous tunnel.

Since the layers of encryption cover the entire cell payload, they also covers the anonymous connection header, and the different anonymous connections are therefore not visible for the intermediate nodes. Intermediate nodes only care about the cell headers to forward the payload (which includes the anonymous connection header) correctly along an anonymous tunnel.

5.2.4 Anonymous End-to-End Communication

The application data to be anonymised are transported within anonymous connections. One anonymous TCP connection from the initiator's computer to a server is mapped onto exactly one anonymous connection. In general,

MorphMix can anonymise any TCP-based application but in its first version, we focus on web browsing and MorphMix supports HTTP (both versions 1.0 and 1.1) and HTTPS. Since MorphMix works as illustrated in Figure 2.6(a), client applications access MorphMix through an access program that acts as a proxy. As we do no longer distinguish between clients and mixes, this access program is part of the MorphMix software itself and a MorphMix node listens for anonymous communication requests by client applications on TCP port p_{appl}, which is 8080 by default, but which can be changed by the node operator. To communicate anonymously with a server application identified with ip_s:p_s, the client application connects to port p_{appl} on the local computer and sends ip_s and p_s to the access program. The node then establishes an anonymous connection within a previously set up anonymous tunnel by sending an end-to-end message to the final node that contains ip_s and p_s. The final node connects to the server and, assuming anything worked correctly, sends back an end-to-end message to the initiator to indicate the connection has been established. The initiator itself then notifies the client application that the connection has been established and application data can be exchanged between the client and server applications. Once this end-to-end communication relationship has been established, application data can be exchanged between client and server application. Figure 5.4 illustrates how the application data are transported within the anonymous connection along the anonymous tunnel, and how the forwarding is done using only anonymous tunnel information.

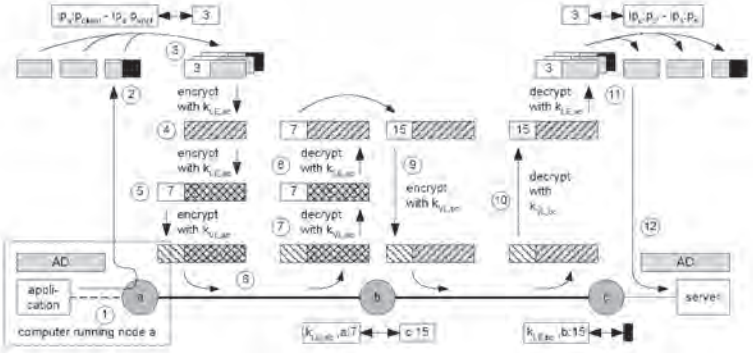


Figure 5.4: *Anonymous connections and cell forwarding.*

There exists an anonymous tunnel from a via b to c. We assume that 7 is the identifier on the virtual link from a to b and 15 on the virtual link from b to c. As explained in Section 2.3, each node along an anonymous tunnel must know what to do with cells it receives. In our example, this means that b knows that every cell arriving from a with identifier 7 must be forwarded to c with identifier 15. The identifier also tells b to use key $k_{LE,ab}$ to remove the corresponding layer of encryption. Similarly, c knows that cells arriving from b with identifier 15 have reached their final node and that the layer of encryption can be removed with $k_{LE,ac}$. We assume the anonymous connection within the anonymous tunnel is identified with identifier 3. Similarly, the connection from the client application to a is identified with the socket pair ip_a:p_{client}–ip_a:p_{appl} and consequently, a knows that everything arriving from the connection corresponding to this socket pair must be sent through the anonymous tunnel using the anonymous connection with identifier 3. Finally, the connection between c and the server application is identified with the socket pair ip_c:$\text{p}_{c'}$–ip_s:p_s, which tells c to forward the data it receives through the anonymous connection identified with identifier 3 must be sent to the socket identified with this socket pair.

To send application data (AD) from the client to the server, the following steps are carried out:

1. The client application sends AD to a.
2. Node a chops AD such that each resulting piece (while leaving room for the anonymous connection header) fits into the payload of one fixed-length cell. From the MorphMix' protocol point of view, each of these resulting pieces is an end-to-end message that must be sent from one endpoint of the tunnel (a) to the other (c). In Figure 5.4, we assume that this results in three pieces and the last piece is padded with random bits to its maximum length.
3. An anonymous connection header that contains the identifier of the anonymous connection (3) is prepended to each piece.
4. Each resulting piece of data is encrypted with $k_{LE,ac}$ corresponding to the layer of encryption between a and c and then with $k_{LE,ab}$ corresponding to the layer of encryption between a and b. Note that for simplicity, we illustrate this and also the following cell transport along the anonymous tunnel for only one piece in Figure 5.4.
5. The resulting data are put into the payload of a fixed-length cell and the cell header is filled with the correct identifier to identify the cell on the virtual link between a and b.

6. The cell header is encrypted using key $k_{VL,ab}$ that corresponds to the virtual link between a and b and the cell is sent to b.
7. Upon receiving the cell, b decrypts the cell header.
8. Node b sees the identifier (7), and knows that the layer of encryption can be removed from the payload with $k_{LE,ab}$.
9. Node b also knows that the payload must be forwarded to c with identifier 15. Consequently, a new cell header is prepended, encrypted with $k_{LE,bc}$, and the resulting cell is forwarded to c.
10. Node c decrypts the cell header, sees the identifier 15, knows that the layer of encryption can be decrypted with $k_{LE,ac}$, and also knows that the cell has reached its final node.
11. Node c consults the anonymous connection header in the payload, sees that the identifier is 3 and knows that the rest of the payload (after removing padding bits) must be sent out on the connection given by the socket pair ip_c:p_c–ip_s:p_s.
12. The same is done for all cells c receives and consequently, the server eventually receives AD. Sending data back from the server to the client works exactly in the opposite way.

It should be noted that although we have clearly separated setting up the end-to-end communication relationship from the actual data exchange between the client and the server application in the example above for illustrative reasons, this is not necessary in practice. In fact, the client application can send the information about the server to contact and the application data together to the access program, as is done by web browsers when they use a web proxy. Since this saves one round-trip, MorphMix includes the application data in the request to establish an anonymous connection whenever possible.

Another important detail is that if the client application does not know ip_s but only the host name of the server, the client application must not resolve the host name itself, as this would easily tell an eavesdropper the server the client wants to access. Instead, the host name is sent instead of ip_s to the access program, which sends it to the final node when setting up an anonymous connection. Only the final node resolves the host name, which does not leak any information as the final node will contact the server directly anyway.

Note also that end-to-end messages are not only sent through completely set up anonymous tunnels like in the example above. During the setup of anonymous tunnels (see Section 5.5.1), nodes are appended hop by hop to the tunnel. At any time, the initiator can exchange end-to-end messages with the

node that is currently the final node in the tunnel.

5.3 Requirements to Break the Anonymity

In this section, we identify the requirements for an adversary to break the anonymity of a user, i.e. to link the initiator and the server of a communication relationship.

An advantage of MorphMix compared to static mix networks is that in the latter, the first mix in a circuit or an eavesdropper observing the cells exchanged on the virtual link between the client and the mix can easily identify the client because the roles of clients and mixes are clearly separated. In MorphMix, however, this is not the case because every client is also a mix. This means that in Figure 5.1, b cannot be sure if a is the initiator of the tunnel or if a is just another intermediate node in this tunnel that relays data for yet another node. The MorphMix protocol guarantees that no such information is leaked by the content of the messages, neither during the anonymous tunnel setup nor during the exchange of application data between initiator and server. This property of MorphMix is often referred to as *plausible deniability* because assuming a user is accused of having contacted a server anonymously, she can always claim she only relayed the data for another node.

We first analyse the internal attacker controlling a subset of all nodes. Recalling our discussion in Section 4.3.2 about static mix networks, we should say that the relationship anonymity is broken if the adversary controls the first intermediate and the final node in the tunnel. This allows the adversary to correlate the cells exchanged between the initiator and the first intermediate node and data exchanged between the final node and the server. If sufficient data are exchanged between initiator and server, this allows the adversary to link the initiator and the server with high probability. However, due to the plausible deniability property of MorphMix, the operator of the first intermediate node cannot be sure that his node is indeed the first intermediate node. In practice, however, it can be assumed that the number of nodes along anonymous tunnels is reasonably low, e.g. five, and the operator of an intermediate node should be able to at least guess how many nodes are following in an anonymous tunnel. For instance, the patterns of cells flowing through a node when nodes are appended to a tunnel are quite typical (see Figure 5.5) and an intermediate node can derive with high probability how many nodes are following in the anonymous tunnel. Assuming most tunnels have a length

of five and an intermediate node recognises by analysing the traffic patterns that three additional nodes are appended to the tunnel, it can derive with high probability that it is indeed the first intermediate node. There are additional ways to guess the position of a node in an anonymous tunnel. Assuming the adversary controls the final node in a tunnel and knows that he controls another node in this tunnel, he can measure the time it takes for the fixed-length cells to travel between the two nodes he controls. This gives him an indication about how many other nodes there may be in between. Another property that can be exploited by an intermediate node is measuring the time it takes for the initiator to react to data it receives from the server. For instance in the case of web browsing, embedded objects in a web page are automatically requested by the browser, which results in quickly sending back data to the web server once the web browser has received the index file of a web page. If the measured time between sending cells towards the initiator and receiving cells from it is small, the intermediate node can conclude with high probability it is indeed the first intermediate node. We therefore assume that if an adversary controls the first intermediate and final node in an anonymous tunnel and data are exchanged between initiator and server through this tunnel, then the adversary can always break the relationship anonymity and link the initiator and the server. This is a worst case assumption because everything else is difficult to quantify. In practice, it may not always be easy for the first intermediate node to correctly determine its position in a tunnel with high probability. In addition, especially if only few data are exchanged between initiator and server, it may not even possible for the first intermediate and final nodes do determine that they are part of the same anonymous tunnel. Nevertheless, assuming there are n nodes in MorphMix, the adversary controls n_c of them, and there is at least one intermediate node in every tunnel, the adversary is able to break the relationship anonymity between initiator and server with a probability of $p_b = (n_c/n)^2$ according to (4.23).

With the external attacker, we distinguish between two cases. In the first case, the attacker observes the initiator directly, which means he sees all traffic entering and exiting the node on all virtual links. Since no cover traffic is employed, this should enable the observer to separate the data that originate or terminate at this node from the data that are relayed for other nodes. The second case is an observer that only sees the cells on the virtual link between the initiator and the first intermediate node in a tunnel. Using similar arguments as for the adversary controlling the first intermediate node above, we can argue this observer may be able to derive which cells originate and terminate

at the initiator in some cases, although this is significantly more difficult than for the first intermediate node because the observer cannot easily separate the data belonging to different circuits. However, again for the reasons that anything except the worst case is difficult to quantify, we assume an adversary observing the virtual link between the initiator and the first intermediate node can always learn which cells originate and terminate at the initiator. Consequently, and recalling our discussion in Section 4.3.1, we therefore assume an eavesdropper can always break the relationship anonymity between the initiator and the server if he observes the cells on the virtual link between the initiator and the first intermediate node and the data exchanged between the final node and the server. Assuming the adversary observes a fraction of f_o of all Internet traffic and assuming the traffic exchanged between nodes in the mix network and between mixes and servers is similarly distributed across the Internet as all traffic, the adversary is able to break the relationship anonymity between initiator and server with a probability of $p_b = f_o^2$ according to (4.22).

Interestingly, the probabilities to break the relationship anonymity are the same as in static mix networks (see 4.3). However, there is an important difference in the sense that in static mix networks, the adversary always knows who the initiator is. Consequently, it can be expected that attacks on static mix networks are nearly as successful in practice as the formulas indicate. In the case of MorphMix, on the other hand, there is often the uncertainty whether the suspected initiator is really the initiator of a tunnel or merely relaying the data for another node because of the plausible deniability property. Consequently, the probability an adversary manages to break the relationship anonymity between initiator and server can be expected to be significantly smaller than in our worst case assumptions. Nevertheless we will use these worst case assumption as a reference during the remainder of this thesis.

5.4 Threat Model

In this section, we state our threat model. We look at the two most likely attackers on large mix networks that we have introduced in Section 4.3, the passive external attacker and the active internal attacker.

5.4.1 The Passive External Attacker

With MorphMix, we want to provide anonymity for the masses and aim at a large number of nodes distributed all around the world. According to our definition of a realistic threat model (see Section 4.3), this makes it is very unlikely any passive external attacker can observe a significant portion of all data that are processed by MorphMix nodes. On the other hand, observing a small portion of the traffic is certainly possible for certain potential adversaries such as ISPs and as discussed in Section 5.3, such an attacker will occasionally succeed in breaking the relationship anonymity between an initiator and a server. However, recalling that even monitoring 10% of all traffic only allows to link the initiator and the server in 1% of all cases and that one goal of MorphMix is to provide protection from long-term profiling and not to guarantee the anonymity of every single transaction, we accept this limited vulnerability to the partial external attacker. Increasing the resistance of MorphMix to this attack depends on the development of efficient cover traffic mechanisms, which it outside the scope of this thesis.

5.4.2 The Active Internal Attacker

There is no admission control and the first goal in Section 5.1 states that everybody with a computer connected to the Internet having access to a computer can easily join the system. As a result, we must assume there are *honest* nodes, which are nodes that do not try to break the anonymity of other users and there are *malicious* nodes, which collude with other malicious nodes to break the anonymity of honest users. As mentioned previously, a MorphMix node is identified with its IP address and is therefore usually associated with a single computer. Consequently, we assume honest users typically run exactly one node on their own computer.

An adversary that possesses several IP addresses can run several malicious nodes. However, it is not necessarily required for the adversary to operate a dedicated physical computer for each node. In particular, if an adversary owns a contiguous range of IP addresses, he can operate multiple nodes on a single computer by using special software (e.g. Linux-VServer [1]) that allows a computer with a single network interface to have multiple IP addresses. There are additional possibilities for an adversary that owns many IP addresses to operate many nodes using only a few physical computers. An access ISP, for

[1] http://www.13thfloor.at/vserver/project

instance, could run a node for each of its IP addresses it currently has not assigned to its customers. One place to do so could be at the border gateway(s) to the access ISP's backbone provider(s) where all data entering and exiting the access ISP's network must pass. In general,we do not claim that operating many nodes is a trivial task for an adversary that owns a range of n IP addresses, but we must not argue that it is as complicated or as expensive as running n different physical computers. Consequently, we assume it is feasible for an adversary to run as many nodes as he owns IP addresses. In the case of a class B network, this would allow the adversary to operate 65533 MorphMix nodes. Even if there were 100000 honest nodes in MorphMix, such an attacker could relatively easily control nearly 40% of all nodes. According to our discussion in Section 5.3 and assuming the nodes along an anonymous tunnel are picked randomly, this would allow him to break the relationship anonymity in 16% (because $0.4^2 = 0.16$) of all cases. This is a serious and very realistic threat to MorphMix users.

So it is quite easy for certain adversary such as ISPs or large institutions in general to run very many MorphMix nodes. To defend against such an adversary, we must first understand how IP addresses are assigned. For now, we focus on IP version 4 (IPv4); the influence of IP version 6 (IPv6) will be discussed in Section 5.9. Although the traditional notion of class A, B, and C networks has been blurred due to subnetting [78] and classless interdomain routing (CIDR) [96], it is still the case that usually contiguous ranges of IP addresses are under a single administrative control. Consequently, although it is easy for an adversary that possesses many IP addresses to run many nodes, all of them are "similar" in the sense they usually all have the *same IP address prefix*, i.e. the first few bits of the different IP addresses are equal. Using again the example above with 100000 honest nodes, we can assume these nodes have IP addresses with 1000s of different 16-bit prefixes. The adversary owning an entire class B network can still operate 65533 nodes, and looking at the complete IP addresses of all 165533 nodes present in this system, he controls about 40% of all. But if we only look at the 16-bit prefix of the IP addresses, the adversary "controls" only one out of 1000s of different 16-bit prefixes.

This assumption of the similarity of IP addresses of the malicious nodes controlled by the adversary is the key to deal with the problem of defending against an internal attacker. We say that all IP addresses with the same 16-bit IP address prefix are in the same */16 subnet*. There are exactly 56559 public /16 subnets in the Internet: the unicast address range between 1.0.0.0

and 223.255.255.255 corresponds to 57088 different /16 subnets, but 256 of them are assigned for local addresses (127.0.0.0 – 127.255.255.255) and 273 are reserved by the three private address ranges (10.0.0.0 – 10.255.255.255, 172.16.0.0 – 172.31.255.255, and 192.168.0.0 – 192.168.255.255). Following our discussion above, we conclude that while it is easy for an adversary running many nodes, it is much more difficult for him to control nodes in a significant portion of all 56559 /16 subnets. An adversary owning a class B network controls nodes in only one of 56559 possible /16 subnets. Even an adversary owning an entire class A network, which corresponds to the largest address ranges that were assigned to single institutions[2], can only run nodes in 256 different /16 subnets, which is less than 0.5% of all public /16 subnets.

Consequently, all three core components of MorphMix we will present in Sections 5.5–5.7, which includes the anonymous tunnels setup protocol, the collusion detection mechanism, and the peer discovery mechanism, do not operate on the whole IP address of a node, but only on its 16-bit IP address prefix. Although we will describe this in much more detail later in this chapter, the basic idea is that from the point of view of the collusion detection mechanism and the peer discovery mechanism, all nodes in the same /16 subnet are equal. So for an adversary to be effective, he must not only control many nodes, but he must control nodes in a wide variety of /16 subnets. Note that the choice of a 16-bit prefix is a compromise between making it difficult for an adversary to run nodes in a significant portion of all subnets (see below) and keeping the overhead of the collusion detection mechanism and the peer discovery mechanism within reasonable limits (see Section 5.8).

We have seen that an adversary won't manage to control nodes in very many different /16 subnet if he runs them only in the subnets he owns. Another strategy for the adversary is to operate nodes also in /16 subnets he does not own. Acquiring IP addresses under his own name in a wide variety of /16 subnets could soon become suspicious. So instead of acquiring IP addresses under his own name, the adversary could provide private persons with the necessary equipment to operate nodes at their homes and pay them, for instance, 1000 US$ a year in addition. Assuming the infrastructure (a decent network connectivity and a PC) costs 4000 US$ a year per node, there are yearly costs of 5000 US$ per node. Convincing 1000 people to run 1000 nodes in 1000 different /16 subnets would therefore cost five million US$ per year. This is certainly not a barrier for certain well-funded organisations that would like to

[2] http://www.iana.org/assignments/ipv4-address-space

break the anonymity of the MorphMix users. To do so, the adversary would somehow have to advertise that he is looking for users operating nodes for him, which again could eventually become suspicious. In any case, even if the adversary does not care if he is detected, getting control over 1000s of nodes in as many different /16 subnets either by operating them himself or by private persons he provides with money and equipment is very difficult.

We say that an adversary *controls* a fraction between 0 and 1 of a /16 subnet depending on the ratio of malicious nodes to all nodes in this subnet. If a subnet contains only malicious nodes, the adversary controls a fraction of 1 of the subnet, and we also say he has full control of the subnet. If it does contain only honest nodes, the adversary controls a fraction of 0 of the subnet, and we also say he has no control of the subnet. If it contains both honest and malicious nodes, the adversary controls any fraction greater than 0 and smaller than 1. With n_s nodes in a /16 subnet s where $n_{h,s}$ are honest and $n_{m,s}$ are malicious, the fraction $f_{c,s}$ the adversary controls is given with $f_{c,s} = n_{m,s}/n_s$. There is one difference for the adversary between owning a subnet and running a node in a subnet he does not possess: In the first case, he can operate as many nodes as there are IP addresses available, and he can force $f_{c,s}$ of this subnet close to 1 even if there are a few honest nodes. Another option is to simply block all MorphMix traffic from and to the honest nodes. Conversely, if the adversary runs nodes either by himself or by private persons in subnets he does not own, $f_{c,s}$ is often smaller than 1 if there are also honest nodes in the subnet. The reason is that the adversary controls only one or a small range of IP addresses and cannot run as many nodes as he likes.

5.4.3 Summary

We summarise the threat model based on the discussion in this Section. We do not consider the external observer as a significant threat because like in any large mix network distributed all over the world, it is unlikely such an adversary can observe a significant portion of MorphMix.

Since it is easy for an adversary owning a range of IP addresses to operate many node, we consider the internal attacker a much more serious threat. To reduce the potential impact from such an adversary, the core components of MorphMix do not operate on the whole IP address of a node, but only on its 16-bit IP address prefix. This is based on the assumption the adversary can operate nodes in a limited number of /16 subnets. He either owns the subnets and has full controls of them, but we do not assume it is realistic a single

adversary will ever own more than 1000 /16 subnets, which corresponds to about four class A networks or 1000 class B networks. Even the largest ISPs do not control addresses in so many /16 subnets, which can be seen by querying the whois servers of the Regional Internet Registries such as RIPE NCC[3] or ARIN[4]. The other option for the adversary is to run nodes in subnets he does not possess, either by himself or by private persons. Usually, he controls a fraction that is smaller than 1 in these subnets because there are also honest nodes present. Again, running nodes in several 1000 subnets is very difficult, in particular if the adversary wants to avoid that his activities to break the anonymity of the users of MorphMix become public. We also assume that malicious nodes can mark every cell they exchange with their neighbours and all application data they exchange with servers with a precise timestamp and send all relevant data (at least the IP addresses, ports, length of the application data, and the timestamp) to a centralised place where the traffic handled by different malicious nodes can be correlated to break tunnels as discussed in Section 5.3.

5.5 Establishing Anonymous Tunnels

In this section, we present and analyse the protocol to set up anonymous tunnels, which is the first major component of MorphMix. We first describe how anonymous tunnels are set up and analyse the procedure. Then we talk about the policy about how virtual links to neighbours should be used and why MorphMix provides incentive to relay the data of other nodes.

5.5.1 Anonymous Tunnel Setup

As we will see below, setting up an anonymous tunnel is a relatively complex process that usually takes several seconds to complete. In addition, as we will see in Section 5.6 when presenting the collusion detection mechanism, some tunnels will be rejected by the initiator and not used to contact a server. Consequently, anonymous tunnels are not established on demand to contact a server anonymously, because this would take too long until a tunnel were ready with high probability. Rather, setting up anonymous tunnels is a background process in the sense that at any time, there should be a few of

[3] whois.ripe.net
[4] whois.arin.net

them ready to be used. In Chapter 8 when we analyse the performance of MorphMix based on a simulator, we assume that a node has established five tunnels at any time. The reason for having multiple tunnels ready at any time is to quickly be able to switch to another tunnel in case one of them fails or offers poor performance. In addition, anonymous tunnels are only used for a limited time. We use the policy that anonymous connection may only be established within ten minutes after a tunnel has been set up. Note that after ten minutes, a tunnel is not simply torn down but stays alive until all anonymous connections within this tunnel have been terminated. There are two main reasons for this policy. The first is that the correct functioning of the collusion detection mechanism and the peer discovery mechanism rely on frequently setting up anonymous tunnels. The second is based on the observation that the more data that are exchanged through an anonymous tunnel, the higher the probability an adversary can break the anonymity in practice if he indeed controls the first intermediate and the final node in a tunnel or eavesdrops on the virtual link between initiator and the first intermediate node and on the route between the final node and the server (see Section 5.3). Consequently, limiting the time an anonymous tunnel can be used limits the amount of data that can be exchanged through this tunnel, which complicates the task for the adversary to break the anonymity. With five tunnels that can be used at any time and a lifetime of ten minutes per tunnel, this results in setting up a tunnel every two minutes on average.

An important design decision we made during the development of MorphMix is that an anonymous tunnel is set up hop-by-hop in the sense that the initiator picks the first intermediate node and establishes the layer of encryption with it. Then the initiator tells the first intermediate node to append another node to the tunnel and establishes the layer of encryption with the second intermediate node. This continues until the initiator decides the tunnel is long enough. The key is that the initiator selects only the first intermediate node and each node along the anonymous tunnel then picks the following node.

This has one big advantage: at any time, a node a only needs to have a few neighbours that it can append to anonymous tunnels if a is requested to do so. In addition, a can communicate with its neighbours over the virtual links it has established to them to learn which of them have spare resources and are willing to be appended to an anonymous tunnels. Conversely, assume the initiator would select all nodes of an anonymous tunnel itself. Except for the first intermediate node, it would have no idea about the current status of

the other nodes, e.g. whether they are still participating in MorphMix, and if yes, whether they are actually willing to accept further anonymous tunnels. For such a system to work efficiently, a lookup service would be required. The lookup service could be queried to get nodes that are currently willing to accept anonymous tunnels. Considering MorphMix is supposed to scale up to millions of nodes, a centralised lookup service that keeps track of the nodes that are currently participating in MorphMix is out of the question. An alternative are distributed scalable peer-to-peer lookup services such as Chord [120] where (a subset of) the MorphMix nodes themselves organise the Chord Ring to provide information about all participating nodes. However, the frequent joins and leaves of nodes and the continuously changing state of each node would generate a lot of traffic to keep the Chord Ring itself intact and to keep the information provided by the lookup service up-to-date. Letting each node select the next hop makes MorphMix highly scalable because independent of the system size, a node only has to manage its local environment and therefore only cares about a relatively small number of other nodes at any time. Of course, MorphMix nodes still need a way to learn about potential neighbours, but the peer discovery mechanism we employ (see Section 5.7) is especially tailored to meet the needs of MorphMix and is much simpler than a lookup service that continuously tries to keep track of the nodes that are currently participating in MorphMix.

However, there is also one major problem with this design decision. According to Section 5.3, we assume an adversary controlling a subset of all nodes can break the relationship anonymity if he controls both the first intermediate and the final node in a tunnel. If an initiator picks a first intermediate node that is controlled by the adversary, it can be therefore expected that the first intermediate node picks another node controlled by the adversary as the next hop and so on to make sure that the adversary controls both the first intermediate and the final node. Using this strategy, assuming there are n nodes, and n_c of them are controlled by the adversary, the adversary manages to break the relationship anonymity between initiator and server with a probability of $p_b = n_c/n$. If the initiator would pick all nodes along the tunnel randomly, the probability of success for the adversary would only be $p_b = (n_c/n)^2$. Note that is a significant difference. Assuming the adversary controls 10% of all nods, he only succeeds in breaking the relationship anonymity in 1% of all cases if the initiator picks all nodes in the tunnel. But if every node along the tunnel picks the following node, he can expect to break the relationship anonymity in 10% of all cases. So by letting each

node along a tunnel select the following node, we have actually significantly increased the chances for the adversary to break the anonymity of MorphMix users. To deal with this problem, we require that the node that selects the following node in a tunnel must first select multiple possible next hop nodes from the set of its own neighbours and offer these nodes in a *selection* to the initiator. The initiator simply picks one of them randomly and the node is appended to the tunnel. This alone does not prevent the attack above because a malicious node can simply offer exclusively malicious nodes in the selection, but this selection can be used by the collusion detection mechanism (see Section 5.6) to detect malicious nodes in tunnels with high probability.

Following the discussion above, setting up an anonymous means adding nodes hop-by-hop. The main goal when adding a node c is to establish a symmetric key for the layer of encryption (see Figure 5.3) between the initiator and c that is only known to the initiator and c. Except for the first intermediate node, the initiator does not know the nodes that will be added hop-by-hop along a tunnel as it is set up. Consequently, the initiator does also not know the public keys of these nodes beforehand and we therefore use the Diffie-Hellman (DH) key-exchange algorithm [34]. If the initiator simply sent its half[5] of the DH key-exchange to node b responsible for selecting the next hop c, b could easily play the role of c (and of other nodes following c) itself without the initiator noticing this. To counter this attack, we must not allow b to see the initiator's half of the DH key-exchange in the clear. To solve this problem, we introduce the notion of a *witness*. For each hop, the initiator selects a witness randomly from the nodes it currently knows (see Section 5.7). The witness' task is to act as a third party in the process of establishing a symmetric key between the initiator and the newly appended node. Figure 5.5 illustrates the principal procedure to append a node to an anonymous tunnel and to establish the layer of encryption. $\{d\}_{PK_i}$ denotes the encryption of data d with the public key PK_i of node i, and $\{d\}_{SK_i}$ denotes the signature (including the signed data) on data d with the secret key SK_i of node i. Figure 5.5 only illustrates the most important fields in the protocol messages. In Appendix A, we will describe the whole MorphMix protocol in much more detail.

Node a is the initiator. We assume the tunnel has already been set up to node b (via zero or more intermediate nodes). In addition, b has currently three virtual links established with nodes c, d, and e that are willing to accept

[5] One half of the DH key exchange corresponds to g^x where g is the publicly known group element and x the secretly chosen exponent

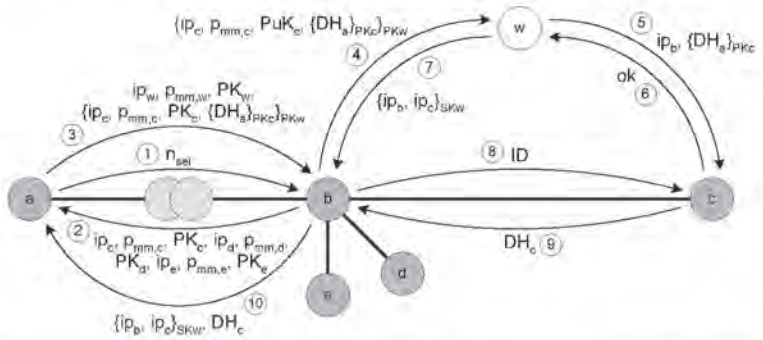


Figure 5.5: *Appending a node to a tunnel and establishing the layer of encryption.*

further anonymous tunnels. To append a new node to the tunnel, all messages exchanged between the initiator and the currently final node b (messages 1, 2, 3, and 10 in Figure 5.5) are transported within the tunnel that has been set up so far and are therefore end-to-end messages identified with the special anonymous connection identifier (0) for control data (see Section 5.2.3). All the other messages are directly exchanged between neighbours using the control data identifier (0) in the cell header. To append a node to a tunnel and to establish the layer of encryption with this node, the following messages are exchanged:

1. a sends a message to b that tells b to append another node to the tunnel. The message contains n_{sel}, which is the number of nodes b has to offer to a in message 2. Here, we assume $n_{sel} = 3$.
2. b receives the message and selects $n_{sel} = 3$ potential next hop nodes (c, d, and e) among its neighbours. It sends a message back to a that contains the IP addresses (ip_c, ip_d, and ip_e), ports (p_{mm_c}, p_{mm_d}, and p_{mm_e}), and public keys (PK_c, PK_d, and PK_e) of the three potential next hop nodes. We name the list of IP addresses offered by b the *selection* from b.
3. a receives the message and randomly picks one node from the selection from b as the next hop. Here, we assume c is selected. Then a picks a witness w randomly from the set of nodes it currently knows, generates its half DH_a of a DH key-exchange, encrypts it for c with PK_c, and

encrypts the resulting data together with ip_c, p_{mm_c}, and PK_c using w's public key PK_w. This results in in $\{ip_c, p_{mm_c}, PK_c, \{DH_a\}_{PK_c}\}_{PK_w}$, which is put into a message together with the witness' IP address ip_w, the port p_{mm_w} on which it accepts connections, and its public key PK_w, and is sent to b.

4. b receives the message, establishes a virtual link to w using ip_w, p_{mm_w}, and PK_w, and forwards the encrypted data.
5. w receives the message, decrypts the encrypted data to get ip_c, p_{mm_c}, PK_c and $\{DH_a\}_{PK_c}$, and establishes a virtual link to c using ip_c, p_{mm_c}, and PK_c. w generates a message consisting of ip_b and $\{DH_a\}_{PK_c}$ and sends it to c.
6. c gets the message and checks if it is indeed willing to accept an anonymous tunnel from b. If yes, c decrypts DH_a, and sends a message back to w telling it that everything is OK. In addition, c builds its own half DH_c of the key-exchange and uses DH_a to compute the key $k_{LE,ac}$ for the layer of encryption between a and c.
7. w receives the message and generates the *receipt* for a. The receipt contains the IP addresses of b and c and is signed by w using SK_w.
8. b receives the message from w and learns that w has selected c as the next hop. It generates a message containing the identifier ID to be used to identify data belonging to this anonymous tunnel on the virtual link between b and c and sends it to c.
9. c gets the message and sends its half DH_c of the DH key-exchange back to b.
10. b generates a message consisting of DH_c and the receipt from w and sends it to a. Upon receiving this message, a checks if the receipt is indeed signed by w and if it contains b's and c's IP addresses. a then uses DH_c to compute the key $k_{LE,ac}$ for the layer of encryption between a and c.

There are two important points to notice about the procedure to append a node to a tunnel. The first is making sure that b does not learn a's half of the DH key-exchange as this would easily enable b to simulate all remaining hops by itself. This is achieved by encrypting DH_a first for c and then for w, which guarantees DH_a is never seen in the clear except at c. In particular, b never sees DH_a in non-encrypted form. The second is preventing b from selecting the next hop purely by itself. This is achieved by having b offering a selection of possible next hops to a and a selecting one of them. This guarantees that b cannot predict which of the nodes in the selection is going to be picked as

the next hop and makes it much more complicated for b to determine the next hop. In particular, if b wants to make sure that c is in the same set of colluding nodes as itself, then all nodes in the selection from b must be in that collusion.

Note that when appending the first intermediate node, a and b in Figure 5.5 are "the same node" because a itself appends the first intermediate node. Consequently, a simply picks the first intermediate node from its neighbours and only messages 4–9 must be used. However, from the point of view of of the witness or the node c that is appended, appending a node always consists of using messages 4–9. As a result, they cannot distinguish appending the first intermediate node from appending any other node by analysing the content of the messages, which confirms the plausible deniability property of MorphMix (see Section 5.3).

5.5.2 Analysis of the Anonymous Tunnel Setup

We analyse the security of the anonymous tunnel setup. We are interested in the possibilities an adversary has to either learn the key exchanged between the initiator and the newly appended node, to simulate the next hop himself, or to make sure the node that is appended by a malicious node is also malicious. We use Figure 5.5 as the reference for the analysis.

Case 1: b is malicious Let's assume w is honest but b is malicious and no further nodes are collaborating with b. The goal of b is to break the layer of encryption that will be set up between a and c to read the data exchanged between them. To do so, b can try to simulate c itself by carrying out the following steps:

1. In message 2, b replaces the public keys corresponding to the IP addresses with self-generated versions it knows the secret keys of. The IP addresses can be those of nodes participating in MorphMix, but this is not required.
2. b intercepts the data sent from w to c to set up the virtual link and acts as c itself. This allows b to decrypt DH_a in message 5 because it knows the corresponding secret key.
3. b generates the OK message for c and sends it to w in message 6 to get the receipt from w in message 7.
4. b generates c's half of the DH key-exchange and inserts it in message 8.

The main difficulty for b is getting active control on the virtual link between w and c with the capability to intercept and inject data. Since b cannot predict which witness a is going to choose, b cannot prepare itself in advance and it is difficult to intercept data close to w. It seems more realistic for b to intercept data close to c, especially as it is b that selects the list of nodes in message 2. To make the attack as complicated as possible, we require that all IP addresses offered by b and b's own IP address must be in different /16 subnets. This does not prevent the attack, but makes it much more difficult because it introduces a greater diversity of the physical paths of the possible virtual links between w and c, and b needs active control over all of them to make sure the attack can be carried out successfully. Nevertheless, even if we assume the adversary has indeed managed to simulate the next hop itself, it gets even more difficult for him if the initiator requests to append yet another node to the tunnel. The problem for the adversary is that when appending this additional hop, the receipt generated by the witness contains b's IP address. Since the initiator expects c's IP address in the receipt, the initiator notices the attack and does not use the tunnel. The only way to avoid detection is if all witnesses to set up further steps are colluding with b, as will be discussed in case 2.

To avoid these problems, b can carry out a man-in-the-middle attack using the following steps:

1. In message 2, b replaces the public keys corresponding to the IP addresses with self-generated versions it knows the secret keys of. This time, the IP addresses must be those of nodes that are currently participating in MorphMix.
2. b intercepts the data sent from w to c to set up the virtual link and performs a man-in-the middle attack on the key exchange to generate the symmetric key for this virtual link.
3. b decrypts DH_a in message 5, replaces a's half of the DH key-exchange with a self-generated version DH_a' and includes it in message 6. The protocol then continues normally until b has received message 9.
4. Before sending message 10 to a, b replaces c's half DH_c of the DH key-exchange with an own version, DH_c'.

Node b has now broken the layer of encryption between a and c because it has split it into two parts: between a to b, the data is encrypted using keys generated from DH_a and DH_c', whereas between b and c, it uses keys generated from DH_a' and DH_c. In contrast to the case where b simulates the next hop b by itself, the initiator will not detect the attack if another hop is appended be-

cause the receipt will contain the correct IP addresses. However, controlling further nodes or breaking additional layers of encryption is very difficult for b because setting up the next hop will be handled by c and the IP addresses in the selection are no longer offered by b. Note that the IP addresses in the selection offered by c to a could be replaced by b since it has broken the layer of encryption between a and c. But the receipt from the witness will uncover this attack because c detects that the IP address of the newly appended node was not in the selection it offered to the initiator.

Only a very powerful attacker having active control over significant parts of the Internet could be able to carry out both these attacks. Following our discussion in Section 4.3.1, such an attacker is extremely unlikely to exist.

Case 2: b and w are malicious and colluding In this case, b attacks in the same way as above to simulate the next hop itself. Since b and w collude, b trivially has active control on the virtual link between w and c. As discussed above, this attack will be detected when the next hop is appended unless the next witness is also colluding with b. So the attack only succeeds if all remaining witnesses collude with b. In addition, b only learns about w after it has offered the selection to a. Consequently, all b can do is to include fake public keys in the selection and "hope" the initiator selects a malicious witness. In general, if the percentage of malicious nodes is relatively large and only a few nodes (e.g. one or two) nodes are appended after b, the attack may be successful with significant probability. Since the attack does not require active control over virtual links, we will analyse its impact in Section 6.3.

Malicious witnesses can also help if they are following later in the tunnel. If b has managed to simulate the next hop by itself or has broken a layer of encryption according to case 1 and an additional hop needs to be appended, a malicious witness can produce an appropriate receipt to please the initiator. Still, this does not make the attack significantly easier because b still needs active control over several virtual links and the probability all following witnesses are colluding with b is very small unless the percentage of malicious nodes is relatively large. In the latter case, the attack described in case 3 is much more likely.

Case 3: b is part of a set of cooperating malicious nodes If we assume that b is not alone but is part of a larger set of cooperating malicious nodes, then b can simply list a subset of these malicious nodes in message 2 and it is

guaranteed that the next hop is also part of the cooperating set. As we have required that the IP addresses must all be in different /16 subnets, the malicious nodes must reside in different subnets, which makes the attack more difficult according to our threat model (see Section 5.4). Nevertheless, if an adversary manages to control several nodes located in different /16 subnets, then this attack is quite easy to carry out.

Summarising the attacks discussed in this section, we conclude that the most realistic attack is the one where a set of cooperating malicious nodes tries to learn more about anonymous end-to-end connections (case 3). To defend against this attack, we introduce a collusion detection mechanism in Section 5.6. The first attack (case 1) requires active control over several virtual links and is therefore much harder to carry out. The second attack (case 2) does not require active control over virtual links, but we will show in Section 6.3 that it is not more successful than the attack described in case 3.

5.5.3 Policy For Using the Virtual Links to Neighbours

Basically, the virtual links to neighbours are bidirectional which means that although a has initiated the connection to b to establish a virtual link, it can be used by both endpoints to advertise the other node in selections. Similarly, a can pick b as its first intermediate node in a tunnel and vice versa. However, such a policy makes an attack possible where the adversary has several of the nodes he controls establish virtual links to an honest node a such that a ends up with many malicious neighbours. The first consequence is that the probability a picks a malicious node as the first intermediate node in one of its tunnels is significantly higher than in the average case. Since controlling the first intermediate node is a necessary requirement for the adversary to break the anonymity, this would significantly increase his chances. To avoid this problem, a should pick the first intermediate node only among those nodes to which a has established the virtual link itself. The second consequence is that if a should append the next hop to a tunnel of another node, it will offer many malicious nodes in the selection it sends back to the initiator in message 2 of Figure 5.5. Assuming malicious nodes offer mainly other malicious nodes in their selections, this implies the selection of this honest node looks very similar to one of a malicious node. However, this second consequence is not a big problem because if there are too many malicious nodes in the selection, the initiator will detect this with high probability and reject the tunnel (see Section 5.6.3). Nevertheless, we will use the policy that a node only uses

those nodes as first intermediate nodes or in selections it offers to which it has established virtual link itself.

5.5.4 Why Relaying Data for Other Nodes is Good

Many peer-to-peer systems suffer from the "free rider" problem [39]. Especially in popular file-sharing systems, most users only consume but do not provide the files they downloaded to others. The main problem is that there is no real incentive to offer content to others because everything is for free and the systems seem to work well enough even if most users are free riders. Solving the free rider problem by technical means is very difficult and proposals to do so include micropayments and reputation systems, both of which have been discussed in the context of the Free Haven project (see Section 3.3.4).

MorphMix suffers from the same problem. Nodes can simply choose not to relay data of others by never accepting virtual links being established to them. This poses a problem because assuming that, say, only 10% of all nodes are relaying data of others, the load on them could get quite high and the performance may suffer. However, the advantage of MorphMix compared to other peer-to-peer systems is that MorphMix provides an incentive to relay the data of others. This has to do with the plausible deniability property of MorphMix (see Section 5.3). Recalling our discussion about the requirements for an attacker to break the relationship anonymity between an initiator and a server, we concluded that it is not always trivial for the adversary to learn whether the initiator of a tunnel is really the initiator of a tunnel or merely relaying the data for another node. However, if a node a is a free rider, an adversary (or in general any other node) can learn about this by trying to establish virtual links to a. If this always fails or if a never accepts relaying tunnels, it can be concluded with very high probability that all data sent or received by a belong to tunnels of which a is the initiator. We therefore conclude a node increases his protection from attacks by relaying data for other nodes.

5.6 Collusion Detection Mechanism

In this section, we present the collusion detection mechanism, which is the second major component of MorphMix. We first describe its principal idea by describing the concept of the *correlation* and the *correlation distribution* and

give a proof of concept. Then we discuss the dependence of the mechanism of different parameters and provide reasonable values for them. Finally, we examine how a node can make use of the correlation distribution to detect anonymous tunnels that contain many malicious nodes with high probability.

5.6.1 Correlation and Correlation Distribution

The collusion detection mechanism makes use of the fact that if a colluding set of malicious nodes want to control many nodes in an anonymous tunnel, they have to offer many or only malicious nodes in their selections. Corresponding to honest nodes, we name the selections they offer *honest selections* and the selections from malicious nodes *malicious selections*. Note that the initiator always learns which node has offered what selection by inspecting the messages during the anonymous tunnel setup (see messages 2 and 10 in Figure 5.5).

We identify the combination of a selection and the node that has offered the selection with *extended selection* (ES). Like with selections, we distinguish between *honest extended selections* and *malicious extended selections* depending on whether the selection has been offered by an honest or malicious node. A node remembers the extended selections it has accumulated during the setup of anonymous tunnels in a *extended selections list* L_{ES}.

An extended selection does not contain IP addresses, but the corresponding 16-bit IP address prefixes. If $\mathrm{prefix}_{16}(\mathrm{ip})$ gives the 16-bit IP address prefix of an IP address ip and if node b has offered the selection $\{\mathrm{ip}_c, \mathrm{ip}_d, \mathrm{ip}_e\}$, the resulting extended selection is $\{\mathrm{prefix}_{16}(\mathrm{ip}_b), \mathrm{prefix}_{16}(\mathrm{ip}_c), \mathrm{prefix}_{16}(\mathrm{ip}_d), \mathrm{prefix}_{16}(\mathrm{ip}_e)\}$.

For each new extended selection, a node computes the *correlation* by comparing it to all extended selections stored in the extended selections list L_{ES}. The idea bases on the assumption that honest nodes pick the nodes they offer in their selections more or less randomly from the set of all nodes. Consequently, there is no subset of nodes that appear more frequently together in honest extended selections than others, which should result in a small computed correlation of the nodes in a honest extended selection. Conversely, since malicious nodes are expected to offer mainly other malicious nodes in their selections, malicious nodes tend to show up together in extended selections, i.e. there is a stronger correlation among them. Consequently, a malicious extended selection should result in a larger computed correlation than an extended selection that contains nodes that are more randomly selected

from all MorphMix nodes. The correlation of a new extended selection is computed according to Algorithm 1:

Algorithm 1 *Computing the correlation of a new extended selection*

1. *Build a set ES_N consisting of the 16-bit IP address prefixes of the nodes in the new extended selection.*
2. *Define a result set ES_R which is empty at first.*
3. *Compare each extended selection ES_L in the extended selections list L_{ES} with ES_N. If ES_N and ES_L have at least one element in common, add the elements of ES_L to ES_R.*
4. *Count each occurrence of elements that appear more than once in ES_R and store the result in m.*
5. *Count the number of elements that appear only once in ES_R and store the result in u.*
6. *Compute the correlation c which is defined as $c = \frac{m}{u}$ if $u > 0$, or ∞ otherwise.*

We argue that based on the assumption that honest nodes offer nodes from a wide variety of all /16 subnets that contain MorphMix nodes, this correlation is in general large if the new extended selection contains many or only colluding malicious nodes and small otherwise. The reasons are that malicious nodes (1) select other malicious nodes with high probability and (2) are selected by other malicious nodes with high probability. This follows from our assumption that attacks by a cooperating malicious set of nodes are most likely. Similarly, honest nodes (3) pick nodes for the selections they offer from the set of all other nodes and (4) are picked by all other honest nodes. In step 3 of Algorithm 1, we want to find out what the nodes in the same /16 subnets as those in the new extended selection have done before, i.e. in what extended selections they have appeared before. Therefore, we collect all elements of those extended selections in L_{ES} that contain at least one element of the new extended selection in a set ES_R. Recalling our threat model (see Section 5.4) where we stated that any adversary can only control nodes in a limited number of all public /16 subnets, assuming that there are honest nodes in a much larger number of /16 subnets, and for reasons (1–4) given above, we can state the following properties about the set ES_R:

1. If the new extended selection ES_N mainly consists of the /16 subnets of *malicious nodes*, ES_R will contain /16 subnets from only a *small* fraction of all public /16 subnets that contain MorphMix nodes. In

addition, most of the /16 subnets in ES_R are present several times in the set. According to Algorithm 1, this implies a large m and a small u, and since $c = m/u$, the resulting c is large.

2. If the new extended selection ES_N mainly consists of the /16 subnets of *honest nodes*, ES_R will contain /16 subnets from a *large* fraction of all public /16 subnets that contain MorphMix nodes. In addition, most of the /16 subnets in ES_R are present once or only a few times in the set. According to Algorithm 1, this implies a small m and a large u, and since $c = m/u$, the resulting c is small.

Simply counting how many times the /16 subnets in ES_N show up in L_{ES} does not work. Although subnets with malicious nodes may show up in extended selections more frequently on average because malicious nodes offer many or only other malicious nodes in their selections, many /16 subnets consisting of only honest nodes will also show up frequently, either because there are many nodes in them or because some nodes are very popular because they have a lot of bandwidth and computing power to spare. In addition, there is an attack the adversary could exploit if simply counting the occurrences of /16 subnets were used that we will point out in Section 6.6.

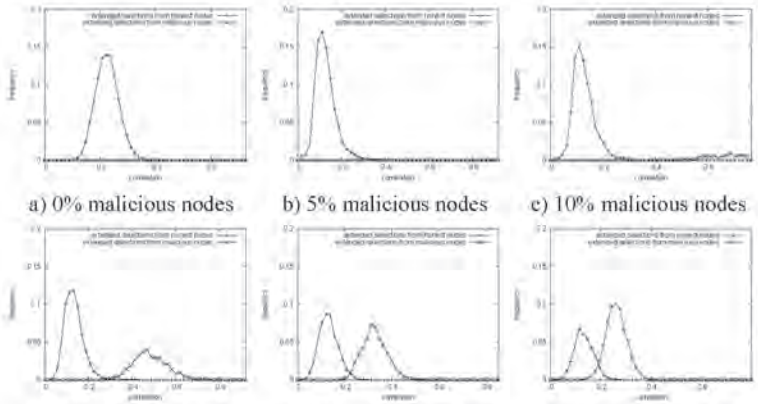
Note that the complexity to compute the correlation of a new extended selection is proportional to the number of extended selections in L_{ES}. For scalability reasons, we cannot keep all extended selections in L_{ES} forever. Rather, we "forget" old extended selections and to keep only the k *most recently received extended selections* in L_{ES}. We will talk in Section 5.6.2 about reasonable values for the number of extended selection in L_{ES}.

A node remembers the correlations it has computed over time and represents them as a *correlation distribution*. In our MorphMix prototype, this is implemented as an array with 50 slots[6], whereas each slot of the array corresponds to a particular discrete correlation. If a new correlation c is computed, it basically affects the slot closest to c by incrementing its value by one. However, in order not to let grow the values in the array indefinitely, they follow an exponential weighted moving average (EWMA) with parameter α. α is slightly larger than zero and depends on the number of extended selections in L_{ES}: if k_{ES} is the number of extended selections in L_{ES}, then $\alpha = 1/k_{ES}$. After a new correlation has been computed, the value in each slot is first multiplied with $(1 - \alpha)$, and α is added to the value in the slot that corresponds to the new correlation. For details about the implementation of

[6] Analyses with our node simulator (see Section 6.1.1) have shown that increasing the number of slots beyond 50 does not provide better results

the correlation distribution, refer to the MorphMix prototype implementation (see Appendix A.7).

As a proof of concept, we analyse how the correlation distribution looks using our node simulator (see Section 6.1.1). We assume a system with 10000 nodes, where some of them are malicious and in the same colluding set. All 10000 nodes are in different /16 subnets and every node has the same probability of being offered in a honest selection. We set up 5000 anonymous tunnels, whereas each tunnel consists of five nodes in total, which is a reasonable choice for the tunnel length (see Section 8.3.7). This means that the initiator gets three different selections during the setup of each tunnel, one from each of the intermediate nodes. Each selection contains 14 nodes, which is a reasonable selection size in a system with nodes in 10000 different /16 subnets (see Section 5.6.2). For now, we assume that malicious nodes offer only other malicious nodes from their collusion in their selections, i.e. malicious selections contain 14 malicious nodes. Figure 5.6 shows the correlation distribution when 0, 5, 10, 20, 30, and 40% of all nodes are malicious.



a) 0% malicious nodes b) 5% malicious nodes c) 10% malicious nodes

d) 20% malicious nodes e) 30% malicious nodes f) 40% malicious nodes

Figure 5.6: *Correlation distribution with 10000 nodes.*

We can see the contributions of honest and malicious nodes to the correlation distribution. In general, this results in two peaks, one on the left from the honest nodes and one on the right from the malicious nodes. The more malicious nodes there are in the system, the bigger the right peak gets and

the closer the two peaks move together. Remembering that each node is in a different /16 subnet, this also means that the larger the fraction of /16 subnets that contain malicious nodes, the bigger the right peak gets and the closer the two peaks move together.

5.6.2 Selection Size and Size of Extended Selections List

Our initial analysis [97] have shown that the shape of the correlation distribution depends on the selection size and the number of extended selections in L_{ES}. In general, both a larger L_{ES} and a larger selection size help to separate the peaks in the correlation distribution, but it has its limits. On the other hand, increasing the sizes also means that the time to compute the correlation and the memory requirements to store the extended selections list grow (see Section 5.8). Using analyses based on our node simulator (see Section 6.1.1), we have derived reasonable values for both sizes. They depend on the number of different /16 subnets that contain nodes. If s is the number of different /16 subnets that contain nodes, the selection size n_{sel} to be used is given by:

$$n_{sel} = \max(3, \lceil 7.75 \cdot \log_{10} s - 17 \rceil) \tag{5.1}$$

The selection size is logarithmically dependent on the number of different /16 subnets that contain nodes. Since there are only 56559 different /16 subnets, there is an upper bound for n_{sel}, which is given by $n_{sel,max} = (\lceil 7.75 \cdot \log_{10} 56559 - 17 \rceil) = 20$. So the selection size is always between three and 20. The size of L_{ES} is also dependent on the number of different /16 subnets that contain nodes. If $\overline{n_{sel}}$ is the average number of nodes in a selection, the number of extended selections k_{ES} in L_{ES} is given by:

$$k_{ES} = \lceil 2 \cdot \frac{s}{\overline{n_{sel}}} \rceil \tag{5.2}$$

Like for the selection size, there is also an upper bound for the size of L_{ES}, which is given by $k_{ES,max} = \lceil 2 \cdot 56559/20 \rceil = 5656$. So if there are noes in all possible /16 subnets, the list contains 5656 extended selections with 21 nodes each, and the list won't grow any further.

Note that when a node joins MorphMix for the first time, it does not know how many nodes there are in the system, i.e. it does not know what value to

use for s in (5.1) and (5.2). Consequently, the node starts with the minimum selection size of three, but tries to estimate the actual number of different /16 subnets that contain nodes by observing how frequently a new extended selection has elements in common with extended selections in L_{ES}. If there are only a few different /16 subnets that contain nodes, this frequency will already be high after only a few tunnels have been set up and vice versa. For more details about estimating the number of /16 subnets that contain nodes, refer to the MorphMix prototype implementation (see Appendix A.7).

To summarise, there is an upper limit on both the selection size and the size of the extended selections list. We carried out some performance tests on a system with a 1GHz AMD Athlon CPU, 256 MB RAM, running Linux as operating system with a 2.4.17 kernel. With both selection size and size of the extended selections list set to their maximum values, it takes about 50 ms to compute the correlation of a new extended selection. Assuming a node sets up a tunnel every two minutes (see Section 8.2.3) and a tunnel has a reasonable length of five (see Section 8.3.7), which means three correlations must be computed per tunnel, the computational overhead resulting from computing the correlations is only about 0.125% on the above-mentioned system, which can be neglected.

5.6.3 Detecting Malicious Tunnels

Based on our discussion in Section 5.3, we say that a tunnel is *malicious* if the adversary controls both the first intermediate and the final node in this tunnel. We also say such a tunnel is *compromised*. Otherwise, the tunnel is considered as *good*. Looking at the correlation distributions in Figure 5.6, the strategy a node follows to detect malicious anonymous tunnels is as follows: At any time, the node knows the correlation distribution it has generated based on selections it received previously. Based on this distribution, the node determines a *correlation limit*, which should have the property that if the correlation of a new extended selection is smaller than this limit, then the node that offered the corresponding selection is honest with a high probability. Similarly, the extended selection corresponding to the selection from a malicious node should yield a correlation that is above the limit with high probability. As an example, using the correlation distribution in Figure 5.6(d), a correlation limit of 0.28 would be reasonable. In general, the correlation limit is not a fixed value but depends on the system size and the percentage of malicious nodes. For instance, the correlation limit for Figure 5.6(e) should be about

0.2 instead of 0.28. The difficulty of determining this limit is that the initiator only knows the correlation distribution of all nodes, i.e. the sum of the contributions of honest and malicious nodes in Figure 5.6. Furthermore, as we will see in Chapter 6, the peaks cannot always be separated so clearly as in Figure 5.6 and we will explain in Section 6.2.1 how the correlation limit is determined in practice. For now, we simply assume the initiator can determine a reasonable correlation limit based on the correlation distribution. If the correlations of all extended selections of an anonymous tunnel are below that limit, then the initiator considers the anonymous tunnel as *good*. But if the correlation of at least one extended selection is above the limit, it is assumed the node that has offered that selection is malicious. Consequently, the initiator considers the tunnel as *malicious* and it will not be used to contact a server anonymously. Note that this decision is made *before* anonymous connections to contact servers are established within the anonymous tunnel, so in case a tunnel is indeed malicious and rejected, the adversary cannot learn anything to break a users anonymity. Note also that if only the final node in the tunnel is malicious, then this is difficult to detect because it does not offer a selection. However, this final node cannot learn anything about the anonymous tunnel only by itself.

The steps the initiator carries out during the setup of an anonymous tunnel to determine whether it is considered good or malicious are listed in Algorithm 2:

Algorithm 2 *Determining if an anonymous tunnel is good or malicious*

1. *Initialise a variable* `rejectTunnel` *to false.*
2. *Get the next extended selection* ES_N *of the anonymous tunnel.*
3. *Compute the correlation* c *of* ES_N.
4. *Determine the correlation limit* c_l *from the correlation distribution.*
5. *If* c *is greater than* c_l, *set* `rejectTunnel` *to true.*
6. *Add* c *to the correlation distribution and add* ES_N *to the extended selections list.*
7. *If there are more intermediate nodes following in the tunnel, go to step 2.*
8. *If* `rejectTunnel` *is true, reject the tunnel. Otherwise it is considered good.*

One might wonder why we reject the tunnel if *any* of the computed correlations is above the correlation limit and not only consider the extended

selection offered by the first node. The latter makes sense because a tunnel is only malicious if the adversary controls both the first intermediate and the final node. However, based on our analyses with the node simulator it has turned out that the probability to detect malicious tunnels is higher if all extended selections are tested and not only the first. The reason is that in case the first malicious extended selection was not detected and the second intermediate node is also malicious, there is another chance by examining the malicious extended selection from the second intermediate node and so on. On the downside, examining all extended selections also means the probability a tunnel that is not compromised is rejected slightly increases, but the increased chances to detect compromised tunnels outweighs this disadvantage.

5.7 Peer Discovery Mechanism

In this section, we present the peer discovery mechanism that enables MorphMix nodes to learn about other nodes. The peer discovery mechanism is the third major component of MorphMix.

In general, resource discovery is an important and fundamental task that must be solved in every self-organising system, which includes peer-to-peer systems. Often, especially in peer-to-peer file-sharing systems, finding a resource is equivalent to locating one or more peers that store a particular file (or a part of the file). In MorphMix, however, the case is different because there are no resources in the sense of files to discover. Rather, the other nodes themselves are the resources. Therefore, all a MorphMix node needs to do is learning about other nodes that can be used as new neighbours or witnesses. In MorphMix, there are two different types of peer discovery: initial and continuous peer discovery. The first one is used by a node to join MorphMix for the first time. The second type of peer discovery happens all the time while a node is participating in MorphMix. We first discuss these two types of peer discovery. Afterwards, we describe how the information about other nodes is organised internally and accessed to select nodes as new neighbours or witnesses.

5.7.1 Initial Peer Discovery

The goal of initial peer discovery is to quickly learn about a few other nodes when a node joins MorphMix for the first time. Once a node knows some

other nodes, it can start establishing virtual links to them and set up anonymous tunnel, which directly leads to continuous peer discovery described below.

Of course, there are always offline methods (not part of MorphMix) such as the Usenet or the Web to learn about other peers. But these approaches are cumbersome to use because they cannot be integrated well into MorphMix. In addition, one can imagine methods based on portscans, address resolution protocol (ARP) [85] broadcast, or IP multicast [27] to discover other nodes, but such methods may take a long time to discover a node (portscans), are of limited reach (ARP broadcast), or are based on technology that is not widely deployed (IP multicast). Consequently, MorphMix itself offers a way to easily learn about other nodes when a node knows about at least one other node. To do so, the MorphMix protocol includes peer discovery messages (see Appendix A.3.3), which allow a node to ask another MorphMix node for information about other nodes, i.e. their IP addresses, ports on which they are listening for connection requests, public keys, and node levels [7].

To facilitate joining for nodes that do not know any other node, there are "official" *introductory nodes* in MorphMix. The contact information of these nodes is included in every distribution of the MorphMix program, and different distributions may contain different introductory nodes. These nodes are basically just MorphMix nodes that are always participating and the method to query them for information about other nodes is the same as described above. The user has the choice to make use of these official introductory nodes and the trust she puts in them depends on the trust she puts in the capability of the developers of a distribution to pick only honest nodes as official introductory nodes. We agree that that the approach with introductory nodes is merely a "hack" to solve the bootstrapping problem, but there is simply no alternative if we want to offer a built-in way to easily join MorphMix if no other node is known. Users that do not trust the introductory nodes can always choose offline methods to learn about other nodes. Another potential problem with introductory nodes is that they are the only centralised (even if there are several of them) component in the otherwise completely distributed MorphMix system and provide therefore a potential point of attack for legal attacks.

One general problem with querying other noes is that if the node that is contacted is malicious, it will inform only about other malicious nodes and

[7] The concept of different node types that have different node levels will be introduced in Section 7.3.2 and is specified in more detail in Appendix A.2.3

these nodes again inform only about malicious nodes and so on, which means a node may end up in a completely malicious MorphMix subsystem where it is the only honest node. To minimise this risk, multiple nodes should be queried, which increases the probability that there is at least one honest node among them.

Querying other nodes is usually only needed right after having joined MorphMix for the first time. A returning node that has been participating before has usually accumulated so much information about other nodes during the continuous peer discovery that other nodes that are currently participating can be easily found. Nevertheless, if it happens at any time that a node knows too few other nodes, it can always query other nodes again, but this is very unlikely to happen considering the continuous peer discovery mechanism described below.

5.7.2 Continuous Peer Discovery

While participating in MorphMix and setting up anonymous tunnels, a node learns about a variety of other nodes. Every selection it gets contains the IP addresses, ports, public keys, and node levels of several nodes. This gives the initiator all necessary information to contact new nodes to establish virtual links to them, or to select a witness to append a node to its own tunnels.

In addition, a node always includes its own node information when establishing a virtual link to another node (see Appendix A.3.1). This is necessary because otherwise, a node could never inform other nodes about itself and consequently, no other node would establish a virtual link to it. According to our policy on using virtual links (see Section 5.5.3), this implies it would never be chosen as intermediate node in tunnels of other nodes and could therefore never relay data for other nodes.

5.7.3 Organising and Accessing Information about other Nodes

The information a node learns about other nodes is stored in an internal data structure. One possibility would be to use a simple list, where each list element corresponds to a node. However, we have seen in Section 5.6 that the collusion detection mechanism is based on the assumption that honest nodes pick the nodes they offer in their selections from a wide variety of /16 subnets

that contain MorphMix nodes. To avoid that an honest node offers always the same set of nodes in its selections, and since these nodes are selected from the set of its current neighbours, honest nodes must change their neighbours from time to time. To do so, a newly established virtual link to a new neighbour is only kept for a limited time of 30 minutes (see Appendix A.5.1). After this lifetime, the virtual link is not simply torn down because there may still be tunnels using it, but the node at the other end of the virtual link is no longer advertised in selections. Consequently, we should organise the data structure in a way that easily supports picking nodes from a wide variety of /16 subnets.

Selecting the neighbours from a large range of /16 subnets has an additional benefit. To break the relationship anonymity between initiator and server, it is a necessary requirement for the internal adversary to control the first intermediate node. Therefore, selecting the neighbours (and therefore potential first intermediate nodes) from a wide variety of all /16 subnets reduces the probability the adversary controls this node because of the assumption that the adversary can only control nodes in a limited number of all public /16 subnets (see Section 5.4).

The data structure to store the information about other nodes is implemented as follows: the initiator remembers the nodes it has received in selections in a *node lookup list*. There is at most one entry in the list per /16 subnet, which implies that this list has at most 56559 entries. Each entry contains the corresponding 16-bit IP address prefix and a *list of nodes* that contains the information about nodes in this subnet. Each entry in a list of nodes contains itself the IP address, port, public key, and node level of the corresponding node. Figure 5.7 illustrates the basic idea of the node lookup list and the principal concept of how nodes are inserted into and selected from the list. Note that Figure 5.7 only sketches the basic concept. In particular, the cases of inserting a node (b, c, and e) correspond to nodes that have been learned as part of selections offered by other nodes. For simplicity, we only show the IP addresses of the nodes in the lists of nodes and leave out their public keys, port numbers, and node levels.

In this example, we assume that in the beginning, the list contains three different subnets and four nodes, as illustrated in Figure 5.7(a). 129.132 is the 16-bit IP address prefix of the first /16 subnet in the list, there are two nodes for the /16 subnet 8.245 and one node for the two other subnets. Now we assume that the initiator learns about a new node with IP address 132.101.23.66. Figure 5.7(b) shows how the information about the new node is processed: since the 16-bit IP address prefix (132.101) corresponding to the new node is

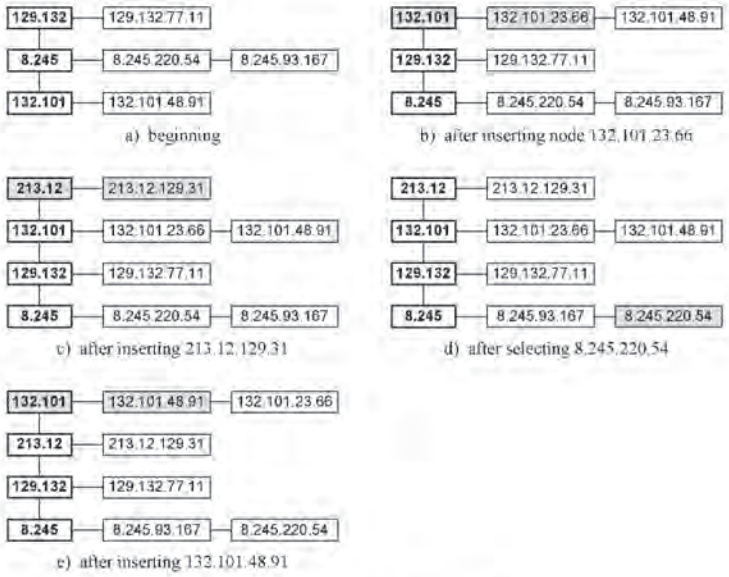


Figure 5.7: *Node Lookup list.*

already in the node lookup list, the node is inserted *at the first position* in the list of nodes of this /16 subnet. In addition, the entry for the subnet is moved *to the first position* of the node lookup list. Figure 5.7(c) shows how another node 213.12.129.31 is inserted into the list: this time, the corresponding 16-bit IP address prefix (231.12) is not yet in the list. Therefore, a new /16 subnet is inserted *at the first position* in the node lookup list with the new node as the single entry in the corresponding list of nodes. Figure 5.7(d) shows what happens if the node lookup list is accessed to select a node from the /16 subnet identified with 8.245: the first element in the corresponding list of nodes, 8.245.220.54 is returned and moved *to the last position* of the list of nodes. Finally, Figure 5.7(e) illustrates when a node 132.101.48.91 that is already in the node lookup list is inserted: the old entry is removed from the list of nodes, the information about the new node is inserted *at the first position* in the list of nodes, and the subnet is moved to the first position of the node lookup list.

For practical reasons, the information about at most ten different nodes is stored in the list of nodes of a subnet, which implies a node knows of at most ten nodes in the same subnet at any time. If information about a new node is learned as part of a selection, the corresponding list of nodes already contains ten nodes, and the new node is not yet in the list, then the last entry of the list of nodes is simply discarded before the new node is inserted.

Organising the node lookup list in this way has two properties: (1) the nodes belonging to the same /16 subnet are ordered in the corresponding list of nodes such that the more recently the information about a node has been received as part of a selection, the closer to the first position it is, and (2) the subnets themselves are ordered such that the more recently the information about a node has been received, the closer the corresponding subnet is to the top of the node lookup list.

Whenever the initiator wants to contact a new neighbour, it randomly picks a subnet from the node lookup list. Then it picks the first node in the corresponding list of nodes and moves it to the last position, as illustrated in Figure 5.7(d). It then tries to establish a virtual link to this node. If the node cannot be contacted or the virtual link can not be established for any reason, the information about the node is *removed* from the corresponding list of nodes and the next node is tried in the list of nodes of the same subnet. If this fails for all nodes in the list of nodes of the subnet, the subnet itself is removed from the node lookup list and another subnet is tried. This procedure guarantees that every honest node picks other nodes from a wide spectrum of /16 subnets, and this is exactly what we wanted to achieve.

Since the /16 subnets corresponding to nodes that have been received recently are moved to the top of the node lookup list, picking a node from a subnet that is close to the top of the list guarantees with high probability that the node is currently participating in MorphMix. This is not critical for nodes that should be contacted as new neighbours because the initiator can simply try another node when a timeout occurs during the connection attempt. However, the initiator also must select witnesses from the node lookup list. To make sure that a high percentage of the attempts to set up an anonymous tunnel succeed, it is desirable that the witnesses the initiator selects are currently participating in MorphMix with high probability. Witnesses should therefore be picked from a subnet that is close to the top of the node lookup list. Note that like with nodes that are selected as neighbours, the information about a node that has been selected as a witness is removed from the list of nodes if it cannot be contacted during the setup of an anonymous tunnel.

The nodes in newly arriving selections are only inserted into the node lookup list if the corresponding computed correlation is not above the correlation limit. This means we have combined the peer discovery and collusion detection mechanisms to minimise the number of malicious nodes in the node lookup list. This is an important property because for the adversary to compromise a tunnel, he must necessarily control the first intermediate node. Since an initiator picks the first intermediate node from the set of its current neighbours, which are selected from the node lookup list, the adversary should make sure that many malicious nodes are present in the node lookup lists of honest nodes to increase his chances to control the first intermediate node. To achieve this, he should include many or only malicious nodes in the selections his nodes offer. But since the collusion detection mechanism detects malicious selections that contain many malicious nodes with high probability (see Chapter 6), the adversary cannot advertise malicious nodes as aggressively as he would like.

We mentioned that Figure 5.7(b, c, and e) correspond to the case of inserting a node that has been learned as part of a selection. However, information about other nodes are also learned when other nodes establish virtual links to the own node or when other nodes are queried. Since these methods of learning about other nodes are not coupled to the collusion detection mechanism, it is easy for the adversary to force the information about the nodes he controls into the node lookup lists of honest nodes. In particular, if inserting nodes into the node lookup list were always done as in Figure 5.7, frequently establishing virtual links to a honest node would allow the adversary to move the information about malicious nodes to the first positions of the subnets where he controls nodes. Consequently, we employ a different strategy when information about other nodes is not learned as part of a selection. The idea is that an adversary that operates several nodes in a /16 subnet that also contains honest nodes should not be able to continuously put a malicious node into the first position of the corresponding list of nodes of an honest node by establishing virtual links to the honest node. On the other hand, the strategy should still make it possible for an honest node that has joined MorphMix and that wants to relay data for other nodes to disseminate its contact information by establishing virtual links to other honest nodes. The strategy to insert information about nodes that have *not* been received as part of selections is as follows:

1. In general, inserting this information never moves the corresponding subnet to the first position in the node lookup list.

2. If the corresponding subnet is not yet in the list, it is inserted *at the last position* of the node lookup list with the information about the new node as the only entry in the list of nodes.
3. If the subnet is already in the node lookup list, the information about the new node is not in the corresponding list of nodes, and the list of nodes contains fewer than ten elements, the information about the new node is inserted at the *first free position*, i.e. at the end of the list of nodes.
4. If the subnet is already in the node lookup list and the information about the new node is already in the corresponding list of nodes, the old entry is *replaced* with the information about the new node.
5. If the subnet is already in the node lookup list, the information about the new node is not in the corresponding list of nodes, the list of nodes contains ten elements, and there is at least one node in the list of nodes that is located in the same /24 subnet as the new node (i.e. the IP addresses of the two nodes have the same 24-bit prefix), the *last of these entries is replaced* with the information about the new node.
6. If the subnet is already in the node lookup list, the information about the new node is not in the corresponding list of nodes, the list of nodes contains ten elements, and there is no node in the list of nodes that is located in the same /24 subnet as the new node, then the *last entry* in the list of nodes is *replaced* with the information about the new node.

Analysing this strategy more closely and looking at a single /16 subnet that contains honest and malicious nodes, we can see that as long as the list of nodes of an honest node contains fewer than ten entries, the adversary (1) cannot remove honest nodes from the list of nodes and (2) cannot insert the information about malicious nodes at the first position in the list of nodes (unless the subnet has not been in the node lookup list before). Consequently, the adversary can only make sure honest nodes store the information about malicious nodes in their list of nodes, but he cannot enforce honest nodes to select a particular malicious node more frequently than any other node in its list of nodes. If we again look at a single /16 subnet, assume that the corresponding list of nodes of an honest node contains ten entries, and there are honest nodes in different /24 subnets, we can state that an adversary must either control nodes in the same /24 subnets as the honest nodes to effectively remove all of them from the list of nodes, or he must control nodes in many different (at least ten) /24 subnets to have a chance to introduce many or only malicious nodes in the list of nodes. In the second case, however, it is not

possible to remove honest nodes that are close to the first position in the list of nodes because information about new nodes is never inserted at the first position.

On the other hand, an honest node a that has joined MorphMix for the first time can effectively disseminate its contact information. By contacting other nodes, it is inserted into their node lookup lists. Since this information cannot easily be removed by the adversary, node a will eventually be contacted by honest nodes and offered in selections to other nodes, which tells these other nodes about node a and so on. Note that especially if the system is large, it may take a while until node a is contacted regularly by other nodes because before a is offered in selections, it is never inserted at the first position of lists of nodes (unless the corresponding /16 subnet has not been in the node lookup list before). However, once a is being offered in selections, it is inserted at the first position of the lists of nodes and the contact information about a is spread quickly.

Note that the strategy described above fails if the adversary controls nodes in very many different /24 subnets of a /16 subnet. In this case, the adversary should be able to insert mainly malicious in the corresponding lists of nodes of honest nodes. The countermeasures are either increasing the length of the lists of nodes beyond ten, or using shorter prefixes than 24 bits, for instance /20 subnets instead of /24 subnets. However, longer lists of nodes means it takes even longer until the information about a new node is disseminated if there are already several nodes in the /16 subnet and the memory requirements for the node lookup list also increases (see Section 5.8.1). Shorter prefixes, on the other hand, increase the probability the adversary can overwrite entries of honest nodes in a list of nodes. Using /24 subnets is a good compromise because it hinders an institution owning a class C network (which is quite common) from inserting significantly more than one of the nodes it controls into the lists of nodes of honest nodes if there are several honest nodes in the /16 subnet.

There is a possible attack on MorphMix that exploits the fact that a node may eventually directly connect to a node that it received earlier as part of a selection. We will analyse the impact of this attack in Section 6.5.

5.8 Scalability and Requirements to Run a Node

MorphMix aims at providing anonymity for the masses and should therefore scale well up to a large number of nodes. In this section, we first state why MorphMix indeed scales very well and why any modern personal computer with a dial-up Internet connection is sufficient to run a node. We also analyse the possibilities for users with computers that are located behind NAT gateways and the influence of dynamic IP addresses.

5.8.1 Scalability and General Requirements

The complexity of all three core components of MorphMix grows as the number of nodes increases. The most critical parameters that affect scalability in MorphMix regarding to these three components are the following:

1. **Collusion detection mechanism**:
   - the selection size
   - the size of the extended selections list
   - the complexity to compute the correlation
2. **Anonymous tunnel setup**:
   - the length of message 2 in Figure 5.5
   - the number of neighbours a node must have at any time
   - the computational overhead imposed by public-key cryptography operations
3. **Peer Discovery**:
   - the memory requirements to store the node lookup list

In general, the key to scalability in MorphMix bases on the fact that all these critical parameters depend on the number of different /16 subnets that contain MorphMix nodes and not on the total number of nodes. As a result, we can expect the complexity to grow slower than the number of nodes gets larger because if many nodes that are already participating in MorphMix, it gets less likely that new nodes reside in a "new" /16 subnet that does not yet contain a node. But even more important, the complexity for these parameters has an upper bound because the number of public /16 subnets is limited to 56559 (see Section 5.4.2). Consequently, if we can show that MorphMix can cope well with an environment with nodes in all public /16 subnets, then we can expect MorphMix to be able to handle as many nodes as there are public IP addresses.

We first analyse the collusion detection mechanism. According to (5.1), the selection size grows logarithmically with the number of /16 subnets. Using the maximum number of /16 subnets, the selection size reaches its maximum of 20. Similarly, the size of the extended selections list grows linearly with the number of /16 subnets and reaches its maximum size with 5656 entries with 21 IP addresses each according to (5.2). This corresponds to less than 0.5 MB memory space, which is hardly an issue for state-of-the-art computers. According to Algorithm 1, the computational overhead to compute the correlation of a new extended selection grows linearly with the number of /16 subnets, and we have already stated in Section 5.6.2 that with nodes in all /16 subnets, this takes about 50 ms on a system with a 1GHz AMD Athlon CPU. Assuming an average tunnel length of five and even if a node sets up significantly more tunnels than one every two minutes because tunnels may fail during the setup or may be rejected according to Algorithm 2, the computational overhead resulting from computing the correlations is well below 1% on the above-mentioned system, which is insignificant. We therefore conclude that the overhead from the collusion detection mechanism is small and can easily be handled by virtually any personal computer that is in use as of November 2003.

The selection size determines the number of nodes a node must offer in its selections and therefore it also determines the minimal number of neighbours a node must have at any time. As a result, this minimal number of neighbours also grows logarithmically with the number of /16 subnets and has an upper limit of 20. Similarly, the selection size determines the amount of data exchanged during anonymous tunnel setup. Consequently, message 2 in Figure 5.5 contains the information of at most 20 different nodes that are offered to the initiator. For now, we can only state that since there is an upper limit for the minimal number of neighbours and the amount of data exchanged during the anonymous tunnel setup, the data overhead is so small that even dial-up Internet connections are sufficient to participate in MorphMix. A detailed analysis of this data overhead will be provided in Section 8.3. Similarly, we state that since the data overhead is small, the computational overhead imposed by public-key cryptography operations that are used to establish virtual links and in general during the anonymous tunnel setup can be well handled by reasonably modern personal computers, and we will also analyse this in more detail in Section 8.3.

Finally, the length of the node lookup list grows linearly with the number of /16 subnets that contain nodes. Consequently, its maximum length is

56559. Every entry in this list contains four bytes for the IP address, two bytes for the port on which the MorphMix node is listening, 256 bytes for the RSA modulus, and one byte for the node level. There may be up to 10 entries per /16 subnet and consequently, the maximum size of the node lookup list is about 150 MB. Since this data structure is stored in memory during the time a node is participating in MorphMix, this is not insignificant. However, it can well be handled by modern personal computer systems that are usually equipped with at least 256 MB RAM as of November 2003. In addition, there is always the possibility to reduce the number of entries per /16 subnets to make the list smaller. Reducing it to two entries per subnet, for example, brings down the memory requirements of the extended selections list to 30 MB.

Summarising this discussion concerning the computational, memory, and bandwidth requirements depending on the number of nodes, we conclude that MorphMix scales indeed very well and can handle as many nodes as there a publicly available IP addresses. The computational requirements are modest, although we have yet to show that the overhead imposed by the public-key cryptography operations is indeed small (see Section 8.3). The memory requirements of the peer discovery mechanism are dominant and imply that it may not yet be possible to run a MorphMix node on a handheld computer. On the other hand, the requirements are comparable with other typical applications such as office packages or graphical web browsers and we therefore conclude that any state-of-the-art personal computer can run a MorphMix node. Bandwidth requirements are also modest and even dial-up connections are sufficient to participate, and we will give a detailed analysis of this claim in Section 8.3.

5.8.2 NAT Gateways and Dynamic IP Addresses

So far, we have assumed all MorphMix nodes have public static IP addresses. What remains to be discussed is how participating in MorphMix is possible for users with a computer that is located in a private network behind a NAT gateway and the influence of dynamic IP addresses.

We first look at users that access the Internet through a NAT gateway. One approach is to simply run the node on a computer behind the NAT gateway and access MorphMix in the same way nodes with public IP addresses do. This allows to establish anonymous tunnels via other nodes, but the own computer cannot be accessed by others because it is hidden behind the NAT

gateway. If the NAT gateway is under control of the user's ISP and the user's computer simply gets assigned a private IP address, there is nothing that can be done about this situation and the user cannot relay anonymous tunnels of other MorphMix user.

On he other hand, NAT gateways are also often used by administrators of home or small office networks that get assigned one public static IP address from their ISP. In this case, the NAT gateway is often under control of the user or the user has indirect control of it through the network administrator. Connecting to a computer behind the NAT gateway from computers outside the private network can now easily be enabled by using port forwarding, which is supported by virtually all NAT gateways, including (A)DSL/Cable routers. When establishing a virtual link to another node, a node behind a NAT gateway simply includes the public IP address and the corresponding port of the NAT gateway instead of its own IP address and port to disseminate its contact information (see Appendix A.3.1).

If multiple users with multiple computers in the same private network want to access MorphMix from behind the NAT gateway, one could simply extend the idea above by giving each MorphMix node a dedicated port such that it can be accessed from outside the NAT gateway. Note that this results in multiple nodes having the same public IP address, but different ports. However, since MorphMix identifies nodes with IP addresses, they are equal from the system's point of view. This can also be seen by recalling that the peer discovery mechanism *replaces* a node with the same IP address as a new node is inserted in the node lookup list. So for the other nodes, this looks like a node that changes its port and its key pair quite frequently, but they always only know about one of them. Although there is no problem to make use of such a configuration, there is a much more elegant and cleaner solution to access MorphMix from behind a NAT gateway if there are multiple users: by choosing one of the internal computers to run a single node, having all users access MorphMix through this single node, and granting access to the node from outside the NAT gateway by using port forwarding. Another option is to run the node directly on the NAT gateway. Since applications access their own MorphMix node like a proxy, there is no need for this application to run on the same computer as the MorphMix node. Another advantage of this scenario is that only a single computer must be kept up and running to be participating in MorphMix all time. A potential disadvantage of this approach is that the anonymity of users may be compromised by an attacker sniffing on the private network, but usually, a private network behind a NAT gateway can

be considered as trustworthy. In fact, this concept of using a single MorphMix node on a dedicated, powerful computer for multiple users in a trustworthy environment is an attractive option to participate in MorphMix for small companies, departments of larger companies, or institutes of a university.

The effect of dynamic IP addresses is that users get a different IP address after a period (e.g. 24 hours) expires or each time they connect their computer to the Internet. The effect is that nodes may get a new identity from time to time. One consequence is that when a node gets a new IP address, it cannot be accessed any longer by other nodes that remember this node under its former identity in their node lookup list. However, this is no problem because a node will easily detect this when trying to contact this node as a potentially new neighbour and simply pick another node. When picking such a node as a witness, the case is different, but selecting witnesses from the top of the node lookup list as discussed in Section 5.7.2 reduces the probability a node has changed its identity in the meantime. In general, nodes that change their IP addresses have the same effect as nodes that leave the system. Since MorphMix is designed to cope with disappearing nodes (see Sections 5.7 and 7.3.2), it can also deal with nodes that have dynamic IP addresses. As an interesting side note, it can be expected that nodes that get dynamic IP addresses always get them from the same /16 subnet. So for the collusion detection mechanism, such a node always looks the same.

We conclude that having a public static IP address is no requirement, because even users with computers that are located behind NAT gateways or that get dynamic IP addresses can run a node and participate in MorphMix.

5.9 An Outlook on IPv6

Although this thesis focuses on IP version 4 (IPv4), it is foreseeable that IPv4 will eventually be replaced with its successor IP version 6 (IPv6) [28], mainly because the IPv4 address space may become too small. In this section, we analyse the implications of MorphMix if IPv6 gets widely deployed and show that MorphMix should still work well on top of IPv6.

In this chapter, we have seen that MorphMix heavily depends on the IPv4 addressing structure and on the way IPv4 addresses are assigned to ISPs or institutions in general. This has directly lead to our choice of using the 16-bit prefix of IP addresses as the basis for the three core components of MorphMix. The two main reasons to use /16 subnets was that it bases on the

assumption that adversaries cannot control nodes in very many of all public /16 subnets and that it makes MorphMix scalable because the maximum number of /16 subnets is much smaller than the maximum number of public IP addresses. Consequently, the goal when moving to IPv6 is to organise IPv6 addresses in a similar way, i.e. to arrange them in "subnets" such that it is difficult for an adversary to run nodes in a significant fraction of them and that the total number of these "subnets" has a similar size as the number of /16 subnets in IPv4 to maintain scalability.

IPv6 addresses [106] are 128 bits long and consist of a 64-bit network part and a 64-bit host part. A trivial approach would therefore be to move from /16 subnets in IPv4 to /64 subnets in IPv6. However, this would result in a potentially very large number of different subnets and the resulting complexity of the three core components of MorphMix could not be handled by any computer in the foreseeable future. To find a more suitable solution, we must analyse how address allocation and assignment will be handled in IPv6 [4] in more detail.

The Internet Assigned Number Authority (IANA) manages the whole IPv6 address space. The address range 2000::/3 has been designated to be the global unicast address space in IPv6 [106]. Below IANA, there are the Regional Internet Registries (RIR) such as ARIN, APNIC, or RIPE NCC. IANA has allocated initial ranges of global unicast address space from the 2001::/16 address block to the existing RIRs. Below the RIRs, there are the Local Internet Registries (LIRs), which are usually ISPs. The LIRs get one or more /32 address blocks from their respective RIRs and end users get /48 address blocks from their LIR. /48 address blocks should be enough even for the largest companies, because it allows operating 2^{16} subnets with 2^{64} hosts each.

Address allocation in IPv6 is therefore much more structured than in IPv4. In IPv6, one can always state that all addresses within the same /32 address block have been allocated by one and the same ISP. This is not the case in IPv4 where there is no such clear boundary. Small ISPs can have relatively small /19 IPv4 address blocks while larger ISPs often administrate several /16 address blocks or even larger ones. Similarly, IPv6 addresses within the same /32 address block can be expected to be topologically (and therefore also geographically) close, but two IPv4 addresses within the same /16 address block are often not.

With IPv4, one main motivation for using /16 subnets was to make it difficult for any entity to run MorphMix nodes in a significant portion of all

possible /16 subnets. Since hardly any ISP or institution owns or administrates more /16 subnets than what corresponds to a class A network, this goal was met. Transferring the same principle to IPv6, then /32 subnets are reasonable to hinder ISPs from operating several nodes. This also implies that it is at least as difficult to do so for individual companies owning a /48 address block. According to Cyberatlas[8], there are about 12000 ISPs worldwide as of March 2003. Assuming each of them gets one /32 IPv6 address block, there will be 12000 /32 subnets, which is smaller than the number of /16 subnets that are possible in IPv4. In general, the problem with using /32 subnets directly it that it is difficult to predict how many of them there will be in use. If the number grows significantly beyond to the number of /16 subnets in IPv4, scalability problems arise. Conversely, if the number of /32 subnets is much smaller than the number of /16 subnets in IPv4, it becomes easier for an institution (or a few colluding institutions) to run nodes in a significant portion of all /32 subnets, which reduces the protection from the internal attacker. So what we need is a function that maps different /32 subnets into a space that approximately corresponds to the number of different /16 subnets in IPv4, and this is what we will solve in the remainder of this section.

The idea is to not use the /32 subnet of an IPv6 address directly, but the last 16 bits of a cryptographic hash (for instance SHA1 [46]) over the 32-bit prefix, which we denote as the $/16_h$ subnet of this address. Independently of the number of ISPs, the complexity of MorphMix is now bound by the 2^{16} different $/16_h$ subnets. If the number /32 subnets will be much larger than today, either because there will be more ISPs or because large ISPs get assigned several /32 subnets, at least one /32 subnet will hash into most of the $/16_h$ subnets and the situation for MorphMix is very similar to what we have today in IPv4. Of course this implies that different /32 subnets may be mapped onto the same $/16_h$ subnet, but this is not a problem, because an adversary controlling a /32 subnet still only gets control in a single $/16_h$ subnet.

It looks different if the number of /32 subnets will be significantly smaller than 2^{16}. This could for instance be the case if the number if IPv6 ISPs gets significantly smaller during the next years and most of them only need one /32 subnet. As a result, the number of $/16_h$ subnets will also be small and it will be easier for an adversary to operate nodes in many different $/16_h$ subnets. It this case, it is therefore reasonable to not only hash the first 32 bits to generate

[8] http://cyberatlas.internet.com

the $/16_h$ subnets, but also make use of the following 16 bits. For instance, assuming there are only 2000 ISPs left and each of them controls one /32 subnet, we could use the last 10 bits of the hash over the 32-bit IP address prefix and append the last 6 bits of the hash over the next 16 bits in the IP address, which should exhaust nearly the entire $/16_h$ subnet space. Assuming an adversary that either operates nodes by himself or by private persons, he must therefore again control nodes in thousands of different /48 networks to control a significant subset of all $/16_h$ subnets. Looking at a single ISP, we can state that it is still capable of controlling only a small subset of all $/16_h$ subnets (2^6 of 2^{16} in this example), which corresponds approximately to the inverse of the number of different ISPs. This is the best we can do to protect from a single ISP because the fewer different ISPs there are, the more significant the portion of the address space a single ISP controls. As another example, assuming there will still be 12000 ISPs and each of them controls one /32 subnet, then using the last 13 bits of the hash over the 32-bit prefix and appending the last 3 bits of the hash over the next 16 bits is reasonable.

Note that this mapping of IPv6 prefixes to $/16_h$ subnets such that it is difficult for any institution to control nodes that correspond to many different $/16_h$ subnets can also be considered as a generalisation of simply using 16-bit prefixes in IPv4. The simple mapping in IPv4 is possible because of the relatively small address space and the fact that most public /16 subnets have already been assigned to RIRs, ISPs, or end users. Due to the uncertainty about how many /32 subnets will be assigned to ISPs in IPv6, we cannot yet tell what mapping should be used. However, it is reasonable to assume that eventually, the number of /32 subnets assigned to ISPs will grow beyond 2^{16} and consequently, hashing the first 32 bits of an IPv6 address will be an adequate mapping into the $/16_h$ subnet space for the reasons given above.

We conclude that using these modifications, MorphMix can still be operated efficiently on top of IPv6 without any limits in the number of users that can be supported. Its maximum complexity is slightly higher than with IPv4, because there will be $2^{16} = 65536$ different $/16_h$ subnets compared to 56559 different /16 subnets with IPv4. As a result, the maximum selection size in (5.1) increases from 20 to 21 and the maximum length of the extended selections list increases from 5656 to 6242 according to (5.2). In addition, the length needed to store the IP addresses in extended selections list and in the node lookup list increases from 4 to 16 bytes, and the length of message 2 in Figure 5.5 also gets slightly larger.

5.10 Summary

In this section, we have presented the basic idea and functionality of MorphMix, a peer-to-peer-based dynamic mix network for low-latency applications. In contrast to static mix networks, there is no distinction between clients and mixes. Rather, every participating node is both a client and a mix at the same time. The principal goal of MorphMix is to enable practical anonymous Internet access for a large number of users. We have defined a threat model which states that due to the openness of MorphMix where anybody can easily run a node, the internal attacker controlling a significant subset of all nodes is the biggest threat. On the other hand, we have also stated that while it is easy for an adversary to run many MorphMix nodes, it is much more difficult to operate nodes in a significant portion of all public /16 subnets. To achieve the principal goal and to protect from this internal attacker, MorphMix is based on three core components.

The first component is the protocol to establish anonymous tunnels. One key decision to make MorphMix scalable is that every node along an anonymous tunnel picks its immediate successor node. This guarantees that at any time, a node only needs to have a few neighbours that it can append to anonymous tunnels. In addition, neighbouring nodes can communicate over their virtual link to learn which nodes have spare resources to accept new anonymous tunnels. But since everyone owning a computer with a public IP address can join MorphMix, a colluding set of malicious nodes would simply pick the next hop among themselves to compromise the anonymity of honest users. For this reason, we have designed the protocol to append a node to a tunnel in such a way to make this attack as complicated as possible. In particular, the node that is appending a node to a tunnel must offer a selection of several potential next hop nodes that are located in different /16 subnets to the initiator and the initiator picks one of them. This alone is not enough to stop the adversary because he can offer exclusively malicious nodes in his selections to control all remaining hops in a tunnel, and we have identified this attack as by far the most promising one assuming the adversary controls several nodes.

To counter this attack, we have developed a collusion detection mechanism, which is the second core component. The goal of the collusion detection mechanism is to detect malicious tunnels with high probability before the server is contacted. It makes use of the selections that are offered to the initiator when appending a node to a tunnel with the goal to detect those selections

that contain several malicious nodes with high probability. The collusion detection mechanism makes use of our assumption in the threat model that the adversary cannot control nodes in many different /16 subnets. Consequently, the collusion detection mechanism is not directly based on the IP addresses that are offered in selections, but on their 16-bit prefix. We have delivered a proof of concept that assuming 10000 nodes reside in as many different /16 subnets and that the adversary always offers exclusively malicious nodes in his selections, the mechanism indeed works by producing a correlation distribution with two peaks that can be clearly separated if the adversary does not control nodes in significantly more than 30% of all /16 subnets.

However, since the collusion detection is based on the assumption that honest nodes pick the nodes they offer in their selections from a wide variety of /16 subnets, MorphMix requires a mechanism that supports this. Consequently, the third core component provides a peer discovery mechanism that allows nodes to easily learn about a large number of nodes. The information about other nodes is organised in a node lookup list, which allows honest nodes to pick their neighbours from a wide variety of /16 subnets. In addition to providing the basis for the correct functioning of the collusion detection mechanism, this has an additional benefit because it is a necessary requirement for the internal adversary to control the first intermediate node to break the relationship anonymity between initiator and server. Selecting the neighbours from a wide variety of all /16 subnets reduces the probability the adversary controls this node because of the assumption in our threat model that the adversary can only control nodes in a limited number of all public /16 subnets.

We have also shown that MorphMix scales very well and can handle as many nodes as there are public IP addresses. Joining MorphMix is possible for a user independent of whether her computer has a static or dynamic public IP address or is located in a private network behind a NAT gateway. The computational and memory requirements to run a node are reasonable and can easily be handled by a modern personal computer, although we will be able to show that the computational overhead imposed by the public-key cryptography operations is indeed small only after we have analysed the data overhead in Section 8.3. Furthermore, even a dial-up Internet connection is sufficient to participate, and we will provide a detailed analysis to support this claim in Section 8.3.

Finally, MorphMix makes heavily use of the IPv4 addressing structure and on the way IPv4 addresses are assigned. However, we have demonstrated

that with minor modifications, MorphMix should be able to cope well with IPv6.

Chapter 6

Attacks on MorphMix

In this chapter, we analyse various attacks on MorphMix. We first describe the basic attack model we will use throughout this chapter. Then we analyse attacks where the adversary inserts different numbers of malicious nodes into malicious selections. Afterwards, we look at attacks including malicious witnesses and DoS attacks. We also discuss attacks exploiting the peer discovery mechanism and show that simply counting the occurrences of subnets in extended selections would not work well to determine whether an anonymous tunnel is good or malicious.

6.1 Basic Attack Model

We have mentioned in Section 5.5.2 that the most realistic threat on MorphMix is an adversary controlling several nodes in several /16 subnets. We therefore focus primarily on attacks where the adversary tries to control as many nodes in tunnels initiated by honest nodes by offering many or only malicious nodes in malicious selections. However, it should be kept in mind that there is a whole range of other attacks that are still possible against MorphMix, although we believe they are no significant threat. One of them is the passive observer that can see parts of all traffic handled by MorphMix nodes. If this adversary manages to observe both the cells on the virtual link between initiator and the first intermediate node and the corresponding data on the route between the final node and the server, he may succeed in break-

ing the relationship anonymity between initiator and server (see Section 5.3). However, since our threat model (see Section 5.4) is based on the assumption that only a small fraction of all traffic can be observed by a realistic adversary, we do not consider this attacker as a significant threat. Furthermore, since we do not employ digital certificates [86], there is no binding between a node's IP address and its public key, which makes man-in-the-middle attack a threat (see Section 5.5.2). But to implement this attack in an effective way to break several virtual links and layers of encryption along an anonymous tunnel, the adversary needs active control over many network links, which is even more difficult than observing the data on these links passively. Finally, there are attacks that MorphMix cannot cope with such as threats from malware. An adversary could manipulate the MorphMix software, the operating system or the application (for instance a web browser) that is used to access the Internet anonymously in a way such that information about end-to-end connections is leaked. Note that as is often the case with software that attempts to increase a user's privacy or security, introducing a back-door is much easier than breaking the system. Since it is likely that implementations of the MorphMix software will be open source, introducing a few lines of code that send information about all communication relationships of a user to a centralised server is very easy. This is much simpler than controlling several nodes or eavesdropping on a significant fraction of all traffic handled by the MorphMix nodes to collect and correlate large amounts of data. The best protection against malware in general is to download the software only from trustworthy servers and to check its integrity with checksums based on cryptographic hashes.

As defined in Section 5.6.3, a tunnel is *malicious* (or *compromised*) if the adversary controls at least the first intermediate and the final node. Correspondingly, an anonymous tunnel is *good* if the adversary does not control both the first intermediate and the final node. Looking at the collusion detection algorithm described in Algorithm 2 in Section 5.6.3, it is obvious that setting up t tunnels results in t_a tunnels being accepted and t_r tunnels being rejected. Similarly, there are t_g good and t_m malicious tunnels. Of the t_a tunnels that are accepted, t_{a_g} are good and t_{a_m} are malicious. Likewise, of the t_r tunnels that are rejected, t_{r_m} are indeed malicious but t_{r_g} are good. The goal is to minimise both t_{a_m} and t_{r_g}. Note that it is trivial to minimise either one without considering the other: rejecting every tunnel means that $t_{a_m} = 0$ and accepting every tunnel implies $t_{r_g} = 0$. The difficulty is to keep both values small simultaneously. Consequently, we are mainly interested in

two figures in our following analyses. The first is the fraction of malicious tunnels among the accepted tunnels, $f_{a_m} = t_{a_m}/t_a$. The second is the fraction of good tunnels that were wrongly classified as malicious, $f_{r_g} = t_{r_g}/t_g$, which is – using intrusion detection systems terminology – nothing else than the fraction of false positives. The prime goal is to minimise f_{a_m}. If we manage to keep f_{a_m} close to zero, then a reasonably low fraction of false positives (e.g. 0.25) is acceptable.

Note that f_{r_g} and f_{a_m} are not independent. With $f_{a_m} = t_{a_m}/t_a$, $t_a = t_{a_g} + t_{a_m}$, $t_{a_g} = t_g - t_{r_g}$, and $t_{r_g} = f_{r_g} \cdot t_g$, it follows

$$f_{a_m} = \frac{t_{a_m}}{t_g(1 - f_{r_g}) + t_{a_m}}, \tag{6.1}$$

which means that a large f_{r_g} implies a large f_{a_m}. So keeping the fraction of false positives low is not only important for good usability of the system, but especially to keep the fraction of malicious tunnels that are wrongfully accepted as good low.

6.1.1 The Node Simulator

To analyse the effectiveness of attacks by an adversary controlling a subset of the nodes, we have developed a *node simulator* that allows simulating attacks on MorphMix from the point of view of a single node. Basically, the simulator simulates setting up anonymous tunnels, focusing on the selections that are offered to the node. The simulator also completely implements the collusion detection mechanism and the peer discovery mechanism. The node simulator has been used for all analyses in the previous, this, and the following chapter. Note that this node simulator is not related to the MorphMix simulator we will describe and use in Chapter 8.

The node simulator allows specifying a wide variety of parameters to analyse different settings. The most important parameters them include:

- The number of /16 subnets that contain nodes
- The number of honest and malicious nodes and how they are distributed over the /16 subnets
- The capabilities of nodes (see Section 7.3)
- The strategy of the adversary (how many malicious nodes to include in selection and when to attack)

- The tunnel length

In addition, the node simulator allows specifying several fundamental parameters of MorphMix, including the selection size and size of the extended selections list. In fact, we have used the node simulator to find reasonable values for several MorphMix parameters, including the selection size in (5.1), the size of the internal table in (5.2), and the number of slots in the internal representation of the correlation distribution (see Section 5.6.1).

The node simulator can provide different outputs, including f_{r_y}, f_{a_m}, and the correlation distribution depending on the number of tunnels that have been set up.

6.1.2 Basic Scenario

For all attacks in this section, we use the same basic setting as in Figure 5.6: there are 10000 nodes in 10000 different /16 subnets and every node has the same probability of being offered in a honest selection. The number of colluding nodes ranges from 500–4000. According to (5.1), a selection contains 14 nodes, which means the adversary can vary the number of malicious nodes he puts into a selection from 0–14. We set up 5000 anonymous tunnels, whereas each tunnel consists of five nodes in total. Note that this is a very "clean" scenario because every node is in a different /16 subnet. In addition, if a fraction f of all nodes is malicious, it actually means that a fraction f of all /16 subnets are completely controlled by the adversary. However this clearly defined scenario is perfectly suited to compare the basic effectiveness of different attack strategies that can be employed by an adversary. We will analyse more realistic scenarios in Chapter 7.

With malicious nodes present in the system, it is never possible to reduce f_{a_m} to zero. Like in any mix network where malicious nodes are present, it may always happen that the adversary controls the endpoints of an anonymous tunnel. If we assume there are n nodes in MorphMix that all reside in different /16 subnets, an adversary controls n_c of them, and he picks the nodes for the malicious selections randomly from the set of all (honest and malicious) nodes, the probability he controls at least the first intermediate and the final node in a tunnel is approximately $(n_c/n)^2$ (see Section 5.3). There is no way to detect such a tunnel because the malicious nodes behave in exactly the same way as honest nodes during the tunnel setup. On the other hand, as we have seen in Section 5.5.1, if MorphMix would not employ any collusion detection mechanism, the adversary could easily control all nodes in a tunnel

if he controlled the first intermediate node. The resulting probability for the adversary to control all intermediate nodes and the final node would be n_c/n. The goal of the collusion detection mechanism is therefore that the adversary cannot do much better than he would if he played fair, i.e. if malicious nodes picked the nodes in their selections randomly.

6.2 Varying the Attack Level

We first analyse what the adversary can achieve depending on his aggressiveness. The more malicious nodes that are included in malicious selections, the more aggressive the adversary is. We denote the number of malicious node in malicious selections the *attack level* of the adversary. We look at two different basic strategies. In the first, the adversary attacks always when one of his nodes offers a selection. In the second, the adversary attacks only if he controls the first intermediate node in the tunnel.

6.2.1 The Adversary Attacks Always

In this attack, the adversary always offers malicious nodes in its selection if a malicious node is hit during the setup of an anonymous tunnel. First, we assume the adversary uses always the same attack level, which means that the number of malicious nodes in malicious selections is always the same. We analyse different attack levels and how they influence the adversary's chances to control as many tunnels as possible, i.e. to maximise f_{a_m}. Figure 6.1 illustrates the average results over ten simulation rounds[1].

Figure 6.1 shows that unless the adversary controls 40% of all nodes (see below for an explanation), it is not advisable for him to always include only malicious nodes in malicious selections. This is not surprising, because Figure 5.6 illustrates that the peaks resulting from the correlations of honest and malicious extended selections are clearly separated as long as the fraction of malicious nodes is reasonably small. Including fewer malicious nodes makes malicious selections more similar to honest selections, and the peaks in the correlation distribution can no longer be easily separated. Below a certain threshold, it is virtually impossible for the correlation detection mechanism to determine if a selection is malicious. For instance, if 10% of all nodes are

[1] Without mentioning this again, we will use the average over ten simulation rounds in general to generate plots of this kind in this and the next chapter.

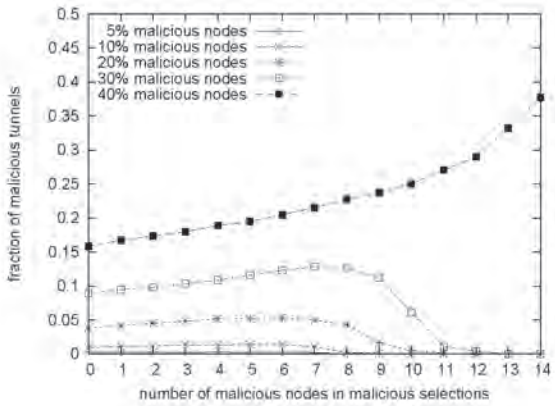


Figure 6.1: f_{a_m} *if the adversary attacks always with the same attack level.*

malicious, f_{a_m} slowly increases as the the number of malicious nodes in malicious selections grows from 0–5, because honest and malicious selections are too similar for the collusion detection mechanism to detect. With more than 5 malicious nodes in a malicious selection, the collusion detection mechanism starts working correctly in the sense that more and more malicious selections are detected if the attack level is increased. This can even better be seen when looking at the correlation distribution. Figure 6.2 illustrates the correlation distribution when 10% of the 10000 nodes are malicious. The number of malicious nodes in selections is increased from 0–14.

For 0–5 malicious nodes in malicious selections, the peak from malicious extended selections overlaps completely with the peak from honest extended selections. Only increasing the number of malicious nodes in malicious selections to 6 and beyond makes it more and more possible to separate the peaks and detect malicious nodes with higher and higher probability. We conclude the collusion detection mechanism cannot completely prevent malicious nodes from offering some malicious nodes in their selections, but it prevents them from offering too many such nodes.

Figure 6.2 also serves well to explain how the correlation limit is determined. We do not try to detect two peaks and pick the minimum between them. This would not always work because if there are no malicious nodes, there is no second peak and if there are different, independent adversaries,

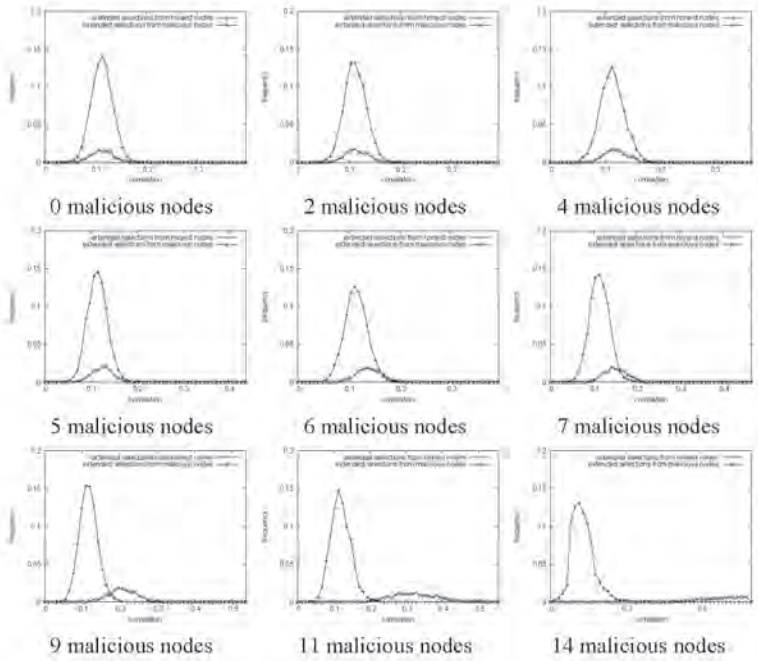


Figure 6.2: *Correlation distribution when varying the attack level from 0–14.*

there are "multiple second peaks", which could also overlap. In addition, as it is the case in Figure 6.2 when about 6–9 malicious nodes are used in malicious selections, determining the minimum is difficult if the peak from malicious extended selections overlaps with the peak from the honest ones and the sum of the two does not result in two peaks but looks more like one peak with a tail on the right end. The strategy is therefore to use the left flank of the first peak and the maximum of this peak as a reference to guess the approximate end of the first peak. This allows to detect at least some malicious selections even when the peaks overlap significantly. For more details about determining the correlation limit, refer to the MorphMix prototype implementation (see Appendix A.7).

Depending on the percentage of malicious nodes, there is an optimum number of malicious nodes that should be offered in malicious selections for

the adversary to be most effective. We name this optimum number the *optimal attack level*. It is defined as the maximum number of malicious nodes in malicious selections such that the two peaks still overlap completely. With 10000 nodes in the system and if 10% of all nodes are malicious, Figure 6.2 tells us that the optimal attack level is 5. This is confirmed by Figure 6.1, which tells us indeed that the adversary manages to control a maximum fraction of 0.0142 of all tunnels if he uses 5 malicious nodes in malicious selections. This is larger than the fraction of $0.1^2 = 0.01$ he would control if he played fair, but is also much smaller than the fraction of 0.1 he would control if no collusion detection mechanism were employed.

Figure 6.1 also shows that the collusion detection mechanism has its limits if the percentage of malicious nodes increases. With 20% malicious nodes, the adversary manages to control a fraction of 0.0535 of all tunnels compared to 0.04 if the malicious nodes played fair and with 30% malicious nodes, this grows to 0.128 compared to 0.09 if the adversary played fair. With 40% malicious nodes, however, the collusion detection mechanism does no longer work and the adversary manages to compromise a fraction of nearly 0.38 of all tunnels instead of 0.16 if he played fair. This can be explained by looking at Figure 5.6 and realising that the peaks resulting from selections from honest and malicious nodes move closer together as the number of malicious nodes gets larger.

A variation of the first attack is to still attack always, but to attack with a higher level if the final node is appended to the anonymous tunnel. This means that if the adversary controls the last intermediate node, this node offers only other malicious nodes in the selection for the final node. If an intermediate node is appended to a malicious node, the node offers fewer malicious nodes in the selection. This attack is difficult to mount in practice because the adversary cannot know when the final node is appended. Assuming the adversary controls indeed the last intermediate node in a tunnel, the predecessor node of the final intermediate node is honest, and there is an additional malicious node in the tunnel, the adversary can try to correlate the cells handled by both malicious nodes during the tunnel setup. Since the number of nodes along anonymous tunnels is reasonably small in practice, e.g. five, this may tell the adversary the position of the last intermediate node in the tunnel. However, there are only a few cells that are handled by both malicious nodes before the last intermediate node must send back the selection for the final node to the initiator. If there are many nodes in the system, these cells are definitely not enough to make a correct guess with high probability.

Another possibility for a malicious node is to guess its position without correlating data with other nodes. The idea is to measure the time that has passed between sending message 9 in Figure 5.5 to append the malicious node and receiving message 1 to append the next node. If the measured time is very small, then it is likely that there is no other node between the initiator and the malicious node. If the time increases, it is more likely that there are "a few" other nodes in between. However, if the initiator introduces a random delay of several seconds between the reception of message 10 (or message 9 if the first intermediate node is appended) and sending out message 1 to append the next hop, it is virtually impossible for a malicious node to determine its position in an anonymous tunnel during the setup. Nevertheless, we analyse the impact of this attack assuming the adversary always knows when the final node is appended to the tunnel to compare it with the attack described above.

We vary the number of malicious nodes in malicious selections from 0–14 when another intermediate node is appended. When the final node is appended, malicious nodes offer 14 malicious nodes in the selection. Figure 6.3 depicts the fraction of malicious tunnels among the accepted tunnels.

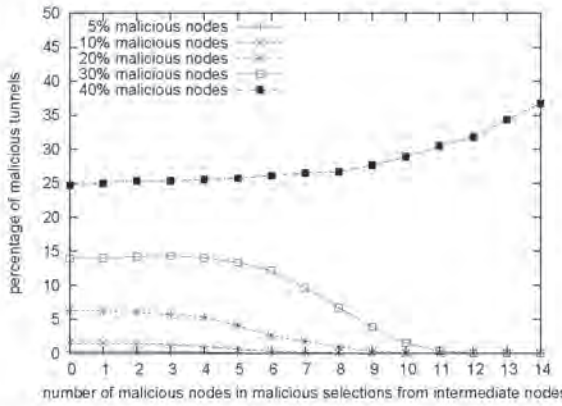


Figure 6.3: *f_{a_m} if the adversary attacks with different attack levels.*

The attack is slightly more efficient than the one above. For instance, with 10% malicious nodes in the system, the adversary controls a fraction of 0.0147 of all tunnels if he uses one malicious node in his selections.

6.2.2 The Adversary Attacks Selectively

A better strategy for the adversary is to attack only if he controls the first intermediate node in an anonymous tunnel. Intuitively, this makes sense because breaking the relationship between initiator and server is only possible if he controls the first intermediate node in the corresponding tunnel. Attacking when the first intermediate node is not controlled delivers unnecessarily an extended malicious selection with several malicious nodes to the initiator. Since a node stores the extended selections it receives in the extended selections list, this increases the correlation of further malicious extended selections with several malicious nodes that are sent to the same initiator. In the following two attacks, malicious nodes offer only honest nodes in their selections whenever the first intermediate node is not controlled by the adversary.

Again, these attacks are difficult to carry out in practice. The adversary must decide during the setup whether a node he controls is the first intermediate node along an anonymous tunnel. This is very difficult since all information a malicious first intermediate node has is to measure the time between sending messages 9 in Figure 5.5 to the initiator and receiving message 1 to append the next hop. In addition, if the first intermediate node is indeed malicious and one or more honest nodes are picked as the next nodes in the tunnel, it is very difficult to determine for a malicious node appended to this tunnel whether the first intermediate node is also malicious, for the same reasons as discussed in Section 6.2.1. We still analyse the impact of the following two attacks in the same way as above.

Like in the case where the adversary attacks always, we first assume the number of malicious nodes in malicious selections is always the same. We analyse different attack levels and how they influence the adversary's changes to control as many tunnels as possible (f_{a_m}). Figure 6.4 illustrates the results.

Compared to Figure 6.1, this strategy gives the adversary better chances to control a malicious tunnel. With 10% malicious nodes, the maximum f_{a_m} increases from 0.0142 with 5 malicious nodes in malicious selections to about 0.0175 with 8 malicious nodes. Interestingly, the adversaries chances decrease compared to Figure 6.1 if he controls 40% of all nodes. The reason is that the initiator now gets fewer malicious selections, which causes the peak from correlations of malicious extended selections to be smaller than in Figure 5.6(f). As a result, the initiator can better detect the leftmost peak and a reasonable correlation limit.

We also examine the adversary's changes if he attacks with a higher attack

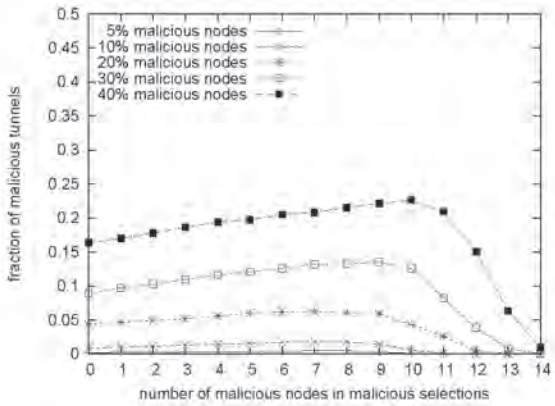


Figure 6.4: *f_{a_m} if the adversary attacks always with the same attack level but only if he controls the first intermediate node.*

level when the final node is appended to the anonymous tunnel. To carry out this attack, the adversary must not only know if he controls the first intermediate node, but also guess when the final node is appended to a tunnel, which is even more complicated than the attack above. Assuming the adversary controls the first intermediate node, we vary the number of malicious nodes in malicious selections from malicious nodes from 0–14 when another intermediate node is appended. When the final node is appended, malicious nodes offer 14 malicious nodes in the selection. Figure 6.5 depicts the fraction of malicious tunnels among the accepted tunnels.

This maximum f_{a_m} that can be achieved with this attack is again larger than before. Compared to Figure 6.1 and an adversary controlling 10% of all nodes, the maximum f_{a_m} increases to about 0.0231.

6.2.3 Summary

Comparing the attacks described above, we conclude the adversary should attack only if he controls the first intermediate node in an anonymous tunnel and attack with a higher attack level when appending the final node. However, as already pointed out, this attack is nearly impossible to mount in practice because decisions have to be made based on very little information. In partic-

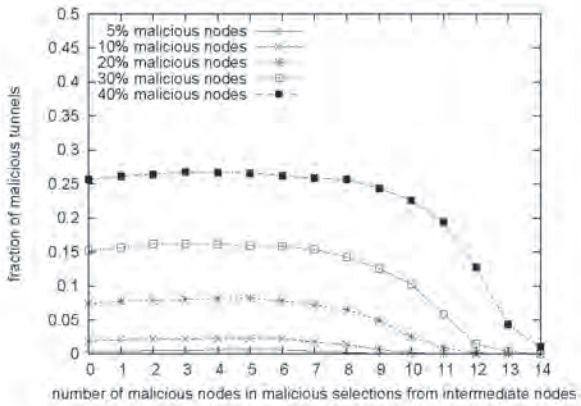
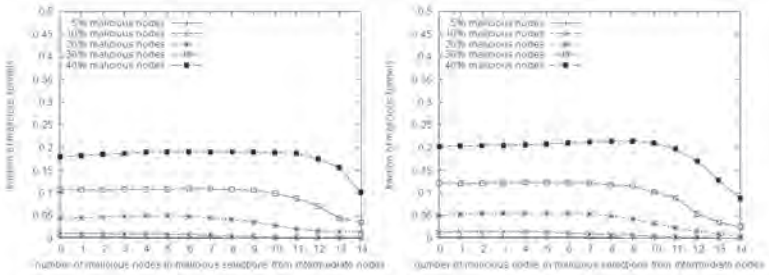


Figure 6.5: *f_{a_m} if the adversary attacks with different attack levels but only if he controls the first intermediate node.*

ular, having the initiator introduce a random delay of several seconds between the reception of message 10 (or message 9 if the first intermediate node is appended) and sending out message 1 to append the next hop makes it virtually impossible for a malicious node to find out if it is the first intermediate node in a tunnel. Similarly, even if a malicious node learns it is the last intermediate node before it offers the selection to the initiator, only very little information is available for the adversary to find out if the first intermediate node of this tunnel is also malicious. Of course, the adversary can always make a guess, but the uncertainty about when to attack will result in several missed opportunities where he should have attacked but did not, and many situations where he wastes a malicious extended selection because he decided to attack although he didn't control the first intermediate node.

To support this claim, we analyse the impact of wrong guesses on the results in Figure 6.5. In this scenario, the adversary has to decide between three options when he controls a node: attack by offering only malicious nodes, attack with the reduced attack level, or do not attack and offer only honest nodes. Simply guessing would tell the adversary the right thing to do in $1/3$ of all cases when he controls a node. To analyse the impact of wrong guesses, we analyse two cases. In the first case, we assume the adversary guesses correctly with a probability of $1/2$; in the second case we increase

this probability to $2/3$. If the adversary make the wrong guess, we assume he chooses any one of the two other options with equal probability. The results are illustrated in Figure 6.6.



a) guessing correctly with prob. $1/2$ b) guessing correctly with prob. $2/3$

Figure 6.6: *f_{a_m} if the adversary attacks with different attack levels but only if he controls the first intermediate node, assuming he does not always guess correctly.*

Comparing Figures 6.5 and 6.6, we see that the higher the probability the adversary makes a correct guess, the larger the number of malicious tunnels among the accepted tunnels, which is not surprising. However, comparing Figures 6.1 and 6.6(b) shows that the adversary must guess correctly with a probability of about $2/3$ or better to be as successful as when he attacks always with the same attack level. Since guessing correctly with a probability of $2/3$ is very unlikely according to our discussion in this section, we can state that the adversary will be more successful in practice by attacking always with the same attack level than by employing the strategy in Figure 6.5. Similar arguments can be made for the other two attacks that depend on correct guesses, illustrated in Figures 6.3 and 6.4. Since these attacks result in a smaller f_{a_m} than the attack in Figure 6.5 assuming the adversary always guesses correctly, we expect they also result in a smaller f_{a_m} than in Figure 6.6(b) in practice. We therefore conclude that of all attacks we have discussed in this section, the one where the adversary attacks always with the same attack level results in the largest f_{a_m} in practice. The adversary can get all information to carry out this attack optimally, because observing the system tells him the approximate number of different /16 subnets with nodes in the system. This can then be used to determine the optimal attack level. One way to determine this optimal

level is by employing the same method we do throughout this thesis: by using our node simulator and testing the effectiveness of different attack levels.

6.3 Attacks Including Malicious Witnesses

Recalling the procedure to append a node to a tunnel in Figure 5.5, there is a potential attack that can be used if the node b that appends the next hop to the tunnel and the witness collude. Although b cannot yet know if the witness will be also malicious, it can hope for a malicious witness and generate a forged selection for the initiator. This selection contains any IP addresses of nodes that are not in the b's collusion and self-generated public keys of which node b knows the corresponding secret keys. If it turns out that the witness is really malicious, b and w can decrypt DH_a sent by the initiator in message 3 and b can simulate the next hop itself. However if the witness is not malicious, the setup will fail because w will either not be able to contact c at all because b did not include the IP addresses of existing MorphMix nodes or c won't be able to decrypt DH_a in message 5 because it is not encrypted with c's real public key. But even if b and w are malicious and b simulates c itself, this will be detected when appending the next hop if that witness is not malicious because it will include b's IP address in the receipt in message 7. Therefore, it does not make sense that the adversary makes use of this attack if he controls a node early in the tunnel, because most attacks would be detected. However, assuming the adversary knows when the final node is appended to the anonymous tunnel, it could make sense to employ this attack when the last intermediate node is malicious. Again, this attack is not easy to mount because finding out when the final node is appended is difficult in practice.

We analyse the impact of this attack. We vary the number of malicious nodes in malicious selections from 0–14 when another intermediate node is appended. When the final node is appended, malicious nodes hope for a malicious witness and offer only IP addresses in /16 subnets that do not contain malicious nodes and also include self-generated public keys of which they know the secret keys. Figure 6.7 depicts the fraction of malicious tunnels among the accepted tunnels.

When controlling 20% of all nodes or fewer, the maximum f_{a_m} is smaller than in Figure 6.1. It seems that the probability that b and w are colluding when appending the last hop is too small to gain anything from this attack. With 30 or 40% malicious nodes, f_{a_m} is slightly larger than in Figure 6.1.

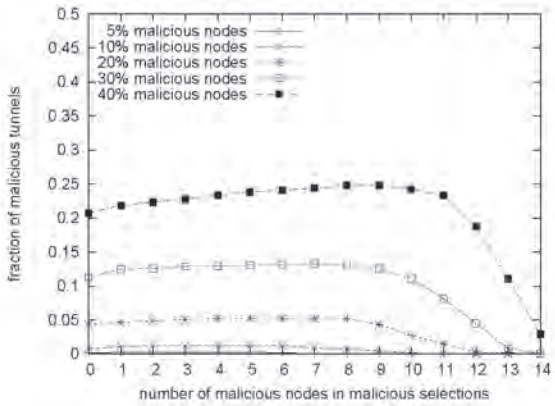


Figure 6.7: f_{a_m} *if the adversary hopes for a malicious witness when the final node is appended.*

Considering that this attack is significantly more difficult to mount than the one in Figure 6.1, it is very unlikely to be more effective in practice for the reasons we discussed in Section 6.2.3. Consequently, it does not make sense for an adversary to use this attack.

6.4 Denial of Service Attacks

Any MorphMix node can always choose not to forward cells, and there is nothing MorphMix can do about it. For the initiator, an anonymous tunnel simply fails to transport data; it cannot distinguish between a congested node, a failure or congestion in the underlying physical network, a node that has crashed, or a node that refuses to forward data. If a tunnel fails, all end-to-end communication relationships that use this tunnel also fail.

This can be exploited by the adversary because it may always happen that a tunnel that is accepted by the initiator contains malicious nodes, but the tunnel itself is not malicious because the adversary does not control both the first intermediate and the final node. If the adversary refuses to transport data through these tunnels after they have completely been set up, t_{a_y} gets smaller and so does t_a, but t_{a_m} stays the same. Since $f_{a_m} = t_{a_m}/t_a$, f_{a_m}

gets larger, i.e. the fraction of malicious tunnels among all tunnels accepted by the initiator gets larger.

Figure 6.8 depicts the impact of this attack. We use the same setting as in Figure 6.1 where the adversary attacks always with the same attack level. However, if the adversary controls at least one node along the tunnel but not both the first intermediate and the final node, he won't forward any data along this tunnel, which means the initiator cannot use it.

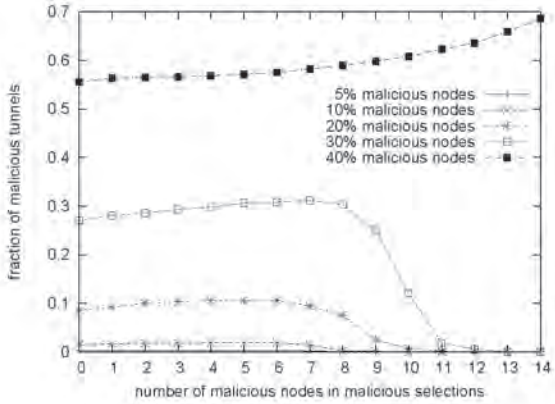


Figure 6.8: *f_{a_m} if the adversary attacks always with the same attack level and refuses to forward data along any tunnel where he controls at least one node but not both the first intermediate and the final node.*

With 5 or 10% malicious nodes, f_{a_m} is not significantly larger than in Figure 6.1. However, as the percentage of malicious nodes grows, f_{a_m} significantly increases. For instance, with 20% malicious nodes f_{a_m} grows to 0.106 compared to 0.0535 in Figure 6.1, and with 30% malicious nodes f_{a_m} is 0.311 compared to 0.128 in Figure 6.1.

The explanation for this is that with 10% malicious nodes, the probability there is at least one malicious node in a tunnel is about 0.34. This is not depicted in Figure 6.8 but was produced as an additional output during the analysis with the node simulator to generate the results in Figure 6.8. With 20% malicious nodes, this probability increases to 0.59 and with 30% malicious nodes to 0.77. It is therefore not surprising that as the percentage of malicious nodes grows, more and more tunnels that are actually good because

the adversary does not control both the first intermediate and last node cannot be used because the adversary blocks the data flowing through them.

Such DoS attacks are not a specific MorphMix problem, but a general problem of mix networks where the adversary operates a subset of the mixes himself: if the adversary blocks the traffic along all chains of mixes where he controls at least one node but not all the nodes he needs to break the chain, his success rate increases. It seems difficult to prevent this attack, but one possibility to reduce its impact is to couple MorphMix with a reputation system. Nodes repeatedly failing to forward data would get a bad reputation over time and would no longer be offered in extended selections from honest nodes. Research on reputation systems is still in its infancy, but initial studies to make mix networks more reliable through reputation have been carried out (see Section 3.5).

Although the theoretical threat from these attacks is significant if the percentage of the nodes controlled by the adversary exceeds 15–20%, it is not trivial to mount in practice. Before refusing to transport data through a particular tunnel, the adversary first must be sure that he does really not control the first intermediate and the final node. Correctly determining this with high probability is only possible after several cells have been transported along the tunnel. The collected data must be sent to a centralised place where the traffic flowing through the various malicious nodes can be correlated to learn which tunnels the adversary controls and which he can block. This takes time and depending on the system size, the number of malicious nodes, and the amount of data transported through a tunnel before it is torn down, this may take longer than the average lifetime of a tunnel. It should be at least possible for a user to send and receive some data through such a tunnel before the adversary starts blocking the traffic, which reduces the impact of the attack. We conclude that in practice, the adversary could be able to find out quickly enough if he controls nodes in a tunnel that is not malicious to block the traffic before the lifetime of the tunnel expires, which increases $f_{a,rc}$ compared to Figure 6.1. However, especially if the percentage of malicious nodes is below 15–20% malicious nodes, the expected gain for the adversary won't be significant.

As a side note, another strategy for an adversary is to participate in MorphMix with several nodes simply to disrupt the service and not to link initiators to servers. To do so, his nodes would accept tunnels being established through them but refuse to transport data once a tunnel has been set up or stop forwarding data after it has been used for a while. Depending on the

application, MorphMix is resilient to tunnel failures up to a certain degree by switching to another tunnel if a tunnel fails and establishing the communication relationship wit the server again. For instance, in the case of web browsing, if downloading a web page is interrupted, the page can simply be downloaded again using another tunnel (see Section 8.3.9). But in general, this attack can be quite effective in the sense that if most tunnels fail, the quality of service as perceived by the users gets so poor that they no longer use MorphMix. As discussed above, a reputation system could help against DoS attacks in general by excluding nodes that have a history of offering poor service.

6.5 Exploiting the Peer Discovery Mechanism

During initial peer discovery, a node a contacts another node b directly to learn about some other nodes. Afterwards, b knows some nodes a is storing in its node lookup list. Let's assume b is malicious and b tells a about m malicious nodes $\{m_1, m_2, \ldots, m_m\}$ that are part of b's collusion. If we look at one of these nodes and identify it with $m_k, 1 \leq k \leq m$, then we can say that if m_k gets picked later as a witness to add a hop to an anonymous tunnel, then it could be that a is the initiator of this tunnel. The adversary can maximise his chances to be successful with this attack by making sure the nodes are advertised only to a and to no other node. However, a could itself tell others about some or all of these nodes during initial or continuous peer discovery, which implies the adversary cannot be sure if the initiator is indeed a if m_k is picked as a witness. In addition, a may have removed m_k from its node lookup list if it has learned about several other nodes in the same /16 subnet or if it has contacted m_k directly to pick it as a new neighbour. But assuming the adversary has told about m_k exclusively to a and assuming a has not informed other nodes about it, then a malicious node c that is colluding with b and m_k and that is appended to a tunnel can be sure that a is the initiator of this tunnel if m_k is picked as the witness for adding this hop. If no further nodes are appended, the tunnel is compromised because the adversary controls the final node and knows how the initiator is.

Looking at continuous peer discovery, there is a similar exploit. A malicious node b offers m other malicious nodes $\{m_1, m_2, \ldots, m_m\}$ exclusively in one selection. Let's assume that a is the initiator of this tunnel, it happens the adversary controls the final node, and, by correlating the data that are

sent along the tunnel, the adversary can determine that the selection above was offered during the setup of this tunnel. Unless the adversary controls also the first intermediate node in this tunnel, he cannot know that a is the initiator at this time, but since the adversary controls the final node, he remembers all servers that were contacted using this tunnel. If later, a contacts any m_k, $1 \leq k \leq m$ directly to pick it as a new neighbour, and if we assume that a has not informed others about m_k, then the adversary can be sure that a was the initiator of the tunnel.

How useful are these attacks? The first attack requires that when m_k is picked as a witness, a malicious node must be appended. In addition, the final node in the tunnel must be malicious, which is trivial is no further nodes are appended after m_k has been picked as the witness. So even if m_k is indeed selected as a witness, it is by no means guaranteed that the final node will also be malicious. The second attack suffers from the problem that the final node does not offer a selection and malicious intermediate nodes cannot predict if the final node in the tunnel will also be in their collusion. Consequently, the malicious intermediate nodes must offer other malicious nodes exclusively in their selections although the probability the final node will be malicious is relatively small. In general, the problem is similar in both cases: in the first attack, the final node must be malicious when m_k is picked as a witness and in the second case, the final node must be malicious when m_k is offered in a selection. But beyond this, both attacks suffer from the problem that they are very uncertain to succeed because the nodes can be overwritten with other nodes from the same /16 subnet in the targeted user's node lookup list or information about them can be forwarded to other nodes during peer discovery. So looking at the first attack, the adversary has no way to tell if the node picking m_k as a witness is indeed the same node that received m_k exclusively earlier. Similarly, in the second attack, the adversary does not know if the node directly contacting m_k is the same node that received m_k exclusively as part of a malicious selection before. Another problem for the adversary is to decide when an exclusive node can be reused. Since the information about a node may remain in a node lookup list for a long time until it is removed (see Section 5.7.3), the adversary does not know if a node has been overwritten or is still out there in the targeted node's list if it has never been contacted directly or used as a witness. If the targeted node has further disseminated information about the node, things get even more complicated because its virtually impossible for an attacker to tell when a node has been removed from the node lookup lists of all nodes.

The main problems with exploiting the peer discovery mechanisms are the uncertainty when deciding if an attack was successful or not and that multiple malicious nodes must be assigned exclusively to a single attack without being reusable for a potentially long time. The second problem gets smaller when the adversary owns several /16 subnets, which gives him quite large reservoir of exclusive nodes. But operating many nodes in one subnet increases the probability that the nodes in these subnets are overwritten quickly in the node lookup lists. Looking at all the complications with the attacks exploiting peer discovery, we argue that they are unlikely to be effective in practice. The attacks may be used and even be effective from time to time on a small scale, but are not well suited to attack on a large scale.

6.6 Why Counting the Occurrences of Subnets does not Work

We stated in Section 5.6.1 that simply counting how many times the subnets in an extended selection show up in the extended selections list would not work well. Here, we show a simple attack how the adversary could exploit this. We use again 10000 nodes in 10000 subnets, 1000 of which are malicious. Malicious nodes offer six malicious nodes in their selections, but if a malicious node is asked by one of its honest neighbours if it is willing to be the next hop in an anonymous tunnel, then it accepts this with a probability of only 0.35. The main idea behind this strategy is to compensate the increase of malicious nodes in malicious selection with the frequency they are included in honest selection. Figure 6.9 illustrates the occurrences of the 10000 /16 subnets. The first 9000 entries are the subnets with honest nodes while the last 1000 subnets contain malicious nodes.

Looking at Figure 6.9, it is virtually impossible to tell what subnets contain many malicious nodes. Using the collusion detection mechanism based on occurrences of subnets in the extended selections list in our node simulator, we found out that the adversary would have managed to control the first intermediate and final node in a fraction of about 0.036 of all tunnels accepted by the initiator. This is significantly larger than the maximum fraction of 0.0142 of all tunnels that are malicious according to Figure 6.1 with 10% malicious nodes and when using the "real" collusion detection mechanism as described in Section 5.6.1. Carrying out the attack above and using the real collusion detection mechanism, the fraction of malicious tunnels drops from 0.036 to

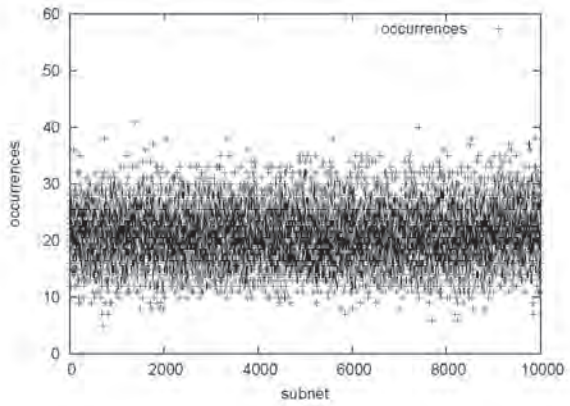


Figure 6.9: *Occurrences of /16 subnets in the extended selections list.*

about 0.0043. We therefore conclude that the correlation as it is computed in Algorithm 1 in Section 5.6.1 is a useful measure to learn which extended selections contain many malicious nodes.

6.7 Summary

Based on the assumption that an adversary controlling a subset of all nodes is the biggest threat to MorphMix, we have analysed several different attack strategies. The most reasonable basic attack the adversary should make use of is attacking always with the same attack level, as analysed in Figure 6.1. Attacking with different levels or attacking only when the first intermediate node is malicious is very difficult in practice and very unlikely to be more successful as we have shown in Figure 6.6.

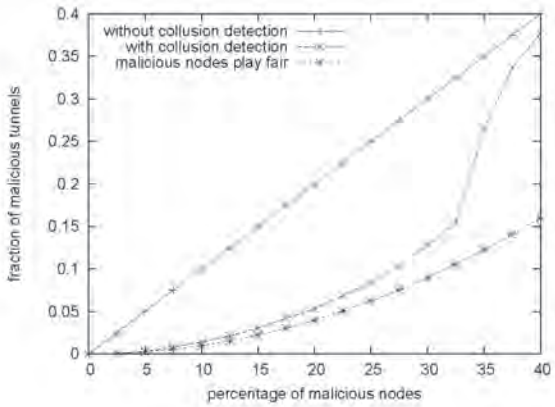
The DoS attack where the adversary refuses to forward data in tunnels where he does not control both the first intermediate and the final node is a potential threat because it significantly increases f_{a_m}, especially if the adversary controls at least 15–20% of all nodes. However, we can expect its practical impact to be much smaller than in the theoretical analysis in Figure 6.8 because the tunnels where traffic would be blocked can only be identified after several cells have been sent through them. In general, any kind

of DoS attack is a problem if the quality of service as perceived by the users gets so poor that they no longer use MorphMix. One possibility to reduce the impact of DoS attacks are future advances in research on reputation systems and their application to mix networks. Exploiting the peer discovery mechanism could give the adversary an advantage for a short time, but turns out to be very problematic in the long run, since the adversary cannot control to which other nodes information about exclusive nodes are propagated after the information has been given to one particular node. Finally, we have shown that simply counting how many times the subnets in an extended selection show up in the extended selections list would not be a useful alternative to our collusion detection mechanism to detect malicious selections.

Consequently, we assume the adversary attacks always with the same attack level in the remainder of this thesis. We also assume he can always determine the optimal attack level to use to maximise f_{a_m}. Using this assumptions, Figure 6.10 depicts the fraction of malicious tunnels among the accepted tunnels depending on the percentage of malicious nodes (0–40%). The top line shows the fraction of malicious tunnels if no collusion detection mechanism is employed, which corresponds to n_c/n if n_c of n nodes are malicious. The bottom line shows the fraction if the adversary plays fair, i.e. if malicious nodes picked the nodes in their selections randomly from the set of all nodes, which corresponds to $(n_c/n)^2$. The line in between shows the fraction of malicious tunnels if the adversary uses the optimal attack level depending on the percentage of nodes he controls. The table below the figure gives the optimal attack level to maximise f_{a_m}. The setting is the one we described in Section 6.1.

As long as the percentage of malicious nodes is reasonably small, the collusion detection mechanism works well in the sense that the adversary cannot compromise significantly more tunnels than if he plays fair. If the adversary controls more than about a third of all nodes, the collusion detection mechanism starts failing because the peaks resulting from selections from honest and malicious nodes are too close together to determine a reasonable correlation limit (see Section 5.6.1).

Note that since we used a setting where every node is located in its own /16 subnet, the results in Figure 6.10 can also be interpreted by replacing *percentage of malicious nodes* with *percentage of subnets where the adversary has full control*. Consequently, Figure 6.10 also tells the fraction of malicious tunnels depending on the percentage of all /16 subnets where the adversary has full control assuming all /16 subnets contain the same number of nodes



malicious nodes (%)	2.5	5	7.5	10	12.5	15	17.5	20
optimal attack level	3	4	5	5	5	5	6	6
malicious nodes (%)	22.5	25	27.5	30	32.5	35	37.5	40
optimal attack level	6	6	7	7	8	14	14	14

Figure 6.10: *$f_{a,m}$ without and with collusion detection, and if the adversary plays fair.*

and all nodes have the same probability of being offered in honest selections. Considering that fully controlling 10% of all /16 subnets is difficult according to our threat model (see Section 5.4), we can expect already now that assuming there are honest nodes in most public /16 subnets, a realistic adversary should only manage to compromise a small fraction of all tunnels that are accepted by the initiator. Nevertheless, we will analyse this in more detail in the next chapter where we assume more realistic scenarios than the one we used in this chapter to compare different attack strategies.

Chapter 7

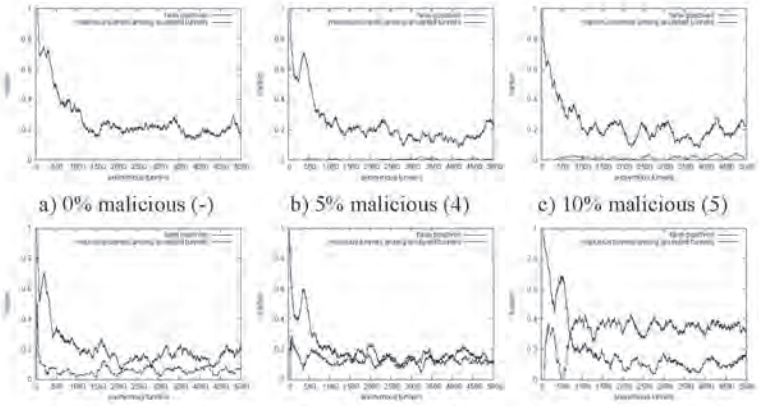
Analysis of the Collusion Detection Mechanism

In this chapter, we evaluate the performance of the collusion detection mechanism. We first use the basic scenario from the previous chapter to see what happens if a node joins MorphMix for the first time. Then, we extend this basic scenario to analyse the effect if there are honest and malicious nodes in the same subnet. Afterwards, we examine realistic scenarios with many nodes and analyse the influence of nodes that have different capabilities and that are not participating in MorphMix all the time. We also look at how optimising the quality of anonymous tunnels in terms of throughput affects collusion detection, analyse the influence if there are different numbers of nodes in the different subnets, and examine the effect of using different tunnel lengths than five.

7.1 Joining MorphMix for the first Time

In our first analysis, we want to examine what happens if a node joins MorphMix for the first time with an empty extended selections list. It is reasonable to assume that a node first has to set up several anonymous tunnels until the correlation distribution starts getting its typical shape and the initiator can make valid decisions about whether a new extended selection is good or malicious.

We use the same setting as in Chapter 6 and vary the percentage of malicious nodes from 0–40%. Malicious nodes attack always using the optimal attack level. We are interested in the the fraction of malicious tunnels among the accepted tunnels (f_{o_m}) and the fraction of false positives (f_{r_g}) depending on the number of tunnels that have been set up. Figure 7.1 shows the results. The data are represented as a rolling average over the 200 most recently set up anonymous tunnels. Below each individual graph, the optimal attack level is given in parenthesis.



a) 0% malicious (-) b) 5% malicious (4) c) 10% malicious (5)

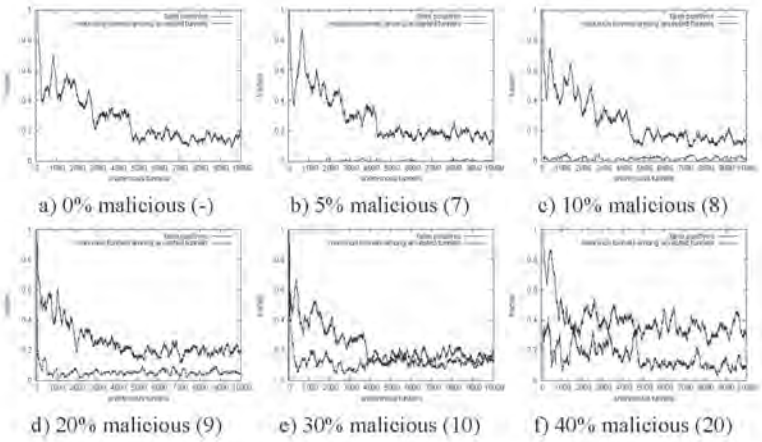
d) 20% malicious (6) e) 30% malicious (7) f) 40% malicious (14)

Figure 7.1: *Performance with 10000 nodes.*

Starting with an empty internal table, the fraction of false positives is quite high at the beginning but starts dropping quickly. Recalling that according to Section 5.5.1, a new tunnel is set up every two minutes on average, and considering that a large fraction of false positives only increases this rate because if a tunnel is rejected, another is set up right away, the fraction of false positives should drop below 0.5 within a few hours after having joined MorphMix. After having set up about 1000 tunnels, the fraction of false positives has reached approximately 0.2 and remains at this level. This implies that in the long run, one out of five tunnels that are actually good are rejected by the initiator. However, this no significant problem because as discussed in Section 5.5.1, setting up anonymous tunnels is a background process to keep a pool of usable tunnels ready at any time. Similarly, Figure 7.1 shows that it

takes setting up several tunnels until the fraction of malicious tunnels among the accepted tunnels has reached a "steady state", but in general, the fraction of malicious tunnels among the accepted tunnels is not especially large in the beginning. This is surprising, but can be explained with the way the correlation distribution is built starting with an empty extended selections list. After a few tunnels have been set up, the shape of the correlation distribution does not yet look as in Figure 6.2, but consist of several small peaks. Since the left flank of the first peak is used as a reference to determine the correlation limit according to Algorithm 2 in Section 5.6.3, the resulting correlation limit is usually chosen too small in the beginning. One can also say the correlation limit is selected conservatively in the beginning to guarantee only a small fraction of malicious tunnels, but at the cost of more false positives.

Since the size of MorphMix is determined by the number of different /16 subnets in which the nodes are located, we conclude that joining the system for the first time when there are 10000 different subnets works well. In Section 5.6, we mentioned there are 56559 public /16 subnets in the Internet. What about joining the system for the first time if there are nodes present in nearly all /16 subnets? Figure 7.2 repeats the analysis for a system with 50000 nodes in 50000 different /16 subnets. This time, we set up 10000 tunnels.



a) 0% malicious (-) b) 5% malicious (7) c) 10% malicious (8)

d) 20% malicious (9) e) 30% malicious (10) f) 40% malicious (20)

Figure 7.2: *Performance with 50000 nodes.*

The results are very similar to those in Figure 7.1, with the exception

that it takes setting up about 4000 anonymous tunnels until the fraction of false positives reaches and remains at approximately 0.2. But still, joining the system for the first time is not a problem, although the fraction of false positives will be relatively high during the first hours. To reduce this learning phase, a node could try to fill its extended selections list much more quickly by asking other nodes that have been participating in MorphMix for a while about extended selections they have stored in their lists. But carelessly giving away the information about extended selections collected during the setup of the own anonymous tunnels could allow others to learn more about these tunnels. In addition, malicious nodes could distribute forged extended selections to confuse honest nodes. We therefore choose not to employ any such mechanism to speed up a node's filling of its extended selections list.

The fact that it takes setting up some anonymous tunnels until a node can make reasonable judgements about whether a tunnel is good or malicious has some implications. First of all, we should keep the knowledge about previously established tunnels in case a node leaves MorphMix and joins again later. Therefore, the complete extended selections list is periodically stored on disk. Furthermore, we have already discussed in Section 5.5.4 why relaying traffic for others is good to increase the own anonymity. The collusion detection mechanism provides additional incentive for a user to keep her node up and running and relay data for others even when she does not need to access the Internet anonymously: the node continues to set up anonymous tunnels to collect information about the system, which keeps the data in the extended selections list table up-to-date.

7.2 Honest and Malicious Nodes in the same /16 Subnet

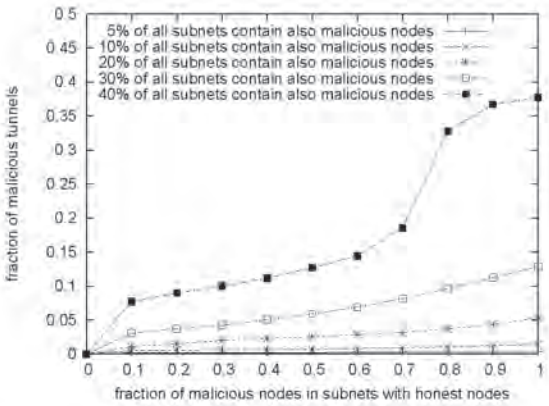
In Chapter 6 and Section 7.1, we have always assumed that some /16 subnets contain only honest and some other subnets contain only malicious nodes. In reality, however, it is more likely that the adversary controls some nodes in subnets where there are also honest nodes.

We analyse a system with 100000 nodes in 10000 different /16 subnets. Every subnet contains 10 nodes, and all nodes have the same probability of being offered in honest selections. We vary the number of subnets that contain both honest and malicious nodes from 500–4000; all other subnets contain only honest nodes. We examine f_{a_m} depending on the fraction of malicious

nodes in the subnets where there are both honest and malicious nodes, which corresponds to the fraction the adversary controls in these subnets according to Section 5.4.2. We vary this fraction from 0–1, where 0 corresponds to no malicious nodes in the particular subnet and 1 corresponds to exclusively malicious nodes in the subnet (which corresponds to the basic scenario used in Chapter 6. We also assume that there are at most ten nodes in a single /16 subnet that are all located in different /24 subnets. This implies that malicious nodes can never overwrite honest nodes in the node lookup lists of honest nodes (see Section 5.7.3). We set up 5000 anonymous tunnels, whereas each tunnel consists of five nodes in total. The adversary attacks always using the optimal attack level. Figure 7.3 illustrates f_{a_m} depending on the fraction controlled by the adversary in subnets with malicious nodes and the percentage of subnets that contain both honest and malicious nodes. The table lists the optimal attack levels to maximise f_{a_m}.

We can see that if the fraction the adversary controls in subnets with malicious nodes gets larger, the number of malicious tunnel among the accepted tunnels also increases. This is not surprising because recalling the peer discovery mechanism in Section 5.7 and looking at a single /16 subnet, the fraction of malicious nodes that are stored in the corresponding list of nodes grows as the fraction the adversary controls of this subnet gets larger. Consequently, the probability an honest node picks a malicious node from this subnet as a new neighbour increases, which implies the probability that the first intermediate node in one of the tunnels of an honest node is malicious also gets larger. For the same reason, the probability that a malicious node from this subnet is included in a selection offered by a honest node gets larger, which increases the probability a malicious node is selected from this subnet as the final node in a tunnel of an honest node. Figure 7.3 also illustrates that if there are malicious nodes in 40% of all subnets, the collusion detection mechanism only fails if the adversary controls more than a fraction of 0.7 in these subnets.

Note that what counts is mostly the fraction controlled by the adversary in a /16 subnet, and not so much the absolute numbers of malicious and honest nodes. If the adversary has full control over a subnet (corresponding to a fraction of 1), then it does not matter if he runs only a few or 65533 nodes because an honest node stores exclusively malicious nodes in the list of nodes at the corresponding entry in the node lookup list. Similarly, assuming the nodes are located in different /24 subnets (see Section 5.7.3), it does not matter if there one malicious and four honest or two malicious and eight honest



% of subnets with malicious nodes	fraction controlled by the adversary in subnets with malicious nodes									
	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
5%	9	8	7	6	6	6	5	5	5	4
10%	10	8	7	7	7	6	6	5	5	5
20%	10	9	8	8	7	7	7	6	6	6
30%	11	10	10	9	9	8	8	8	7	7
40%	11	10	10	10	9	9	9	14	14	14

Figure 7.3: *f_{0m} depending on the fraction controlled by the adversary in subnets with malicious nodes.*

nodes in a /16 subnet, because in both cases, the adversary controls a fraction of 0.2 of the /16 subnet, which means it is likely that a fraction of 0.2 of all nodes that are stored in the list of nodes of this /16 subnet in node lookup list are malicious. We can also say that a /16 subnet that contains many honest nodes is more resistant to the adversary than one with only a few such nodes. If there are eight honest nodes in a /16 subnet and the adversary manages to run two nodes in this subnet, he controls only a fraction of 0.2. If he runs two malicious nodes in a subnet with only two honest nodes, he controls a fraction of 0.5. So every additional honest node in a /16 subnet increases the resistance of this subnet to the adversary. Note that due to the restriction for practical reasons of the length of a list of nodes to ten and if there are more

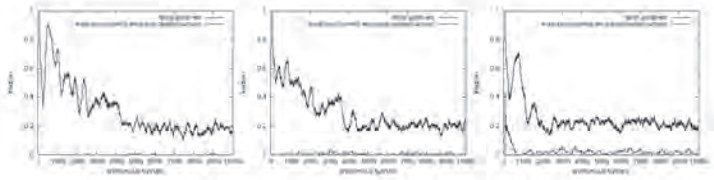
than ten nodes in a /16 subnet, it is possible that the fraction of malicious nodes an honest node stores in the list of nodes of this particular /16 subnet is larger than the fraction the adversary controls in this /16 subnet. However, this requires the malicious nodes in this /16 subnet to be located in several different /24 subnets.

7.3 Large Realistic Systems

One goal of MorphMix is to provide anonymity for a large number of users. We therefore analyse the performance of the collusion detection mechanism assuming there are nodes in nearly all public /16 subnets. We always look at two systems: one system with 100000 honest nodes in 50000 subnets and a large system with 1000000 honest nodes in 50000 subnets. We assume the adversary manages to control 10000 malicious nodes that are located in 1000, 2000, 5000, or 10000 different subnets that also contain honest nodes. In addition, we assume that malicious nodes establish virtual links to other nodes as frequently as honest nodes and all nodes in the same /16 subnet are located in different /24 subnets. The latter implies that a node that is inserted into the node lookup list of an honest node can never overwrite another node that is located in the same /24 subnet (see Section 5.7.3). When offering selections, malicious nodes attack always using the optimal attack level. First, we assume that every node has an abundant capacity, i.e. every node can always accept relaying further anonymous tunnels. In addition, all nodes are continuously participating in MorphMix. In Section 7.3.2, we look at what happens if the nodes have different capabilities and if not all of them are participating in MorphMix all the time.

7.3.1 The Nodes have Abundant Capabilities and are Continuously Participating in MorphMix

We start by examining the smaller system with 100000 nodes. Starting with an empty internal table, 10000 anonymous tunnels are set up. Figure 7.4 shows the fraction of false positives and the fraction of malicious tunnels among the accepted tunnels. The data are again represented as a rolling average over the 200 most recently set up anonymous tunnels. Note that we do not graphically depict the results for malicious nodes in 2000 subnets, but still include them in the table.



a) 1000 subnets with malicious nodes (7) b) 5000 subnets with malicious nodes (9) d) 10000 subnets with malicious nodes (12)

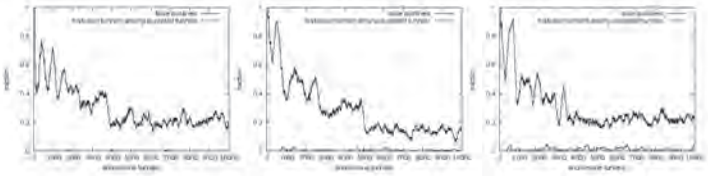
subnets with mal. nodes	optimal attack level	f_{a_m} with collusion detection	f_{a_m} without collusion detection
1000	7	0.0009	0.0142
2000	9	0.0022	0.0261
5000	9	0.0068	0.0482
10000	12	0.0248	0.0811

Figure 7.4: *100000 honest, 10000 malicious nodes (abundant capabilities, always participating).*

We see that for the adversary, it is much better to control only one or a few nodes in as many different subnets as possible than to have nearly full control over a smaller number of subnets. Owning ten nodes in 1000 different subnets corresponds to controlling ten of twelve nodes in these subnets (a fraction of 0.83) and allows the adversary to control the first intermediate and final node in a fraction of 0.0009 of all tunnels that are accepted by the initiator. Having one malicious node in 10000 different subnets means controlling one of three nodes in these subnets (a fraction of 0.33), but allows the adversary to control the first intermediate and final node in a fraction of 0.0248 of all tunnels. The table in Figure 7.4 also lists f_{a_m} if no collusion detection were employed. In this case, the adversary would offer always only malicious nodes in malicious selections. So looking at the example with malicious nodes in 1000 different subnets, we can see that a fraction of 0.0142 of all tunnels used be the initiator would be malicious if no collusion detection were used. This is already quite a low value because the node lookup list (see Section 5.7) contains malicious nodes for at most 1000 subnets and honest nodes for about 50000 subnets. Therefore, the probability a malicious node is picked as the first intermediate node is already quite small. Still, employing the cover traffic mechanism reduces f_{a_m} about 16 times to 0.0009. With an increasing number of subnets in which the adversary controls nodes, this reduction factor becomes smaller.

with 2000 subnets it is about 12, with 5000 subnets about seven, and with 10000 subnets about 3.5.

Increasing the number of honest nodes to 1000000 gives the results in Figure 7.5. The main difference is that the adversary controls a smaller fraction of the nodes in the subnets with malicious nodes. For instance, if the adversary operates two nodes in 5000 subnets, he no longer controls a fraction of 0.5 (two of four nodes) of these subnets, but only a fraction of about 0.09 (two of 22 nodes).



a) 1000 subnets with malicious nodes (9)

b) 5000 subnets with malicious nodes (14)

c) 10000 subnets with malicious nodes (17)

subnets with mal. nodes	optimal attack level	f_{a_m} with collusion detection	f_{a_m} without collusion detection
1000	9	0.0004	0.0086
2000	12	0.0009	0.0093
5000	14	0.0042	0.0181
10000	17	0.0128	0.0354

Figure 7.5: *1000000 honest, 10000 malicious nodes (abundant capabilities, always participating).*

Compared to the setting with 100000 nodes, f_{a_m} gets smaller both with and without collusion detection. This is not surprising when looking at the results in Figure 7.3 and undermines that increasing the number of honest nodes adds to the resistance of MorphMix to attacks.

7.3.2 The Nodes have Different Capabilities and Up-Times

Up to now, we have always assumed that all nodes are equal: they are always capable to accept further anonymous tunnels and all nodes are participating in MorphMix all the time. In reality, this is not the case. Some users have slow dial-up connections and pay for the time they are online, which means

their nodes are usually participating in MorphMix for only a relatively short period during a day. In addition, these nodes do not have the capabilities to relay many anonymous tunnels of other nodes because of the bandwidth limitations and are therefore offered less frequently in selections. Then there are nodes with very good network connectivity that are potentially participating in MorphMix continuously. These are nodes run at universities or by users having fast DSL connections. And there is a range of nodes in between these two extremes.

It is difficult to estimate what nodes would participate in MorphMix. As a basis, we use a measurement study [109] about the peers participating in the Napster [39] and Gnutella [39] file-sharing systems. One main result of the study is the distribution of the bandwidths of the peers, and based on these results, we define a distribution for the bandwidths of MorphMix nodes that we assume to be realistic. For instance, according to the study, only about 10% of all peers have slow dial-up connections (at most one ISDN channel) and about 15% of all peers have very fast ones (T1 or T3). In between, there is a range of peers with ADSL, Cable, and DSL connections. We assume the bandwidth of MorphMix nodes is similarly distributed as in Napster/Gnutella. Depending on the bandwidth of a MorphMix node, we also define acceptance probabilities, which is the probability a node accepts to relay anonymous tunnels when it is picked as a new neighbour. If the new neighbour does not accept to relay tunnels, the virtual link is terminated and a new neighbour is picked from the same /16 subnet. It is reasonable to assume that nodes with good Internet connections accept to relay the data of others with a higher probability than nodes with slower Internet connections. Table 7.1 illustrates the distribution for the bandwidths of MorphMix nodes and the acceptance probabilities. Note that these assumptions are only valid for honest nodes. We describe a different model for malicious nodes below.

Table 7.1: *Realistic bandwidth distribution of MorphMix nodes.*

node type	bandwidth (Kb/s) up-stream	bandwidth (Kb/s) down-stream	percentage of all nodes	acceptance probability
ISDN	64	64	10	0.05
$ADSL_{256}$	64	256	25	0.1
$ADSL_{512}$	128	512	25	0.2
DSL_{512}	512	512	25	0.5
T1	1544	1544	10	0.8
T3	4632	4632	5	0.95

Looking at Table 7.1, we can see that we assign ISDN nodes a very small acceptance probability of 0.05, which implies that these nodes are only capable of accepting anonymous tunnels in one out of 20 cases when they are picked as a new neighbour. Conversely, we assume nodes with fast Internet connections can handle most of the requests to relay further anonymous tunnels and we therefore assign T1 and T3 nodes an acceptance probability of 0.8 and 0.95, respectively. Note that we have not explicitly listed nodes with Cable connections because the bandwidths they offer are the same as ADSL or DSL connections. Therefore, the ADSL and DSL nodes in Table 7.1 also include nodes with Cable connections.

A second valuable result from the measurement study are the up-times of the peers. It shows that the probability a Napster/Gnutella peer is connected to the Internet at any time is nearly evenly distributed between zero and one, with the exception that hardly any peer is nearly never or nearly always connected. It is reasonable to assume that the peers with dial-up connections are connected to the Internet for only a relatively short time and that peers with fast T1 and T3 connections are nearly always online. In MorphMix, we assume that nodes are always participating in MorphMix during the time they are connected to the Internet. Using the results of the measurement study, we therefore model the probability a MorphMix node is participating at any time as follows:

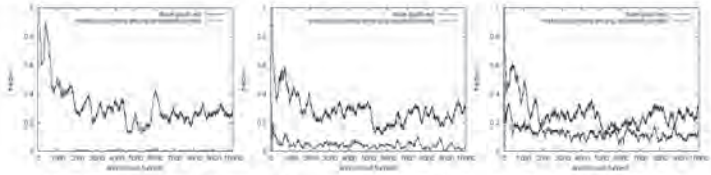
- The ISDN nodes are participating in MorphMix during one hour a day, which means the participation probability is $1/24$.
- The T1 and T3 nodes have a participation probability of 0.9.
- All other nodes get randomly a participation probability between $1/24$ and 0.9.

To be most effective, the adversary makes sure that the malicious nodes are participating in MorphMix as often as possible. In addition, to be involved in as many anonymous tunnels as possible, the malicious nodes should always accept further anonymous tunnels. We therefore assign all malicious nodes per default an acceptance probability and a participation probability of one.

When analysing the performance of the collusion detection mechanism, the participation probabilities are used as follows: we assume the initiator sets up one anonymous tunnel every two minutes (see Section 5.5.1). This implies that 30 anonymous tunnels are set up during one hour. It can also be assumed that nodes are connected to the Internet for a certain period and then offline for another; it is not likely that on- and offline periods follow each other rapidly. Consequently, at the beginning, it is determined for each node if it

is participating in MorphMix during the following hour (which corresponds to 30 anonymous tunnels) according to its participation probability. After 30 anonymous tunnels have been set up, this procedure is repeated. Since an ISDN node has a participation probability of 1/24, this implies that on average, such a node is continually participating in MorphMix for one hour during 24 hours, which is a reasonable assumption.

We analyse how well the collusion detection mechanism copes with the realistic acceptance and participation probabilities defined above. We start with 100000 honest nodes. Figure 7.6 illustrates the performance of this scenario.



a) 1000 subnets with malicious nodes (4)

b) 5000 subnets with malicious nodes (7)

c) 10000 subnets with malicious nodes (10)

subnets with mal. nodes	optimal attack level	f_{a_m} with collusion detection	f_{a_m} without collusion detection
1000	4	0.0028	0.0534
2000	5	0.0091	0.1004
5000	7	0.0447	0.2290
10000	10	0.1242	0.4254

Figure 7.6: *100000 honest, 10000 malicious nodes (different capabilities and participation probabilities).*

The fraction of malicious tunnels among the accepted tunnels is larger than in Figure 7.4. This makes sense because taking the participation probabilities into account means that at any time, there are about $15 \cdot 0.9 + 75 \cdot (0.9 + 1/24)/2 + 10 \cdot 1/24 \approx 46.5\%$ of all honest nodes participating. With 100000 honest nodes in 50000 /16 subnets, this means that at any time, there are several subnets where no honest node is participating. In addition, using the data from Table 7.1, 60% of all nodes have an acceptance probability of at most 0.2. Recalling how honest nodes pick their neighbours (see Section 5.7.2), this implies that even if there are honest nodes for a given /16 subnet in the initiator's node lookup list, it may happen that none of them can be used as a

new neighbour. This significantly increases f_{a_m} compared to Figure 7.4.

Nevertheless, the collusion detection mechanism still works very well because f_{a_m} is also increased if no collusion detection mechanism is used. Comparing f_{a_m} with and without employing the collusion detection mechanism, then the relative gain in Figures 7.4 and 7.6 depending on the number of /16 subnets with malicious nodes is comparable.

Considering that an average honest node is no longer always participating in MorphMix and does not always accept further anonymous tunnels leads us to the notion of the *overall acceptance probability* of honest and malicious nodes. If there are n_h honest nodes with acceptance probabilities acc_{h_i} and participation probabilities $part_{h_i}, 1 \leq i \leq n_h$, the overall average acceptance probability of any honest node is defined as:

$$acc_h = \frac{1}{n_h} \sum_{i=1}^{n_h} acc_{h_i} \cdot part_{h_i} \tag{7.1}$$

Similarly, if there are n_m malicious nodes with acceptance probabilities acc_{m_i} and participation probabilities $part_{m_i}, 1 \leq i \leq n_m$, the overall average acceptance probability of any malicious node is defined as:

$$acc_m = \frac{1}{n_m} \sum_{i=1}^{n_m} acc_{m_i} \cdot part_{m_i} \tag{7.2}$$

Looking at the scenario analysed in Figure 7.6(b) with the adversary operating nodes in 5000 subnets and applying (7.1), we get

$$acc_h = 0.1 \cdot 0.05 \cdot \frac{1}{24} + (0.25 \cdot 0.1 + 0.25 \cdot 0.2 + 0.25 \cdot 0.5) \cdot \frac{\frac{1}{24} + 0.9}{2} + (0.1 \cdot 0.8 + 0.05 \cdot 0.95) \cdot 0.9 \approx 0.207$$

This means that picking any honest node at any time, it is both participating and can accept further anonymous tunnels with probability of about 0.207. Since malicious nodes are always participating and can always accept anonymous tunnels, $acc_m = 1$. So compared to Figure 7.4 where $acc_h = 1$,

honest nodes in the scenario of Figure 7.6 are only about one fifth "as useful" on average.

Taking different capabilities and participation probabilities into account, we have to modify our notion about the adversary controlling a certain fraction of a /16 subnet (see Section 5.4). The fraction the adversary controls of a subnet depends on the nodes that are online on average in this subnet and their acceptance probabilities. If there are $n_{h,s}$ honest nodes in a /16 subnet s with acceptance probabilities $acc_{h_i,s}$ and participation probabilities $part_{h_i,s}, 1 \leq i \leq n_{h,s}$, and $n_{m,s}$ malicious nodes in the same subnet s with acceptance probabilities $acc_{m_i,s}$ and participation probabilities $part_{m_i,s}, 1 \leq i \leq n_{m,s}$, the average fraction $f_{c,s}$ the adversary controls of this subnet is defined as:

$$f_{c,s} = \frac{\sum_{i=1}^{n_{m,s}} acc_{m_i} \cdot part_{m_i}}{\sum_{i=1}^{n_{h,s}} acc_{h_i} \cdot part_{h_i} + \sum_{i=1}^{n_{m,s}} acc_{m_i} \cdot part_{m_i}} \quad (7.3)$$

If the honest nodes are evenly distributed among all s /16 subnets, (7.3) can also be written as:

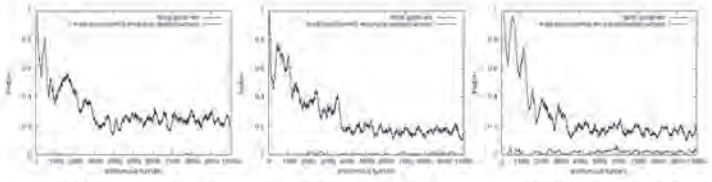
$$f_{c,s} = \frac{\sum_{i=1}^{n_{m,s}} acc_{m_i} \cdot part_{m_i}}{\frac{n_h}{s} \cdot acc_h + \sum_{i=1}^{n_{m,s}} acc_{m_i} \cdot part_{m_i}} \quad (7.4)$$

Applying the scenario from Figure 7.6 with 5000 malicious nodes to (7.4), we get

$$f_{c,s} \approx \frac{2}{2 \cdot 0.207 + 2} \approx 0.829$$

for the subnets that contain malicious nodes. Considering the decrease in the overall acceptance probability of honest nodes (acc_h) from 1 to 0.207 and the increase of the fraction an adversary controls of the subnets with malicious nodes from 0.5 to 0.829 in the case when there are malicious nodes in 5000 subnets explains the significantly larger f_{a_m} when comparing Figures 7.6 and 7.4.

Increasing the number of honest nodes to 1000000 leads to the results illustrated in Figure 7.7.



a) 1000 subnets with malicious nodes (7)

b) 5000 subnets with malicious nodes (11)

c) 10000 subnets with malicious nodes (14)

subnets with mal. nodes	optimal attack level	f_{a_m} with collusion detection	f_{a_m} without collusion detection
1000	7	0.0006	0.0171
2000	8	0.0014	0.0295
5000	11	0.0052	0.0550
10000	14	0.0220	0.0923

Figure 7.7: *1000000 honest, 10000 malicious nodes (different capabilities and participation probabilities).*

Like in Figure 7.6, f_{a_m} is larger than when all honest nodes are participating all the time and always accept further anonymous tunnels (see Figure 7.5). Applying (7.3) again assuming the adversary operates nodes in 5000 subnets results in

$$f_{c,s} \approx \frac{2}{20 \cdot 0.207 + 2} \approx 0.326,$$

which is smaller than the $f_{c,s}$ of 0.5 of the scenario in Figure 7.4(b) but also larger than the $f_{c,s}$ of 0.09 of the scenario in Figure 7.5(b). It is therefore reasonable that the f_{a_m} in Figure 7.7 are between those of Figures 7.4 and 7.5.

7.4 Optimising the Quality of Anonymous Tunnels

Taking into account nodes with very different bandwidths, we must think about the quality of the nodes along an anonymous tunnel. Basically, the slowest node in a tunnel determines the maximum throughput of the tunnel:

if one intermediate node is an ISDN node and all the others, including the initiator, are T3 nodes, the throughput of the tunnel will be at most 64 Kb/s. This is a significant problem because hardly any user is willing to sacrifice her fast Internet connection for anonymity if all she gets is the equivalent of a slow dial-up connection.

The only way to cope with this problem is to make sure no nodes with slow Internet connections are present along tunnels of initiators that have good Internet connections. In practice, this means that the initiator specifies a minimum quality in terms of bandwidth for the nodes it accepts in its anonymous tunnels, which causes the intermediate nodes to offer only nodes that fulfil these requirements in their selections. Table 7.2 specifies what we believe could be acceptable intermediate and final nodes depending on the capabilities of the initiator.

Table 7.2: *Acceptable intermediate and final nodes*

	intermediate and final nodes.					
initiator	ISDN	$ADSL_{256}$	$ADSL_{512}$	DSL_{512}	T1	T3
ISDN	•	•	•	•	•	•
$ADSL_{256}$			•	•	•	•
$ADSL_{512}$				•	•	•
DSL_{512}				•	•	•
T1				•	•	•
T3				•	•	•

For initiators that are ISDN nodes, every other node is suitable to be used in their anonymous tunnels because no other node has a worse Internet connectivity than the initiator. For $ADSL_{256}$ nodes, ISDN and other $ADSL_{256}$ are no good choice for the intermediate or final nodes in their tunnels because this would limit the throughput to 64 Kb/s, which is only one fourth of the initiator's down-stream bandwidth. Even $ADSL_{512}$ nodes in tunnels of $ADSL_{256}$ nodes limit the throughput to 128 Kb/s, but we believe that a 50% throughput reduction is acceptable for getting anonymity. If the initiator has an $ADSL_{512}$, DSL_{512}, T1, or T3 connection to the Internet, the nodes along a tunnel should be at least DSL_{512} nodes to guarantee a reasonably good end-to-end performance to the initiator.

However, we can expect that these measures to improve the throughput of anonymous tunnels will increase the adversaries chances to compromise an anonymous tunnel. Just like when we introduced different capabilities and

participation probabilities of the nodes in Section 7.3.2, the effective number of honest nodes for initiators with fast Internet connections becomes smaller because nodes with slow connections are no longer offered in selections to them. Figure 7.9 illustrates the performance with 100000 honest nodes. We assume the initiator has an Internet connection corresponding to ADSL_{512} or faster, which corresponds to the worst case since the spectrum of nodes that can be offered in selections to these nodes is smallest according to Table 7.1.

a) 1000 subnets with malicious nodes (7)

b) 5000 subnets with malicious nodes (11)

c) 10000 subnets with malicious nodes (14)

subnets with mal. nodes	optimal attack level	f_{a_m} with collusion detection	f_{a_m} without collusion detection
1000	4	0.0045	0.0574
2000	5	0.0113	0.1126
5000	7	0.0523	0.2404
10000	10	0.1417	0.4464

Figure 7.8: *100000 honest, 10000 malicious nodes (with tunnel optimisation).*

As expected, f_{a_m} is larger than in Figure 7.6 where no optimisation of the throughput of anonymous tunnels was made. However, the difference is quite small although DSL_{512}, T1, and T3 account for only 40% of all nodes according to Table 7.1. The reason is that from the point of view of a node with a fast Internet connection, not very much has changed because looking at the acceptance probabilities in Table 7.1 shows that nodes with slow Internet connection accept only infrequently to relay tunnels after they have been picked as a new neighbour. This implies that by requesting a minimum quality for the nodes offered in selection for nodes with good Internet connections, we have merely removed occasional occurrences of nodes with slow connections in these selections. It should be noted that for ADSL_{256} nodes, f_{a_m} is slightly smaller than in Figure 7.8 and for ISDN nodes, nothing has changed compared to the results in Figure 7.6.

The small difference between the results shown in Figures 7.6 and 7.8 can also be explained using our notation introduced in Section 7.3.2. Only considering DSL_{512}, T1, and T3 nodes and using (7.1), we get the following overall average acceptance probability of any honest node:

$$acc_h = 0.25 \cdot 0.5 \cdot \frac{\frac{1}{24} + 0.9}{2} + (0.1 \cdot 0.8 + 0.05 \cdot 0.95) \cdot 0.9 \approx 0.174$$

This is relatively close to the 0.207 when no optimisation of tunnels was made. Similarly, using (7.4) to compute the average percentage $f_{c,s}$ the adversary controls in the subnets that contain malicious nodes for the case illustrated in Figure 7.8(b) results in

$$f_{c,s} \approx \frac{2}{2 \cdot 0.174 + 2} \approx 0.852.$$

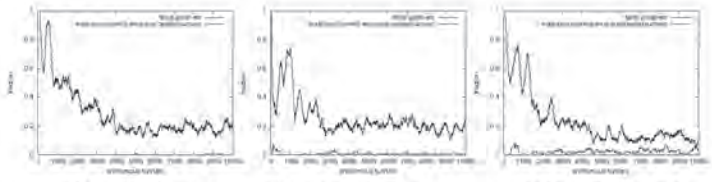
Compared with the results without any optimisation, this is a small increase from 0.829 to 0.852, which again explains the small increase of f_{a_m}. Increasing the number of honest nodes to 1000000 leads to the results illustrated in Figure 7.9.

Again, f_{a_m} is slightly larger than without optimising the throughput of a tunnel. Using (7.4) to compute the average fraction $f_{c,s}$ the adversary controls in the subnets that contain malicious nodes for the case illustrated in Figure 7.9(b) results in

$$f_{c,s} \approx \frac{2}{20 \cdot 0.174 + 2} \approx 0.365,$$

which is close to the 0.326 without any optimisation. We conclude that optimising the throughput of anonymous tunnels does not significantly increase the adversaries chances to compromise anonymous tunnels. In addition, we will see in Section 8.3.3 when analysing the end-to-end performance MorphMix offers that the benefits by optimising the throughput of anonymous tunnels greatly outweighs the small increase of f_{a_m}.

As a side note MorphMix demonstrates how poorly asymmetric access technologies such as ADSL and often also Cable connections are suited for peer-to-peer systems. For traditional client/server applications, such asymmetric access technologies are very well suited but in peer-to-peer systems,



a) 1000 subnets with malicious nodes (7)

b) 5000 subnets with malicious nodes (11)

c) 10000 subnets with malicious nodes (14)

subnet with mal. nodes	optimal attack level	f_{a_m} with collusion detection	f_{a_m} without collusion detection
1000	7	0.0005	0.0179
2000	8	0.0018	0.0316
5000	11	0.0060	0.0571
10000	14	0.0237	0.0951

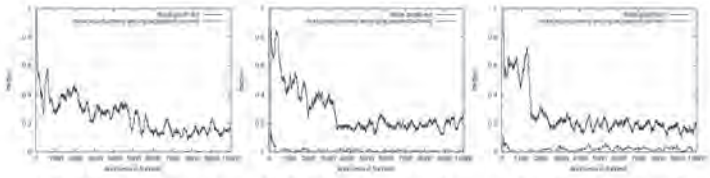
Figure 7.9: *1000000 honest, 10000 malicious nodes (with tunnel optimisation).*

the slow up-link of these nodes becomes a bottleneck. What we have mainly done with our throughput optimisation to make sure that nodes with slow up-links are not present in the tunnels of nodes with fast down-links. Actually, this is not completely true because we have allowed ADSL_{512} nodes with relatively slow 128 Kb/s up-links in tunnels of ADSL_{256} nodes. If these nodes had up-links as fast as their down-links, they would be much more useful for MorphMix and Table 7.2 would look differently.

One can ask if the measures to increase the quality of anonymous tunnels decrease the chances to set up a tunnel successfully in the sense that all intermediate nodes along a tunnel have enough neighbours of the adequate node type they can offer in their selections. But this can be easily solved in practice (and is done in this way in the MorphMix protocol in Appendix A) in the sense that if the number of nodes n_{sel} to be offered in a selection is larger than the number $n_{type \geq type_{min}}$ of neighbours that fulfil the minimal node type requirements as requested by the initiator, the remaining positions in the selection are filled with the $(n_{sel} - n_{type \geq type_{min}})$ next best nodes regarding the node type.

7.5 The Subnets Contain Different Numbers of Honest Nodes

In reality, the participating nodes are not evenly distributed over the /16 subnets in which they are located. We analyse how much this affects the performance of the collusion detection mechanism. The basic setting is the same as in Figure 7.9, with the exception that the number of honest nodes per subnet is distributed linearly over all 50000 subnets such that the subnet with the largest number of honest nodes contains ten times as many honest nodes as the subnet with the smallest number of honest nodes. We also use tunnel throughput optimisation according to Table 7.2 and assume the initiator has an Internet connection corresponding to at least ADSL_{512} or better. Since some subnets contain ten times as many nodes as others, we only consider the case with 1000000 honest nodes and Figure 7.10 illustrates the results.



a) 1000 subnets with malicious nodes (6)

b) 5000 subnets with malicious nodes (10)

c) 10000 subnets with malicious nodes (13)

subnets with mal. nodes	optimal attack level	f_{a_m} with collusion detection	f_{a_m} without collusion detection
1000	6	0.0007	0.0192
2000	7	0.0023	0.0334
5000	10	0.0069	0.0611
10000	13	0.0251	0.0984

Figure 7.10: *1000000 honest, 10000 malicious nodes (different numbers of honest nodes in the /16 subnets).*

The adversary's probability to compromise any tunnel is again slightly larger than in Figure 7.9. This can be explained with the fact that although there are fewer than average honest nodes in half of all subnets, there also more than average honest nodes in the other half. Similarly, half of the malicious nodes are in subnets with relatively many honest nodes and half of them

are in subnets with relatively few honest nodes on average. Since according to Figure 7.3, malicious nodes are the more powerful the fewer honest nodes there are in their subnets, the results in Figure 7.10 indicate that the effect of having some malicious nodes in sparsely populated subnets does slightly outweigh the effect of having some malicious nodes in subnets with many honest nodes.

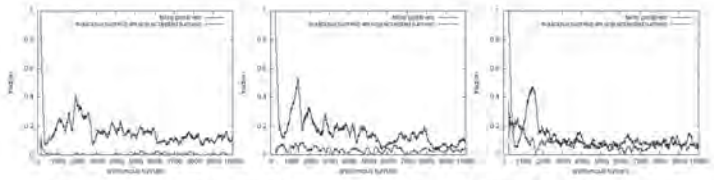
7.6 Varying the Tunnel Length

Finally, we analyse the effect of the tunnel length. Longer tunnels imply more extended selections per tunnel for the initiator to test, which should increase the probability to detect malicious tunnels according to Algorithm 2 in Section 5.6.3. On the other hand, longer tunnels in general also mean worse end-to-end performance as we will analyse in Section 8.3.7. Figure 7.11 illustrates f_{a_m} and the fraction of false positives for a tunnel length of three, four, six, and ten. All other parameters are set as in Figure 7.10, which means we assume the subnets contain different numbers of nodes and that tunnel optimisation is used according to Table 7.2. For completeness, we also include the figures for a tunnel length of five in the table below the graphs.

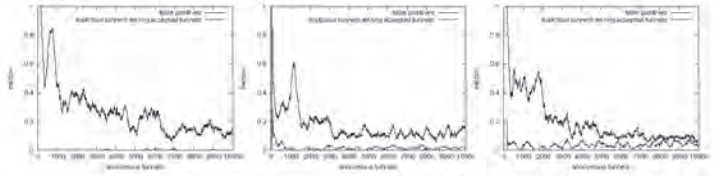
The fraction of malicious tunnels among the accepted tunnels gets indeed smaller if the tunnel length gets longer. However, it does not seem to make sense to increase the tunnel length more and more, because the relative gain by adding a hop gets smaller and smaller. For instance, with malicious nodes in 5000 subnets, the relative decrease in f_{a_m} when going from a tunnel length of five to six is about the same as when going from six to ten. In addition, the rate of false positives also increases as the tunnel length gets longer. With a tunnel length of ten, the fraction of false positives is about 0.4 compared to about 0.2 with five nodes in a tunnel. Note that f_{a_m} without collusion detection is independent of the tunnel length. We will continue analysing the influence of the tunnel length in Section 8.3.7.

7.7 Summary

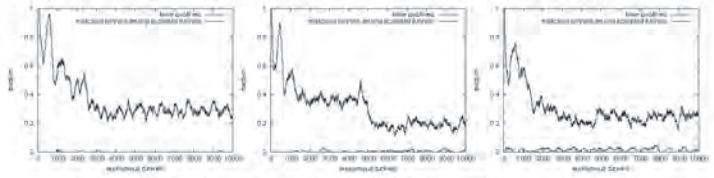
Looking at the realistic scenarios we analysed in this section, we can state that our collusion detection mechanism works indeed well. It significantly reduces f_{a_m} compared to the case if no such mechanism were employed and



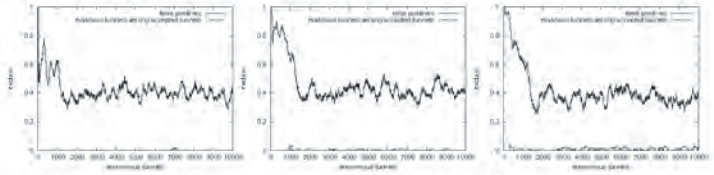
a) tunnel length = 3; malicious nodes in 1000, 5000, and 10000 subnets



b) tunnel length = 4; malicious nodes in 1000, 5000, and 10000 subnets



c) tunnel length = 6; malicious nodes in 1000, 5000, and 10000 subnets



d) tunnel length = 10; malicious nodes in 1000, 5000, and 10000 subnets

subnets with mal. nodes	optimal attack level	f_{a_m} with collusion detection depending on tunnel length 3	4	5	6	10	f_{a_m} without collusion detection
1000	6	0.0070	0.0014	0.0007	0.0004	0.0003	0.0192
2000	7	0.0098	0.0028	0.0023	0.0017	0.0013	0.0334
5000	10	0.0288	0.0157	0.0069	0.0052	0.0038	0.0611
10000	13	0.0752	0.0426	0.0251	0.0161	0.0100	0.0984

Figure 7.11: *1000000 honest, 10000 malicious nodes (diff. tunnel lengths).*

it is very difficult for an adversary to control the first intermediate and final node in a significant percentage of all anonymous tunnels. According to our most realistic scenario in Figure 7.10 that takes into account that nodes are not participating in MorphMix continuously, that nodes are connected to the Internet with different bandwidths, that there are different numbers of honest nodes in the /16 subnets, and that employs tunnel throughput optimisation, the collusion detection mechanism can reduce the number of compromised tunnels that are accepted by the initiator about 27 times with malicious nodes in 1000 subnets, about 14 times with malicious nodes in 2000 subnets, about eight times with malicious nodes in 5000 subnets, and about 4 times with malicious nodes in 10000 subnets.

We conclude that especially when the system gets large, i.e. when there are nodes in most public /16 subnets, the task for the adversary becomes very complicated, because he cannot simply run many nodes in a few subnets but must be present in a large number of different subnets. Of course it could be the case that the adversary owns a part of the public IP address space, for instance a whole class A network. But this only gives him full control over 256 /16 subnets, which only enables him to control the first intermediate and final node in very few tunnels. To be effective, the adversary must have nodes under his control in very many different /16 subnets. Assuming a large system with honest nodes in nearly all public /16 subnets, the results in Figure 7.10 show that the adversary must control nodes in more several 1000 subnets to compromise more than a fraction of 0.01 of the tunnels that are accepted by the initiator.

We have also seen that not only the collusion detection mechanism, but also the peer discovery mechanism helps to keep the fraction of compromised tunnels small. In particular, the way in which information about other nodes is arranged in the node lookup list (see Section 5.7.2) guarantees that the probability an adversary controls the first intermediate node in a tunnel does not depend on the fraction of all nodes he controls, but on the fraction of different /16 subnets in which he controls nodes. Consequently, the fraction of malicious tunnel is already relatively small simply because honest nodes pick their neighbours randomly among all /16 subnets, but the collusion detection mechanism helps to significantly reduce this fraction further.

One minor problem is the learning phase that requires a newly joining node to set up many anonymous tunnels until the collusion detection mechanism starts working correctly. This results in s significantly higher fraction of false positives in the beginning because the correlation limit is selected con-

servatively during this phase to keep the fraction of malicious tunnels small. However, if a node remains active most of the time, this only happens once and even during this learning phase, a user can already use her node to contact servers anonymously, although anonymous tunnels must be set up at a higher rate to compensate for the high fraction of false positives. One could reduce this learning phase by asking other nodes about their extended selections, but we have given arguments in Section 7.1 that this is not necessarily a good idea.

The results in this section have demonstrated that to estimate the strength of an adversary, we cannot simply count the absolute number of honest and malicious nodes, but must take their capabilities and participation probabilities into account. We have therefore introduced the notion of the overall average acceptance probabilities acc_h and acc_m. Honest nodes that are participating in MorphMix more frequently or accept relaying anonymous tunnels more often than others are more valuable to increase the resistance to attacks because they contribute more to keep the fraction $f_{c,s}$ an adversary controls in a particular subnet small.

In general, longer anonymous tunnels increase a user's anonymity because it decreases the percentage of compromised tunnels by the adversary. If minimising f_{a_m} were the only goal, tunnels lengths of ten or even more would be a reasonable choice. However, we will see in Section 8.3 that longer tunnels usually imply worse end-to-end performance and we will therefore continue to use a tunnel length of five nodes as the basis for our evaluation of MorphMix.

Chapter 8

MorphMix Simulation and Results

Although we have shown in the previous chapters that MorphMix protects well from an internal attacker that wants to break the anonymity of MorphMix users, it is still not clear how practical the system is. Since MorphMix users contact servers indirectly, the expected performance is certainly worse than when communicating with a server directly. In addition, there is overhead because nodes not only handle their own data, but also set up tunnels and relay the data of other nodes. To evaluate the expected performance MorphMix can offer to its users, we have implemented a simulator. In this chapter we first describe the MorphMix simulator and the basic settings we have used in our simulation runs. Afterwards, the simulation results are presented.

8.1 The MorphMix Simulator

We decided to implement our own simulator, mainly because existing generic network simulators such as ns-2 [14] simulate the underlying network protocols in great detail and are therefore not capable of simulating a large number of nodes (e.g. 1000) over a large simulated time period (several hours) within a reasonable execution time. The prime goal of the MorphMix simulator is not to reflect the real world as closely as possible, but to deliver valuable information about how the heterogeneity and dynamism of the nodes affect

the end-to-end performance. We therefore basically simulate the MorphMix overlay network consisting of nodes and virtual links and omit many of the details of the underlying physical communication infrastructure.

The core elements of the simulator are the MorphMix *nodes*. Every node has an up- and down-stream bandwidth that specifies the number of bytes it can send and receive at most during a time interval. In addition, nodes may be shut down by their operators or simply crash or may temporarily not be reachable due to network problems. There is a subtle difference between a crash or a shut down of the MorphMix application and a crash of the computer running the application: in the first case, the TCP connections from and to the computer will be correctly terminated [41], which tells the neighbours immediately that the connections and all tunnels using them are no longer usable. These nodes can then send a TERM message (see Appendix A.3.5) through all tunnels that have failed and the initiators of these tunnels can simply terminate all connections between the access program and the client applications (see Section 5.2.4) that are using these tunnels. Since client applications usually inform their users about unexpectedly terminated connections, the users can then re-establish the communication relationships with the servers using other tunnels. In the second case, however, the TCP connections won't be torn down and the crashed computer simply won't react to data sent to it. If the computer is eventually rebooted, it will respond with a TCP RST if it still receives data but it may take a long time until the computer is rebooted at all. In this situation, we must assume that the user eventually notices that a tunnel has failed without being notified by the client application. A computer crash will therefore be more harmful for the end-to-end performance than an application crash or shut down because it takes longer for a user to detect that a particular tunnel is no longer usable. Nodes that are temporarily not reachable due to network problems produce the same effects as a crashed computer: they simply do not respond anymore.

If two nodes are connected (i.e. they are neighbours), there is a *virtual link* between them. A virtual link has a certain delay to simulate the propagation delay when data are sent from a node to another and is an abstraction of a "perfect" TCP connection. This means that if none of the two nodes communicating over a virtual link has crashed or is temporarily blocked, then all data are transmitted with exactly the specified delay. However, virtual links do simulate the TCP flow control mechanism to account for the limited buffer space in hosts. Depending on the delay on a virtual link, this flow control mechanism puts an upper limit on the maximum throughput of any

virtual link.

The simulator itself is event-driven. The continuous time line is chopped into slots of 10 ms. Sending data at a node during a certain time slot triggers the complete reception of these data during a future time slot at another node depending on the delay of the virtual link between the two nodes, their up- and down-stream bandwidths, the TCP flow control mechanism, and the other data that are handled concurrently by the two nodes. Since we assume the delay on any virtual link to be always at least 20 ms, time slots of 10 ms are small enough to accurately analyse the time it takes for data to travel through anonymous tunnels. Of course one could always make the time slots smaller, but this would increase the simulation time inversely proportionally.

We analyse the performance of MorphMix using web browsing as the example application. It can be expected that web browsing will be one of the prime applications if any system for anonymous low-latency Internet access ever gets widely deployed. But web browsing is also a very challenging application for anonymity-providing systems, since web pages often contain several small embedded objects, which results in many sequential request/reply pairs being exchanged between client and server.

8.2 Basic Simulator Settings

There are a variety of parameters that can be specified in the MorphMix simulator. All of our simulations in this section will be based on the same basic setting. These setting can be separated into protocol settings, virtual link settings, tunnel settings, node settings, and web browsing scenario settings. The simulator simulates the entire MorphMix protocol as specified in Appendix A and the basic settings for many parameters are directly inherited from the protocol specification.

8.2.1 Protocol Settings

These settings follow directly from the MorphMix protocol in Appendix A: the *fixed cell length* is 512 bytes and the *cell* and *anonymous connection header length* is 16 bytes. To take into account the overhead from lower-level protocols such as TCP, IP, and link layer protocols, we assume 10% of a node's bandwidth is used for the corresponding headers and trailers and simply deduct these 10% from a node's bandwidth.

8.2.2 Virtual Link Settings

Virtual link settings describe the properties of our abstraction of a TCP connection between two nodes. The *virtual link delay* is the time it takes for the data to travel from one to the other node. For every virtual link between two nodes, we choose a random delay evenly distributed between 20 and 150 ms. The *TCP buffer size* determines the maximum throughput of the virtual link and is set to 64 KB (65536 bytes). For the collusion detection mechanism to work, the nodes that are offered in selections from the same node must change from time to time, which means honest nodes must change their neighbours from time to time. To do so, a newly established virtual link to a new neighbour is only kept for a limited *virtual link lifetime*, which is set to 30 minutes. After this lifetime, the virtual link is not simply torn down, but the node at the other end of the virtual link is no longer advertised in selections. The virtual link itself is kept alive until all tunnels using it have been torn down. Finally, the *virtual link status message interval* determines the interval between two subsequent STAT_REQ/STAT_REP pairs (see Appendix A.3.4) being sent over one virtual link. After a STAT_REP message has been received, we assume it takes a random time evenly distributed between 0 and four minutes before the next STAT_REQ is sent.

8.2.3 Tunnel Settings

The *tunnel length* is the number of nodes in a tunnel and is set to five, which corresponds to the standard tunnel length we used in the previous chapters. Tunnel settings also specify the *tunnel setup interval* time, which is set to two minutes and the *minimum number of tunnels* that should be available at a node at any time, which is set to five. This means that tunnels are regularly set up in the background while making sure that the number of tunnels never falls below a certain minimum. There is a *tunnel lifetime* set to ten minutes, which means that after this lifetime, a tunnel is no longer used for new anonymous connections. Limiting the tunnel lifetime guarantees that virtual links can eventually be torn down after all tunnels using it have reached their lifetime. The *tunnel end-to-end ping interval* specifies the time between two subsequent E2E_PING/E2E_PONG pairs (see Appendix A.4.3) are used to measure the round-trip time (RTT) of a tunnel and is set to two minutes. To detect tunnels that have failed for any reason, for instance because a node along the tunnel has left MorphMix, the *tunnel timeout parameter*, which is

set to 30 seconds, is used. If the initiator sends an E2E_PING message and no reply arrives within the time specified by the tunnel timeout parameter, the tunnel is considered to have failed and is torn down. Similarly, if no reply arrives within this time during the tunnel setup after the initiator has sent a message, the tunnel is also considered to have failed and is torn down.

Our basic assumption throughout this chapter is that setting up anonymous tunnels only fails if one of the nodes along the tunnel fails (for instance by leaving MorphMix) during the setup or a witness that is used to append a node fails. Similarly, we assume anonymous tunnels are never rejected by the initiator, i.e. every tunnel that has been set up successfully can potentially be used for anonymous communication. The influence of other failures during tunnel setup or tunnels that are rejected by the initiator because they are identified as malicious by the collusion detection mechanism will be analysed in Section 8.3.6.

8.2.4 Node Settings

We always set the *number of nodes* in the system to 1000. More nodes are possible but the simulation time grows linearly with the number of nodes. However, we argue that even a system with 1000 nodes delivers reasonable information about how a very large system would perform if certain parameters are set accordingly. To do so, we will always use the maximum selection size of 20 (see Section 5.6.2), which implies the messages to set up a tunnel have their maximum length. We also make sure that at any time, every node has at least 30 neighbours that are willing to relay more anonymous tunnels, which implies that 20 nodes can easily be offered in selections at any time. So even if the system consisted of a million nodes, the tunnel setup messages would not be longer and the local environment every node has to handle would not be larger. We use the same distribution of the nodes' capabilities and participation probabilities (see Table 7.1) as we used for the large realistic systems in Section 7.3.2. This determines the *up- and down-stream bandwidths*, the *participation patterns*, and the *acceptance probabilities* of the nodes. If the system were ten times bigger, there would be ten times as much traffic, but also ten times as many nodes to handle it. Since the distribution of the nodes' capabilities and participation probabilities would be unchanged, we could expect the simulation results to be very similar.

The participation pattern determines when a node joins MorphMix to participate and when it leaves the system again because it is shut down by its op-

erator. When a node leaves the system because of its participation pattern, we always assume the MorphMix application has been shutdown, which means the information about tunnels that have failed is quickly propagated to the initiators (see Section 8.1).

We specify the *cell processing delay* with ten ms, which means that after a cell has been completely received by a node, we assume it can be forwarded to the next node after ten ms. Processing a cell includes reading the data from the socket into the application, performing the cryptographic operations, modifying the cell header, and writing the data into the socket to forward the cell. Only the cryptographic operations are computationally expensive (see Appendix A.2.1 and A.2.2): the 16-byte header of the incoming cell must be decrypted and the one of the outgoing cell encrypted, a layer of encryption must be added or removed resulting in encrypting or decrypting 496 bytes, and computing two hashes over 502 bytes of data, one to check the checksum of the incoming cell and one to build the checksum of the outgoing cell. Modern computers are capable of computing hashes and symmetric encryptions or decryptions at rates of several 10 Mb/s. Since several 10 Mb/s is significantly faster than the bandwidth of any node (see Table 7.1), we can assume that cells can be processed at line speed, and the assumption all cells can be processed withing 10 ms is reasonable. Similarly, we also assume the initiator can handle the multiple encryptions or decryptions of the cell it sends and receives at line speed, i.e. the limiting factor to the number of cells that can be handled is never the rate of encryption or decryption. Finally, the *public-key operation processing delay* is specified with 100 ms and denotes the delay induced by public-key operations during the setup of anonymous tunnels. Using state-of-the-art computers, 100 ms is certainly enough to perform any of the public-key operations we are dealing with, which includes public-key encryptions and decryptions and DH key-exchange operations. Note that although RSA public-key operations take much less time than private-key operations due to the small public-key exponent (see Appendix A.2.1), we assume both of them introduce a processing delay of 100 ms to be on the safe side. We also assume that public-key operations never delay the processing of other cells, which can be achieved in practice by performing public-key operations not in the main loop of the MorphMix program, but in a dedicated thread. We will give a more detailed analysis of the computational overhead imposed by the cryptographic operations in Section 8.3.5.

Per default, nodes or the computers they run on never crash and nodes that are online can always be reached, i.e. their connection to the Internet is never

blocked temporarily. This means that nodes do only leave MorphMix when they are willingly shut down by their operator according to their participation pattern. Note that we will analyse the influence of nodes that crash or that cannot be reached temporarily in Section 8.3.9.

8.2.5 Web Browsing Scenario Settings

The nature of web traffic and web users has been thoroughly analysed and we use appropriate values from scientific literature to model it. The *web request length* is 300 bytes with a probability of 0.8 and 1100 bytes with a probability of 0.2 [72]. The *web reply lengths* (in bytes) follow a ParetoII distribution with parameters $k = 800$ and $\alpha = 1.2$ [44]. For simplicity, we limit the maximum reply size to 1 MB. The number of *embedded objects per page* also follow a ParetoII distribution, this time with parameters $k = 2.4$ and $\alpha = 1.2$ [44]. For simplicity, we limit the maximum number of embedded objects per page to 50. The *reading time* (in seconds) is the time it takes between having completely downloaded a page and initiating the next download. Again, this follows a ParetoII distributed with parameters $k = 60$ and $\alpha = 2.0$ [44]. For simplicity, we limit the maximum reading time to 60 seconds. The *browsing time* is the time a user running a node is browsing the Web per day. For ISDN nodes, we assume the user is always browsing when the node is running; for all other nodes, the browsing time is set to two hours per day. Like the virtual links between two MorphMix nodes, the connection between the final node in a tunnel and the web server has also a delay that is evenly distributed between 20 and 150 ms and also simulates the TCP flow control mechanism using buffer sizes of 64 KB. In addition, the time it takes to establish the TCP connection to the web server is simulated by inserting an additional delay of two times the delay on that connection. Unlike nodes, web servers never crash and can always be reached. The *web request processing delay* is specified with ten ms, which is the time it takes for the web server to start sending a web reply after having completely received a web request. Considering the performance of modern web servers, ten ms is certainly not too small. As mentioned in Section 5.2.4, the initiator must not perform the address resolution using the domain name system (DNS) [76, 77] by itself because this would reveal the identity of the host it intends to contact. Resolving the name is therefore done by the final node in a tunnel. However, querying a name server takes very little time (a few ms) because the name server is usually located nearby the final node (for instance at the access ISP)

and queries and replies use the UDP protocol. In addition, caching replies of name servers means that name servers need not be contacted every time a web request arrives at the final node of a tunnel. We therefore do not take the time to resolve a name into account and assume it is included in the processing delay of ten ms by the final node.

If downloading a page has failed because the tunnel that has been used to download (parts of) the page has failed for any reason, we assume the user reacts by downloading the entire page again. We specify this *download failure reaction time* with five and 30 seconds. For the reasons discussed in Section 8.1 two different reaction times make sense: five seconds is used if the initiator, and therefore also the client application, can be informed about the failed tunnel and the terminated connections; 30 seconds is used if this is not possible and the user has to detect herself whether downloading the page has failed.

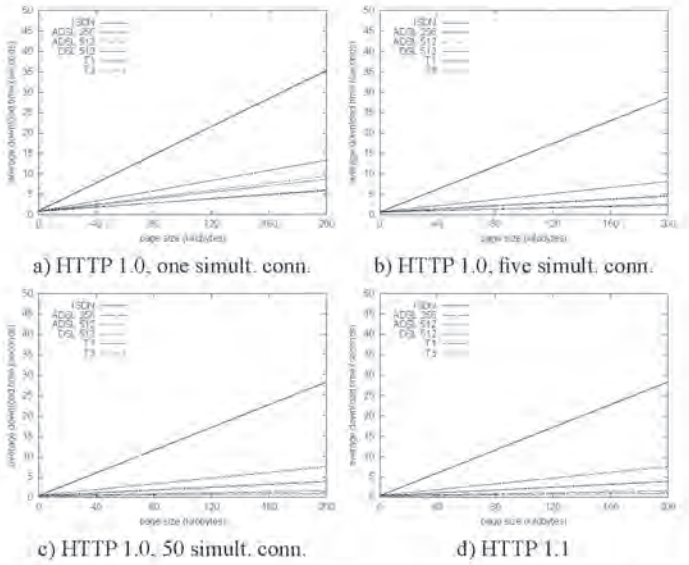
8.3 Simulation Results

We analyse the performance of MorphMix using web browsing as the example application. In all simulations, we simulate four hours of real-time. We usually evaluate the time it takes to completely download a web page, which is defined as the time between sending the first byte of the web request for the index file and receiving the last byte of the complete page. We do not take the time it takes to display a completely downloaded page in the web browser into account. Since there are six different types of nodes depending on their up- and down-stream bandwidths (see Table 7.1), the graphs show the download times for each of these types separately. In general, it turns out that the page download time is nearly linearly dependent on the page size; we therefore use linear regression to plot the graphs.

8.3.1 Contacting the Web Server Directly

We first analyse the performance if the web server is contacted directly. We use both versions 1.0 [8] and 1.1 [45] of HTTP and also vary the upper limit of simultaneous connections when using HTTP 1.0, because browser usually have such an upper limit. The main difference between HTTP 1.0 and 1.1 is that with HTTP 1.0, a dedicated TCP connection is established to download a single web object, e.g. the index file or a single embedded object of a page.

With HTTP 1.1, the entire web page is usually downloaded over a single TCP connection, which significantly reduces the number of TCP connections that have to be established to download an entire web page. Consequently, we expect HTTP 1.1 offers better performance than HTTP 1.0. During the four hours of simulated real-time, about 130000 pages were downloaded with a total size of about 3.2 GB. Figure 8.1 depicts the results.



a) HTTP 1.0, one simult. conn. b) HTTP 1.0, five simult. conn.

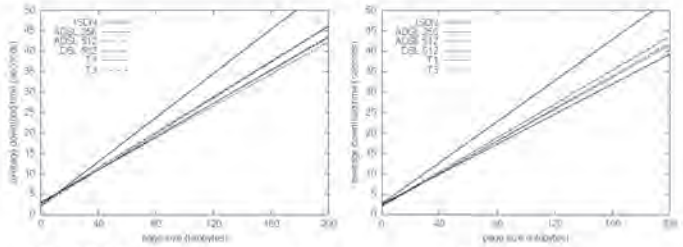
c) HTTP 1.0, 50 simult. conn. d) HTTP 1.1

Figure 8.1: *Download times, accessing the web server directly.*

Not surprisingly, the download times are the shorter the higher the bandwidth with which users are connected to the Internet. We can also see that the performance of HTTP 1.0 is increased if more simultaneous connections are allowed. As expected, HTTP 1.1 generally outperforms HTTP 1.0, but the advantage gets smaller if many simultaneous connections are allowed in HTTP 1.0. We will use the results in Figure 8.1 as a reference for all forthcoming performance measurements in this section. As most browsers have at least a default limit of about five simultaneous connections with HTTP 1.0, we will focus on this case when using HTTP 1.0.

8.3.2 Contacting the Web Server through MorphMix

We analyse the performance when the web server is contacted anonymously through MorphMix. Figure 8.2 shows the download times for HTTP 1.0 with five simultaneous connections and HTTP 1.1.



a) HTTP 1.0, 5 simult. connections b) HTTP 1.1

Figure 8.2: *Download times, accessing the web server through MorphMix.*

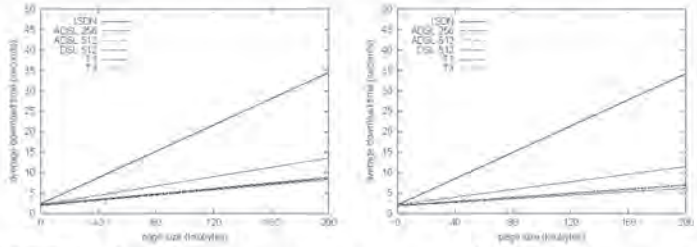
Compared to Figure 8.1, the download times are significantly longer. The better the bandwidth of the node, the more severe the performance penalty from which it suffers. We already discussed this problem in Section 7.4 when we analysed the influence of optimising the throughput of anonymous tunnels on the collusion detection mechanism. The problem stems from intermediate or final nodes with poor Internet connections in tunnels of initiators that have good Internet connections. While this is not a problem for ISDN nodes because they are the slowest node type in our analyses, the impact gets the larger the faster the initiator is. In particular, the end-to-end performance of any node falls below the performance ISDN nodes experience if the web server is contacted directly, as is shown in Figure 8.1. Comparing HTTP 1.0 with HTTP 1.1, Figure 8.1 shows that like when contacting the web server directly, the latter offers slightly better performance. This is again not surprising because using HTTP 1.0, each object of a web page results in setting up one anonymous connection within the anonymous tunnel and one TCP connection between the final node and the web server, whereas with HTTP 1.1, all object of a web page are downloaded through a single anonymous connection and a single TCP connection between the final node and the web server.

We strongly believe that a performance loss so significant as shown in Figure 8.2 would be unacceptable for most users with reasonably fast Internet

connections and hinder MorphMix from acquiring a critical mass.

8.3.3 Optimising the Throughput of Anonymous Tunnels

To optimise the throughput of anonymous tunnels, we employ the minimum quality for intermediate and final nodes depending on the node type of the initiator as introduced in Table 7.2. We have seen in Section 7.4 that the optimisation has only a marginal effect on the adversary's chances to compromise anonymous tunnels. Figure 8.3 shows the corresponding download times.



a) HTTP 1.0, 5 simultaneous connections b) HTTP 1.1

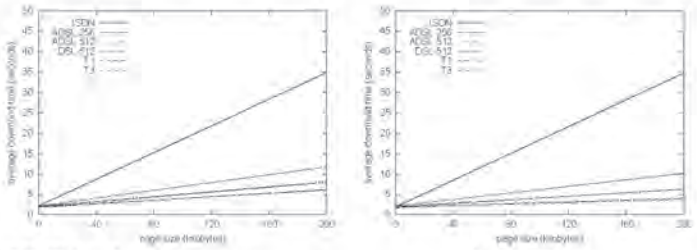
Figure 8.3: *Download times with optimised tunnel throughput.*

Compared to Figure 8.2, the end-to-end performance could be significantly improved. We can also clearly state that the benefits from optimising the throughput of anonymous tunnels greatly outweighs the small increase in the number of compromised tunnels. When optimising the throughput, ISDN nodes have better end-to-end performance than T3 nodes without any optimisation, which is remarkable. Interestingly, the performance of ISDN nodes has also significantly improved although they still accept all other nodes in their anonymous tunnels. The explanation is that ISDN nodes are now only intermediate or final nodes in tunnels of other ISDN nodes. As a result, ISDN nodes must donate only very little bandwidth to handle the traffic of other nodes, which increases the bandwidth available for their own data.

Compared with the results in Figure 8.1 when contacting the web server directly, the download times for large web pages have increased about 20% for ISDN nodes and about 50% for ADSL_{256} nodes. All other nodes only accept nodes with at least DSL_{512} speed in their tunnels and the performance they experience is therefore approximately equal. Their download times are

now about 50% longer than those of $ADSL_{512}$ or DSL_{512} nodes when the web server is contacted directly.

Further optimisation is of course still possible. Looking at Table 7.2, it could make sense to increase the minimum node type for intermediate nodes if the initiator is a $ADSL_{256}$ to DSL_{512} to avoid the slow 128-Kb/s up-stream bandwidth of $ADSL_{512}$ nodes. Similarly, DSL_{512} nodes can be considered as too slow if the initiator is a T1 or T3 node and we therefore raise the minimum node type for intermediate or final nodes for these initiators to T1 nodes. Analysing the download times for this scenario with even more optimised tunnels, we get the download times as illustrated in Figure 8.4.



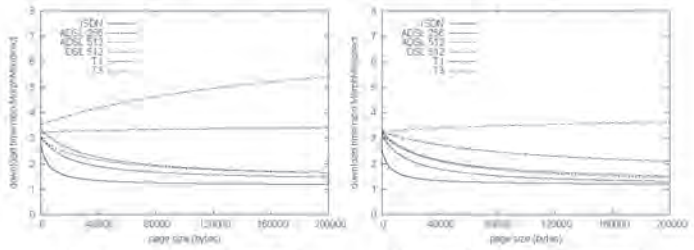
a) HTTP 1.0, 5 simultaneous connections b) HTTP 1.1

Figure 8.4: *Download times with even more optimised tunnel throughput.*

Compared with the results in Figure 8.3, the throughput could be again improved for all nodes with at least $ADSL_{256}$ speed. The biggest relative gain is experienced by T1 and T3 nodes because they got rid of "slow" DSL_{512} nodes in their tunnels. However, the improvement compared to Figure 8.3 is in general relatively small and we do not believe it is worth the greater risk of compromised tunnels. We therefore will continue using the optimisation according to Table 7.2 and the corresponding results in Figure 8.3 as a basis. Another argument to continue our analysis based on this less aggressive optimisation strategy is that in practice, the higher the minimum node type specified by the initiator, the smaller the probability all nodes along a tunnel the initiator sets up fulfil this minimum node type requirements. The reason is that the higher this minimum node type, the smaller the probability a node has enough neighbours of this minimum node type it can offer in selections. As we have discussed in Section 7.4, selections are filled up with the next best neighbours that do not fulfil the minimal requirements and it may there-

fore happen that not all nodes along a tunnel fulfil the minimum node type requirements as specified by the initiator. It is therefore indeed reasonable to use optimisation according to Table 7.2 as a basis, although in practice, a user may always choose to override the default minimum node type she accepts in her tunnels.

For completeness, Figure 8.5 illustrates the ratio of the download times between accessing the web server directly and through MorphMix based on the optimisations in Figures 8.3 and 8.4. We only illustrate the ratio for HTTP 1.1 because the results for HTTP 1.0 are similar.



a) optimisation as illustrated in Figure 8.3(b)

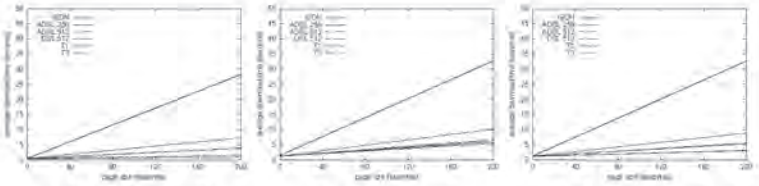
b) optimisation as illustrated in Figure 8.4(b)

Figure 8.5: *Ratio between the download times when accessing the web server through MorphMix and directly using HTTP 1.1.*

We see that for small web pages, the ratio is nearly the same for all tunnels. The reason is that the download time is mainly determined by the RTT, which is several times larger if the web server is accessed through MorphMix. As the web pages get larger, the throughput of the anonymous tunnel becomes more important and we can confirm the results we observed above that especially T1 and T3 nodes profit from the even more optimised tunnels we used to generate the results in Figure 8.4.

Using a single anonymous tunnel to download the web page, it seems we are approaching the limits in terms of end-to-end performance. However, the size of a web page does not unambiguously reflect the complexity to download it because it could be just one file that requires one request/reply pair or it could be composed of an index file and several embedded object, resulting in several request/reply pairs being sent through MorphMix. To assess the pure performance of MorphMix, we simulate file transfers with random file sizes

between one and 200000 bytes. Every file transfer results in exactly one request/reply pair, which means there is no uncertainty as in the web browsing case about the number of embedded objects in a web page. Figure 8.6 depicts the download times when the server is contacted directly, when the optimisations according to Table 7.2 are used, and when the further optimisations as in Figure 8.4 are employed.



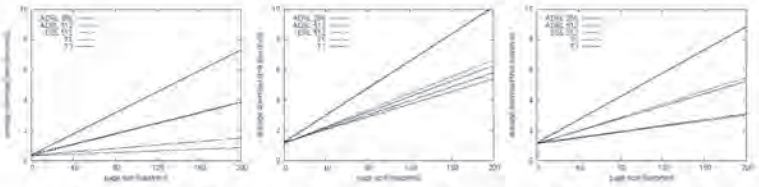
a) direct download b) optimisation according to Table 7.2 c) further optimisation as in Figure 8.4

Figure 8.6: *Download times for a single file.*

Again and not surprisingly, downloading a single file through MorphMix takes longer than downloading it directly. When contacting the server directly, the download times in Figures 8.1(d) and 8.6(a) are virtually the same, which means that downloading a single web page possibly composed of several objects is only marginally slower than downloading a single file of the same size. This can be explained with the relatively short RTT, which means that the time between requesting the embedded objects and starting receiving them is relatively short. Contacting the server through MorphMix using tunnel throughput optimisation according to Table 7.2 and comparing Figures 8.3(b) and 8.6(b), we can see that downloading a single file is notably faster than downloading a web page of the same size. The main reason is the significantly longer RTT than when contacting the server directly: since the delay on a virtual link and on the connection to the web server is 85 ms on average (see Section 8.2), accessing the server through MorphMix and using a tunnel length of five results in an the average RTT of 850 ms compared to 170 ms when the server is contacted directly. If a web page with embedded objects is downloaded through MorphMix, this implies it takes at least 850 ms on average between completely receiving the index file and starting receiving the first bytes of embedded objects. Consequently, there is a gap of at least 850 ms on average during which no data are downloaded to the ini-

tiator. In the single file transfer case, this gap is not present, which explains the difference of the download times. Using the even more optimised tunnel throughput scenario and comparing Figures 8.4(b) and 8.6(c), we observe exactly the same.

With the results in Figure 8.6, we can definitely see the limits of the end-to-end performance MorphMix users may expect. To analyse this in more detail, we depict the download times for a single file again in Figure 8.7, but this time with an y-axis that only ranges from 0-10 seconds. We do not include the download times for initiator that are ISDN nodes because for large files, their download times are significantly longer than ten seconds.



a) direct download b) optimisation according to Table 7.2 c) further optimisation as in Figure 8.4

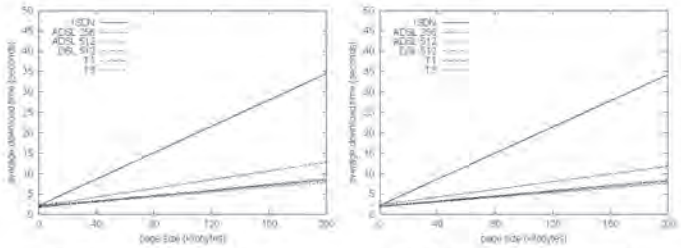
Figure 8.7: *Download times for a single file (more detailed illustration).*

There are three factors that account for the increased download times. The first is the increased RTT, which is clearly visible looking at Figure 8.7 for small file sizes: contacting the server through MorphMix (Figures 8.7(b) and(c)) results in download times that are about one second longer than when contacting the server directly (Figure 8.7(a)). The second component are nodes with poor Internet connections along the tunnels of initiators with good Internet connections. Looking at initiators with ADSL_{256} connections and comparing Figures 8.7(a) and (b), the increased download times with MorphMix cannot only be explained with the increased RTT, but also with the slow up-stream bandwidth of ADSL_{512} nodes that are allowed to be present in the tunnels according to Table 7.2. With the even more optimised tunnels, this is no longer the case and as a result, the increased download times of ADSL_{256} nodes in Figure 8.6(c) can again be explained with the RTT. Finally, the third factor are congested nodes. In general, MorphMix makes use of statistical multiplexing, which means that even if a node handles several tunnels simultaneously, it is likely that it must send or receive data of one or

only a few of them at the same time. However, it may always happen that this is not the case at a node for a short while, which makes this node a temporary bottleneck for all tunnels using it. This can be seen by the increased download times for large files of DSL_{512} nodes comparing Figures 8.7(a) and (b), which shows the difference is larger than what can be explained with the increased RTT.

8.3.4 Using Multiple Anonymous Tunnels in Parallel

Another way to get better end-to-end performance is to use multiple tunnels in parallel. This should decrease the download times of web pages with several embedded objects because they can be requested in parallel through different tunnels. We analyse the download times when using three or five tunnels in parallel. We use HTTP 1.0 with a maximum of five parallel connections. Note that using tunnels in parallel to download a single web page does not make sense with HTTP 1.1 where the index file and all embedded objects of a page are fetched through the same anonymous connection, and therefore through the same anonymous tunnel. Figure 8.8 illustrates the download times.



a) three tunnels in parallel b) five tunnels in parallel

Figure 8.8: *Download times using multiple tunnels in parallel.*

Again, the download times could be decreased compared to Figure 8.3(a) and using five tunnels in parallel results in an end-to-end performance that is only slightly worse than using HTTP 1.1 as in Figure 8.3(b). However, using multiple tunnels in parallel to request a web page from a single server greatly increases the risk that an adversary breaks the anonymity because all he needs is to compromise one of the tunnels used to communicate with the server. Assuming f_{a_m} is the fraction of compromised tunnels among the tunnels an

initiator accepts and t_p tunnels are used in parallel to access a single server, the probability p_{obs} that this can be observed by the adversary is

$$p_{obs} = 1 - (1 - f_{a_m})^{t_p}. \tag{8.1}$$

As an example, assuming $f_{a_m} = 0.01$ and $t_p = 5$ results in $p_{obs} \approx 0.049$, which means the probability of being observed has increased nearly five times. Considering that even using five tunnels in parallel does in general not yield shorter download times than when using HTTP 1.1, it definitely does not make sense to use tunnels in parallel. We therefore continue to use the good compromise with one tunnel per web page and the optimisations according to Table 7.2. Since HTTP 1.1 always outperforms HTTP 1.0 if one tunnel is used, we will stick exclusively with HTTP 1.1 from now on.

8.3.5 Bandwidth Usage and Overhead

Appendix A.6 gives a quantitative analysis of the overhead produced by MorphMix and concludes that the data to set up and maintain anonymous tunnels result in an average overhead of sending and receiving about 1090 B/s for each node. Here, we analyse the bandwidth usage and the data overhead of MorphMix in more detail assuming our web browsing scenario. We distinguish between six different types of data:

1. **Web requests/replies at initiator**: the web requests sent and replies received by initiators. This corresponds to the the application data sent and received if the web server is contacted directly.
2. **Cell header/padding**: the additional data that are needed to generate fixed-length cells from the web requests and replies. This includes the cell headers, anonymous connection headers, and the random bits for the padding.
3. **Forwarding web requests/replies**: the web requests and replies forwarded by intermediate and final nodes. It is the total length of all data that are forwarded, including application data, cell and anonymous connection headers, and padding.
4. **Tunnel setup overhead**: all data associated with tunnel setup and teardown. It includes data sent and received by all nodes along a tunnel to establish append nodes and to establish the layer of encryption, including setting up the virtual links from and to the witnesses. We separate

between data sent and received by the nodes along a tunnel and by the data sent and received by witnesses to append a node. It also includes TERM messages to tear down tunnels.

5. **E2E ping/pong overhead:** all data sent and received to test the RTT of anonymous tunnels. It includes the data sent and received to test the own tunnels and the data forwarded for other nodes.
6. **Virtual link message overhead:** all virtual link messages sent and received between two neighbours to set up a virtual link and exchange virtual link status information. It also includes CREDIT messages (see Appendix A.3.6) for flow-control and cell headers and padding to exchange the virtual link messages.

The first three types of data are needed to fulfil the prime task of a mix network: to send and receive application data through anonymous tunnels. We therefore do not count the anonymous connection headers, cell headers, and padding bits to generate the fixed-length cells from the application data and forwarding the resulting cells along anonymous tunnels as data overhead. Consequently, we identify the first three types of data as *tunnel data*. On the other hand, the other three types of data are needed to provide the anonymous tunnel infrastructure including tunnel setup, testing, and teardown overhead and management of the overlay network, and are therefore collectively identified as *data overhead*.

We first analyse how much of the available bandwidth is actually used by MorphMix using the scenario in Figure 8.3(b) where web pages are requested through one anonymous tunnel using HTTP 1.1 and tunnel optimisation according to Table 7.2 is used. We distinguish between data sent and received and between tunnel data and data overhead. Figure 8.9 shows the bandwidth usage for all nodes together and for the different node types.

Looking at the leftmost bars in Figure 8.9, we can make two important observations:

1. Overall, only about 3% of the total bandwidth available to all nodes is used by MorphMix for sending and receiving data assuming our web browsing scenario described above. This is quite a small burden and means in general that most users can easily run a node without noticing a significant drop in terms of network performance for other applications.
2. Of all MorphMix data sent and received by all nodes, about 61% are tunnel data and 39% are data overhead. This means that the overhead is relatively large compared to the tunnel data, but since the total Mor-

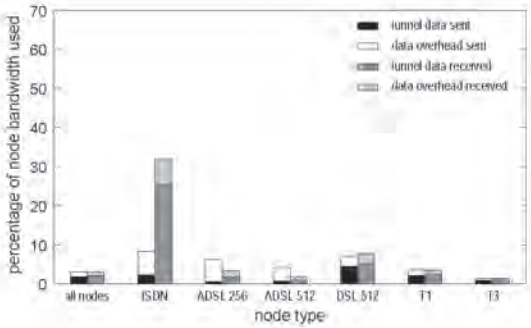


Figure 8.9: *Bandwidth usage.*

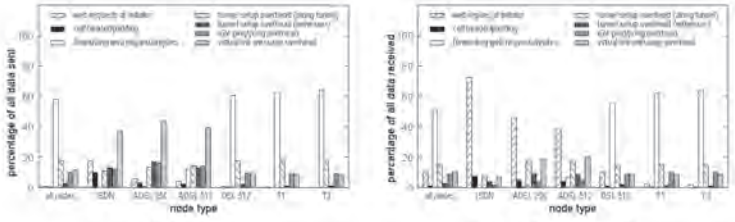
phMix load is so small compared to the bandwidth available to the nodes, it can easily be dealt with this data overhead.

Looking at the different node types, we can see that about one third of the down-stream bandwidth of ISDN nodes is used, which can be explained with their small bandwidth and our assumption that users with ISDN nodes are always browsing when they are online. Of the up-stream bandwidth of ISDN nodes, less than 10% is used. The explanation is that web requests are much shorter than web replies and that ISDN nodes only relay very little data of other nodes. In general, the percentage of the used bandwidth decreases as the Internet connection of the nodes gets faster. The exception in Figure 8.9 are DSL_{512} nodes because they are the nodes with the slowest Internet connections that are accepted in anonymous tunnels of all other nodes (see Table 7.2).

The calculations in Appendix A.6 result in an average data overhead of sending and receiving about 1090 B/s per node. According to Figure 8.9, ISDN nodes spend about 6% of their total available bandwidth for data overhead, which corresponds to approximately 430 bytes/s. This is less than the average because ISDN nodes relay fewer than average data from other nodes and 430 B/s is definitely an acceptable overhead for slow ISDN nodes. This overhead increases as the nodes get faster and T3 nodes use about 0.47% of their available bandwidth for overhead, which is equal to 2450 B/s and above the average. This is the maximum data overhead any node can expect and is insignificant compared to the available bandwidth to fast nodes.

We can also use the results in Figure 8.9 to estimate the computational overhead imposed by the cryptographic operations. We only consider T3 nodes because they handle the largest amount of data. These nodes send and receive about 7.8 KB of data per second. On our test system that is equipped with a 1GHz AMD Athlon CPU and 256 MB RAM, the symmetric key operations and the cryptographic hashes we employ (see Appendix A.2.1) can be computed at rates of 80 and 350 Mb/s, respectively. Even recalling that the payload of some cells must be encrypted multiple times, the resulting computational overhead is significantly below 1%. Looking at public-key cryptography and assuming RSA private-key operations with a 2048-bit key (see Appendix A.2.1), our test system manages to process about 160 Kb of data per second. Public-key cryptography is only used to process the data overhead and even there, only to establish virtual links and during the anonymous tunnel setup. In fact, less than 10% of all data overhead involves public-key operations (see Appendix A.6), and only half of these operations are the significantly more expensive private-key operations. Considering a T3 node sends and receives about 20 Kb of data overhead per second, less than 2 Kb of them must be processed using expensive private-key operations, which results in a computational overhead of about 1–2%. We therefore conclude that the computational overhead imposed by cryptographic operations is below 2% and can be handled well by a reasonably modern computer and that our assumptions in Section 8.2.4 were correct.

To analyse the data sent and received by the nodes in more detail, Figure 8.10 illustrates how much of the used bandwidth is spent on which type of data.



a) data sent b) data received

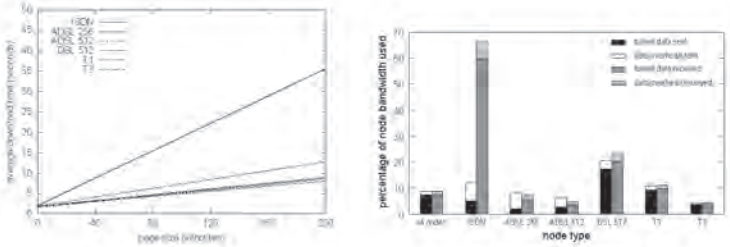
Figure 8.10: *Data sent and received by the nodes.*

By far the biggest part – about 55% – is spent on forwarding the web requests and replies of other nodes and only relatively little is spent to handle the own application data. This is reasonable because sending a web request as an initiator means that all other nodes along the anonymous tunnel must send and receive this request, too. Similarly, the reply sent back by the web server must be sent and received by the final and all intermediate nodes along the tunnel. The effective web replies received at the initiators only account for about 11% of all data nodes receive. Since web requests are usually much shorter than web replies, requests issued at an initiator account for less than 1% of all data sent. Figure 8.10 also shows that the additional amount of data produced by cell and anonymous connection headers and padding bits is relatively small compared to the user data: looking at the web replies, it is about 10% of the application data.

Tunnel setup and teardown overhead is responsible for about half of all data overhead and for about 19% of all all data. About 16.5% stem from the nodes along the tunnel and 2.5% from the witnesses when appending a node. End-to-end status information is responsible for about 9% and the various virtual link messages for about 11% of all data. Looking at the different node types, the bandwidth that is spent for handling the data of other nodes gets the bigger the faster the Internet connection of the node is. This is reasonable because according to our assumptions about realistic capabilities and participation probabilities in Section 7.3.2, nodes with good Internet connections accept relaying anonymous tunnels more frequently and are participating in MorphMix more often. In particular, ISDN nodes have to deal with relatively little data overhead, which can clearly be seen by inspecting Figure 8.10(b). More than 70% of all data received by ISDN nodes are web replies they have requested themselves and the data overhead accounts for only about 20%. The main reason is that we assume that the owners of the ISDN nodes are always browsing the web when the node is up. In addition, ISDN nodes nearly never accept relaying anonymous tunnels and as a result, their tunnel setup overhead mainly stems from setting up their own tunnels and acting as a witness for others. Similarly, ISDN nodes forward only little end-to-end status information messages of other nodes, which means most of the data overhead also stems from testing their own tunnels. As the nodes' bandwidth increases, they accept relaying anonymous tunnels more frequently and also become more attractive for others to be used in their tunnels. As a result, the fraction of the bandwidth that is used to forward data of other nodes gets larger and the fraction of the bandwidth that spent on handling the own web

requests and replies gets smaller.

The results depicted in Figures 8.9 and 8.10 are of course heavily dependent on the scenario and the different parameters we specified in Section 8.2. For instance, reducing the interval between two subsequent tunnels setups to one minute on average would approximately double the data overhead imposed by tunnel setups. Similarly, increasing the amount of application data decreases the relative data overhead. To analyse the impact of an increased amount of application data, we reduce the reading time (see Section 8.2.5) to zero, which means that as soon as a web page has completely been downloaded, the next request is initiated right away. Figure 8.11 depicts the download times and the bandwidth usage.

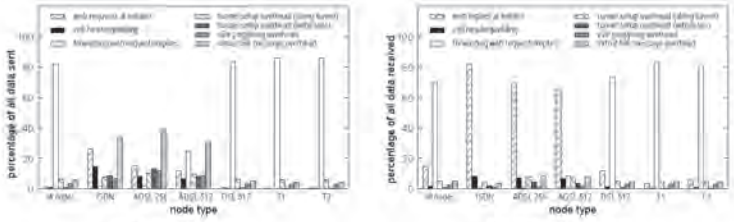


a) download times b) bandwidth usage

Figure 8.11: *Download times bandwidth usage with reading time = 0.*

Figure 8.11(b) shows that the total bandwidth used by MorphMix has nearly tripled from 3% to about 8.9% compared to Figure 8.9. Still the download times in Figure 8.11(a) have only increased a little bit compared to Figure 8.2(b), which means the MorphMix nodes could cope quite well with the increased traffic volume. Furthermore, the additional computational overhead by symmetric key operations and the cryptographic hashes can easily be handled by the nodes following our discussion in Section 8.3.5. Since the data overhead has remained the same, it now only accounts for about 15% of all data. For completeness, Figure 8.12 shows the data sent and received by the nodes in more detail.

Compared to Figure 8.10, the bars corresponding to tunnel data have become longer and those corresponding to data overhead have become smaller. Otherwise, the basic characteristics of how many data are used for which type depending on the node type have remained similar to those in Figure 8.10.



a) data sent b) data received

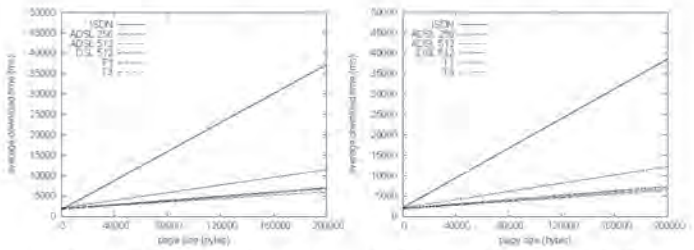
Figure 8.12: *Data sent and received by the nodes with reading time = 0.*

8.3.6 The Influence of Failed Tunnel Setups and Rejected Tunnels

Throughout this chapter, we have always assumed that setting up anonymous tunnels only fails if one of the nodes along the tunnel fails (for instance by leaving MorphMix) during the setup or a witness that is used to append a node fails. Similarly, we have assumed tunnels are never rejected by the initiator. However, in practice it may happen that the tunnel fails during the setup for any reason, for instance because the witness specified by the initiator cannot be contacted. In addition, some tunnels will be rejected by the initiator because they are identified as malicious by the collusion detection mechanism. Even without malicious nodes, a fraction of about 0.2 of all tunnels will be rejected because of false positives (see Chapter 7). We analyse the impact of failed or rejected tunnels in Figure 8.13, assuming that a fraction of 0.5 or 0.8 of all anonymous tunnels fails or is rejected. Note that although tunnels can fail at any time during the setup, we assume a worst case scenario in the sense that the failure occurs at the end of setting up a complete tunnel. Consequently, failed and rejected tunnels both "cost" setting up a complete tunnel.

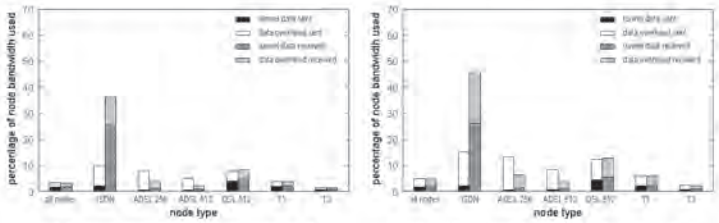
Comparing Figure 8.13 with Figure 8.3(b), we can see that the download times have marginally increased. This is not surprising, as there is additional load because more tunnels have to be set up. This can also bee seen by analysing the corresponding bandwidth usage illustrated in Figure 8.14.

In general, the overhead has increased compared to Figure 8.3 and consequently, the total load per node has increased. The overhead has grown from about 39% of the total amount of data in Figure 8.3 to approximately



a) failure/reject probability 0.5 b) failure/reject probability 0.8

Figure 8.13: *Download times with failed and rejected tunnels.*



a) failure/reject probability 0.5 b) failure/reject probability 0.8

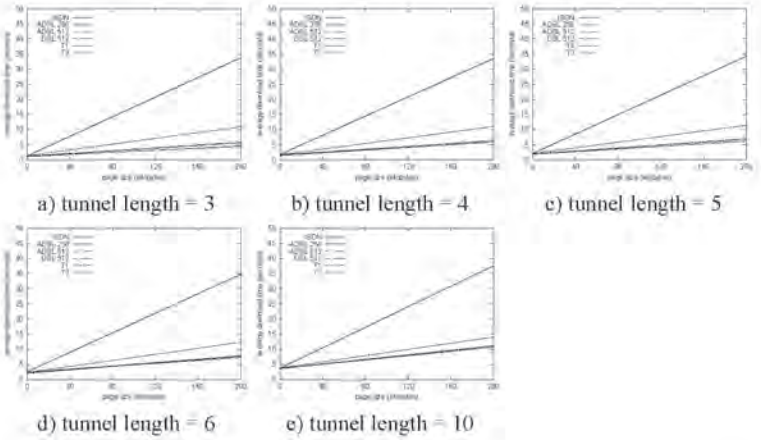
Figure 8.14: *Bandwidth usage with failed and rejected tunnels.*

49% in Figure 8.14(a) and 64% in Figure 8.14(b). However, the total load of all MorphMix traffic is still below 5% of the total bandwidth available to all nodes even if a fraction of 0.8 of all tunnels cannot be used. For T3 nodes, the data overhead in Figure 8.14(b) has about tripled compared to Figure 8.3, which implies the computational overhead imposed by cryptographic operations has also approximately tripled. On a computer equipped with a 1GHz AMD Athlon CPU, this result in a computational overhead of about 6% (see Section 8.3.5), which can easily be handled. Consequently, we conclude that MorphMix copes well with a significant fraction of failed or rejected tunnels.

8.3.7 The Influence of the Tunnel Length

In Section 7.6, we have analysed the effect of different tunnel lengths on the probability the adversary manages to compromise anonymous tunnels. In

general, we have seen that longer tunnels imply better protection but they also increase the fraction of false positives. To examine the influence of the tunnel length, Figure 8.15 illustrates the download times depending on the tunnel length. We use again HTTP 1.1 to download a whole web page through one anonymous tunnel and the optimisations according to Table 7.2. As a reference, we also include Figure 8.3(b) with a tunnel length of five nodes.



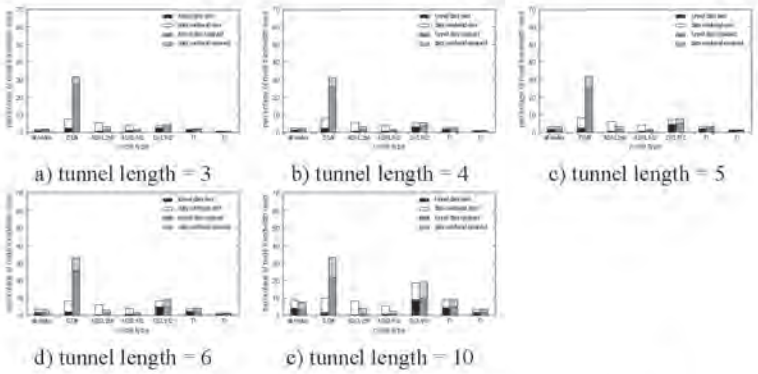
a) tunnel length = 3 b) tunnel length = 4 c) tunnel length = 5

d) tunnel length = 6 e) tunnel length = 10

Figure 8.15: *Download times depending on the tunnel length.*

Not surprisingly, the download times increase as the tunnels get longer. In addition, the RTT gets longer, which can be clearly seen by comparing the download times when the page sizes are small. On the other hand, the download times if one uses three nodes in a tunnel are significantly faster than with five nodes, in particular if the initiator has a good Internet connection. Figure 8.16 depicts the bandwidth usage for the scenarios in Figure 8.15.

The longer the tunnel length, the larger the percentage of the total bandwidth that is used by MorphMix. The relative data overhead also gets larger as the tunnels get longer because the tunnel setup overhead grows faster than the tunnel data if the tunnel length increases. Similarly, the computational overhead imposed by cryptographic operations and the collusion detection mechanism increases as the tunnel length grows. Note that longer tunnels also means an increased risk that any of the intermediate or final nodes suddenly leaves the system and thereby breaks the tunnel. Nevertheless, assuming the



a) tunnel length = 3 b) tunnel length = 4 c) tunnel length = 5

d) tunnel length = 6 e) tunnel length = 10

Figure 8.16: *Bandwidth usage depending on the tunnel length.*

web browsing scenario, the amount of data can easily be handled by the nodes even if all tunnels have a length of ten.

As a conclusion, we state there is simply no optimal tunnel length in MorphMix. Choosing the tunnel length is a compromise between usability and protection from attacks. A user who prefers good end-to-end performance over best possible anonymity should be happy with a tunnel length of three. Another user interested in minimising the probability of compromised tunnels may be willing to use tunnels with ten nodes even if this implies worse end-to-end performance and a higher rate of false positives. A third user could try to find a good trade-off between performance and protection from attacks and chooses five nodes in her tunnel. Recalling the results in Figures 7.11, 8.15, and 8.16, a tunnel length of five still seems to be a good compromise and a reasonable default value to be used in MorphMix. However, any implementation of a MorphMix node should give the user the choice to change the default value to anything between three and ten. A tunnel length below three does not make sense because this means the initiator directly appends the final node to itself, which implies there is no selection offered to the initiator during the entire tunnel setup and consequently, there are no data for the collusion detection to operate on. Tunnel lengths above ten are also not recommended because they add virtually nothing to further increase the resistance to attacks and are therefore not worth the additional performance penalty and the increased load on other MorphMix nodes.

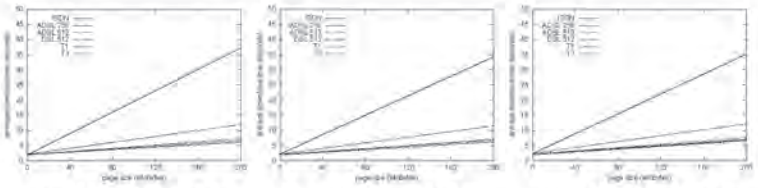
8.3.8 The Influence of the Cell Length

We have always used 512 bytes for the length of the cells, but other cell lengths could be used. In general, shorter cells mean more header data overhead per cell but also fewer padding bits on average in the last cell of a MorphMix protocol message. In addition, the end-to-end delay for a single cell is slightly smaller because nodes must always completely receive a cell before they can forward it. Conversely, longer cells mean less header data overhead and more padding bits in the last cell of a message, and completely receiving and processing a cell takes a bit longer. As an example, we analyse the total amount of data that must be transmitted to handle different application data lengths from 100 to 100000 bytes. To get the effective payload per cell that can be used to carry application data, we must subtract the cell and anonymous connections headers lengths from the cell length. Looking at cell lengths of 256, 512, and 1024 bytes, the available payloads for the application data are 224, 480, and 992 bytes, respectively. Table 8.1 illustrates the number of cells needed and the total data that must be transmitted depending on the application data length and the cell length.

Table 8.1: *Data volume depending on the cell length (all lengths in bytes)*

appl. data length	256-byte cells		512-byte cells		1024-byte cells	
	#cells	tot. data	#cells	tot. data	#cells	tot. data
100	1	256	1	512	1	1024
1000	5	1280	3	1536	2	2048
10000	45	11520	21	10752	11	11264
100000	447	114432	209	107008	101	103424

Looking at the results in Table 8.1, we can say that small amounts of application data profit from small cells while longer amounts benefit from long cells. If we were sure MorphMix would only be used to download large files, we would choose a cell length of 1024 or even longer. If it were mainly used for anonymous remote terminal access, even 256 bytes would be too much because a single character would be transmitted in one cell most of the time. Using our web browsing scenario and examining the download times depending on the cell length, we get the results in Figure 8.17. We use the standard settings, i.e. requesting a page through one tunnel via HTTP 1.1 and using tunnel optimisation according to Table 7.2. For easy comparison, we also include Figure 8.3(b).



a) cell length = 256 B b) cell length = 512 B c) cell length = 1024 B

Figure 8.17: *Download times using different cell lengths.*

There is virtually no difference between the download times, with the exception that with a cell length of 1024 bytes, the times for fast nodes are slightly worse than with shorter cells. We conclude our cell length of 512 bytes is reasonable, although 256 bytes would make sense too when looking at the download times. One argument in favour of 512 bytes is that about 80% of all web requests are approximately 300 bytes long (see Section 8.2.5) and therefore fit into one cell, which means exactly one cell must be processed by each node along a tunnel for most web request.

8.3.9 Crashing Nodes and Blocked Virtual Links

So far, nodes did never crash and could always be reached. The only way they could disappear and render tunnels useless was when their operators willingly shut them down. In practice, we can expect that nodes or the computers they run on crash from time to time and that nodes can temporarily not be reached due to problems with their Internet connection. We have already mentioned the subtle differences between a MorphMix application that crashes or is shut down and a computer running a MorphMix node that crashes or that cannot be reached temporarily in Section 8.1. Here we analyse the impact on the end-to-end performance.

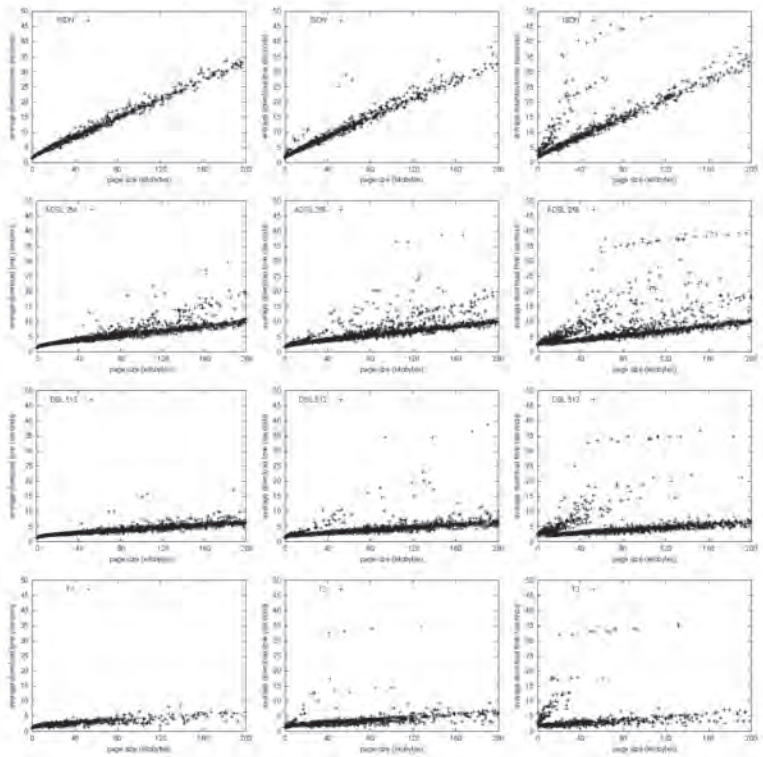
We use the following model: MorphMix applications or the computers they run on can crash at any time. If a node crashes, we assume it is because of a computer crash in 50% of all cases and because of a MorphMix application crash in the other 50% of all cases. Once a node has crashed, we assume it will be back online within a random time between one and five minutes. In addition, nodes can temporarily not be reachable due to connection problems for a random time between one and five minutes. Note that the effect

of a node that cannot be reached is similar as a crashed computer because the initiator, and therefore the client application, cannot be notified about the terminated connection. Since the download failure reaction time (see Section 8.2.5) is at most 30 seconds, any of the failures described above always result in downloading the entire page again through another tunnel.

We analyse the impact of node crashes and temporarily blocked virtual links assuming every node crashes once during a day and is temporarily blocked from its neighbours 5, 10, or 20 times during a day. Figure 8.18 illustrates the download times for the four node types ISDN, $ADSL_{256}$, DSL_{512}, and T3 for the cases where a node is blocked from its neighbours 5 or 20 times a day. Since some page downloads take much longer because they page must be downloaded more than once, we do not use linear regression this time but show the effective download times of each individual page download. We use the standard settings, i.e. requesting a page through one tunnel via HTTP 1.1 and using tunnel optimisation according to Table 7.2.

First of all, Figure 8.18(a) shows the download times are indeed nearly linearly dependent on the page size, which justifies our practice of using linear regression to plot the graphs. Without any node crashes or blocked virtual links, only very few downloads take significantly longer than they should because the probability a tunnel is interrupted is small. In this case, interrupted tunnels occur only if an intermediate or final node leaves the system according to its participation pattern (see Section 8.2.4). With crashing nodes and with an increasing probability of blocked virtual links, more and more tunnels fail and as a result, the probability a page must be requested more than once increases. However, even in Figure 8.18(c), the vast majority of all download times are still the same as with no node crashes or blocked virtual links. Corresponding to Figure 8.18, Table 8.2 lists the percentage of pages that failed during their first download.

With no node crashes or blocked virtual links, only about one of 1000 page downloads fail. With every node crashing once and being blocked temporarily from the Internet five times a day, the failure rate increases to about 1%. The results for even more frequently blocked nodes are of rather theoretical interest, as we do not believe nodes crash or are blocked from the Internet so often. We conclude that MorphMix should still be able to deliver satisfactory performance if nodes crash or are blocked from their neighbours from time to time.



a) no crashes, never blocked | b) 1 crash, 5 times blocked | c) 1 crash, 20 times blocked

Figure 8.18: *Download times when nodes crash or are temporarily blocked from their neighbours.*

8.4 Summary

We have presented a thorough discussion of the performance MorphMix offers to its users using our own simulator and web browsing as the example application. Naturally, we cannot expect the same performance as when contacting the web server directly, but summarising all results, we state that de-

Table 8.2: *Percentage of pages that failed during their first download.*

node type type	no crashes, no blocks	1 crash, 5 blocks	1 crash, 10 blocks	1 crash, 20 blocks
ISDN	0.09%	0.90%	2.04%	4.47%
$ADSL_{256}$	0.11%	0.95%	2.00%	3.97%
$ADSL_{512}$	0.09%	0.96%	1.90%	3.95%
DSL_{512}	0.07%	0.95%	1.85%	3.75%
T1	0.06%	1.05%	1.95%	3.78%
T3	0.15%	0.88%	1.74%	4.93%
total	0.09%	0.96%	1.91%	3.95%

spite the heterogeneity and the fact that nodes may no longer be reachable at any time, MorphMix offers good performance. In particular, using HTTP 1.1 and downloading the whole page through a single tunnel, a tunnel length of five nodes, and tunnel optimisation according to Table 7.2 results in an acceptable performance penalty and provides a reasonable compromise between download times and protection from attacks. We have also demonstrated that participating in MorphMix and getting adequate performance is not only possible for users with computers that have a broadband Internet connection, but also for users with computers with slow 64 Kb/s Internet connections. Finally, we have shown that even if a significant fraction of all anonymous tunnels cannot be used, either because the tunnel fails during the setup or is rejected by the initiator because the tunnel is identified as malicious by the collusion detection mechanism, the performance gets only marginally worse.

Besides the "normal" mix network overhead induced by fixed-length cells and relaying the data of other nodes, there is additional data overhead resulting from tunnel setup, testing, and teardown and management of the overlay network, which accounts for nearly 50% of all MorphMix traffic assuming every second anonymous tunnel cannot be used because it has either failed during the setup or is rejected by the initiator. This is the price MorphMix users must pay to deal with a dynamic environment with potentially malicious nodes, but we have shown that this data overhead can be easily handled. Similarly, we have also demonstrated that considering the nodes that handle the largest amounts of data and assuming that most tunnels cannot be used after they have been set up, the cryptographic operations impose a computational overhead that consumes up to 6% of the available computing power on a computer that is equipped with a 1GHz AMD Athlon CPU. Con-

sequently, the computational overhead can be easily handled by reasonably modern computers.

Although we have simulated a system consisting of only 1000 nodes, we state that our results are also representative for significantly larger systems. In particular, we have set the selection size to its maximum value, which implies the messages to set up a tunnel have their maximum length. In addition, every node has 30 neighbours at any time to guarantee 20 nodes can be easily offered in selections, which means the overhead to manages the local environment is not smaller than it would be with nodes in all public /16 subnets. Furthermore, we use the realistic assumptions about the capabilities of the nodes we introduced in Section 7.3.2. As a result, the tunnel data, data overhead, and computational overhead for a single node in our simulation are the same as if there were nodes in all /16 subnets. Since the load on a single node is low, we state that our results would be very similar to the results based on a significantly larger system. In general, it can be expected that the performance offered by MorphMix is virtually independent of the number of /16 subnets that contain nodes if the distribution of the nodes' capabilities remains approximately the same, because the data and computational overheads are always small, no matter how large the system is.

We conclude that MorphMix is indeed practical in the sense that its data overhead an the computational overhead resulting from cryptographic operations are reasonably small and the performance it offers is good enough such that users are not turning away from the system for performance reasons. It should be noted that we have measured only the time to download the pages and have not taken the time it takes to display a completely downloaded page in the web browser into account. Since displaying the pages can take a few seconds for complex pages independent of whether the page was downloaded directly or through MorphMix, the relative performance penalty to download *and* display a page should be even smaller than what our simulation results revealed.

One final remark about the variations of different parameters to optimise the performance: it is all in hands of the initiators. A user that wants to maximise her protection from being observed and does not care much about the performance she gets can always choose to accept every node in her anonymous tunnels and to use long tunnels. Another user aiming at a good compromise between anonymity and performance would probably make use of the optimisations according to Table 7.2 and use a tunnel length of five to contact a host anonymously. Finally, users that wish to maximise the performance

but still get a certain degree of anonymity make sure that only fast nodes are present in their tunnels and use short tunnels with only three or four nodes.

Chapter 9

Conclusions

In this chapter, we conclude our work. We first provide a brief summary of our work. Then we review the goals we have stated in Section 5.1 and analyse if we could achieve them. We also point out some limitations of MorphMix. Afterwards, we compare MorphMix with other peer-to-peer-based systems that aim at providing anonymous Internet access. Finally, we identify several challenging topics for future research on mix networks in general.

9.1 Summary

In thesis, we have presented MorphMix, a novel peer-to-peer-based dynamic mix network. The main motivation for developing MorphMix was that static mix network, operated commercially or by volunteers, seem not to be well suited to provide anonymous Internet access for a large number of users. Static mix networks operated by volunteers suffer from the problem of acquiring enough mixes and from the threat of an internal attacker controlling a significant portion of all mixes. Commercially operated static mix network have yet to show whether they can indeed be operated profitably. Peer-to-peer-based mix networks, on the other hand, seem to have some intrinsic advantages over static mix networks. Since every user brings her own mix, they should be able to support a very large number of users. In addition, the potentially large number of participating users makes it more difficult for an adversary to operate a significant subset of all mixes. However, peer-to-peer

mix networks are still a relatively new area of research and they had yet to demonstrate their usefulness in practice.

The goal of our work on MorphMix was to provide a practical system that enables anonymous low-latency Internet access for a large number of users. To achieve this goal, MorphMix is composed of three core components. The first of these components is the anonymous tunnel setup protocol. One key design decision regarding this protocol is that every node along a tunnel picks its immediate successor. This has the advantage that every node must only handle its local environment consisting of its neighbours. A node can easily communicate with its neighbours to learn which of them are currently participating in MorphMix and have spare resources to accept new anonymous tunnels. The disadvantage is that a malicious node would simply pick another malicious node as its successor to compromise anonymous tunnels of honest nodes. To counter this attack, we designed the anonymous tunnel setup protocol in a way such that a node cannot simply choose its successor itself, but must offer a selection of several nodes to the initiator and the initiator picks one of them. The idea is that a malicious node must now offer many or only malicious node in its selections to guarantee the following node in the tunnel is also malicious with high probability.

This is where the second core component, the collusion detection mechanism, comes into play. Based on the assumption that an adversary can operate nodes only in a small subset of all public /16 subnets, the collusion detection mechanism can detect selections that contain many malicious nodes with high probability. if such a selection is detected, the tunnel is suspected as potentially malicious and is rejected by the initiator. Therefore, a malicious node can only offer relatively few malicious nodes in its selections without being detected and consequently, the adversary manages to compromise only slightly more tunnels than if he played fair. The collusion detection mechanism bases on the assumption that honest nodes pick their neighbours they offer in selections from a wide variety of all /16 subnets that contain MorphMix nodes. Consequently, MorphMix requires a mechanism that supports this.

This is achieved with the third component, the peer discovery mechanism. The idea is that a node remembers the information about other nodes it receives in selections. This information about other nodes is stored in a way that allows honest nodes to pick their neighbours from a wide variety of /16 subnets. In addition to providing the basis for the correct functioning of the collusion detection mechanism, this has an additional benefit because it is a

necessary requirement for the internal adversary to control the first intermediate node to break the relationship anonymity between initiator and server. Selecting the neighbours from a wide variety of all /16 subnets reduces the probability the adversary controls this node because of the assumption that the adversary can only control nodes in a limited number of all public /16 subnets.

After having described the basic design of MorphMix, we have analysed the impact of different attack strategies that can be employed by the adversary and have come to the conclusion that the most promising attack the adversary should make use of is attacking always with the same attack level. This means that whenever a malicious node is picked as an intermediate node in a tunnel, it should offer always the same number of malicious nodes in its selection. The number of malicious nodes in a selection corresponds to the attack level and depending on the number of different /16 subnets that contain MorphMix nodes, there is an optimal attack level that maximises the adversary's chances to compromise a tunnel.

Based on this attack, we have analysed the performance of the collusion detection mechanism assuming a realistic scenarios with a large number of nodes that are located in many different /16 subnets. In addition, we have analysed the impact of different capabilities of the nodes and the influence of the fact that many nodes are not continuously participating in MorphMix, but may join or leave at any time. The main result was that the collusion detection works well in the sense that it can significantly reduce the number of compromised tunnel compared to the case if no such mechanism were employed. In particular, assuming a large system with honest nodes in nearly all public /16 subnets, the adversary must control nodes in several 1000 subnets to compromise more than a fraction of 0.01 of the tunnels that are accepted by the initiator.

Finally, we have implemented a simulator to evaluate the performance MorphMix users can expect and to analyse the data overhead produced by MorphMix. Using web browsing as the example application, we have shown that although the nodes are very heterogeneous, may no longer be reachable at any time, tunnels may fail during the setup, and tunnels may be rejected by the initiator because it is identified as malicious by the collusion detection mechanism, the performance MorphMix offers is good enough such that users are not turning away from the system for performance reasons. In addition, both the data overhead and the computational overhead imposed by cryptographic operations are reasonably small and can be easily handled by

any participating node.

For completeness, we have also provided the full MorphMix protocol specification and a prototype implementation, both of which are described in the appendix.

9.2 Achievement of Goals and Assessment

The principal goal of our work was to develop a practical system that enables anonymous low-latency Internet access for a large number of users. In Section 5.1, we have stated four more detailed goals we wanted to achieve with MorphMix to fulfil the principal goal of this thesis, and we can say we have fulfilled all four of them:

1. **Requirements to Participate**: Recalling our discussion on computational and memory requirements in Sections 5.8.1 and 8.3, participating in MorphMix is possible for anyone owning a state-of-the art computer that is connected to the Internet and capable of running modern graphical application such as web browsers or office packages. We have shown in Section 8.3 that the bandwidth requirements are also modest and even users with computers that have slow 64 Kb/s dial-up Internet connections can participate. In addition, participating is possible independent of whether the computer has a static or dynamic public IP address or is located in a private network behind a NAT gateway, as we have shown in Section 5.8.2. Finally, joining MorphMix for the first time is easy because the peer discovery mechanism (see Section 5.7) makes it possible to quickly learn about other nodes.
2. **Scalability**: MorphMix scales very well and can handle as many nodes as there are public IP addresses (see Section 5.8.1). The key to scalability in MorphMix is that the complexity of its three core components does not depend on the number of nodes, but on the number of /16 subnets that contain MorphMix nodes. Since there is an upper bound on the number of /16 subnets (see Section 5.4.2), the complexity of the three components is also limited, and we have shown in Sections 5.8.1 and 8.3 that a node that fulfils the requirements above can cope well with an environment with MorphMix nodes in all /16 subnets. Furthermore, we have demonstrated in Section 8.3 that since the data overhead is so small, the performance offered by MorphMix is virtually independent on the system size.

3. **Protection from Attacks**: Based on our assumption that an adversary can operate nodes in only a small fraction of all public /16 subnets (see Section 5.4.2), we have shown in Chapter 6 that the collusion detection mechanism works well in the sense that an attacker cannot compromise significantly more tunnels than if he played fair, i.e. if malicious nodes behaved like honest nodes and picked the nodes in their selections randomly. Note that this is close to the optimum we can achieve because it is never possible to detect an adversary that plays fair based on his behaviour. Furthermore, we have demonstrated in Chapter 7 that assuming a realistic scenario with a large number of nodes that have different capabilities and that are spread across many different /16 subnets, the combination of the peer discovery mechanism and the collusion detection mechanism prevents an adversary from compromising more than a very small fraction of all tunnels that are accepted by the initiator. In particular, assuming a large system with honest nodes in nearly all public /16 subnets, the adversary must control nodes in several 1000 subnets to compromise more than a fraction of 0.01 of the tunnels. Consequently, MorphMix indeed provides good protection from long-term profiling attacks by an internal attacker. However, it must be remembered that MorphMix cannot guarantee the anonymity of every single transaction and does therefore not offer perfect anonymity. In addition, MorphMix does not employ measures to protect from an external observer that observes a subset of all nodes because we do not consider this attacker as a significant threat (see Section 5.4). However, as mentioned in Section 5.3, it must be expected that an adversary observing both the first intermediate and the final node of a tunnel manages to break the relationship anonymity between the initiator and the server that is contacted through this tunnel. Developing efficient mechanisms that significantly increase protection from external observers in mix networks in general is a topic of further research.
4. **Performance**: According to our performance analysis in Chapter 8 based on a realistic web browsing scenario, we conclude MorphMix indeed offers good performance despite the heterogeneous environment with nodes that have significantly different capabilities and that may no longer be reachable at any time, either because they have been shut down by their operators, have crashed, or can temporarily not be reached due to network problems. Naturally, there is a performance penalty when accessing servers through MorphMix, but especially the

measures to optimise the throughput of anonymous tunnels by making sure no nodes with slow Internet connections are present in the tunnels of initiators with broadband Internet connections (see Section 8.3.3) result in acceptable performance. We therefore can state that the performance MorphMix offers is good enough such that MorphMix users are not turning away from the system for performance reasons.

Since we have achieved all four partial goals, we conclude that MorphMix fulfils the principal goal of our work and is indeed a practical system that enables anonymous low-latency Internet access for a large number of users.

However, there is still room for improvements. The most significant limitation of MorphMix is that if any node along a tunnel can no longer be reached for any reason, the tunnel fails. Consequently, all anonymous communication relationships between the initiator and servers that use this tunnel are terminated. This is a general problem of mix networks that are operated similar as illustrated in Figure 2.6(a), but static mix networks suffer less from it because their mixes are usually available all the time. We have thought about possible solutions to mitigate this problem, for instance by bypassing nodes in a tunnel that are no longer reachable. However, doing so could enable an attack where malicious nodes claim that their honest successor node in a tunnel can no longer be reached and we therefore decided not to make use of this approach. Consequently, and until there is no proposal to solve this problem, MorphMix is not well suited for long-standing communication relationships such as remote logins.

For many other applications, however, MorphMix can be well used. We have already shown in Chapter 8 that web browsing is one such application. In addition, MorphMix can be used to anonymise FTP downloads or in general to enable anonymous file transfer, with the risk that files must be downloaded again if a tunnel fails. MorphMix is also very well suited to enable anonymously searching and downloading files from other peers in peer-to-peer file-sharing communities. One could go even further: by incorporating ideas to use mix networks to enable both client and server anonymity via rendezvous points[1] and assuming every peer runs also a MorphMix node would enable a completely anonymous file-sharing community where offering, downloading, and searching for files would be anonymous.

Another potential problem for MorphMix are DoS attacks. As discussed in Section 6.4, an adversary may participate in MorphMix with several nodes

[1] http://freehaven.net/tor

simply to disrupt the service. To do so, his nodes would accept tunnels being established through them but refuse to transport data once a tunnel has been set up or stop forwarding data after it has been used for a while. In practice, this attack can be quite effective in the sense that if most tunnels fail in the middle of a file transfer, the quality of service as perceived by the users gets so poor that they no longer use MorphMix. Again, this problem is less severe in static mix networks with a limited number of mixes where it is much easier to identify and exclude mixes that fail to process data correctly. The solution to this problem is to couple MorphMix with a reputation system. Nodes that repeatedly fail to forward data would get a bad reputation over time and would no longer be offered in extended selections from honest nodes. Research on reputation systems is still in its infancy, but initial studies to make mix networks more reliable through reputation have been carried out (see Section 3.5). Note that such a reputation system would not only protect better from DoS attacks by an adversary, but decrease the failure rate of anonymous tunnels in general because nodes that frequently leave the system would also get a poor reputation.

One final remark about the fact that MorphMix protects from long-term profiling attacks but does not guarantee the anonymity of every single transaction. Although the latter would be more desirable from an anonymity point of view, the acceptance of MorphMix could actually benefit from this. The reason is that if a person is suspected to be involved in criminal activities by communicating with a particular server through MorphMix, it is relatively easy to uncover this communication relationship because all that is needed is to eavesdrop on both the person's computer and the server (see Section 5.3). It is likely that if there is a strong suspicion of ongoing criminal activity, court orders would be issued to facilitate this action. Consequently, MorphMix is probably not the right tool to be used for criminal activities because the risk of being detected by a well targeted attack is too high.

9.3 Comparison with Other Systems

The advantages of dynamic, peer-to-peer-based mix networks have already be pointed out in Chapter 4 and we therefore do not compare MorphMix with static mix networks. Rather, we compare MorphMix with other, peer-to-peer-based approaches where hosts that are not part of the system are contacted via some other nodes. In particular, we compare MorphMix with Crowds (see

Section 3.3) and Tarzan (see Section 3.1.2).

9.3.1 Comparison with Crowds

There are three main differences when comparing MorphMix with Crowds: (1) Crowds requires a centralised lookup server to keep track of nodes that are currently participating, (2) Crowds does not employ a collusion detection mechanism, and (3) Crowds does neither make use of fixed-length cells nor of layered encryption.

The requirement of a lookup service is definitely a major drawback, first of all because it provides a single point of failure and attack and second because the lookup server must inform all participating nodes about joining or leaving nodes because every node must know about all other nodes in the crowd. The second makes Crowds not well suited to support many nodes (e.g. several 1000s) where nodes come and go. In contrast, leaving out the optional introductory nodes to join for the first time, MorphMix does not relay on such a lookup service and in particular, MorphMix does not require a node to know about all other nodes at any time and we have shown in Sections 5.8 and 8.3 that MorphMix scales very well up to as many nodes as there are public IP addresses. Note that due to the lack of simulation results or analyses of the overhead that is produced by the communication of the nodes with the lookup server, it is difficult to compare the data overheads of Crowds and MorphMix. But it can be expected that in Crowds, the relative data overhead compared to the actual application data that are processed grows with the number of nodes because more and more bandwidth must be devoted to keep all nodes informed about joining and leaving nodes. In MorphMix, on the other hand, the data overhead is nearly independent of the number of nodes and can easily be handled by the nodes even if there are very many participants (see Section 8.3).

The second difference is the lack of a collusion detection mechanism in Crowds. Assuming the requester picks a malicious node to which it forwards the request and that node can find out that its predecessor is indeed the requester, it has broken the anonymity. To protect from this attack, the last node along a chain retrieves the page including all embedded objects before sending it back to the requester. This prevents the malicious node from easily making use of a timing attack to learn whether it is directly following the requester or not because embedded objects would be requested by the browser automatically. It also improves the performance because round-trips

to request embedded objects between requester and web server are avoided. Still, this approach also has disadvantages because it requires the last page to parse an HTML object to get all embedded objects, which could be difficult with web pages that contain executable scripts. In addition, the approach does not work if HTTPS is used because the last node along the chain cannot access the HTML object. Furthermore, the requester clicking on a link can itself leak information that can be used for a timing attack by the first node in the chain to determine its position with high probability. The Crowds' designers propose to introduce random delays to complicate this attack, but this reduces the end-to-end performance and could refrain potential users from using the system. Finally, HTTP redirects [45] may be inserted by the malicious node to force the browser to issue another request after a specified amount of time. Of course one can always filter such content on the requesters computer, but this always implies limiting the capabilities of the system a bit. In general, this entire approach has the serious drawback that Crowds needs to be application-aware and cannot easily be used for other applications than web browsing. In contrast, the collusion detection mechanism as employed in MorphMix is a much cleaner solution because it increases the probability that anonymous paths are "secure" before the server is contacted. Consequently, no such measures as employed by Crowds are required and the nodes along a tunnel can always simply forward the data of others without having to inspect the content. Note also that during the analysis of attacks on MorphMix, we have assumed that during the exchange of data between initiator and server, the first intermediate node in the tunnel always learns that it is directly following the initiator. This may not always be easy in practice, but it will often be possible to find this out with high probability if sufficient data are sent forth and back through the tunnel (see Section 5.3).

The third major difference is the lack of fixed-length cells and layered encryption in Crowds. From the point of view of correlating data at different places in the Internet, layered encryption and fixed-length cell do not help much because the combined application data volume and timing attack at the endpoints is difficult to prevent (see Section 4.1) without employing cover traffic. The reason for using layered encryption in MorphMix is mainly motivated by hiding the data sent by the initiator from any of the nodes along the tunnel except from the final one. Otherwise, a malicious first intermediate node could for instance see a web request and the whole collusion detection mechanism would be pointless. Crowds, on the other hand, does not make use of layered encryption because it assumes the first node cannot eas-

ily learn that it follows directly the initiator. Even if layered encryption were employed in Crowds, it would not help because malicious nodes could simply pick other malicious nodes (or themselves) as their successors. Using fixed-length cells has a performance advantage because application data can be "streamed" along an anonymous tunnel in the sense that data can be forwarded as soon as a (short) cell has been received. In Crowds, the entire data corresponding to a web request or reply must be received by a node before they are forwarded, which introduces long end-to-end delays if the chain gets long and if a web page is large. The MorphMix design with fixed-length cells and layered encryption also has the advantage that cover traffic could easily be added if an efficient mechanism will be ever developed.

We conclude that MorphMix is superior to Crowds, mainly because of its scalability, its application independence, and its capabilities to detect malicious tunnels with high probability before any critical data are sent through that tunnel. Since Crowds cannot detect a malicious node directly following the requester, it mainly focuses on making it difficult for this node to detect its predecessor is indeed the requester.

9.3.2 Comparison with Tarzan

The main differences between MorphMix and Tarzan are: (1) Tarzan builds an universally verifiable set of neighbours (the mimics) for every node, and (2) Tarzan employs cover traffic streams between neighbours. The first requires Tarzan nodes to know about all other nodes, which again makes it unlikely Tarzan can function well in a large and dynamic environment where nodes come and go. Apart from this drawback, the fact that every node selects its neighbours in a pseudo-random but verifiable way makes it virtually impossible for a malicious node to have only other malicious neighbours. Since the initiator picks the nodes along a tunnel (each node is picked from the mimics of it predecessor), it is therefore very unlikely all nodes along an anonymous path are malicious. In comparison to the *collusion detection* mechanism employed in MorphMix, we can identify the mechanism employed by Tarzan as *collusion prevention*. Assuming the mimics of a node in Tarzan are indeed selected randomly, it can be expected that the node following the initiator and the last node in an anonymous path are also selected nearly randomly from the set of all nodes. Consequently, the probability of a compromised anonymous path can be expected to be slightly better than in MorphMix and close to the optimum (i.e. the bottom line in Figure 6.10). However, this comes

with a heavy price because every node must know about all other nodes for the system to work correctly. In addition, there is only little room for throughput optimisation because the potential next hop nodes are limited to a node's mimics. Due to the lack of quantitative simulation results, we can only estimate the data overhead of Tarzan, but like in Crowds, it is reasonable to assume that keeping the information about the entire system up-to-date at every node grows faster than the number of participating nodes.

The second difference is the decision to make use of cover traffic in Tarzan. The main motivation was to provide protection from a global eavesdropper and we agree that this requires cover traffic between neighbours (see Section 4.1.1). However, according to our threat model (see Section 5.4), we believe that like in MorphMix, internal attackers are a more significant threat and no cover traffic scheme helps against this internal active attacker. In addition, cover traffic could reduce the performance offered by Tarzan so significantly that users interested in anonymity would not use the system at all. Just imagine a DSL_{512} nodes with six neighbours. Using the same constant cover traffic rates on all links would reduce the bandwidth of each link to 64 Kb/s even if all neighbours could handle at least as many data as the DSL_{512} node. In MorphMix and assuming the node is currently only handling the data of one tunnel, the full 512 Kb/s were available to forward the data. It is exactly reasons like this that refrain us from employing any cover traffic mechanism. It should be noted that Tarzan does not require a node to employ the same fixed stream bit-rates with all its neighbours. In fact, data rates of the bidirectional cell streams between two neighbours can vary within an upper and a lower bound. This seems to be a good idea because different nodes have different capabilities but it is not entirely clear how much protection such a scheme really offers.

We conclude that Tarzan could work well if the number of nodes is relatively small or if the nodes in the system do not change too frequently. Assuming such a scenario, Tarzan provides good protection against internal attackers and even against the global observer. In fact, the number of compromised tunnels can be expected to be slightly smaller than in MorphMix. But as long as there are only a few honest nodes, it is also relatively easy for an adversary to operate a significant subset of all Tarzan nodes by himself. On the other hand, Tarzan is unlikely to cope well with large (e.g. with several 1000 nodes) and dynamic systems because of the requirement for every node to know about all other nodes. MorphMix can cope much better with large systems because there is no need to know about all other nodes. Looking at

the cover traffic, it should be remembered that Tarzan provides better protection from external observers, but at the cost of a significant performance penalty. It all depends on the threat model: if the eavesdropper is considered to be the biggest threat, cover traffic may be a good idea. With our threat model, cover traffic would help only little and its drawbacks would greatly outweigh its benefits.

9.4 Further Work

Besides the limitations of MorphMix we have identified in Section 9.2, there are several open issues regarding mix networks in general. The following list contains some of the challenging questions that remain to be answered. Note that these are general problems that are not dependent on a particular mix network design. Consequently, MorphMix would profit from solving these problems as well.

1. **Cover traffic schemes**: The concept of dummy traffic is still not well understood. In general, there is the question whether there are alternatives to constant streams of cells between a pair of nodes that are much more efficient without reducing protection from attacks. If not, then maybe there are schemes that produce significantly less overhead while reducing the anonymity only marginally. In particular with peer-to-peer-based mix networks, there is the question whether there are efficient cover traffic schemes that would significantly improve the protection from eavesdroppers without introducing too much data overhead and hurting the performance so much such that nobody would to use the system.
2. **Deployment**: How should a mix network be deployed? The problem is that without several other users, there is only little anonymity at all. But users really interested in anonymity won't join before there is a reasonable number of system users. When deploying a peer-to-peer-based mix networks, this problem is even more significant because there is not even a fixed set of static mixes for the first users to begin with. In this case, the only reasonable way to solve the problem is by collecting several users – for instance through mailing lists focusing on anonymity and privacy aspects – that are interested in running a node to provide a basic infrastructure to attract additional users.

3. **Incentives**: What incentives are there for volunteers to operate a mix? This problem of acquiring enough mixes is one of the most crucial issues of static mix networks. In peer-to-peer-based systems, this is a smaller problem and in MorphMix, there are even incentives to relay the data of others because it increases the own protection from attacks. But MorphMix cannot enforce a node to actually act as mix and it may be that like in many peer-to-peer file-sharing systems, 90% of all nodes will be free riders. One way to attack this problem is through reputation systems in the sense that peers that do not offer a service to other peers get a poor reputation over time and are no longer allowed to use services offered by other peers. In general, solving these problems on reputation and incentives in peer-to-peer communities is a very interesting and challenging topic for future research.
4. **Exit abuse**: This is a serious problem in mix networks. It is also coupled to the incentive problem described above: do people really want to handle the web requests of others? What if a Yahoo account is accessed through a mix network and a threatening e-mail message is sent to the President of the USA? Will the operator of the last mix in the chain be prosecuted because the IP addresses in the (possibly available) logs at Yahoo indicate the account has been accessed from her computer? This problem seems more significant in peer-to-peer-based mix networks, because especially in commercially operated static mix networks, the operator can plausibly argue about not having sent the e-mail message himself. One possibility to solve this problem are exit policies in the sense that there is a blacklist at every node that determines what host/port combinations must not be accessed. There is another potential solution to this problem: assuming a system such as MorphMix becomes extremely popular in the sense that there are millions of users that relay traffic for each other. This would significantly "blur" the relationship between IP addresses in IP packets and the computer the data have originally been sent from or are sent to and as a result, IP addresses could possibly no longer be accepted by law enforcement to track down individuals. Another option could be to combine anonymity and accountability such that it is possible to unambiguously identify an anonymous users if certain conditions are met (for instance if a court order to do so has been issued). To do so, mixes would log all data they process including the corresponding keys and the mapping of incoming and outgoing data. If the last mix were accused of having

been the sender of the threatening e-mail message to the President, it would identify the previous mix in the chain. Doing this step-by-step would eventually reveal the true sender. The problem is that the logged information could grow rapidly and pose a burden on the nodes, and a node that accidentally "loses" the logs for any reason could be in trouble. But even worse, any set of mixes could easily blame any user of having contacted a server and having sent or received certain data because there is no binding (for instance a digital signature) between the user and the data she has sent or received that could be used to prove this binding to a third party. Pseudonyms that can be unambiguously linked to an individual's real identity could be a solution to solve this accountability problem. In general, the relation between the concepts of anonymous communication, the systems and operators that implement them, and the needs of society are a topic for future research.

There are several other open problems, but the ones described above are closely related to the work presented in this thesis. We expect that anonymity and privacy-enhancing technologies in general will certainly remain to be a hot and interesting research area during the years to come.

Bibliography

[1] Masayuki Abe. Universally Verifiable MIX with Verification Work Independent of the Number of MIX Servers. In *Proceedings of EUROCRYPT 1998*. Springer-Verlag, LNCS 1403, 1998.

[2] Alessandro Acquisti, Roger Dingledine, and Paul Syverson. On the Economics of Anonymity. In Rebecca N. Wright, editor, *Proceedings of Financial Cryptography (FC '03)*. Springer-Verlag, LNCS 2742, January 2003.

[3] Ross Anderson. The Eternity Service. In *Proceedings of Pragocrypt '96*, 1996.

[4] APNIC, ARIN, and RIPE NCC. IPv6 Address Allocation and Assignment Policy. RIPE Network Coordination Center, ripe-267, 2003.

[5] Adam Back, Ian Goldberg, and Adam Shostack. Freedom 2.1 Security Issues and Analysis. White Paper, `http://www.homeport.org/~adam/zeroknowledgewhitepapers/Freedom_Security2-1.pdf`, May 2001.

[6] Adam Back, Ulf Möller, and Anton Stiglic. Traffic Analysis Attacks and Trade-Offs in Anonymity Providing Systems. In *Proceedings of 4th International Information Hiding Workshop*, Pittsburg, PA, USA, April 2001.

[7] Krista Bennett and Christian Grothoff. GAP – Practical anonymous networking. In Roger Dingledine, editor, *Proceedings of Privacy Enhancing Technologies workshop (PET 2003)*. Springer-Verlag, LNCS 2760, March 2003.

[8] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext Transfer Protocol – HTTP/1.0. RFC 1945, 1996.

[9] Oliver Berthold, Hannes Federrath, and Marit Köhntopp. Project "Anonymity and Unobservability in the Internet". In *Proceedings of the Workshop on Freedom and Privacy by Design / Conference on Freedom and Privacy 2000 CFP*, pages 57–65, Toronto, Canada, April 4–7 2000.

[10] Oliver Berthold, Hannes Federrath, and Stefan Köpsell. Web MIXes: A System for Anonymous and Unobservable Internet Access. In H. Federrath, editor, *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*. Springer-Verlag, LNCS 2009, July 2000.

[11] Oliver Berthold and Heinrich Langos. Dummy Traffic Against Long Term Intersection Attacks. In *Proceedings of the 2nd Workshop on Privacy-Enhancing Technologies*, San Francisco, CA, USA, April 14–15 2002.

[12] Oliver Berthold, Andreas Pfitzmann, and Ronny Standtke. The Disadvantages of Free MIX Routes and how to Overcome them. In H. Federrath, editor, *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*. Springer-Verlag, LNCS 2009, July 2000.

[13] Philippe Boucher, Adam Shostack, and Ian Goldberg. Freedom Systems 2.0 Architecture. White Paper, `http://www.homeport.org/~adam/zeroknowledgewhitepapers/Freedom_System_2_Architecture.pdf`, December 2000.

[14] Lee Breslau, Deborah Estrin, Kevin Fall, Sally Floyd, John Heidemann, Ahmed Helmy, Polly Huang, Steven McCanne, Kannan Varadhan, Ya Xu, and Haobo Yu. Advances in Network Simulation. *IEEE Computer*, May 2000.

[15] D. L. Chaum, A. Fiat, and M. Naor. Untraceable Electronic Cash. In *Advances in Cryprology – CRYPTO '88*, volume 403 of *Lecture Notes in Computer Science*, pages 319–327. Springer-Verlag, 1989.

[16] David L. Chaum. Untraceable Electronic Mail, Return Adresses, and Digital Pseudonyms. *Communications of the ACM*, 24(2):84–88, February 1981.

[17] David L. Chaum. Blind Signature System. In David L. Chaum, editor, *Proceedings of Crypto '83*, page 153. Plenum Press, New York, 1984.

[18] David L. Chaum. The Dining Cryptographers Problem: Unconditional Sender and Receiver Untraceability. *Journal of Cryptology*, 1(1):66–75, 1988.

[19] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System. In *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, pages 46–66, July 2000.

[20] Lance Cottrell. Mixmaster Software. `http://www.obscura.com/~loki/remailer/remailer-essay.html`.

[21] Lance Cottrell. PKCS #1: RSA Cryptography Standard. RSA Laboratories. `http://www.rsasecurity.com/rsalabs/pkcs/pkcs-1/index.html`.

[22] Lance Cottrell. The Anonymizer. `http://www.anonymizer.com`.

[23] Wei Dai. PipeNet. `http://www.eskimo.com/~weidai/pipenet.txt`.

[24] Wei Dai. Two attacks against the Freedom Network. `http://www.eskimo.com/~weidai/freedom-attacks.txt`.

[25] George Danezis. Mix-networks with Restricted Routes. In Roger Dingledine, editor, *Proceedings of Privacy Enhancing Technologies workshop (PET 2003)*. Springer-Verlag, LNCS 2760, March 2003.

[26] George Danezis, Roger Dingledine, and Nick Mathewson. Mixminion: Design of a Type III Anonymous Remailer Protocol. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, May 2003.

[27] S. Deering. Host Extensions for IP Multicasting. RFC 1054, 1988.

[28] S. Deering and R.Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460, 1998.

[29] Thomas Demuth and Andreas Rieke. Securing the Anonymity of Content Providers in the World Wide Web. In *Proceedings of SPIE'99*, pages 494–502, San José, CA, USA, January 1999.

[30] Yvo Desmedt and Kaoru Kurosawa. How To Break a Practical MIX and Design a New One. In *Proceedings of EUROCRYPT 2000*. Springer-Verlag, LNCS 1803, 2000.

[31] Claudia Díaz and Andrei Serjantov. Generalising Mixes. In Roger Dingledine, editor, *Proceedings of Privacy Enhancing Technologies workshop (PET 2003)*. Springer-Verlag, LNCS 2760, March 2003.

[32] Claudia Díaz, Stefaan Seys, Joris Claessens, and Bart Preneel. Towards Measuring Anonymity. In Roger Dingledine and Paul Syverson, editors, *Proceedings of Privacy Enhancing Technologies Workshop (PET 2002)*. Springer-Verlag, LNCS 2482, April 2002.

[33] T. Dierks and C. Allen. The TLS Protocol, Version 1.0. RFC 2246, 1999.

[34] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, November 1976.

[35] Roger Dingledine, Michael Freedman, David Hopwood, and David Molnar. A Reputation System to Increase MIX-net Reliability. In *Proceedings of 4th International Information Hiding Workshop*, pages 126–141, Pittsburg, PA, USA, April 2001.

[36] Roger Dingledine, Michael J. Freedman, and David Molnar. The Free Haven Project: Distributed Anonymous Storage Service. In H. Federrath, editor, *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*. Springer-Verlag, LNCS 2009, July 2000.

[37] Roger Dingledine and Paul Syverson. Reliable MIX Cascade Networks through Reputation. In *Proceedings of Financial Cryptography 2002*. Springer-Verlag, March 2002.

[38] R. Droms. Dynamic Host Configuration Protocol. RFC 2131, 1997.

[39] Andy Oram (Editor). *Peer-to-Peer: Harnessing the Power of Disruptive Technology*. O'Reilly, first edition, 2001.

[40] Jonathan B. Postel (editor). Internet Protocol. RFC 791, 1981.

[41] Jonathan B. Postel (editor). Transmission Control Protocol. RFC 793, 1982.

[42] Electronic Frontiers Georgia (EFGA). Anonymous Remailer Information. `http://anon.efga.org/Remailers/`.

[43] K. Egevang and P. Francis. The IP Network Address Translator (NAT). RFC 1631, 1994.

[44] Anja Feldmann, Anna C. Gilbert, Polly Huang, and Walter Willinger. Dynamics of IP Traffic: A Study of the Role of Variability and the Impact of Control. In *Proceeding of SIGCOMM '99*, Massachusetts, USA, September 1999.

[45] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, 1999.

[46] Secure Hash Standard. United States of America, National Institute of Science and Technology, Federal Information Processing Standard (FIPS) 180-1, April 1993.

[47] Advanced Encryption Standard (AES). United States of America, National Institute of Science and Technology, Federal Information Processing Standard (FIPS) 197, November 2001.

[48] Elke Franz, Anja Jerichow, and Guntram Wicke. A Payment Scheme for Mixes Providing Anonymity. In W. Lamersdorf and M. Merz, editors, *Trends in Distributed Systems for Electronic Commerce*, Lecture Notes in Computer Science, pages 94–108. Springer-Verlag, 1998.

[49] Michael J. Freedman. Private communication, January 2003.

[50] Michael J. Freedman and Robert Morris. Tarzan: A Peer-to-Peer Anonymizing Network Layer. In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS 2002)*, Washington, D.C., USA, November 2002.

[51] A. Freier, P. Karlton, and P. Kocher. The SSL Protocol, Version 3.0. Netscape Communications, 1996.

[52] Jun Furukawa and Kazue Sako. An Efficient Scheme for Proving a Shuffle. In Joe Kilian, editor, *Proceedings of CRYPTO 2001*. Springer-Verlag, LNCS 2139, 2001.

[53] E. Gabber, P.B. Gibbons, Y. Matias, and Y Mayer. How to Make Personalized Web Browsing Simple, Secure and Anonymous. In *Proceedings of Financial Cryptography '97*, pages 17–31. Springer-Verlag, February 1997.

[54] Ian Goldberg. *A Pseudonymous Communications Infrastructure for the Internet*. PhD thesis, UC Berkeley, December 2000.

[55] Ian Goldberg and Adam Shostack. Freedom Network 1.0 Architecture and Protocols. White Paper, `http://www.homeport.org/~adam/zeroknowledgewhitepapers/arch-tech.pdf`, November 1999.

[56] Ian Goldberg and David Wagner. TAZ Servers and the Rewebber Network: Enabling Anonymous Publishing on the World Wide Web. *First Monday*, 3(4), August 1998.

[57] David M. Goldschlag, Michael G. Reed, and Paul F. Syverson. Hiding Routing Information. In *Information Hiding*, Lecture Notes in Computer Science, pages 137–150. Springer-Verlag, 1996.

[58] Ceki Gülcü and Gene Tsudik. Mixing E-mail With Babel. In *Proceedings of the Network and Distributed Security Symposium - NDSS '96*, pages 2–16. IEEE, February 1996.

[59] Johan Helsingius. `anon.penet.fi` press release. `http://www.penet.fi/press-english.html`.

[60] R. Housely and W. Polk. Internet X.509 Public Key Infrastructure. RFC 2528, 1999.

[61] Markus Jakobsson. Flash Mixing. In *Proceedings of Principles of Distributed Computing - PODC '99*. ACM Press, 1999.

[62] Markus Jakobsson and Ari Juels. An Optimally Robust Hybrid Mix Network (Extended Abstract). In *Proceedings of Principles of Distributed Computing - PODC '01*. ACM Press, 2001.

[63] Anja Jerichow, Jan Müller, Andreas Pfitzmann, Birgit Pfitzmann, and Michael Waidner. Real-Time Mixes: A Bandwidth-Efficient Anonymity Protocol. *Journal on Selected Areas in Communications*, 16(4):495–509, May 1998.

[64] B. Kantor. BSD Rlogin. RFC 1282, 1991.

[65] R. Keller, L. Ruf, A. Guindehi, and B. Plattner. PromethOS: A Dynamically Extensible Router Architecture Supporting Explicit Routing. In James Sterbenz, Osamu Takada, Christian Tschudin, and Bernhard Plattner, editors, *Proceedings of the Fourth Annual International Working Conference on Active Networks IWAN*, number 2546 in Lecture Notes in Computer Science, Zurich, Switzerland, December 2002. Springer Verlag.

[66] S. Kent and R. Atkinson. Security Architecture for the Internet Protocol. RFC 2401, 1998.

[67] Dogan Kesdogan, Dakshi Agrawal, and Stefan Penz. Limits of Anonymity in Open Environments. In Fabien Petitcolas, editor, *Proceedings of Information Hiding Workshop (IH 2002)*. Springer-Verlag, LNCS 2578, October 2002.

[68] Dogan Kesdogan, Jan Egner, and Roland Büschkes. Stop-and-Go MIXes: Providing Probabilistic Anonymity in an Open System. In *Proceedings of Information Hiding Workshop (IH 1998)*. Springer-Verlag, LNCS 1525, 1998.

[69] D. Kristol and L. Montulli. HTTP State Management Mechanism. RFC 2109, 1997.

[70] Dennis Kügler. An Analysis of GNUnet and the Implications for Anonymous, Censorship-Resistant Networks. In Roger Dingledine, editor, *Proceedings of Privacy Enhancing Technologies workshop (PET 2003)*. Springer-Verlag, LNCS 2760, March 2003.

[71] Steven Low, Nicholas Maxemchuk, and Sanjoy Paul. Anonymous Credit Cards. In *Proceedings of the 2nd Annual ACM Conference on Computer and Communications Security*, pages 108–117, 1994.

[72] Bruce A. Mah. An Empirical Model of HTTP Network Traffic. In *Proceeding of Infocom 1997*, pages 592–600, Kobe, Japan, April 1997.

[73] Tim May. Description of Early Remailer History. `http://www.inet-one.com/cypherpunks/dir.1996.08.29-1996.09.04/msg00431.html`.

[74] David Mazières and M. Frans Kaashoek. The Design, Implementation and Operation of an Email Pseudonym Server. In *Proceedings of the 5th ACM Conference on Computer and Communications Security (CCS'98)*. ACM Press, November 1998.

[75] M. Mitomo and K. Kurosawa. Attack for Flash MIX. In *Proceedings of ASIACRYPT 2000*. Springer-Verlag, LNCS 1976, 2000.

[76] P. Mockapetris. Domain Names – Concepts and Facilities. RFC 1034, 1987.

[77] P. Mockapetris. Domain Names – Implementation and Specification. RFC 1035, 1987.

[78] J. Mogul and J. Postel. Internet Standard Subnetting Procedure. RFC 950, 1985.

[79] Ulf Möller and Lance Cottrell. Mixmaster Protocol – Version 2. Unfinished draft. `http://www.obscura.com/~loki/remailer/remailer-essay.html`, January 2000.

[80] C. Andrew Neff. A Verifiable Secret Shuffle and its Application to E-Voting. In P. Samarati, editor, *Proceedings of 8th ACM Conference on Computer and Communications Security (CCS-8)*, pages 116–125. ACM Press, November 2001.

[81] Ron Newman. The Church of Scientology vs. `anon.penet.fi`. `http://www.xs4all.nl/~kspaink/cos/rnewman/anon/penet.html`.

[82] Miyaku Ohkubo and Masayuki Abe. A Length-Invariant Hybrid MIX. In *Proceedings of ASIACRYPT 2000*. Springer-Verlag, LNCS 1976, 2000.

[83] Andreas Pfitzmann and Marit Köhntopp. Anonymity, Unobservability, and Pseudonymity - A Proposal for Terminology; Draft v0.12. `http://www.koehntopp.de/marit/pub/anon/Anon_Terminology.pdf`, June 17 2001.

[84] Andreas Pfitzmann, Birgit Pfitzmann, and Michael Waidner. ISDN-MIXes: Untraceable Communication with Very Small Bandwith Overhead. In *Kommunikation in verteilten Systemen, 267*, pages 451–463, 1991.

[85] David C. Plummer. Ethernet Address Resolution Protocol. RFC 826, 1982.

[86] W. Polk, R. Housley, and L. Bassham. Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 3279, April 2002.

[87] J. Postel and J. Reynolds. Telnet Protocol Specification. RFC 854, 1983.

[88] J. Postel and J. Reynolds. File Transfer Protocol. RFC 959, 1985.

[89] Jonathan B. Postel. User Datagram Protocol. RFC 768, 1980.

[90] Jonathan B. Postel. Simple Mail Transfer Protocol. RFC 821, 1982.

[91] Sandro Rafaeli, Marc Rennhard, Laurent Mathy, Bernhard Plattner, and David Hutchison. An Architecture for Pseudonymous e-Commerce. In *Proceedings of the AISB'01 Symposium on Information Agents for Electronic Commerce*, pages 33–42, York, UK, March 21–24 2001.

[92] Jean-François Raymond. Traffic Analysis: Protocols, Attacks, Design Issues and Open Problems. In H. Federrath, editor, *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*. Springer-Verlag, LNCS 2009, July 2000.

[93] Joseph Reagle and Lorrie Faith Cranor. The Platform for Privacy Preferences. *Communications of the ACM*, 42(2), February 1999.

[94] Michael Reed, Paul Syverson, and David Goldschlag. Anonymous Connections and Onion Routing. *IEEE Journal on Selected Areas in Communications*, 16(4):482–494, May 1998.

[95] Michael K. Reiter and Aviel D. Rubin. Crowds: Anonymity for Web Transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, November 1998.

[96] Y. Rekhter and T. Li. An Architecture for IP Address Allocation with CIDR. RFC 1518, 1993.

[97] Marc Rennhard. MorphMix: Peer-to-Peer based Anonymous Internet Usage with Collusion Detection (available at `http://www.tik.ee.ethz.ch/~rennhard/publications/morphmix.pdf`). TIK Technical Report Nr. 147, TIK, ETH Zurich, Zurich, CH, August 2002.

[98] Marc Rennhard and Bernhard Plattner. Introducing MorphMix: Peer-to-Peer based Anonymous Internet Usage with Collusion Detection. In *Proceedings of the Workshop on Privacy in the Electronic Society (in association with 9th ACM Conference on Computer and Communications Security)*, pages 91–102, Washington, DC, USA, November 21 2002.

[99] Marc Rennhard and Bernhard Plattner. Practical Anonymity for the Masses with Mix-Networks. In *Proceedings of the IEEE 8th Intl. Workshop on Enterprise Security (WET ICE 2003)*, Linz, Austria, June 9–11 2003.

[100] Marc Rennhard and Bernhard Plattner. Practical Anonymity for the Masses with MorphMix. In *Proceedings of the Financial Cryptography Conference (FC 2004)*, Key West, USA, February 9–12 2004.

[101] Marc Rennhard, Sandro Rafaeli, and Laurent Mathy. From SET to PSET – The Pseudonymous Secure Electronic Transaction Protocol (available at `http://www.tik.ee.ethz.ch/~rennhard/publications/PSET.pdf`). TIK Technical Report Nr. 117, TIK, ETH Zurich, Zurich, CH, August 2001.

[102] Marc Rennhard, Sandro Rafaeli, Laurent Mathy, Bernhard Plattner, and David Hutchison. An Architecture for an Anonymity Network. In *Proceedings of the IEEE 6th Intl. Workshop on Enterprise Security (WET ICE 2001)*, pages 165–170, Boston, USA, June 20–22 2001.

[103] Marc Rennhard, Sandro Rafaeli, Laurent Mathy, Bernhard Plattner, and David Hutchison. Analysis of an Anonymity Network for Web Browsing. In *Proceedings of the IEEE 7th Intl. Workshop on Enterprise Security (WET ICE 2002)*, pages 49–54, Pittsburgh, USA, June 10–12 2002.

[104] Marc Rennhard, Sandro Rafaeli, Laurent Mathy, Bernhard Plattner, and David Hutchison. Towards Pseudonymous e-Commerce. *Electronic Commerce Research Journal, Special Issue on Security and Trust in Electronic Commerce, Kluwer Academics Publisher*, 4(1–2):83–111, January–April 2004.

[105] E. Rescorla. HTTP over TLS. RFC 2818, 2000.

[106] R.Hinden and S. Dering. IP Version 6 Addressing Architecture. RFC 2373, 1998.

[107] Ron Rivest, Adi Shamir, and Leonard Adleman. A Method for Obtaining Difital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[108] E. Rosen and Y. Rekhter. BGP/MPLS VPNs. RFC 2547, 1999.

[109] Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems. In *Proceedings of Multimedia Computing and Networking 2002 (MMCN '02)*, San Jose, CA, USA, January 2002.

[110] Bruce Schneier. *Applied Cryptography*. John Wiley & Sons, Inc., second edition, 1996.

[111] Andrei Serjantov and George Danezis. Towards an Information Theoretic Metric for Anonymity. In Roger Dingledine and Paul Syverson, editors, *Proceedings of Privacy Enhancing Technologies Workshop (PET 2002)*. Springer-Verlag, LNCS 2482, April 2002.

[112] Andrei Serjantov, Roger Dingledine, and Paul Syverson. From a Trickle to a Flood: Active Attacks on Several Mix Types. In *Proceedings of 5th International Information Hiding Workshop*, Noordwijkerhout, Netherlands, October 2002.

[113] Andrei Serjantov and Richard E. Newman. On the anonymity of timed pool mixes. In *Proceedings of the Workshop on Privacy and Anonymity Issues in Networked and Distributed Systems*, pages 427–434, Athens, Greece, May 2003. Kluwer.

[114] Adi Shamir. How to Share a Secret. *Communications of the ACM*, 22(11), November 1979.

[115] Claude Shannon. The Mathematical Theory of Communication. *Bell Systems Technical Journal*, 27(3):379–423; 623–656, 1948.

[116] Rob Sherwood, Bobby Bhattacharjee, and Aravind Srinivasan. P5: A Protocol for Scalable Anonymous Communication. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, May 2002.

[117] Adam Shostack. Private communication, January–October 2003.

[118] Adam Shostack and Ian Goldberg. Freedom 1.0 Security Issues and Analysis. White Paper, `http://www.homeport.org/~adam/zeroknowledgewhitepapers/Freedom-Security.pdf`, November 1999.

[119] W. Simpson. IP in IP Tunnelling. RFC 1853, 1995.

[120] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proceedings of the ACM SIGCOMM '01 Conference*, San Diego, CA, USA, August 2001.

[121] Paul Syverson, Michael Reed, and David Goldschlag. Onion Routing Access Configurations. In *DARPA Information Survivability Conference and Exposition (DISCEX 2000)*, volume 1, pages 34–40. IEEE CS Press, 2000.

[122] Paul Syverson, Gene Tsudik, Michael Reed, and Carl Landwehr. Towards an Analysis of Onion Routing Security. In H. Federrath, editor, *Proceedings of Designing Privacy Enhancing Technologies: Workshop*

on Design Issues in Anonymity and Unobservability. Springer-Verlag, LNCS 2009, July 2000.

[123] Marc Waldman and David Mazières. Tangler: a censorship-resistant publishing system based on document entanglements. In *Proceedings of the 8th ACM Conference on Computer and Communications Security (CCS'01)*, pages 126–135, November 2001.

[124] Marc Waldmann, Aviel D. Rubin, and Lorrie Faith Cranor. Publius: A Robust, Tamper-Evident, Censorship-Resistant Web Publishing System. In *Proceedings of the 9th USENIX Security Symposium*, August 2000.

[125] Nathalie Weiler. Secure Anonymous Group Infrastructure for Common and Future Internet Applications. In *Proceedings of 17th Annual Computer Security Applications Conference 2001 (ACSAC'01)*, pages 401–411, New Orleans, LO, USA, December 2001.

[126] Nathalie Weiler. *SANGRIA – Secure ANonymous GRoup InfrAstructure*. PhD thesis, Swiss Federal Institute of Technology Zurich, 2002.

[127] Matthew Wright, Micah Adler, Brian Neil Levine, and Clay Shields. An Analysis of the Degradation of Anonymous Protocols. In *Proceedings of ISOC Network and Distributed System Security Symposium (NDSS 2002)*, San Diego, USA, February 2002.

[128] Matthew Wright, Micah Adler, Brian Neil Levine, and Clay Shields. Defending Anonymous Communication Against Passive Logging Attacks. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, May 2003.

[129] Phil R. Zimmermann. *The Official PGP User's Guide*. Boston: MIT Press, 1995.

Acknowledgements

I want to express my gratitude to Prof. Dr. Bernhard Plattner for guidance and providing me with an environment to freely explore the world of scientific research. Furthermore, I'd like to thank Dr. Laurent Mathy, not only for valuable comments on my work, but also for fruitful collaboration when we were working together on a European research project.

Many thanks to my colleagues at the Computer Engineering and Networks Laboratory at ETH Zurich for inspiring discussions and enjoyable social activities. I'm especially happy about having made new friends; notably Nathalie Weiler, Matthias Bossardt, and Lukas Ruf. I'm also thankful for the contributions of various students who wrote semester or diploma theses under my supervision.

Finally, my biggest thanks go to my wife Brigitte and my parents for love, support, and understanding.

Biography

I was born in Chur, Switzerland on 10th February 1972 where I also attended primary and high schools and graduated with a Matura degree (Typus C) in June 1991. After attending compulsory military service and visiting a language school abroad, I started my studies in electrical engineering at the Swiss Federal Institute of Technology (ETHZ) in October 1992. I interrupted my studies for one year to work as a software engineer and received the diploma (masters) degree in electrical engineering (Dipl. El.-Ing. ETH) in April 1998.

From Mai 1998 until September 1999, I was working as an IT consultant before I decided to join the Computer Engineering and Networks Laboratory (TIK) at ETHZ as a research assistant. After spending a few months on a project dealing with charging and accounting technologies for the Internet, I started working on a research project in the area of pseudonymous e-commerce in April 2000. While working on this project, I became familiar with problems surrounding anonymity in the Internet in general and mix networks in particular. Influenced by my own experiences while designing and implementing a traditional mix network within the context of the project, I started my work on MorphMix in February 2002.

While working at TIK, I also undertook postgraduate studies in higher education at ETHZ starting in October 2000 and graduated with the diploma for didactics in computer science (Didaktischer Ausweis in Informatik) in March 2003.

Appendix A

MorphMix Protocol and Prototype Implementation

In this appendix, we describe the details of the MorphMix protocol. We first introduce the notation we are using throughout this appendix. Then, we describe the protocol basics which includes the various cryptographic algorithms we employ, the cell format, the different node levels, and the encoding of various fields in messages. Afterwards, we give the meaning and precise format of all messages exchanged between neighbouring nodes and between the endpoints of an anonymous tunnel. We also give detailed descriptions of the life cycle of virtual links and tunnels, and the policy for using virtual links. Finally, we make a quantitative analysis of the data overhead induced by MorphMix and give a brief description of the MorphMix prototype implementation.

A.1 Notation

We use the following notation: a is the node that sets up a virtual link to node b. Similarly, when a node is appended to a tunnel, a is the initiator and b is the final node of the tunnel that has been set up so far. w is the witness. Node c is the node that is selected as the next hop when a node is appended to a tunnel, and $d, e, f, \ldots$ are the other nodes b offered in the selection but that were not chosen. When end-to-end messages are sent through a tunnel, they are sent

from a via b to c and we assume they are currently on the virtual link between a and b. $\mathrm{k}_{VL_{xy}}$ is the symmetric key for the virtual link between nodes x and y. $\mathrm{k}_{LE_{xy}}$ is the symmetric key for the layer of encryption between nodes x and y. PK_x and SK_x are the corresponding public and secret keys of node x. ip_x is the IP address of x. lev_x is the node level of node x. MorphMix node x listens for incoming connections on port p_{mm_x}. The MorphMix port is 28080 per default, but can be specified by the node operator. This port is used to contact a node that has been chosen as a new neighbour. In addition, there is a witness port p_{wit_x}, which is always specified as $\mathrm{p}_{wit_x} + 1$. This port is used to establish a virtual link to a witness and from the witness to the node that is appended to a tunnel. The reason for having two ports is that if a node does not want to accept being selected as a neighbour, it simply stops listening on the MorphMix port. On the other hand, a node should always listen on its witness port even if it does not accept being selected as a new neighbour. Before setting up a virtual link, we assume a knows ip_b, p_{mm_b} (which implies a also knows p_{wit_b} $(= \mathrm{p}_{mm_b} + 1)$, PK_b, and lev_b and before appending a node to a tunnel, a knows ip_w, p_{mm_w}, and PK_w. All these assumption make sense due to the peer discovery mechanism in Section 5.7. Finally, h is the host that is contacted by a through the anonymous tunnel.

A.2 Basic Protocol Properties

In this section, we describe the basic protocol properties. We first define the cryptographic algorithms we employ. Then, we describe the format of a cell, the different node levels, and the encoding used in various fields of protocol messages.

A.2.1 Cryptographic Algorithms

MorphMix makes use of different cryptographic algorithms. For symmetric key cryptographic operations, we employ the Advanced Encryption Standard (AES) [47] with 128-bit keys in cipher block chaining (CBC) mode [110] with no padding. For public key operations, we use RSA [107] with 2048-bit moduli. The same key pair is used for signing and encrypting. To encrypt with the public key, we basically use RSA in electronic codebook (ECB) mode [110] with PKCS1 (version 1.5) padding [21], which operates on 245-byte blocks. If more than 245 bytes must be encrypted with the public key,

we use a hybrid scheme as follows to encrypt n bytes: first, a 16-byte initialisation vector iv and a 16-byte symmetric key k are randomly chosen, which serve as the input to an AES cipher that is operated as described above. Then, the concatenation of iv, key, and the first $k = n - (16 \cdot \lceil (n - 245 + 32)/16 \rceil)$ bytes of the data are encrypted using RSA with the recipients public key, which results in d_1. Since this input is guaranteed to be smaller than 245 bytes, only one block is encrypted. In addition, the remaining $(n - k)$ bytes of the data are encrypted using the AES cipher, which results in d_2. The concatenation of d_1 and d_2 is the output, i.e. the ciphertext of the hybrid encryption. Decrypting the ciphertext works vice versa. To sign, we employ RSA with SHA1 [46]. The public exponent of every node is fixed to $2^{16} + 1 = 65537$ and the modulus of node x is given by mod_x. To get the symmetric keys for a layer of encryption, we use the Diffie-Hellman (DH) key-exchange algorithm [34] with a 1024-bit modulus.

A.2.2 Cell Format

The format of a MorphMix cell exchanged between two neighbouring nodes is depicted in Figure A.1.

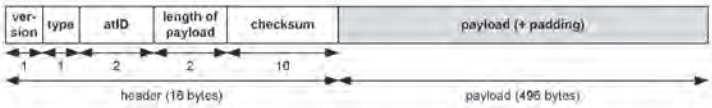


Figure A.1: *Cell format.*

All cells have a length of 512 bytes and consist of a 16-byte header and a 496 bytes payload (including the padding). The *version* and *type* fields are defined as unsigned 8-bit integers. The current version is 1.0, and the various types are needed to identify the type of message that is transported in the payload of the cell. The *atID* is an unsigned 16-bit integer and is needed to multiplex anonymous tunnels on a virtual link between two nodes. The atID has only local significance on the virtual link between two nodes. The unsigned 16-bit integer field *length of payload* identifies the number of bytes in the *payload* field, excluding the padding. If needed, the payload is padded with random bits to make sure the cell has its fixed length.

To protect the integrity of a cell, there is a 10-byte *checksum*, which is made over the concatenation of the first 6 bytes of the header and the whole

payload, including the padding. SHA1 produces a 20-bytes output, of which we XOR the left and right 10 bytes to generate the checksum. 10 bytes are enough to detect if the cell has been tampered with in transit between two nodes if the header is encrypted. It also makes replay attacks virtually impossible, because if the header is encrypted, the decryption of the header of a replayed cell results in a completely different plaintext header because we are using AES in CBC mode. The probability a replayed cell is not detected is 2^{-80}.

If messages are exchanged between neighbours, they are directly transported within the payload of cells. In the case of end-to-end messages that are exchanged between the endpoints of an anonymous tunnel, an additional anonymous connection header is inserted at the beginning of a cell payload. The anonymous connection header looks exactly the same as a cell header and the atID corresponds to the anonymous connection identifier (acID). Using the concept of anonymous connections (see Section 5.2) allows to use a single anonymous tunnel for multiple anonymous communication relationships in parallel.

The atIDs are assigned as follows: the node that initiated establishing the virtual link uses odd, the other even unsigned 16-bit integers. Note that according to our policy about how virtual links are used (see Appendix A.5.2), this implies that even number will not be used at this time, but we nevertheless separate between odd and even numbers in case this policy changes in future versions. atIDs are assigned sequentially for each new anonymous tunnel transported within a virtual link between the nodes, starting with 1 (1,3,5,7,...) and 2 (2,4,6,8,...), respectively. If they ever reach their maximum (65535 and 65534), they simply wrap around, although it is extremely unlikely a virtual link between two nodes remains active for such a long time (see Appendix A.5.1). It works in the same way for anonymous connections: the initiator uses odd acIDs and the final node uses even acIDs to multiplex anonymous connections within one tunnel (although at this time, we do not allow connections being established by the final node to the initiator) . The atID/acID 0 is reserved for control data exchanged between neighbours or between the endpoints of an anonymous tunnel that do not belong to a particular anonymous tunnel/connection or if no such identifier has been assigned yet (for instance during the setup of anonymous tunnels).

A.2.3 Node Levels

To optimise the quality of anonymous tunnel, each node has a specific node level that depends on its up- and down-stream bandwidths. Table A.1 defines the node levels used in MorphMix. The table also lists the encoding of the different levels and the default minimum level the nodes along a tunnel should have depending on the node level of the initiator.

Table A.1: *Node levels in MorphMix*

node level	bandwidth (Kb/s)		enc.	default min. level
	up-stream	down-stream		of nodes along tunnel
slower than ISDN	< 64	< 64	1	1
ISDN	64	64	2	2
ADSL/Cable_{256}	64	256	3	4
ADSL/Cable_{512}	128	512	4	5
ADSL/Cable_{1024}	256	1024	5	6
DSL_{512}	512	512	6	5
T1	1544	1544	7	6
T3 or faster	≥ 4632	≥ 4632	8	6

To make use of the node levels, the operator of a node specifies the level of its node according to Table A.1. If none of the levels matches her Internet connection, she picks the highest level that both up- and down-stream bandwidths of her connections are at least as large as those of the chosen level. In addition, she may modify the minimum level for the nodes she accepts in her anonymous tunnels or simply use the default one. It is important to remember that selecting the minimum acceptable level is a trade-off between performance and protection from attacks. The default levels are similar to those used in our analyses in Sections 7.4 and 8.3.3, which have turned out to be reasonable compromises.

A.2.4 Encoding

Whenever control messages are exchanged between nodes (for instance to set up virtual link and anonymous tunnels), the payload itself contains formatted control data fields. To make sure that two nodes can communicate, the data in these fields must be encoded in a clearly specified way. In general, we always encode values in big-endian order, i.e. the most significant byte comes first. In addition, most fields have a fixed length. One exception are the DH key-exchange parameters, where the public key is represented in ASN.1 encoding

for public keys [86], which corresponds to at most 296 bytes. Table A.2 lists the encoding of several fields.

Table A.2: *Encoding of fields in the payload.*

field	length	description
IP address (ip_x)	4	Each of the four bytes of an IP address is stored in one of the four unsigned bytes
port (p_x, p_{mm_x})	2	Ports are encoded as unsigned 16-bit integers
RSA modulus (mod_x)	256	The 2048 bits are encoded as 256 unsigned bytes
node level (lev_{min}, lev_x)	1	The node level is encoded as an 8-bit unsigned integer
nonce ($nonce_{x,y}$)	16	A nonce is always 16 unsigned bytes long. We use nonces throughout the protocol to recognise replies to a particular request and to guarantee the freshness of these replies.
symmetric key ($k_{\{VL\|LE\}_{x,y}}$)	16	The 128 bits are encoded as 16 unsigned bytes
initialisation vector (iv)	16	The 128 bits are encoded as 16 unsigned bytes
selection size (n_{sel})	1	The selection size is encoded as an 8-bit unsigned integer
length (l_x)	2	Length fields are represented as unsigned 16-bit integers and identify the number of bytes of the following field
DH parameters (DH_x)	≤ 296	The DH public key (of one party) is represented in ASN.1 encoding for public keys [86]
number of nodes for peer discovery (n_{req}, n_{rep})	1	The requested and replied number of nodes for discovery is encoded as an 8-bit unsigned integer
information (info, $info_x$)	1	Status information exchanged between two neighbours or information about the host x, represented as unsigned 8-bit integers
RSA public key encryption ($\{\ldots\}_{PK_x}$)	$n \cdot 256$	Data encrypted with x's RSA public key are represented as unsigned bytes; the length is a multiple of 256 bytes
RSA signature ($\{\ldots\}_{SK_x}$)	256	Data hashed with SHA1 and then encrypted with x's RSA private key, represented as 256 unsigned bytes

We describe the various message types that are exchanged between neighbours and between the endpoints of an anonymous tunnels. According to Appendix A.2.2, the types are encoded as 8-bit unsigned integers. Since the length of some messages can be longer than what fits into a single cell, there is a simple rule to recognise when all cells of a message have been received:

The last cell of a message always has a different type than the others. For instance, the selection a node offers when appending a node to the tunnel is an end-to-end message that uses multiple cells. Therefore, the type field in the anonymous connection headers is *SEL_REQ* in all cells except the last, where it is *SEL_REQ_FINAL*. Table A.3 lists the encoding of all message types used in MorphMix.

Table A.3: *Encoding of message types.*

type	encoding	type	encoding
messages between neighbours			
LINK_REQ	11	LINK_REP	12
WIT_REQ	16	WIT_REQ_FINAL	17
WIT_REP	18	WIT_FAIL	19
NEXT_REQ	21	NEXT_REP	22
NEXT_FAIL	23		
ADD_REQ	26	ADD_REP	27
ADD_FAIL	28		
NODES_REQ	31	NODES_REP	32
NODES_REP_FINAL	33		
STAT_REQ	36	STAT_REP	37
STAT_PUSH	38		
TERM	41		
CREDIT	46		
LINK_DATA	51		
end-to-end messages			
SEL_REQ	61	SEL_REP	62
SEL_REP_FINAL	63	SEL_FAIL	64
APP_REQ	66	APP_REQ_FINAL	67
APP_REP	68	APP_REP_FINAL	69
APP_FAIL	70		
CON_REQ	71	CON_REQ_FINAL	72
CON_REP	73	CONDATA_REQ	74
CONDATA_REQ_FINAL	75	CON_CLOSE	76
CON_FAIL	77		
E2E_PING	81	E2E_PONG	82
E2E_DATA	86		

A.3 Messages between Neighbours

In this section, we precisely describe all messages in MorphMix that are exchanged between neighbours. They are used to set up virtual links, to set up and terminate anonymous tunnels, for peer discovery, to get status information about neighbours, for flow control, and to carry end-to-end messages. With the exception of the case when carrying end-to-end messages, layered encryption is not relevant if messages are exchanged between neighbours. Therefore, both the header and payload are usually encrypted using the key of the corresponding virtual link.

A.3.1 Establishing a Virtual Link

Whenever two nodes in MorphMix communicate directly with each other, they establish a virtual link. To do so, a establishes a TCP connection with b using ip_b and p_{mm_b} (or p_{wit_b} if a or b act as a witness). Node a then selects randomly three 128-bit values: a *nonce*$_{a,b}$, an *initialisation vector (iv)* for the AES cipher because it operates in CBC mode, and a *key* $k_{VL,ab}$, which will be used to encrypt data sent across the virtual link. Node a also includes its IP address ip_a, port p_{mm_a}, RSA modulus mod_a, and node level lev_a. If a is located in a private network behind a NAT gateway and port forwarding to connect to a has been enabled on the NAT gateway, a includes the public IP address and the corresponding port of the NAT gateway instead of its own. All these data are concatenated, encrypted with b's public key PK_b, and sent to b in a *LINK_REQ* message. Since the resulting data have a length of 368 bytes, they fit into the payload of a single cell. The atID is set to 0 because the messages do not belong to any particular anonymous tunnel. Cells carrying messages of type LINK_REQ are the only cells where the header is transmitted in the clear.

Upon receiving the message, b decrypts the payload to get the data that were encrypted by a. iv and $\text{k}_{VL,ab}$ are used as input to the AES cipher to encrypt the data on this virtual link. In addition, ip_a, p_{mm_a}, mod_a, and lev_a are passed to the peer discovery mechanism (see Section 5.7). If b is willing to accept the connection, it generates message of type *LINK_REP* that contains $\text{nonce}_{a,b}$. The header and the whole payload of the resulting cell are encrypted with $\text{k}_{VL,ab}$ and sent back to a. Node a gets the cell, decrypts the header, sees that it is a message of type LINK_REP, checks the included nonce to make sure it really is a valid reply to its own message and the virtual link

is established. If there is any problem with the LINK_REP message, a simply tears down the TCP connection to b. If b cannot decrypt the LINK_REQ message or does not want to accept the connection from a for any reason, it simply closes the TCP connection. As mentioned in Appendix A.1, b can simply stop listening on p_{mm_b} if it does not want to be chosen as a neighbour. However, a node should always accept virtual links being established to p_{wit_b} to guarantee setting up anonymous tunnels succeeds with high probability. Table A.4 summarises all fields and cells used during the setup of a virtual link.

Table A.4: *Fields and cells to establish a virtual link.*

field	length	description
$nonce_{a,b}$	16	Nonce for a to recognise b's reply
iv	16	Initialisation vector for the AES cipher
$k_{VL,ab}$	16	Symmetric key used for the virtual link

ver-sion	type	at ID	pld len	payload	key to encrypt header	key to encrypt payload
1.0	LINK_REQ	0	368	$\{nonce_{a,b} \parallel iv \parallel k_{VL,ab} \parallel ip_a \parallel p_{mm_a} \parallel mod_a \parallel lev_a\}_{PK_b}$	no	no
1.0	LINK_REP	0	16	$nonce_{a,b}$	$k_{VL,ab}$	$k_{VL,ab}$

A.3.2 Appending a Node to a Tunnel

Messages 1–3 and 10 in Figure 5.5 are end-to-end messages and will be handled in Appendix A.4.1. Here, we describe messages 4–9.

Upon receiving message 3 in Figure 5.5, b sets up a virtual link to w. As mentioned in Appendix A.5.2, this is only needed if b and w are currently not neighbours. Node b then generates a *WIT_REQ* message. Since the length of the message is longer than the payload of a cell, it results in two cells. The first cell has type WIT_REQ and the second WIT_REQ_FINAL. The message includes a nonce and the encrypted payload b received from a in message 3.

The witness w decrypts the encrypted data to get ip_c, p_{mm_c}, and mod_c. It establishes a virtual link to c by connecting to c's witness port p_{wit_c} ($= p_{mm_c} + 1$) if w and c are currently not neighbours. Then, w generates a *NEXT_REQ* message, which contains a new nonce ($nonce_{w,c}$), $nonce_{b,c}$, ip_b, and a's DH public key that is encrypted for c. If w cannot establish a virtual

link to c or if there is a problem with the WIT_REQ message, w sends a *WIT_FAIL* message to b that contains $\text{nonce}_{w,c}$.

Node c decrypts a's DH public key and completes the key-exchange to get the shared secret. The first 16 bytes of this shared secret are used as the initialisation vector and the next 16 bytes as the symmetric key $k_{LE,ac}$ for the AES cipher that is used for the layer of encryption between a and c. Node c then generates its own DH public key, DH_c of the key-exchange and replies to w with a *NEXT_REP* message that contains $\text{nonce}_{w,c}$. If c does not accept being the next node in the tunnel or if there is another problem, c replies with a *NEXT_FAIL* message to w. Upon receiving the NEXT_REP message, w generates the receipt for a, which includes $\text{nonce}_{a,w}$, the IP addresses of b and c, and a digital signature over these data. It then sends $\text{nonce}_{b,w}$ and the receipt to b in a *WIT_REP* message. If w has received a NEXT_FAIL message from c, it sends a WIT_FAIL message to b.

After having received the WIT_REP message, b sends an *ADD_REQ* message containing $\text{nonce}_{b,c}$ and an identifier ID to be used to multiplex the anonymous tunnel on the virtual link from b to c later. If c accepts the anonymous tunnel, it replies with an *ADD_REP* message, which contains $\text{nonce}_{b,c}$ and c's DH public key. If there is a problem with the ADD_REP message, it replies with an *ADD_FAIL* message. Table A.5 summarises the fields and cells that are used used during steps 4–9 in Figure 5.5.

A.3.3 Peer Discovery Messages

A node can ask another node about further nodes. To do so, a establishes a virtual link to b by connecting to b's MorphMix port, i.e. b must first become a neighbour of a. Then, a sends a *NODES_REQ* message to b. The message contains a nonce and the maximum number of nodes a wants to learn about. Node b responds with a *NODES_REP* message. Since the reply is likely to be longer than the payload of a single cell, the last cell has type NODES_REP_FINAL and the others NODES_REP. A NODES_REP message contains $\text{nonce}_{a,b}$, the number of noes about which information is provided in the message (b can choose to inform a about fewer nodes than a requested), and the IP addresses, ports, RSA moduli, and node levels of these nodes. The fields and cells used for peer discovery are given in Table A.6.

Table A.5: *Fields and cells (corresponding to messages 4–9) to append a node to an anonymous tunnel.*

field	length	description
$\text{nonce}_{b,w}$	16	Nonce for b to recognise w's reply
$\text{nonce}_{b,c}$	16	Nonce for b to recognise c's reply
$\text{nonce}_{a,w}$	16	Nonce for a to recognise w's reply
ip_c	4	c's IP address
p_{mm_c}	2	c's MorphMix port
mod_c	256	c's RSA modulus
DH_a	≤ 296	a's DH public key
$\text{nonce}_{w,c}$	16	Nonce for w to recognise c's reply
ip_b	4	b's IP address
ID	2	the identifier to multiplex the anonymous tunnel on the virtual link between b and c
DH_c	variable	c's DH public key

ver-sion	type	at ID	pld len	payload	key to encrypt header	key to encrypt payload
1.0	WIT_REQ (_FINAL)	0	≤ 720	$\text{nonce}_{b,w} \parallel \text{nonce}_{b,c} \parallel \{\text{nonce}_{a,w} \parallel \text{ip}_c \parallel \text{p}_{mm_c} \parallel \text{mod}_c \parallel \{\text{DH}_a\}_{PK_c}\}_{PK_w}$	$k_{VL,bw}$	$k_{VL,bw}$
1.0	NEXT_REQ	0	≤ 382	$\text{nonce}_{w,c} \parallel \text{nonce}_{b,c} \parallel \text{ip}_b \parallel \{\text{DH}_a\}_{PK_c}$	$k_{VL,wc}$	$k_{VL,wc}$
1.0	NEXT_REP	0	16	$\text{nonce}_{w,c}$	$k_{VL,wc}$	$k_{VL,wc}$
1.0	NEXT_FAIL	0	16	$\text{nonce}_{w,c}$	$k_{VL,wc}$	$k_{VL,wc}$
1.0	WIT_REP	0	296	$\text{nonce}_{b,w} \parallel \text{nonce}_{a,w} \parallel \text{ip}_b \parallel \text{ip}_c \parallel \{\text{nonce}_{a,w} \parallel \text{ip}_b \parallel \text{ip}_c\}_{SK_w}$	$k_{VL,bw}$	$k_{VL,bw}$
1.0	WIT_FAIL	0	16	$\text{nonce}_{b,w}$	$k_{VL,bw}$	$k_{VL,bw}$
1.0	ADD_REQ	0	18	$\text{nonce}_{b,c} \parallel \text{ID}$	$k_{VL,bc}$	$k_{VL,bc}$
1.0	ADD_REP	0	≤ 314	$\text{nonce}_{b,c} \parallel \text{DH}_c$	$k_{VL,bc}$	$k_{VL,bc}$
1.0	ADD_FAIL	0	16	$\text{nonce}_{b,c}$	$k_{VL,bc}$	$k_{VL,bc}$

Table A.6: *Fields and cells to learn about other nodes.*

field	length	description
$\text{nonce}_{a,b}$	16	Nonce for a to recognise b's reply
n_{req}	1	The number of nodes a requests at most ($1 ldots 255$)
n_{rep}	1	The number of nodes in b's reply ($1 ldots 255$)
ip_i	4	i's IP address, $i = c, d, e, \ldots$
p_{mm_i}	2	i's MorphMix port, $i = c, d, e, \ldots$
mod_i	256	i's RSA modulus, $i = c, d, e, \ldots$
lev_i	1	i's node level, $i = c, d, e, \ldots$

version	type	at ID	pld len	payload	key to encrypt header	key to encrypt payload
1.0	NODES_REQ	0	17	$\text{nonce}_{a,b} \parallel \text{n}_{req}$	$\text{k}_{VL,ab}$	$\text{k}_{VL,ab}$
1.0	NODES_REP (_FINAL)	0	$17 + \text{n}_{rep} \cdot 263$	$\text{nonce}_{a,b} \parallel \text{n}_{rep} \parallel \text{ip}_c \parallel \text{p}_{mm_c} \parallel \text{mod}_c \parallel \text{lev}_c \parallel \ldots$	$\text{k}_{VL,ab}$	$\text{k}_{VL,ab}$

A.3.4 Virtual Link Status Information Messages

Neighbouring nodes can request status information from each other or decided themselves to inform the other peer about the own status. To request b's status, a sends a *STAT_REQ* message to b, to which b replies with a *STAT_REP* message. In addition, if a wants to tell b about its status, it can send a *STAT_PUSH* message. STAT_REQ and STAT_REP messages contain a nonce and all status information messages contain an info field. The fields and cells used to exchange status information messages are given in Table A.7.

A.3.5 Terminating an Anonymous Tunnel

Any node along an anonymous tunnel can terminate the tunnel at any time. Often, this is done by the initiator, but due to potential failure of nodes, it can also be done by any intermediate node or the final node. To terminate a tunnel, a *TERM* message is sent to the next node in the tunnel. The resulting cell contains the ID that identifies the tunnel to be torn down on the corresponding virtual link. If an intermediate node tears down a tunnel, it sends two TERM messages, one to the previous and one to the next node in the tunnel, unless one of these two nodes is no longer a neighbour. The recipient of a TERM message sends itself a TERM message to the next node again using the appropriate ID and so on, until the initiator or/and the final node receive such a

Table A.7: *Fields and cells to exchange status information between neighbours.*

field	length	description
$\text{nonce}_{a,b}$	16	Nonce to identify which STAT_REQ belongs to which STAT_REP message
info	1	Either the requested information, the reply, or the pushed information 1: ACCEPT_TUNNELS (in STAT_REQ messages to ask if the peer can accept further anonymous tunnels) 2: OK_TUNNELS (in STAT_REP messages to tell that further anonymous tunnels can be accepted) 3: NO_TUNNELS (in STAT_REP or STAT_PUSH messages to tell that no further anonymous tunnels can be accepted)

ver-sion	type	atID	pld len	payload	key to encrypt header	key to encrypt payload
1.0	STAT_REQ	0	17	$\text{nonce}_{a,b} \parallel$ info	$k_{VL,ab}$	$k_{VL,ab}$
1.0	STAT_REP	0	17	$\text{nonce}_{a,b} \parallel$ info	$k_{VL,ab}$	$k_{VL,ab}$
1.0	STAT_PUSH	0	17	info	$k_{VL,ab}$	$k_{VL,ab}$

message. Table A.8 gives the format of a cell containing a TERM message.

Table A.8: *Cell to terminate an anonymous tunnel.*

ver-sion	type	atID	pld len	payload	key to encrypt header	key to encrypt payload
1.0	TERM	ID	0		$k_{VL,ab}$	$k_{VL,ab}$

A.3.6 Flow Control Messages

MorphMix employs a simple flow control mechanism. The main motivation for this is that if a node gets cells much faster than it can forward them to the next node, the cells may pile up in that node and consume a significant amount of its memory. MorphMix employs a credit-based scheme very similar to the one introduced in the context of the Anonymity Network (see Section 3.1.2), which works bidirectionally between two neighbouring nodes along an anonymous tunnel. A node a gets an initial credit, which corresponds to the number of cells it is allowed to send to one specific neighbour b for a particular anonymous tunnel identified with atID ID on the virtual link between a and b. This initial credit is set to 50 cells, which means that a is

allowed to send 50 cells belonging to this anonymous tunnel to b before it must wait. Node b counts itself the cells with atID ID it has received from a and already forwarded to the next node along the tunnel (or to the client application(s) if the b is the initiator or to the host(s) if the b is the final node). Whenever b has forwarded 30 cells, it sends a *CREDIT* message back to a which sets the credit for the tunnel with atID ID back to 50. The atID of the cell containing the CREDIT message contains the ID of the tunnel for which the credit should be reset. Sending the CREDIT message already after 30 and not only after 50 cells have been forwarded should guarantee that a can reset the credit for the corresponding tunnel before all credits have been used up. If b cannot forward the cells from a anymore, it simply stops sending back CREDIT messages, which prevents a from sending cells along the tunnel identified with atID ID after its credit is used up. Note that flow control only affects LINK_DATA messages (see Appendix A.3.7) that carry the actual end-to-end messages. All other messages between neighbours can always be sent and do not affect the credit of cells a node is allowed to send. Table A.9 gives the format of a CREDIT message.

Table A.9: *Cell to reset the credit of a tunnel on a virtual link.*

version	type	atID	pld len	payload	key to encrypt	
					header	payload
1.0	CREDIT	ID	0		$k_{VL,ab}$	$k_{VL,ab}$

A.3.7 Virtual Link Data Messages

Besides all control messages, there are simple *LINK_DATA* messages. A LINK_DATA message is used to transport (parts of) an end-to-end message across a virtual link between two neighbours. Table A.10 gives the format of a cell containing a LINK_DATA message.

Table A.10: *Cell to transport end-to-end messages.*

version	type	atID	pld len	payload	key to encrypt	
					header	payload
1.0	LINK_DATA	ID	496	(parts of an) end-to-end message	$k_{VL,ab}$	$k_{LE,ab}$

A.4 End-to-End Messages

In addition to messages between neighbours, there are end-to-end messages exchanged between the endpoints of an anonymous tunnel. They are used to set up anonymous tunnels, to set up and terminate anonymous connections, to exchange end-to-end status information, and to carry data exchanged between the client application and the host. End-to-end messages are always transported between two neighbours within one or more LINK_DATA messages. The first 16 bytes of the corresponding cell payloads are used for the anonymous connection header, which looks exactly like a cell header. Since the cell header of end-to-end messages always contains the message type LINK_DATA, the atID of the corresponding tunnel, and a payload length of 496 bytes (see Appendix A.3.7), we only illustrate the cell payload (including the anonymous connection header) and not the entire cells.

A.4.1 Appending a Node to a Tunnel

We follow the messages in Figure 5.5 and describe the messages in steps 1–3 and 10. When a wants to append a node to the tunnel, it sends a *SEL_REQ* message to b. The message includes a nonce ($\text{nonce}_{a,b}$, the number of nodes b must offer in its selection to a, and the minimum level these nodes should have.

Node b replies with a *SEL_REP* message. This message usually needs several cells to be transported, which means the anonymous connection header in the last cell has type SEL_REP_FINAL, the others SEL_REP. The message includes $\text{nonce}_{a,b}$, the number of nodes in the selection, and the IP addresses, ports, public keys, and levels of these nodes in the selection. If b cannot offer a selection to a, it replies with a *SEL_FAIL* message. Note that b should try to only offer nodes that satisfy the minimum node level specified by a, but the prime goal is to offer a selection at all. So if b can offer a selection to a but has not enough neighbours that satisfy the minimum node level specified by a, the rule is that b still offers the entire selection. Node a then picks randomly a node from the selection (here we assume node c is picked). It is important that a picks the node at random, even if some nodes in the selection do not meet the minimum level. Otherwise, a malicious node could offer a selection where it includes one malicious node with a very high level and only honest nodes with low levels and hope the initiator picks the one with the highest level. Node a then generates its public key DH_a of the DH

key-exchange and encrypts it with c public key, which results in $\{\mathrm{DH}_a\}_{PK_c}$. It then picks a witness w from the set of nodes it knows (see Section 5.7) and encrypts the concatenation of a nonce ($\mathrm{nonce}_{a,w}$, c's IP address, port, public key, and $\{\mathrm{DH}_a\}_{PK_c}$ with w's public key. It generates an *APP_REQ* message for b, which results in multiple cells where the anonymous connection header in the last has type APP_REQ_FINAL and the others APP_REQ. The message contains $\mathrm{nonce}_{a,b}$, w's IP address ip_w, port p_{mm_w}, RSA modulus mod_w, ip_c, p_{mm_c}, and the encrypted data for w. After b has completed appending c to the anonymous tunnel, it sends a *APP_REP* message to a, which again results in multiple cells where the anonymous connection header in the last cell has type APP_REP_FINAL and the others APP_REP. The message contains $\mathrm{nonce}_{a,b}$, the signed receipt from w (see Section A.3.2), and c's DH public key DH_c. Node a checks the receipt if it indeed contains the correct nonce ($\mathrm{nonce}_{a,w}$) and the IP addresses of b and c. If this is not the case, the tunnel is terminated. If the receipt can be correctly verified, a uses DH_c to create the initialisation vector and the symmetric key $\mathrm{k}_{LE,ac}$ that are used as inputs to the AES cipher for the layer of encryption between a and c. If anything has failed and b could not append c, b sends an *APP_FAIL* message to a. Table A.11 summarises the fields and cell payloads used during steps 1–3 and 10 in Figure 5.5 when appending a node to a tunnel.

A.4.2 Initiating and Terminating an Anonymous Connection

Once an anonymous tunnel has been set up, the initiator can establish anonymous connections. To do so, it sends a *CON_REQ* message to c. The message contains the IP address or host name hn_h and the port p_h of the service to contact. The message can potentially result in multiple cells, which means the anonymous connection header in the last cell has type CON_REQ_FINAL and the others CON_REQ. Note that the initiator must not perform the address resolution by itself because this would reveal the identity of the host it intends to contact. The convention for c is that if hn_h has a length of four bytes and has the format of a valid IP address, it will be treated as an IP address, otherwise as a host name. Node c tries to contact h and sends back a *CON_REP* message, which contains an information field info_h. To increase end-to-end performance, there is an additional message type to establish an anonymous connection, the *CONDATA_REQ* message. Basically it works similar to a CON_REQ message but in addition contains end-to-end data to

Table A.11: *Fields and cell payloads (corresponding to messages 1–3 and 10) to append a node to an anonymous tunnel.*

field	length	description
$\text{nonce}_{a,b}$	16	Nonce for a to recognise b's reply
n_{sel}	1	The number of nodes b must offer in the selection
lev_{min}	1	The minimum level the nodes in b's selection should have
ip_i	4	i's IP address, $i = c, d, e, \ldots$
p_{mm_i}	2	i's MorphMix port, $i = c, d, e, \ldots$
mod_i	256	i's RSA modulus, $i = c, d, e, \ldots$
lev_i	1	i's node level, $i = c, d, e, \ldots$
ip_w	4	w's IP address
p_{mm_w}	2	w's MorphMix port
mod_w	256	w's RSA modulus
$\text{nonce}_{a,w}$	16	Nonce for a to recognise w's reply
ip_b	4	b's IP address
DH_a	≤ 296	a's DH public key
DH_c	≤ 296	c's DH public key

ver-sion	type	ac ID	pld len	payload	key to encrypt header	key to encrypt payload
1.0	SEL_REQ	0	18	$\text{nonce}_{a,b} \parallel \text{n}_{sel} \parallel \text{lev}_{min}$	$\text{k}_{LE,ab}$	$\text{k}_{LE,ab}$
1.0	SEL_REP (_FINAL)	0	$17 - s \cdot 263$	$\text{nonce}_{a,b} \parallel \text{n}_{sel} \parallel \text{ip}_c \parallel \text{p}_{mm_c} \parallel \text{mod}_c \parallel \text{lev}_c \parallel \ldots$	$\text{k}_{LE,ab}$	$\text{k}_{LE,ab}$
1.0	SEL_FAIL	0	16	$\text{nonce}_{a,b}$	$\text{k}_{LE,ab}$	$\text{k}_{LE,ab}$
1.0	APP_REQ (_FINAL)	0	≤ 972	$\text{nonce}_{a,b} \parallel \text{ip}_w \parallel \text{p}_{mm_w} \parallel \text{mod}_w \parallel \text{ip}_c \parallel \text{p}_{mm_c} \parallel \{\text{nonce}_{a,w} \parallel \text{ip}_c \parallel \text{p}_{mm_c} \parallel \text{mod}_c \parallel \{\text{DH}_a\}_{PK_c}\}_{PK_w}$	$\text{k}_{LE,ab}$	$\text{k}_{LE,ab}$
1.0	APP_REP (_FINAL)	0	≤ 592	$\text{nonce}_{a,b} \parallel \text{nonce}_{a,w} \parallel \text{ip}_b \parallel \text{ip}_c \parallel \{\text{nonce}_{a,w} \parallel \text{ip}_b \parallel \text{ip}_c\}_{SK_w} \parallel \text{DH}_c$	$\text{k}_{LE,ab}$	$\text{k}_{LE,ab}$
1.0	APP_FAIL	0	16	$\text{nonce}_{a,b}$	$\text{k}_{LE,ab}$	$\text{k}_{LE,ab}$

be sent to the host as soon as the connection between the final node and the host has been established. Looking at applications such as web browsing, this saves one RTT because using the the normal CON_REQ/CON_REP pair, the web request can only be sent through the anonymous tunnel to the web server after the CON_REP has arrived at the initiator. CONDATA_REQ may be longer than what fits into a single cell, which implies the anonymous con-

nection header in the last cell has type CONDATA_REQ_FINAL and the others CONDATA_REQ. Note that there is no corresponding CONDATA_REP message because the reply from the server is directly sent back to the initiator using E2E_DATA messages (see Appendix A.4.4). If connecting to the host is not possible, the final node sends back a *CLOSE_FAIL* message that contains an info field to inform the initiator about the reason why connecting was not possible. Finally, a *CON_CLOSE* message is sent along an anonymous connection if either the host has closed the connection to the final node or if the client application has closed the connection to the access program. The CON_CLOSE message informs the other endpoint of the tunnel to itself close the connection to the client application or the host. The fields and cell payloads used to initiate and terminate anonymous connections are given in Table A.12.

Table A.12: *Fields and cell payloads to initiate and terminate anonymous connections.*

field	length	description
l_{hn_h}	16	The length of the following host name or IP address of h
hn_h	variable	h's host name or IP address
p_h	2	The port to contact on h
$info_h$	2	Information about the connection attempt to h 1: NAME_RESOLUTION_FAILURE 2: DESTINATION_NOT_REACHED

version	type	ac ID	pld len	payload	key to encrypt header	key to encrypt payload
1.0	CON_REQ (_FINAL)	ID	variable	$hn_h \parallel p_h$	$k_{LE,ac}$	$k_{LE,ac}$
1.0	CON_REP	ID	0		$k_{LE,ac}$	$k_{LE,ac}$
1.0	CONDATA_REQ (_FINAL)	ID	variable	$l_{hn_h} \parallel hn_h \parallel p_h \parallel$ end-to-end data	$k_{LE,ac}$	$k_{LE,ac}$
1.0	CON_CLOSE	ID	0		$k_{LE,ac}$	$k_{LE,ac}$
1.0	CON_FAIL	ID	1	$info_h$	$k_{LE,ac}$	$k_{LE,ac}$

A.4.3 End-to-End Status Information Messages

To measure the RTT of an anonymous tunnel or to learn if the tunnel is actually still functioning, there are *E2E_PING* and *E2E_PONG* message. The initiator simply sends a nonce in a E2E_PING message through the tunnel to

the final node and remembers the time when it has sent message. Upon receiving the message, the final node immediately sends the same nonce back to the initiator using an E2E_PONG message. The fields and cell payloads used to measure the RTT of a tunnel are given in Table A.13.

Table A.13: *Fields and cell payloads to exchange status information between endpoints of a tunnel.*

field	length	description
$\text{nonce}_{a,c}$	16	Nonce for a to recognise c's reply

ver-sion	type	ac ID	pld len	payload	key to encrypt header	key to encrypt payload
1.0	E2E_PING	0	16	$\text{nonce}_{a,c}$	$k_{LE,ab}$	$k_{LE,ab}$
1.0	E2E_PONG	0	16	$\text{nonce}_{a,c}$	$k_{LE,ab}$	$k_{LE,ab}$

A.4.4 End-to-end Data Messages

To carry the actual data that are exchanged between the client application and the host, *E2E_DATA* messages are used. The anonymous connection is identified by putting the appropriate ID into the acID field of the anonymous connection header and upon receiving an E2E_DATA message, the initiator simply forwards the end-to-end data to the client application and the final node forwards the data to the host. Table A.14 gives the cell payload of E2E_DATA messages.

Table A.14: *cell payloads to transport end-to-end data.*

ver-sion	type	at	pld len	payload	key to encrypt header	key to encrypt payload
1.0	E2E_DATA	ID	≤ 480	end-to-end data	$k_{LE,ab}$	$k_{LE,ab}$

A.5 Virtual Link and Tunnel Usage

In this section, we give the details about how long virtual links and tunnels can be used once they have completely been set up. In addition, we give the policy for using virtual links.

A.5.1 Virtual Links and Tunnel Lifetimes

For the collusion detection mechanism (see Section 5.6) to work correctly, honest nodes must change their neighbours from time to time. We enforce this by specifying the maximum time a virtual link or a tunnel can be used. We first describe the life cycle of a virtual link:

1. During its lifetime, a virtual link from a to b can have four different states: *SETUP*, *READY*, *WAIT1*, and *WAIT2*. In general, a virtual link can be terminated at any time if a or b crashes, but in the normal course of events, a virtual link changes its state from SETUP to READY to WAIT1 and WAIT2. When a starts setting up a virtual link, the state of the virtual link is set to SETUP.
2. When the virtual link has been completely set up, its state changes to READY and it remains in this state for a *virtual link READY lifetime* of 30 minutes. Only while a virtual link is in state READY, a can choose b as the first intermediate node in a tunnel and b may be offered in selections from a unless b has informed a to not to do so via virtual link status information messages (see Appendix A.3.4).
3. When the virtual link READY lifetime expires, the state of the virtual link is changed to WAIT1. The virtual link remains in this state for a *virtual link WAIT1 lifetime* of 15 minutes.
4. When the virtual link WAIT1 lifetime expires, the state of the virtual link is changed to WAIT2. Once a virtual link is in state WAIT2, it is terminated as soon there are no more anonymous tunnels that use this virtual link. After a *virtual link WAIT2 lifetime* of 15 minutes, the virtual link is terminated in any case.

Similarly, there is a life cycle for anonymous tunnels:

1. During its lifetime, an anonymous tunnel can have three different states: *SETUP*, *READY*, and *WAIT*. A tunnel can be torn down at any time if any of the nodes along the tunnel leave MorphMix, but in general, a tunnel changes its state from SETUP to READY and WAIT. When the initiator starts setting up an anonymous tunnel, the state of the tunnel is set to SETUP. If the tunnel cannot be set up completely within five minutes, it is torn down.
2. When the tunnel has been completely set up, its state changes to READY and it remains in this state for a *tunnel READY lifetime* of 10 minutes. Only while a tunnel is in state READY, it may be used to set up new anonymous connections.

3. When the tunnel READY lifetime expires, the state of the tunnel is changed to WAIT. Once a tunnel is in state WAIT, it is terminated as soon as all anonymous connections using the tunnel are terminated. Although there is no tunnel WAIT lifetime, a tunnel is eventually terminated even if there are still anonymous connections using it when any of the virtual links it uses is terminated.

A.5.2 Policy for Using Virtual Links

We have already discussed in Section 5.5.3 that when offering a selection or picking a neighbour as the first intermediate node in a tunnel, nodes should only use those nodes to which they have established the virtual link themselves. Virtual links are also used during tunnel setup by the node b that is appending a new node to contact the witness w (WIT_REQ(_FINAL), WIT_REP, and WIT_FAIL message) and for the witness to contact the node c that is appended to a tunnel (NEXT_REQ, NEXT_REP, and NEXT_FAIL message). The policy for establishing and using a virtual link between b and w to append c is given below. It works in exactly the same way between w and c.

1. When b must contact w, b and w are currently neighbours, and there is a virtual link between them in state READY, this virtual link is also used for all witness messages exchanged between b and w for this particular appending of node c. It does not matter which of the two nodes b and w was the initiator of the virtual link.
2. If b and w will be neighbours in the sense that there is a virtual link in state SETUP between them (but none in state READY), b waits until this virtual link changes its state to READY and uses it to exchange messages with w. Like above, it does not matter which of the two nodes b and w was the initiator of the virtual link.
3. If b and w are currently not neighbours or if the virtual link(s) between them is (are) in state WAIT1 or WAIT2, a new virtual witness link is established. The virtual witness link is basically the same as a virtual link and is established in the same way, but the virtual witness link is only used to exchange the witness messages between b and w for this particular appending of node c and will be torn down after all witness messages between b and w have been exchanged. Note also that a virtual witness link between two nodes does not make these node neighbours.

A.6 Quantitative Analysis of the Data Overhead

We give a quantitative analysis of the data overhead produced by the MorphMix protocol. As discussed in Section 8.3.5, data overhead includes all data that are not directly related to transporting application data: tunnel setup and teardown, virtual link setup, exchange of virtual link status messages, end-to-end ping and pong messages, flow control messages, and peer discovery messages.

A.6.1 Tunnel Setup and Teardown Overhead

We assume a tunnel of length l is set up. We also assume that there are nodes in all /16 domains, which results in a selection size of 20 according to Section 5.6.2. The players when setting up a tunnel are the following: the initiator, the $1^{st} \ldots (l-2)^{th}$ intermediate node(s), the final node, and the $(l-1)$ witnesses. Besides the actual messages to append a node to a tunnel, tunnel setup also includes establishing the virtual links from and to the witnesses. For simplicity and to take into account the maximum possible data overhead, we assume that virtual links from and to the witnesses must always be established to exchange the witness messages (see Appendix A.5.2). Analysing the messages exchanged during the setup of anonymous tunnels, we get the tunnel setup overhead for the initiator as illustrated in Table A.15.

Table A.15: *Overhead for the initiator to set up a tunnel.*

message	payload length	number of messages	# cells send	# cells receive	total length (bytes) send	total length (bytes) receive
LINK_REQ	256	1	1		512	
LINK_REP	16	1		1		512
WIT_REQ	≤ 720	1	2		1024	
WIT_REP	296	1		1		512
ADD_REQ	18	1	1		512	
ADD_REP	≤ 312	1		1		512
		6	4	3	2048	1536
SEL_REQ	18	$(l-2)$	1		512	
SEL_REP	5277	$(l-2)$		11		5632
APP_REQ	≤ 972	$(l-2)$	3		1536	
APP_REP	≤ 592	$(l-2)$		2		1024
		$4 \cdot (l-2)$	4	13	2048	6656

The top six messages are needed to append the node immediately follow-

ing the initiator and each of these messages is needed exactly once. As a result, four fixed-length cells of length 512 bytes each must be sent and three cells must be received, resulting in sending 2048 and receiving 1536 bytes.

For each additional node to be appended to the tunnel, the initiator sends two and receives two messages. Since the tunnel has a total length of l nodes, this results in sending $(l-2)\cdot 4$ cells corresponding to $(l-2)\cdot 2048$ bytes and receiving $(l-2)\cdot 13$ cells corresponding to $(l-2)\cdot 6656$ bytes.

Looking at the overhead for the i^{th} intermediate node, we get the results in Table A.16.

Table A.16: *Overhead for the i^{th} intermediate node to set up a tunnel.*

message	payload length	number of messages	# cells send	# cells receive	total length (bytes) send	total length (bytes) receive
LINK_REQ	256	1	1	1	512	512
LINK_REP	16	1	1	1	512	512
NEXT_REQ	≤ 382	1		1		512
NEXT_REP	16	1	1		512	
WIT_REQ	≤ 720	1	2		1024	
WIT_REP	296	1		1		512
ADD_REQ	18	1	1	1	512	512
ADD_REP	≤ 312	1	1	1	512	512
SEL_REQ	18	1		1		512
SEL_REP	5277	1	11		5632	
APP_REQ	≤ 972	1		3		1536
APP_REP	≤ 592	1	2		1024	
		12	20	10	10240	5120
SEL_REQ	18	$(l-2-i)$	1	1	512	512
SEL_REP	5277	$(l-2-i)$	11	11	5632	5632
APP_REQ	≤ 972	$(l-2-i)$	3	3	1536	1536
APP_REP	≤ 592	$(l-2-i)$	2	2	1024	1024
		$4\cdot(l-2-i)$	17	17	8704	8704

The top twelve messages are needed to append the intermediate node itself and the following node, which results in sending 20 cells (10240 bytes) and receiving 10 cells (5120 bytes). Depending on the position of an intermediate node, it has to relay more or fewer messages when the following nodes are appended. With a tunnel length of l nodes and for the i^{th} intermediate node for $1 \leq i \leq (l-2)$, this results in relaying $l-2-i$ times 17 cells, resulting in sending and receiving $(l-2-i)\cdot 17$ cells ($(l-2-i)\cdot 8704$ bytes).

The final node is a special case of an intermediate node because it is only

appended to a tunnel but does not append additional nodes itself. Table A.17 lists its overhead during the setup of a tunnel, which results in sending and receiving three cells (1536 bytes).

Table A.17: *Overhead for the final node to set up a tunnel.*

message	payload length	number of messages	# cells send	# cells receive	total length (bytes) send	total length (bytes) receive
LINK_REQ	256	1		1		512
LINK_REP	16	1	1		512	
NEXT_REQ	≤ 382	1		1		512
NEXT_REP	16	1	1		512	
ADD_REQ	18	1		1		512
ADD_REP	≤ 312	1	1		512	
		6	3	3	1536	1536

There is also overhead for the witnesses. A witness must send four cells (2048 bytes) and receive five cells (2560 bytes) when a node is appended to the tunnel. Table A.18 shows which messages are responsible for how much overhead.

Table A.18: *Overhead for a witness to set up a tunnel.*

message	payload length	number of messages	# cells send	# cells receive	total length (bytes) send	total length (bytes) receive
LINK_REQ	256	1	1	1	512	512
LINK_REP	16	1	1	1	512	512
WIT_REQ	≤ 720	1		2		1024
WIT_REP	296	1	1		512	
NEXT_REQ	≤ 382	1	1		512	
NEXT_REP	16	1		1		512
		6	4	5	2048	2560

Finally, there is some overhead when tunnels are torn down. To do so, the initiator sends a TERM message consisting of one cell (512 bytes) to the first intermediate node, from where the tunnel is torn down hop by hop. Therefore, the initiator sends one message, each intermediate node sends and receives one message, and the final node receives one message.

Assuming a tunnel length of $l = 5$, the overhead for each node involved in setting up and tearing down an anonymous tunnel is summarised in Table A.19

Using the results from Table A.19 and assuming there are n nodes in

Table A.19: *Overhead summary to set up and tear down a tunnel with length five.*

node	# cells		total length (bytes)	
	send	receive	send	receive
initiator	17	42	8704	21504
1^{st} witness	4	5	2048	2560
2^{nd} witness	4	5	2048	2560
3^{rd} witness	4	5	2048	2560
4^{th} witness	4	5	2048	2560
1^{st} int. node	55	45	28160	23552
2^{nd} int. node	38	28	19456	14848
3^{rd} int. node	21	11	10752	6144
final node	3	4	1536	2536
all nodes	150	150	76800	76800

MorphMix, the average tunnel length is five, every node sets up a tunnel every T_{ts} seconds and every node is equally likely to be selected as a witness, an intermediate node, or a final node, then the average load (sending and receiving) on every node is

$$load_{ts} = \frac{n}{T_{ts}} \cdot \frac{76800}{n} = \frac{76800}{T_{ts}} [B/s] . \quad \text{(A.1)}$$

With $T_{ts} = 120$, this results in an average tunnel setup overhead of sending and receiving of about 640 B/s. While this is less than 0.5% of what a T1 node can handle, it has an impact on slow nodes. Looking at an ISDN node connected to the Internet with a bandwidth of 64 Kb/s, the tunnel setup overhead accounts for approximately 8%.

A.6.2 Virtual Link Setup Overhead

If a node contacts and sets up a virtual link every T_{le} seconds and assuming that the probability to be contacted is the same for every node, this results in an average load of

$$load_{le} = \frac{1}{T_{le}} \cdot 1024 [B/s] . \quad \text{(A.2)}$$

Assuming a node contacts another node once per minute, i.e. $T_{le} = 60$, the resulting overhead is about 17 B/s.

A.6.3 Virtual Link Status Information Overhead

At any time, every node has several neighbours that can be used as potential next hops when appending a node to a tunnel. Assuming a node has established a virtual link with l_n neighbours on average and assuming that a node exchanges one STAT_REQ/STAT_REP pair with each of its neighbours every T_{ls} seconds, the overhead per node is

$$load_{ls} = \frac{l_n}{T_{ls}} \cdot 1024 = [B/s] \quad \text{(A.3)}$$

Assuming that every node has $l_n = 30$ neighbours on average and that status information is exchanged every two minutes, the overhead is about 256 B/s.

A.6.4 End-to-End Status Information Overhead

There is additional overhead from status information messages to test the quality of a tunnel with E2E_PING and E2E_PONG messages. With n nodes, an average tunnel length of l, every node has established t anonymous tunnels on average, and a tunnel is tested with an E2E_PING/E2E_PONG pair every $T_{t_{stat}}$ seconds, the average overhead per node is

$$load_{t_{stat}} = \frac{n \cdot t}{T_{t_{stat}}} \cdot \frac{2 \cdot (l-1) \cdot 512}{n} = \frac{t \cdot (l-1) \cdot 1024}{T_{t_{stat}}} [B/s] \quad \text{(A.4)}$$

With $t = 5$ tunnels established per node at any time, an average tunnel length of $l = 5$, and testing a tunnel every two minutes on average, the overhead is about 171 B/s.

A.6.5 Other Protocol Overhead

There are other overheads. Initial peer discovery produces NODES_REQ and NODES_REP messages, but since they are usually only needed when joining MorphMix for the first time or when joining again after having been offline for a while, their impact is negligible. In addition, a few CREDIT messages must be sent within the LINK_DATA messages stream between two nodes,

but the number of these messages is dependent on the number of LINK_DATA messages and can therefore not be expressed in the same way as we did for the other overheads above.

A.6.6 Protocol Overhead Summary

The overheads produced by the MorphMix protocol are quite significant. Summing the overheads calculated above results in an average overhead of sending and receiving about 1090 B/s. Taking into account additional overheads from peer discovery and flow control messages, the effective overhead is even slightly higher. This is quite a burden for ISDN nodes and even for slower ADSL or Cable connections. However, recalling that slower nodes will simply refuse accepting many anonymous tunnels as discussed in Section 7.3.2 implies that these nodes must handle much less than the average overhead. On the other hand, fast nodes can easily handle more than the average overhead without significantly compromising their bandwidth available for real data. Note that the overhead is analysed in more detail in Section 8.3.5.

A.7 MorphMix Prototype Implementation

We have implemented a MorphMix prototype as a proof of concept. The prototype implements the entire MorphMix protocol including collusion detection and peer discovery mechanisms and represents a fully functioning MorphMix node. The prototype includes support for HTTP (versions 1.0 and 1.1) and HTTPS, and can easily be extended to support other protocols as well. While the implementation did not deliver new fundamental results, it served us well to correctly specify the details of the MorphMix protocol.

The MorphMix prototype is free software and is available with full source code under the terms of the GNU General Public License[1] at the MorphMix project web page[2]. The MorphMix project web page will in general inform about further developments of MorphMix, and also contains information about how to configure and run the prototype. In addition, the source code itself contains many comments to facilitate further development of the prototype. The prototype is implemented in Java 1.4. Since Java 1.4 includes

[1] http://www.gnu.org/copyleft/gpl.html
[2] http://www.tik.ee.ethz.ch/~morphmix

the Java Cryptography Extension (JCE)[3] framework but no implementations of the RSA and AES ciphers, we use version 1.21 of the free Bouncy Castle[4] crypto package, which provides a complete implementation of JCE. This crypto package is not available at our MorphMix web page and the latest release[5] must be downloaded before the MorphMix prototype can be run. The Bouncy Castle web page also gives detailed information about how to install the crypto package. Of course, it is possible to use any other provider of a JCE compatible implementation of the necessary cryptographic algorithm.

The MorphMix prototype mainly serves experimental purposes to test the protocol and to analyse the effect of varying different parameters. In particular, it has not been optimised with respect to performance. Nevertheless, our experiences with the prototype have shown that in its current state, the performance it offers on a state-of-the-art computer and its stability are good enough such that the prototype could certainly be used as a basis for a limited user trial. However, the prototype definitely should be tested and fine-tuned more thoroughly before a wider public release is attempted.

MorphMix is designed to operate in an environment with many nodes distributed across a wide variety of different /16 subnets. However, such an environment is usually not available when testing or experimenting with the software. Consequently, we have added functionality to the MorphMix prototype that allows running either several nodes using different MorphMix ports on a single computer, or several nodes on different computers within the same /16 subnet. Whether a MorphMix node should run in either one of these test modes or in the real mode with nodes in many different /16 subnets can be specified with the appropriate command line arguments. In addition, the MorphMix prototype makes use of a properties file to specify a variety of parameters, which allows easily changing these parameters without having to modify the code.

[3] http://java.sun.com/products/jce
[4] http://www.bouncycastle.org
[5] http://www.bouncycastle.org/latest_releases.html