Eine Analyse des Tor-Protokolls

Kai H. Michaelis

Seminar-Arbeit

am

Lehrstuhl für Netz- und Datensicherheit
Prof. Dr. Jörg Schwenk

betreut durch Juraj Somorovsky

14.3.2012

Horst-Görtz Institut Ruhr-Universität Bochum

Inhaltsverzeichnis

1 Einführung 1

2 Vorgeschichte des Onion Routing 2
- 2.1 Proxys und Proxychains . . . 2
- 2.2 Anonyme E-Mail und ISDN-Mixes . . . 3

3 Tor 8
- 3.1 Onion Routing . . . 8
- 3.2 2nd Generation . . . 10
  - 3.2.1 Kommunikation . . . 14
  - 3.2.2 Hidden Services . . . 14
- 3.3 Verringerung der Latenz . . . 17

4 Alternative Techniken 19
- 4.1 Freenet . . . 19
- 4.2 Crowds . . . 21
  - 4.2.1 Multicast . . . 23

5 Angriffe 24
- 5.1 Verkehrsanalyse . . . 24
- 5.2 Timing Attacks . . . 27
- 5.3 Anwendungsprotokolle . . . 28
- 5.4 Tor-spezifische Angriffe . . . 29
- 5.5 Juristisches . . . 29

6 Fazit 32

Abbildungsverzeichnis

2.1 Proxychain per SOCKS . . . 3
2.2 Formale Beschreibung der Chaum-Mixes . . . 3
2.3 Funktionsweise einer Chaum Mix-Kaskade . . . 4
2.4 Formale Beschreibung von Rücksendeadressen . . . 5
2.5 Aufbau der beiden Hälften einer einfachen Verbindung zwischen $\mathcal{A}$ und $\mathcal{B}$ durch M_m. . . . 5

3.1 Onion von b nach y über x . . . 9
3.2 Einfacher virtueller Kanal über mehrere Onion Router . . . 9
3.3 Virtueller Kanal mit Loose routing . . . 9
3.4 Rückkanal über eine Reply Onion . . . 10
3.5 Aufbau einer Cell abhängig vom Kommando. . . . 11
3.6 Konstruktion eines 2-Hop-Circuit. . . . 12
3.7 Beispiel zur Congestion control via Windows. . . . 13
3.8 Aufbau und Abbau von Streams. . . . 14
3.9 Tors Leaky Pipe Architektur. . . . 14
3.10 Etablierung eines Location Hidden Service in Tor. . . . 15
3.11 Konstruktion einer Verbindung zu einem Hidden Service. . . . 16
3.12 Aufbau eines Circuits mit Preshared DH und dem symmetrischen Schlüssel $K_O = g^{\alpha\beta}$. . . . 16
3.13 Etablierung eines Kanals mit einem Hidden Service über Valet Nodes (Neuerungen hervorgehoben). . . . 17

4.1 Suche nach einem Datensatz im Freenet. . . . 20
4.2 Funktionsweise von Crowds. . . . 21

5.1 Learning Phase einer Disclosure Attack. . . . 25
5.2 Excluding Phase einer Disclosure Attack. . . . 26
5.3 Konstruktion des Vektors aller Kommunikationspartner von $\mathcal{A}$ mit t Beobachtungen. . . . 26
5.4 Endpunkt von der Nachricht von $\mathcal{A}$ in Runde i. . . . 27
5.5 Das auf die Datenrate aufgeprägte Muster kann in Form der Latenz wiedererkannt werden. . . . 28
5.6 Hintertür in der JAP Mix. . . . 31

1 Einführung

Sicherheit in öffentlichen Computernetzen nimmt schon seit Jahrzehnten einen großen Platz in Industrie und Forschung ein. Spätestens seit der Entdeckung des World Wide Web als Marktplatz hat der Bedarf an „sicherer“ Kommunikation erstaunliche, vor allem kommerzielle, Energien freigesetzt. Entwicklungen wie Transport Layer Security und die XML-Security Standards ermöglichen es heute Informationen sicher und über mehrere Plattformen hinweg auszutauschen. Fragen von der formalen Sicherheit eines Systems bis zur hochperformanten Implementierung der Modelle sind heute etablierte Forschungszweige.

Dieser kommerziellen Nutzung des Internets folgt nun seit etlichen Jahren immer sichtbarer die Etablierung als öffentlicher Raum. Der Austausch über politische und gesellschaftliche Themen hat im Jahre 2012 eine Dimension erreicht, die nicht mehr mit der einzelner Usenet-Groups und IRC Chatrooms von vor 10 Jahren vergleichbar ist.

Diese Entwicklung bringt den Fokus auf ein Sicherheitsziel, welches sich nicht mit den vorhandenen wie TLS lösen lässt: Anonymität. So problematisch diese für den Kommerz, so elementar ist sie für einen offenen Austausch über Politik und Gesellschaft. Um diesen „Technischen Datenschutz“ soll es in der folgenden Arbeit gehen.

In den ersten Kapiteln wird ein Überblick der vorhandenen Lösung gegeben. Eine Beschreibung der historischen Entwicklung erleichtert das Verständnis für die besonderen Probleme anonymer Kommunikation in öffentlichen Netzen. Der Schwerpunkt wird auf dem zweiten Kapitel und Tor liegen, eines der größten Netze zur anonymen Kommunikation zurzeit. Es wird das Protokoll an sich, als auch vorgeschlagene Modifikationen erläutert.

Nach Tor wird im dritten Kapitel ein kurzer Exkurs zu alternativen Konzepten auf dem Plan stehen, worauf dann im vierten Kapitel ein Überblick der existierenden Angriffe auf die zugrunde liegende Technik, sowie im speziellen, Angriffe auf Elemente von Tor gegeben wird.

Tor selbst ist ein Netz welches TCP-Ströme über mehrere Server (sogenannte Onion Router) leitet um die Quelle und (für Hidden Services) das Ziel eins Datenstroms zu verschleiern. Verschlüsselung der Nutzlast verhindert, dass passive Angreifer, sowie die Onion Router selbst den Inhalt des TCP-Stroms erfahren können. Darüber hinaus bietet Tor noch responder anonymity in Form von Hidden Services. Teilnehmer des Netzwerks können Dienste bereitstellen, für die einzelne Onion Router als Gateway fungieren. Damit kann auch der Anbieter eines Dienstes, zusätzlich zum Client, anonym bleiben.

2 Vorgeschichte des Onion Routing

Das Versenden von Nachrichten in einem paketvermittelten Netz mit gefälschter Quelladresse wäre wohl die simpelste Möglichkeit anonym zu kommunizieren. Da praktisch alle heute eingesetzten Protokolle in ISO/OSI Ebene 7 verbindungsorientierte Transportprotokolle und damit TCP verwenden, scheitert diese Lösung schon beim Versuch einen 3-Wege-Handshake aufzubauen.

2.1 Proxys und Proxychains

Ein nächster naiver Ansatz, die Absender in paketvermittelten Netzen zu verschleiern ist es, Nachrichten über eine oder mehrere Zwischenstellen zu übertragen. Diese, nach wie vor beliebte, Technik verwendet die in einigen Anwendungsprotokollen [1] enthaltene Fähigkeit, Pakete von einer sogenannten Proxy an das Ziel zu senden. Selbst Anwendungsprotokolle ohne Proxy im Pflichtenheft können mit Protokollen wie SOCKS[15] über mehrere Computer umgeleitet werden. Obwohl damit der Initiator der Verbindung, aus der Sicht des Empfängers, der Proxyserver ist, hat diese Technik gravierende Nachteile. Der Transport der Daten über einen einzelnen Server ermöglicht diesem den gesamten (im Zweifelsfall unverschlüsselten) Verkehr abzuhören. Vor allem wenn SOCKS Server anhand niedriger Latenz ausgewählt werden, ist es wahrscheinlich, dass das Ziel (und damit der „Gegner“) und Proxy im gleichen Autonomen System liegen.

Das SOCKS-Protokoll, welches ursprünglich entwickelt wurde, um beliebige Protokolle durch Firewalls zu zwingen, kann genutzt werden, um ganze Ketten von Proxys zu erstellen. SOCKS Server akzeptieren typischerweise an Port 1080 Anfragen. Nach einer optionalen Authentifikation der Clients in SOCKS5 müssen nur eine Zieladresse und ein Zielport gesendet werden. Der Server baut dann eine Verbindung auf und leitet alle nachfolgenden Daten darüber weiter. Durch diesen Kanal kann dann wieder eine SOCKS-Verbindung aufgebaut werden (siehe Abb. 2.1). Dieses Proxy Chaining erhöht den Aufwand, den Client vom Server aus zurückzuverfolgen, was im eher feudal organisierten Internet schon bei einzelnen Rechnern schwierig ist.

Wenn keine Verschlüsselung eingesetzt wird, ist aber auch diese Technik sehr unsicher. Nun kann jeder SOCKS-Server in der Kette alle Nachrichten vom Client zum Server abhören. Der maximale Durchsatz fällt auf das Minimum der in der Kette beteiligten Proxys, die Laufzeit steigt auf die Summe. Selbst wenn alle Pakete zwischen den Proxys verschlüsselt werden, könnte ein passiver Angreifer, der keinen Zugriff auf die Rechner hat, durch simple Beobachtung des Weges den die Pakete durch das Netz nehmen alle beteiligten Computer identifizieren. Trotz dieser offensichtlichen Probleme sind einzelne

[1] Am beliebtesten in der Praxis zweifelsohne die Proxy-Fähigkeit von HTTP per Host Header

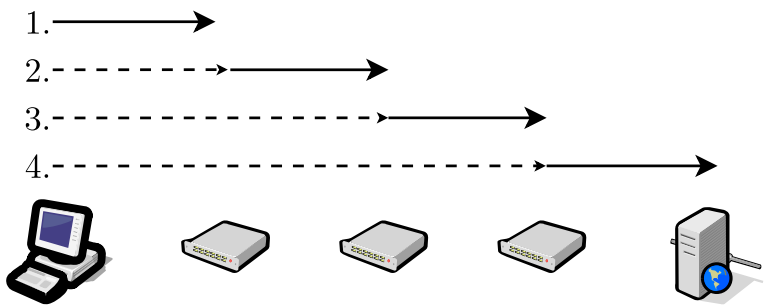

Abbildung 2.1: Proxychain per SOCKS

Proxys und Proxychains so beliebt, dass es unzählige Webseiten mit (gewollt oder ungewollt) öffentlich zugänglichen Proxys gibt, sowie Programme[2] um SOCKS Server zu einer Kette zusammenzufassen.

2.2 Anonyme E-Mail und ISDN-Mixes

Die Probleme des Proxy-Ansatzes behebt die Einführung einer neuen Art von Verteiler: Die Mix (siehe Abb. 2.2). Die erste Beschreibung dieser Mix von Chaum[3] war im Kontext von anonymem E-Mail-Verkehr. Hierbei wird jede Nachricht, zusammen mit einem zufälligen Wert, unter dem öffentlichen Schlüssel des Empfängers verschlüsselt. Diese chiffrierte Nachricht wird ihrerseits mit einer weiteren Nonce und der Zieladresse unter dem öffentlichen Schlüssel einer Mix verschlüsselt. Das gesamte Paket wird dann an diese Mix gesendet. Dort wird die äußere Verschlüsselung wieder entfernt und die Nachricht zusammen mit anderen in einen Stapel sortiert. Von diesem werden dann zufällige Nachrichten an die jeweiligen Empfänger gesendet.

$$
\begin{array}{ll}
N & \text{Nachricht} \\
N = K_i^{-1}(K_i(N)) & \text{PK-Kryptosystem} \\
R_i \neq R_j \quad \forall i \neq j & \text{Nonce} \\
M_i & \text{Mix } i
\end{array}
$$

$$
K_1(R_1, K_B(R_0, N)) \xrightarrow{M_1} K_B(R_0, N) \quad \text{Einzelne Mix}
$$

$$
\begin{array}{lll}
K_n(R_n, K_{n-1}(\cdots)) \xrightarrow{M_n} K_{n-1}(R_{n-1}, \cdots) & & \text{Erste Mix in Kaskade} \\
\xrightarrow{M_{n-1}\cdots M_1} K_B(R_B, N) & & \text{Nach der } n-1\text{-ten Mix}
\end{array}
$$

Abbildung 2.2: Formale Beschreibung der Chaum-Mixes

Da die einzelne Mix immer erst einen Stapel an Nachrichten auflaufen lässt und dann

[2] http://proxychains.sourceforge.net/

in einer anderen Reihenfolge weiterleitet, kann ein Außenstehender nicht durch simple Beobachtung des Verkehrs Nachrichten nachverfolgen. Einzelne Mixes werden in Ketten, ähnlich zu Proxychains, angeordnet (sogenannten Kaskaden, Abb. 2.3). Der Absender legt diese fest, indem er seine Nachricht in bestimmter Reihenfolge mit immer neuen Nonces unter dem öffentlichen Schlüssel einer Mix verschlüsselt.

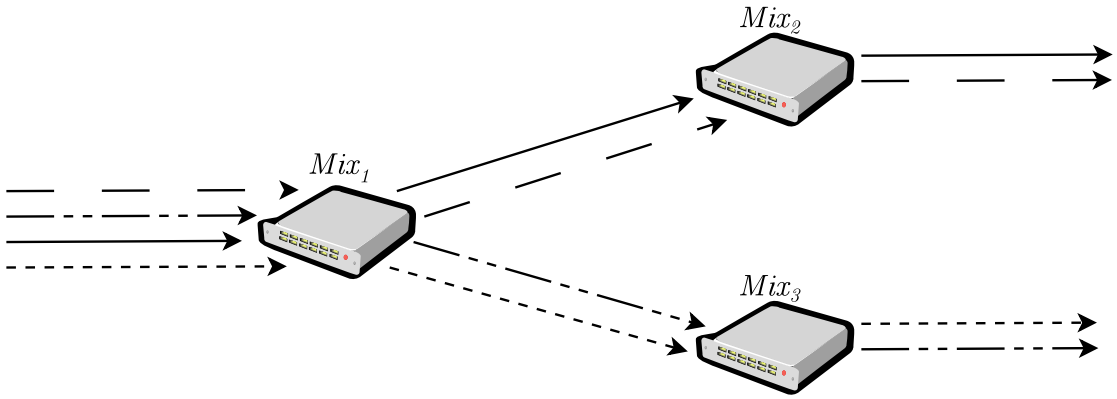

Abbildung 2.3: Funktionsweise einer Chaum Mix-Kaskade

Um Verkehrsanalyse durch doppelte Nachrichten zu vermeiden, müssen Mixes in der Vergangenheit empfangene Nachrichten aus ihrem Stapel löschen, bevor dieser abgearbeitet wird.

Dieses System ermöglicht auch das Beantworten von Nachrichten, ohne dass der Antwortende die Adresse des ursprünglichen Senders kennen muss (siehe Abb. 2.4). Hierbei wird die Nachricht mit einem, für diese Situation generierten öffentlichen Schlüssel, verschlüsselt. Die Absenderadresse ist wiederum in einer verschachtelten Verschlüsselung unter den öffentlichen Schlüsseln der beteiligten Mixes verschlüsselt. Auf dem Weg durch die Kaskade wird immer eine Schicht um die Absenderadresse entfernt und die Nonce, welche vorher nur zum Schutz gegen Verkehrsanalyse eingefügt wurde, als Schlüssel genutzt um die Nachricht mit einem symmetrischen Algorithmus zu verschlüsseln. Damit wird in jeder Mix eine Hülle um die Absenderadresse entfernt und eine um die Nachricht hinzugefügt. Die letzte Mix kann die Adresse extrahieren und die Nachricht an den Empfänger weiterleiten. Da dieser die Nonces erzeugt hat, kann er nun die Verschlüsselungen um die Nachricht sukzessive entfernen.

Chaums Mix-Kaskaden sind ausdrücklich für E-Mail-Anwendungen gedacht und damit nicht direkt auf Protokolle übertragbar, die in Echtzeit arbeiten. Sie bilden dennoch die Grundlage für alle heute verwendeten Onion Routing Strategien. Erst zehn Jahre später wurde Chaums Schema auf Echtzeitkommunikation übertragen: ISDN Mixes[21].

ISDN, Integrated Services Digital Network, ist ein (für unsere Betrachtungen) leitungsvermitteltes, digitales Netzwerk. Jeder Teilnehmer hat zwei Datenleitungen mit je 64 kbit/s. Zusätzlich besitzt jeder Anschluss noch einen Kanal für Signalisierung (Out-of-band signaling). Anstatt nun Sprachsignale in Pakete zu zerlegen und via Mix-Kaskaden zu versenden, werden Leitungen zwischen Mixes aufgebaut und nur Steuer-

$$\begin{array}{ll} A_i & \text{Absenderadresse} \\ R_i(X) & \text{Symmetrisches Kryptosystem} \end{array}$$

$$K_1(R_1, A_A), K_A(R_0, N)) \xrightarrow{M_1} A_A, R_1(K_B(R_0, N)) \quad \text{Einzelne Mix}$$

$$\begin{array}{lll} K_n(R_n, K_{n-1}(\cdots)), K_A(R_0, N) & \xrightarrow{M_n} & K_{n-1}(R_{n-1}, \cdots), R_n(K_A(\cdots)) \\ & \xrightarrow{M_{n-1}\cdots M_1} & A_A, R_1(R_2(\cdots)) \end{array}$$

Abbildung 2.4: Formale Beschreibung von Rücksendeadressen

signale, nach dem aus Chaum[3] bekannten Schema, asynchron übertragen. Ein ISDN-Netzwerk besteht aus dem Ortsvermittlungsnetz, in welchem sich die Mixes befinden und den (trans)nationalen Verbindungen dazwischen.

Ein einfacher Kanal zwischen $\mathcal{A}$ und $\mathcal{B}$ besteht aus zwei Hälften (siehe Abb. 2.5). Dem sendenden von $\mathcal{A}$ zu einer Mix M_m und dem empfangenden Kanal von $\mathcal{B}$ zu M_m. Hierbei sendet $\mathcal{A}$ eine SendEstab Nachricht zu M_m. Diese wird behandelt wie die Nachrichten in Chaum-Mixes: Verschlüsselt mit dem öffentlichen Schlüssel der Mix, inklusive einer Nonce und einem privaten, symmetrischen Schlüssel k_i. Zusätzlich zur Nachricht im OOB-Kanal wird eine Leitung von $\mathcal{A}$ zur Mix alloziert. Analog wird der empfangende Kanal von $\mathcal{B}$ zur Mix durch eine RecEstab Nachricht etabliert, welcher den Schlüssel k_j verwendet. Daten, die von $\mathcal{A}$ zur Mix übertragen werden, werden dort entschlüsselt, Daten von der Mix zu $\mathcal{B}$ verschlüsselt. Dieses Schema kann leicht auf Mix-Kaskaden erweitert werden, indem die SendEstab Nachricht sukzessive mit neuen symmetrischen Schlüsseln unter dem öffentlichen Schlüssel der n-ten Mix auf dem Pfad von $\mathcal{A}$ zu M_m verschlüsselt wird.

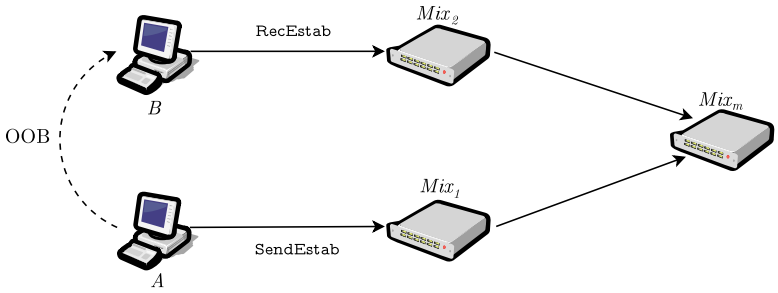

Abbildung 2.5: Aufbau der beiden Hälften einer einfachen Verbindung zwischen $\mathcal{A}$ und $\mathcal{B}$ durch M_m.

Zusätzlich zum symmetrischen Schlüssel enthält die innerste Nachricht noch einen Wert l_i. Dieses Label ist gleich für zwei korrespondierende SendEstab und RecEstab Nachrichten und ermöglicht der Mix M_m die beiden Hälften des Kanals zusammenzufügen. Damit ist ein Kanal etabliert, der folgende Eigenschaften erfüllt.

- $\mathcal{A}$ ist anonym gegenüber $\mathcal{B}$, da $\mathcal{B}$ nur den Weg bis M_m kennt.
- Genauso ist $\mathcal{B}$ anonym gegenüber $\mathcal{A}$, da auch $\mathcal{A}$ nur die Kaskade zu M_m verfolgen kann.
- Ein passiver Angreifer kann, solange er keinen Zugriff auf alle Mixes hat, nicht die Leitungen eines Kanals von denen der anderen unterscheiden.
- Außer M_m kann keine Mix den Verkehr im Klartext lesen.

Aus zwei dieser einfachen Kanäle lässt sich ohne Änderung am Modell einer für die Kommunikation in beide Richtungen aufbauen.

Einige Probleme sind noch zu lösen, die die Kontaktierung zwischen $\mathcal{A}$ und $\mathcal{B}$ betreffen, sowie Verkehrsanalyse durch ungenutzte Leitungen und Latenz während des Verbindungsaufbaus.

Um $\mathcal{B}$ vom Kommunikationswunsch von $\mathcal{A}$ zu informieren, muss eine anonyme incoming call Nachricht an $\mathcal{B}$ geschickt werden. Bei ISDN-Mixes befinden sich die Mixes nur im jeweiligen Ortsvermittlungsnetz. $\mathcal{A}$ würde nun eine incoming call Nachricht durch die Mix-Kaskade innerhalb ihres Vermittlungsnetz L_a zu M_m und von da aus zu L_b senden. Das Ortsvermittlungsnetz von $\mathcal{B}$, L_b überträgt die Nachricht dann per Broadcast an alle Teilnehmer (und damit auch zu $\mathcal{B}$).

Zwar können die einmal etablierten Kanäle für Echtzeitanwendungen verwendet werden, dennoch ist der Aufbau dieser mit der beschriebenen naiven Methode langsam. Eine Mix wird die incoming call Nachricht erst weiterleiten, wenn genügend andere aufgelaufen sind. Genauso kann eine Verbindung erst abgebaut werden, wenn genug andere auf ihre Beendigung warten. Bei ISDN mit seinen nur zwei Leitungen pro Anschluss würde das schnell weitere Anrufe eines Teilnehmers blockieren. Die Lösung besteht darin, das gesamte Netzwerk in Zeitschlitze zu unterteilen. Eine anonyme Verbindung ist nur einen Zeitschlitz lang aktiv und muss danach erneuert oder abgebaut werden. Damit werden am Ende eines jeden Zeitschlitzes genug Pakete an Mixes auflaufen, um neue Verbindungen aufzubauen oder abzubauen.

Eine letzte Schwachstelle sind ungenutzte Leitungen. Zwar ist es nicht möglich durch simple Beobachtung zwei Datenströme zu unterscheiden, allerdings können Teilnehmer ohne Verbindungen zu einer Mix von vornherein als $\mathcal{A}$ oder $\mathcal{B}$ ausgeschlossen werden. Da sich Mixes nur im Ortsvermittlungsnetz befinden, also auch anonyme Verbindungen sich bis dahin verfolgen lassen, könnte das in der Praxis die Sicherheit des Systems gefährden. Als Gegenmaßnahme baut jeder Teilnehmer mit seinen ungenutzten Leitungen eine Verbindung über eine Mix-Kaskade zu sich selbst auf und erneuert diese entweder in jedem Zeitschlitz oder nutzt sie um andere Teilnehmer zu kontaktieren.

Da eine Ortsvermittlungsstelle typischerweise > 5000 Teilnehmer hat ist es praktisch unmöglich, Einzelne aus dieser Menge über nicht-technische Methoden zu demaskieren. Allerdings bleiben einige denkbare Angriffe.

- Ein aktiver Angreifer könnte die incoming call Nachricht abfangen, bevor sie als Broadcast versendet wird und sie einzeln an die Teilnehmer versenden. Wenn der

Anschluss im nächsten Zeitschlitz reagiert, handelt es sich um den gesuchten Empfänger.

- Das ISDN-Mix Schema bietet keinen Schutz gegen Denial of Service-Angriffe. Ein Teilnehmer kann einfach die beiden verfügbaren Leitungen eines Anschlusses allozieren und diesen an der Kommunikation hindern.
- Sollte der Angreifer mit $\mathcal{B}$ kooperieren, könnte er den Verkehr in der Leitung von $\mathcal{A}$ stören. Wenn die Störung bei $\mathcal{B}$ ankommt, ist $\mathcal{A}$ sein Kommunikationspartner.

Auch wenn diese Art von Echtzeit Mixes auf das Kommunikationsmodell von ISDN, inklusive seiner technischen Schwachstellen (nur zwei 64 kbit/s Leitungen) zugeschnitten ist, kann es als Vorläufer aller Onion Routing Protokolle für paketbasierende Netze angesehen werde. Das schließt natürlich Tor ein.

3 Tor

Die Anfänge von Tor liegen beim US-amerikanischen Naval Research Laboratory, wo es unter anderem von einem der ursprünglichen Erfinder des (IP-basierenden) Onion Routing[11] Paul Syverson entwickelt wurde. Inzwischen wird Tor aber von einer eigenen gemeinnützigen Organisation, The Tor Project als Open-Source weiterentwickelt.

Tor[7] (The Onion Routing) ist ein 2nd Generation Anonymisierungsnetzwerk. Es unterscheidet sich von anderen Netzen wie Crowds oder Anonymizer vor allem dadurch, dass es einfach zu Benutzen und zu Betreiben ist, sich ausschließlich auf die Netzwerkebene konzentriert und bewusst bestimmte Gegenmaßnahmen nicht implementiert, um Protokoll und Anwendung überschaubar zu halten. Tor benötigt keine Modifikationen am Betriebssystem, sondern läuft als normaler Benutzerprozess ohne spezielle Rechte. Es anonymisiert einzelne TCP-Verbindungen, welche über eine integrierte SOCKS Proxy entgegengenommen werden, damit kann es mit fast jeder Anwendung kooperieren. Hidden Services, also Server, die auch gegenüber ihrer Clients anonym sind, sind genauso Teil von Tor wie die Möglichkeit einer einzelnen Mix zu bestimmen, für welche Ziele im nicht-anonymen Internet sie Datenströme weiterleiten will (sogenannte Exit-Policies). Obwohl Tor ausdrücklich als Testumgebung für Experimente im Onion Routing entwickelt wurde, ist es durch diese benutzerfreundlichen Designentscheidungen eines der beliebtesten Mittel zur anonymen Kommunikation im Internet. Seit seiner Veröffentlichung 2003 hat Tor etwa 900 Mixes und mehr als 200000 Nutzer pro Woche [27].

3.1 Onion Routing

Die ursprüngliche Beschreibung von Onion Routing als Implementierung von Chaum Mixes in einem TCP/IP Netzwerk wurde von Goldschlag, Reed und Syverson 1996 veröffentlicht[11]. Onions bieten die Anonymität des originären Konzepts der Mixes, erlauben aber Kommunikation in Echtzeit ohne die, in paketbasierenden Netzen verschwenderische, konstante Allozierung von Kanälen wie bei ISDN-Mixes.

Onion Router erstellen einen virtuellen Kanal durch mehrere andere Knoten, von denen jeder eine „Hülle“ des mehrfach verschlüsselten Pakets entfernt, der sogenannten Onion. Jede dieser Verschlüsselungen mit dem öffentlichen Schlüssel des empfangenden Onion Routers enthält zwei symmetrische Schlüssel, mit denen die Nutzlast zum nächsten/vorherigen Knoten verschlüsselt ist, sowie eine Gültigkeitsdauer um Replay zu verhindern.

Zusätzlich zu den Schlüsseln K_{fx}, K_{bx} wird noch der zu verwendende Algorithmus F_{fx}, FK_{bx} übertragen. Das gesamte Paket wird auf eine konstante Größe erweitert, um durch die sonst monoton fallende Länge verfolgbar zu sein. Das angeheftete Padding entspricht der Länge des entfernten Headers ($\text{exp_time}_x, Y, F_{fx}, K_{fx}, F_{bx}, K_{bx}$). Mit dieser

$$\mathcal{O}_{b,x,y} = \{\text{exp_time}_x, Y, F_{fx}, K_{fx}, F_{bx}, K_{bx}, \text{PK}_y(\mathcal{O}_{x,y,z}), \text{Pad}\}$$

Abbildung 3.1: Onion von b nach y über x

Technik (siehe Abb. 3.2) kann dann eine Mix-Kaskade aufgebaut werden, wie sie schon bei Chaum beschrieben wurde.

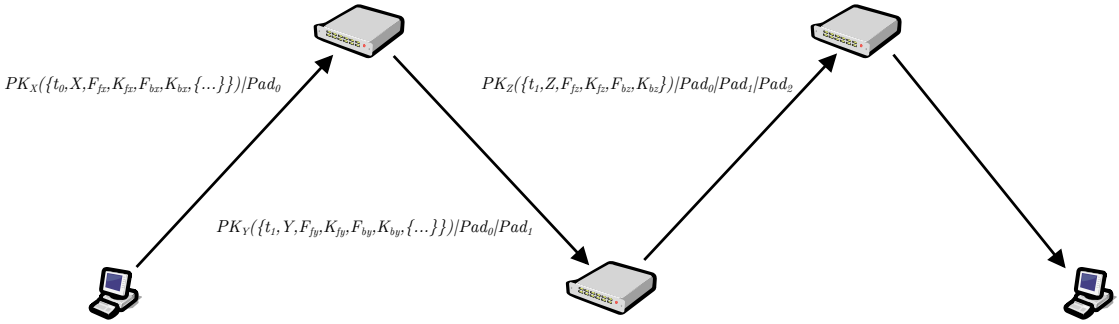

Abbildung 3.2: Einfacher virtueller Kanal über mehrere Onion Router

Um die Sicherheit zu erhöhen, bricht Onion Routing aus dem festen Premixed routing der Chaum-Mixes aus. Eine Mix-Kaskade kann von einem der Konten erweitert werden, indem er den verschlüsselten Inhalt für den nächsten Hop durch eine dazwischen liegende Kaskade transportiert die er vorher selbst aufbaut. Dieses Loose routing wird durch ein weiteres Feld im Kopf der Nachricht gesteuert. Damit ergibt sich folgende Onion.

$$\mathcal{O}_{b,x,y} = \{\text{exp_time}_x, \text{max_loose}, Y, F_{fx}, K_{fx}, F_{bx}, K_{bx}, \text{PK}_y(\mathcal{O}_{x,y,z}), \text{Pad}\}$$

Der Knoten X kann jetzt $\text{PK}_y(\mathcal{O}_{x,y,z})$ über bis zu max_loose weitere Onion Router zu Y leiten (siehe Abb. 3.3).

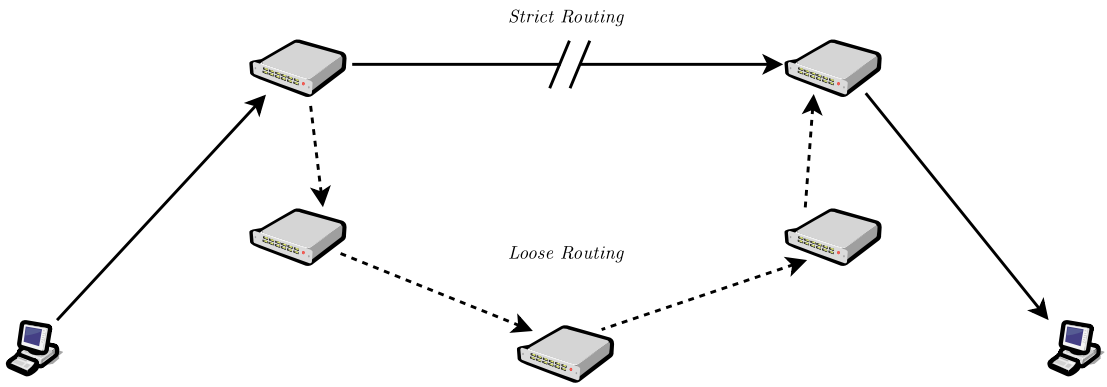

Abbildung 3.3: Virtueller Kanal mit Loose routing

Analog zu den anonymen Rücksendeadressen in Chaum Mixes ist es auch im Onion

Routing möglich, asynchron (z.B. als Antwort auf eine E-Mail) einen Rückkanal zum ursprünglichen Initiator der Mix-Kaskade aufzubauen (siehe Abb. 3.4). Diese Reply Onion unterscheidet sich im prinzipiellen Aufbau nicht von den in Vorwärtsrichtung benutzten. Sie werden vom ursprünglichen Initiator der Verbindung konstruiert und enthalten innerhalb der letzten Verschlüsselung einen Datensatz, der es dem letzten Onion Router erlaubt den Rechner von $\mathcal{A}$ zu erreichen.

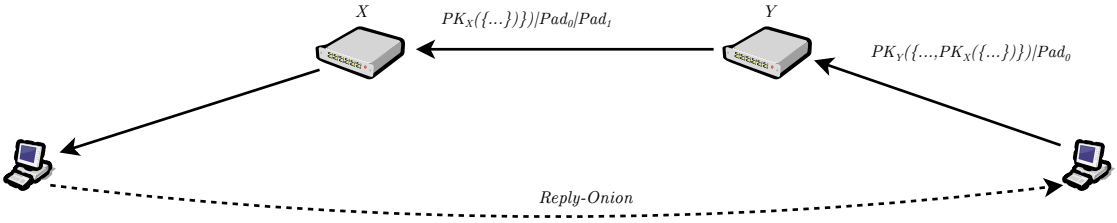

Abbildung 3.4: Rückkanal über eine Reply Onion

3.2 2nd Generation

Die oben beschriebene erste Generation der Onion Router hatte noch einige, hauptsächlich praktische, Schwachstellen.

- Zu anonymisierende Verbindungen wurden direkt vom Onion Router entgegen genommen. Damit musste Unterstützung für jedes Anwendungsprotokoll einzeln implementiert werden.
- Kein zentrales Verzeichnis. Jedem Teilnehmer mussten alle Onion Router im Netz bekannt sein.
- Das im Transport angeheftete Padding machte das Protokoll Ressourcen-intensiver aber nicht sicherer[1].
- Es wurde nicht in nennenswertem Umfang und außerhalb von Laborbedingungen eingesetzt.
- Keine Gewährleistung der Integrität der Pakete.

Aus dem Versuch diese Nachteile anzugehen ging Tor hervor. Tor unterscheidet zwischen zwei verschiedenen Teilnehmern. Onion Proxys nehmen über das SOCKS-Protokoll zu anonymisierende Verbindungen entgegen und Onion Router, welche Nachrichten innerhalb des Netzes weiterleiten. Jeder dieser Teilnehmer unterhält TLS Verbindungen zu jedem anderen (bekannten) Onion Router des Tor-Netzwerks. Durch diese läuft alle Kommunikation. Zusätzlich zu den Kurzzeitschlüsseln des TLS Protokolls besitzt jedes Mitglied noch einen Langzeitschlüssel zur Identifikation. Onion Proxys etablieren Circuits über mehrere Onion Router durch diese werden dann Streams versandt, welche

mit SOCKS Verbindungen korrespondieren. Eine Onion Proxy kann mehrere Circuits unterhalten, von dem jeder mehrere Streams transportieren kann.

Das Tor-Anwendungsprotokoll versendet 512 Byte große Cells (siehe Abb. 3.5). Jede dieser besteht aus einer 2-Byte-Circuit-ID (circID) und einem 1-Byte-Kommando. Danach folgt der Rest des Paketes, welcher abhängig vom Kommando interpretiert wird.

max. 512 Bytes

CircID	create, created, padding, destroy	Length	Nutzdaten

CircID	relay	Length	begin, end, data, connect(ed), extend(ed), truncate, sendme, resolve(d)	Recognized	Stream ID	SHA-1 Hash	Length	Nutzdaten

Abbildung 3.5: Aufbau einer Cell abhängig vom Kommando.

Im Gegensatz zum klassischen Onion Routing werden Circuits bei Tor nicht für jede Verbindung neu aufgebaut, indem eine beliebig geschachtelte Verschlüsselung der Originalnachricht verschickt wird, sondern iterativ aus einem Pool von etablierten ausgewählt.

Diese Circuits werden stückweise über einzelne Onion Router etabliert, wobei $\mathcal{A}$ mit jedem dieser Teilnehmer einen symmetrischen Schlüssel aushandelt. Cells die den Circuit hinunter wandern werden mit AES im Counter-Mode[1] unter diesen Schlüsseln verschlüsselt (In den folgenden Diagrammen mit $K_i(X)$ bezeichnet). Wenn eine Onion Proxy ein solches Kommando erhält, entschlüsselt sie es und überprüft das Recognized Feld. Wenn dieses Null ist und der SHA-1 Hash korrekt, wird die Cell verarbeitet. Wenn nicht, dann wandert sie weiter den Circuit herunter.

Um einen Circuit zu öffnen, sendet die Onion Proxy von $\mathcal{A}$ eine Cell mit create Kommando mit einer zufällig gewählten Circuit ID an einen ersten Onion Router. In der Cell befindet sich der von $\mathcal{A}$ bestimmte Teil einer Diffie-Hellman Key-Exchange g^α, verschlüsselt mit dem öffentlichen Schlüssel des Onion Routers PK_{O_i}. Die Gegenseite antwortet mit created, ihrem öffentlichen Schlüssel g^β und einem Hash über den resultierenden Schlüssel K. Nun ist ein 1-Hop-Circuit zwischen $\mathcal{A}$ und dem ersten Onion Router etabliert. Um einen weiteren hinzuzufügen sendet $\mathcal{A}$ ein relay extend an den ersten Knoten im Circuit. Dieser wird die Nachricht weiterleiten. Der letzte Hop im Circuit (hier: der Erste) wird dann ein create Kommando an den, in der Nachricht von $\mathcal{A}$ enthaltenen, Onion Router senden. Danach ein relay extended zurückschicken (siehe Abb. 3.6).

Um Circuits wieder abzubauen, sendet $\mathcal{A}$ ein relay truncate Kommando an einen Onion Router innerhalb ihres Circuit. Diese schickt eine destroy Cell an die nachfolgenden Onion Router. Diese lösen ihren Teil der Verbindung und leiten die Nachricht weiter. Der ursprüngliche Onion Router antwortet letzten Endes mit einem relay truncated Kommando. Um einen einzelnen Circuit komplett zu schließen, kann $\mathcal{A}$ einfach ein destroy

[1] Die Tor Specifications bezeichnen diese Konstruktion als Stromchiffre (0.3. Ciphers)

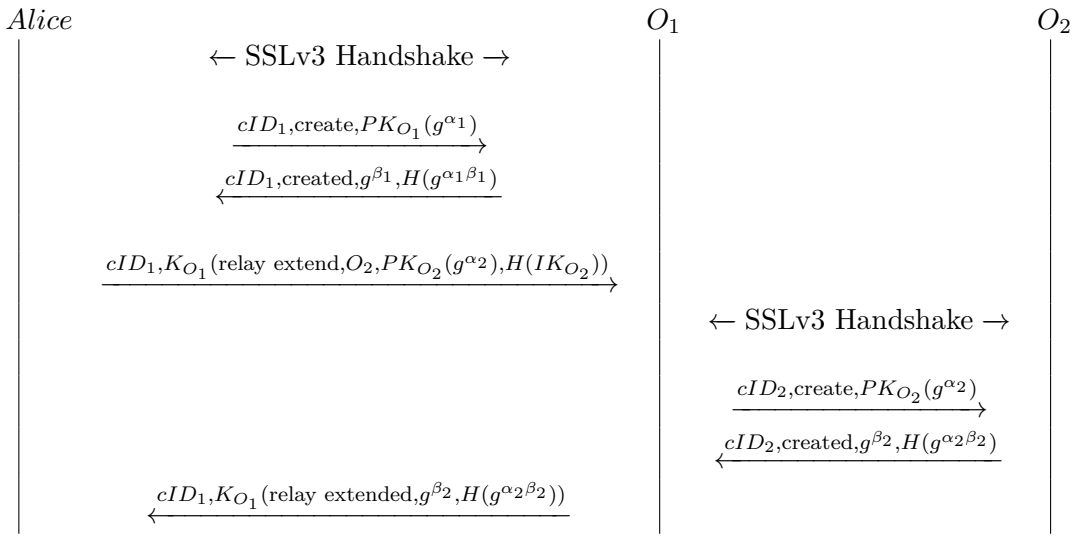

Abbildung 3.6: Konstruktion eines 2-Hop-Circuit.

an den ersten Onion Router schicken (sozusagen ein relay truncate an sich selbst).

Alle Onions die durch diese Circuits fließen, sind zusätzlich noch mit einer SHA-1 Prüfsumme geschützt. Diese wird mit den symmetrischen Schlüssen initialisiert und wird laufend über alle, über den Circuit gesendete Pakete, gebildet. Da die Pakete verschlüsselt übertragen werden, kann die Prüfsumme nur am Ende des Circuits überprüft werden.

Zu lösen bleibt das Problem wie neue Teilnehmer die Adressen von Onion Router erfahren. Andere Netze verwendeten dafür Broadcasts[10], welche in regelmäßigen Abständen von allen Onion Routern als signierte Datensätze gesendet werden. Mit der Ausnahme von ISDN, wo das Transportmedium einen Broadcast-Kanal bietet, ist dieser Ansatz aber zu ressourcenintensiv. Tor nutzt auch hier die ökonomisch sinnvollste aller Lösungen in Form von redundanten Directory Servers. Diese, deren Adressen als bekannt angenommen werden (weil in der Applikation enthalten), kennen alle Onion Router des Tor Netzes. Eine Onion Proxy fragt beim Start per HTTP einen dieser Server nach einer Liste von bekannten Knoten und speichert diese. Onion Router senden ihrerseits Informationen an die Directory Server. Diese Router Descriptors sind, mit dem Identity Key signierte, Textdateien, welche den öffentlichen Schlüssel, die Adresse und die Exit Policy des Routers enthalten. Um die Listen auf allen (im Moment 9) Directory Servern synchron zu halten, sendet jeder Seine an alle Anderen. In regelmäßigen Abständen (15min, 30min) übernehmen die Directory Server alle Router in ihre Liste, die von mehr als der Hälfte der Anderen gespeichert sind. Diese Mehrheitsentscheidung verhindert einen (beabsichtigten) Drift der Server.

Zu den öffentlichen Directory Servern kommen noch nicht-öffentliche. Diese Bridges bieten einen Zugang zum Tor Netz, wenn öffentliche Teile des Netzes blockiert werden.

Da Tor, wie schon erwähnt, sich darauf konzentriert, möglichst viele Menschen zum Teilnehmen zu ermutigen, erlaubt es den Administratoren von Onion Routern das Ver-

halten der Anwendung an die eigenen Vorstellungen anzupassen. Die beiden Werkzeuge dafür sind Exit Policies und Bandbreitenlimitierung. Bei Letzterem kann die Datenrate durch den Onion Router auf das beschränkt werden, was der Betreiber des Rechners bereit ist, zu Verfügung zu stellen. Um die dadurch entstehenden Routen mit sehr unterschiedlichen Kapazitäten effizient zu nutzen, implementiert Tor an TCP angelehnte Windows. Eine Onion Proxy besitzt ein ausgehendes und ein eingehendes Window mit der Größe von 1000 Cells. Wenn nun Cells die Onion Proxy erreichen oder verlassen, wird das Window für diese Richtung um eins kleiner. Wenn das eingehende Window auf 900 verringert wurde, wird ein relay sendme Kommando an den vorherigen Onion Router gesendet und das Window wieder auf 1000 gesetzt. Wenn ein relay sendme Kommando empfangen wird, wird das ausgehende Window auf 1000 gesetzt (siehe Abb. 3.7). Damit behandelt jede Proxy ein Window (das Eingehende) und synchronisiert eines (das Ausgehende) mit dem nächsten Hop. Da Onion Proxys Daten nur weiterleiten, also gleich viele ein- wie ausgehende Cells haben, reicht das um die vom Administrator gesetzte Bandbreite effizient auszunutzen.

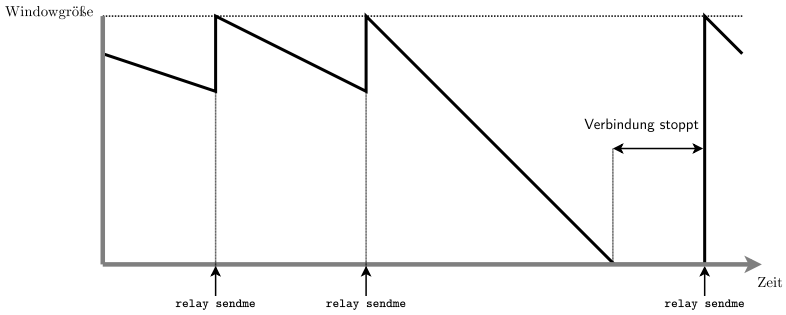

Abbildung 3.7: Beispiel zur Congestion control via Windows.

Weder Sequenznummern noch explizite Bestätigungen müssen zusätzlich zu den Windows implementiert werden, da die einzeln Datenströme TCP Verbindungen sind und Tor hier nur als (verlustbehaftetes) Netzwerkprotokoll behandelt wird.

Die zweite Möglichkeit einen Onion Router zu konfigurieren ist die Exit Policy. Wenn Tor zur Anonymisierung von Verbindungen zu Rechnern außerhalb des Netzwerks genutzt wird, fungiert der letzte Onion Router im Circuit als herkömmliche Proxy. Da dessen IP-Adresse genutzt wird, um die Anfrage zu senden, ist es auch diese, die als Quelle von Missbrauch interpretiert werden kann. Um dies zu unterbinden, ermöglicht Tor es dem Betreibern Verbindungen zu bestimmten IP-Adressen oder Ports zu verweigern. Dieses Regelwerk ist die Exit Policy des Knotens. Die meisten Exit Policies blockieren SMTP, um massives Spamming über Tor zu verhindern. Allerdings ist die überwiegende Zahl der Exit Policies als Blacklists angelegt, welche alles Erlauben und nur die für Missbrauch am anfälligsten Dienste verbieten [12].

3.2.1 Kommunikation

Um nun Nutzlasten über einen Circuit zu übertragen, nimmt die Onion Proxy eine SOCKS-Verbindung entgegen und sendet ein relay begin Kommando mit der Zieladresse und Portnummer an den Onion Router welcher als Endpunkt dienen soll. Dafür generiert sie eine zufällige Stream ID. Als Antwort erhält sie ein relay connected Kommando. Über diesen nun etablierten Stream können dann per relay data die Daten der SOCKS-Verbindung transportiert werden. Um den Stream wieder zu beenden, wird ein relay end Kommando gesendet und von der Gegenseite bestätigt. Im Falle eines nicht beabsichtigten Abbruchs sendet der letzte noch aktive Knoten in der Kaskade ein relay teardown (siehe Abb. 3.8).

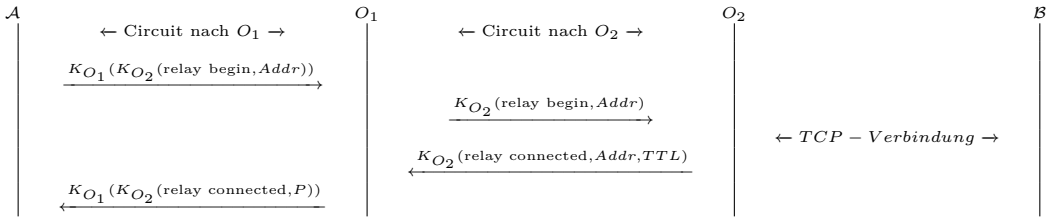

Abbildung 3.8: Aufbau und Abbau von Streams.

Solche Streams müssen nicht zwangsläufig durch den gesamten Circuit hindurchgehen. Die Onion Proxy wählt immer den jüngsten Circuit und in diesem einen Knoten mit einer passenden Exit Policy aus. Da die Kommandos, welche über einen Circuit transportiert werden, nur für den Empfänger zu lesen sind (verschachtelte Verschlüsselung), weiß kein Knoten, welche Art von Cell übertragen wird, noch die Länge der Kaskade. Dass ein Paket für ihn bestimmt ist, erfährt ein Onion Router erst, wenn nach der Entschlüsselung das eigentliche Kommando zu Vorschein kommt. Das erlaubt beim Anonymisieren von TCP-Verbindungen eine Leaky Pipe Architektur: Cells können an jedem Knoten der Kaskade Tor verlassen (siehe Abb. 3.9). Damit würde ein Angreifer, der die Länge des Circuits von $\mathcal{A}$ kennt, immer noch nicht die Länge des konkreten Streams kennen.

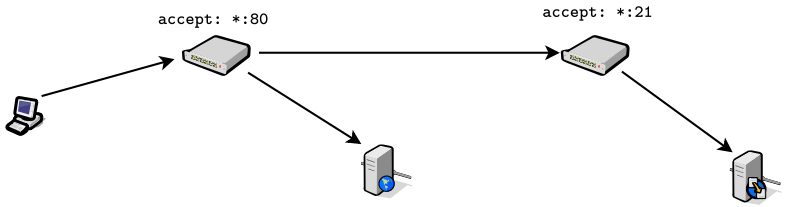

Abbildung 3.9: Tors Leaky Pipe Architektur.

3.2.2 Hidden Services

Tor besitzt zwei Anwendungsfälle. Einmal das beschriebene Anonymisierungsnetzwerk, welches Verbindungen von Teilnehmern Tors zu öffentlichen Servern verschleiert und

zum anderen anonymes Bereitstellen von Services. Diese (Location) Hidden Services bieten auch dem Betreiber $\mathcal{B}$ die Möglichkeit anonym zu bleiben (responder anonymity) und Denial-of-Service Attacken entgegenzuwirken. Der Kern dieses Mechanismus sind die in ISDN-Mixes beschriebenen Rendezvous Points: Onion Router die zwei Circuits verbinden und den Parteien erlauben anonym zu bleiben. Tor unterstützt Server für jedes Anwendungsprotokoll solange diese Verbindungen über die bind Methode von SOCKS annehmen können.

Ein Hidden Service besteht aus einem öffentlichen Schlüssel, mit dem $\mathcal{B}$ Nachrichten, die den Service betreffen, signiert und dessen Hash der den Server eindeutig im Tor Netz identifiziert. $\mathcal{B}$ wählt eine Reihe an Onion Routern als Introduction Points aus, welche er als signierten Datensatz veröffentlicht (siehe Abb. 3.10).

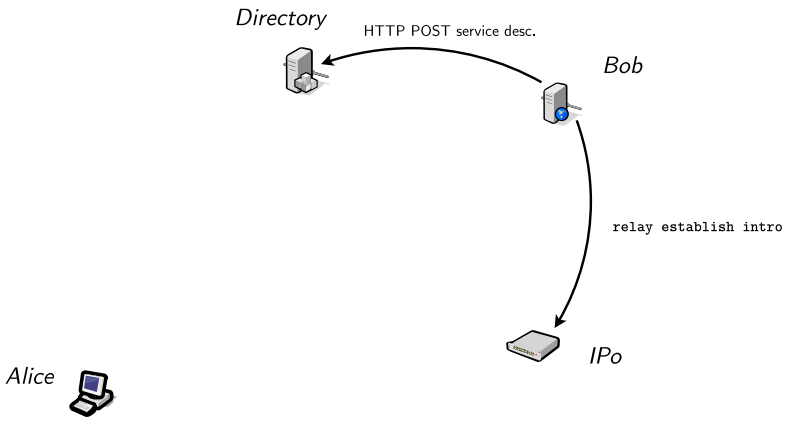

Abbildung 3.10: Etablierung eines Location Hidden Service in Tor.

Wenn nun $\mathcal{A}$ eine Verbindung zum Hidden Service von $\mathcal{B}$ aufbauen will, wählt sie ihrerseits einen Onion Router als Rendezvous Point und baut einen Circuit zu diesem auf. Danach informiert sie $\mathcal{B}$ durch eine Nachricht an einen der Introduction Points, welche den Rendezvous Point, einen zufälligen Wert (Rendezvous Cookie) zur Identifikation von $\mathcal{B}$, ein optionales Authorization Token und die erste Hälfte einer Diffie-Hellman Key-Exchange enthält. Der Introduction Point leitet die Anfrage weiter an $\mathcal{B}$ (ohne dessen Identität zu kennen. $\mathcal{B}$ hält einen Circuit zum IPo offen). $\mathcal{B}$ hat nun die Wahl auf die Anfrage zu reagieren oder nicht. Im Fall eines DoS Angriffs könne $\mathcal{B}$ z.B. es vom Authorization Token abhängig machen, ob er reagiert (und Ressourcen investiert) oder nicht. Wenn $\mathcal{B}$ die Anfrage annimmt, baut er einen Circuit zum Rendezvous Point von $\mathcal{A}$ auf und sendet diesem das Rendezvous Cookie und seine Hälfte der DHKE. Der Onion Router, welcher die Rolle des Rendezvous Point innehat, kann anhand des Cookies die Circuits von $\mathcal{A}$ und $\mathcal{B}$ verbinden. $\mathcal{A}$ sendet ein relay begin um zum Schluss einen Stream zur Onion Proxy von $\mathcal{B}$ und darüber zum Hidden Service zu etablieren (siehe Abb. 3.11).

Seit der Inbetriebnahme von Tor 2003 wurden zwei Verbesserungen für das grundlegende Protokoll vorgeschlagen[26]. Um die Effizienz der Circuit Konstruktion zu verbessern, kann der Standard Diffie-Helmann KE-Algorithmus mit Preshared g^{β} verbessert werden.

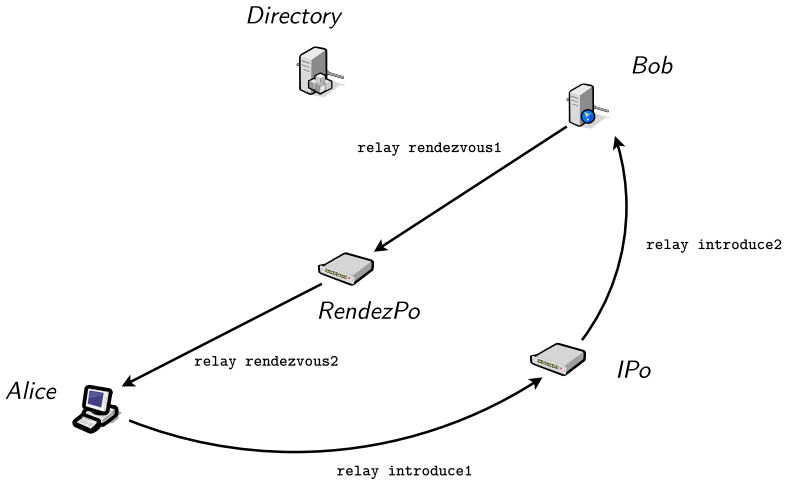

Abbildung 3.11: Konstruktion einer Verbindung zu einem Hidden Service.

Wenn eine Onion Proxy einen Circuit erzeugt, muss jeder Knoten auf dem Weg einen neuen öffentlichen Schlüssel generieren. Um dieses Potenzieren einzusparen, veröffentlichen alle Onion Router einen Langzeitschlüssel. $\mathcal{A}$ kennt dann alle Parameter um einen symmetrischen Schlüssel inklusive der create bzw. extend Nachricht in einem Paket an den Onion Router zu senden (siehe Abb. 3.12). Damit wird eine Round-Trip-Time und einmal Potenzieren eingespart.

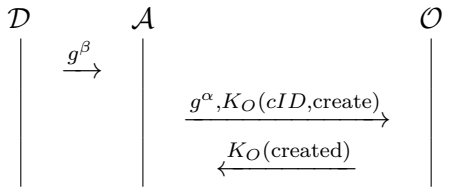

Abbildung 3.12: Aufbau eines Circuits mit Preshared DH und dem symmetrischen Schlüssel $K_O = g^{\alpha\beta}$.

Die zweite große Modifikation des Protokolls betrifft die Hidden Services. Die Introduction Points eines Hidden Service bilden einen Angriffspunkt. Die als IPo fungierenden Onion Router sind allen Teilnehmern bekannt, da diese in den Directory Server veröffentlicht werden müssen. Sie können damit zu Ziel von DDoS Attacken werden, schlimmer noch, Betreiber eines IPo könnten (durch z.B. juristische Mittel) dazu gebracht werden ihre Dienste einzustellen um, damit einen Hidden Service still zulegen. Die beiden Ursachen dieses Problems, dass Introduction Points öffentlich und nur wenige sind, kann durch die Nutzung von Valet Nodes behoben werden. Hierbei wird die, im ursprünglichen Onion Routing Konzept noch vorhandene, Idee der Reply Onions teilweise wiederbelebt. Ein Hidden Service generiert mehrere Valet Tickets welche Out-of-Band an $\mathcal{A}$ geschickt werden. Diese sind mit einem aus dem Hash des öffentlichen Schlüssels abgeleiteten, sym-

metrischen Schlüssel verschlüsselt. $\mathcal{A}$ nutzt diesen um, einen Circuit zu dem im Valet Ticket gespeicherten Valet Node aufzubauen. Diesem sendet sie die im Ticket enthaltenen Contact Information. Mit diesen kann der Valet Node den Introduction Point erreichen, welcher seinerseits die Verbindungsanfrage an den Hidden Service weiterleitet (siehe Abb. 3.13). Zu den wenigen IPos kommen nun viele Valet Nodes hinzu. Die Menge und Art dieser kann der Hidden Service im Voraus bestimmen, wenn er die Valet Tickets generiert. So können diese mit einem weiteren Cookie verschlüsselt werden und damit nur von autorisierten Onion Proxys benutzt werden. Da die Contact Information nicht den Hidden Service enthalten und Valet Nodes nicht die Adresse des IPos wissen die Valet Nodes nicht für welchen Hidden Service sie eine Verbindungsanfrage weiterleiten und die Introduction Points bleiben anonym.

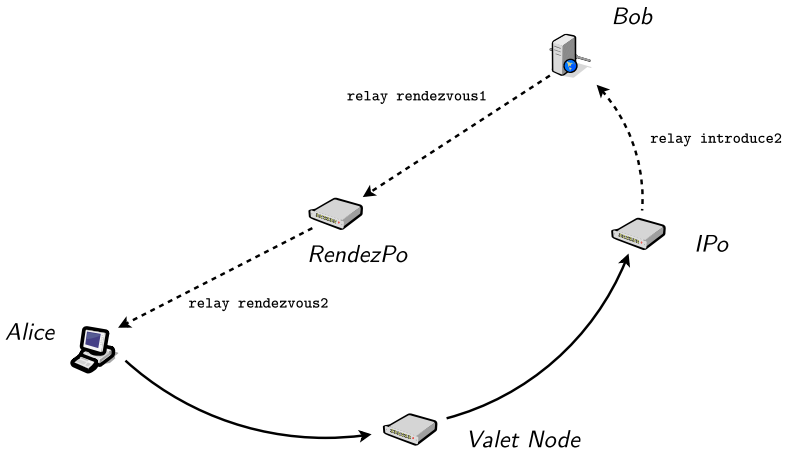

Abbildung 3.13: Etablierung eines Kanals mit einem Hidden Service über Valet Nodes (Neuerungen hervorgehoben).

Der Nachteil der Valet Node ist ein weiteres verkomplizieren von Hidden Services. Mit Valet Nodes sind insgesamt vier Circuits nötig um eine Verbindung aufzubauen: vom Hidden Service zum Introduction Point, vom Client zum Valet Node, dann zum Rendezvous Point und vom Hidden Service zu diesem. Da ein Tor Circuit mindestens Länge drei einhält, sind insgesamt 12 TLS Kanäle an eine einzige Client-zu-Hidden-Service Route gebunden.

3.3 Verringerung der Latenz

Starkes Augenmerk wird bei Tor auf die Erhöhung der Performanz gelegt. Wie z.B. in der Frage ob Padding für einzelne Cell angewendet werden soll, oder nicht geht Tor meistens den Weg, der bessere Leistung verspricht, ohne zu viel Sicherheit einzubüßen. Da Tor, wie schon erwähnt, ein ausgesprochenes low-latency Netzwerk ist, stellt sich immer die Frage wie Verbindungen schneller werden können, ohne ihre Anonymität zu verlieren.

Eine der in [8] besprochenen Möglichkeiten ist es, mehr Onion Router ins Netz zu

bringen. Da Tor nach wie vor ausschließlich auf freiwilliger Basis betrieben wird, wäre es interessant zu ermitteln, was diese Menschen dazu bringt ihre Ressourcen (Bandbreite und Zeit) zur Verfügung zu stellen, um vielleicht mehr dazu zu ermutigen dasselbe zu tun. Ideen wie „Premium“ Service (mehr Bandbreite, Vorzugsbehandlung beim Routing) für zahlende Onion Proxys könnten mit Hilfe von Systemen wie Bitcoins [20] implementiert werden, sollten diese verhältnismäßig stabil[2] werden.

Ein neues Setup für Hidden Services welches den Parteien erlaubt beim Verbindungsaufbau zu wählen ist eine andere, konkretere Möglichkeit. Entweder sie nutzten einem vom Hidden Service ausgewählten Contact Point der mit dem alten Introduction Point vergleichbar ist, oder der Client gibt einen Rendezvous Point vor der aber nicht am Ende eines neuen Circuit sitzt, sondern der letzte Onion Router vor dem Valet Node ist. Damit entfällt der klassische Rendezvous Point, ohne dass Sicherheit eingebüßt wird, da der Hidden Service vom Valet Node keine Informationen über diesen letzten Hop erhält. Der Contact Point übernimmt hier die Aufgabe, die Circuits zu verbinden.

[2] Im Fall von Bitcoins wäre eine Verringerung der Volatilität der Währung durch mehr Marktteilnehmer schon ein großer Schritt vorwärts.

4 Alternative Techniken

Zum Abschluss wollen wir noch einen Blick über den Tellerrand wagen und alternative Arten anonymer Kommunikation im Internet beleuchten. In der fast unüberschaubaren Vielfalt von anonymen Netzwerken und Datenspeichern verfolgt Tor einen gesamtheitlichen Ansatz. Es bietet Hidden Services wie andere Overlay Networks und Anonymisierung von Zugriffen auf Rechner im „normalen“ Internet. Tors Ziel ist es so benutzerfreundlich wie möglich zu sein, da ein Onion Routing Netzwerk nur mit vielen Teilnehmern anonym ist.

Im Gegensatz dazu bietet das Anonymisierungsnetzwerk Crowds nur Onion Routing zu öffentlichen Webseiten und das Overlay Network Freenet einen anonymen und Zensurresistenten Datenspeicher. Beide erreichen das durch Peer-to-Peer Routing.

4.1 Freenet

Das Freenet[5] welches von Ian Clarke 1999 entwickelt wurde konstruiert aus einem Netz aus Computern einen verteilten Datenspeicher, indem sich Informationen speichern und diese mit einem Minimum an Routing Informationen wieder zu extrahieren lassen.

Anstatt auf Onion Routing basiert das Freenet auf vertrauenswürdigen Verbindungen (Trusted Connections). Teilnehmer bauen nur Verbindungen zu Rechnern auf, denen sie vertrauen. Dieses wird technisch erreicht, indem diese Zertifikate austauschen. Das Vertrauensverhältnis hilft Freenet, sein Sicherheitsziel der Senderanonymität zu erreichen. Wenn ein (böswilliger) Knoten im Netz eine Anfrage erhält, kann dieser nicht entscheiden, ob diese vom Absender selbst kam oder ob er diese nur weitergeleitet hat. Da Freenet Peer-to-Peer operiert wäre es ohne die Voraussetzung von vertrauenswürdigen Nachbarn für beliebige Knoten möglich, alle Anfragen ihrer Nachbarn auszuspionieren, wenn diese nur diesen einen Knoten kennen. Diese Netzarchitektur wird als Darknet bezeichnet.

Das Freenet anonymisiert keine beliebigen TCP-Verbindungen wie Tor. Es hat keinen Kontakt zum öffentlichen Internet, sondern bildet ein eigenes Overlay Network indem Daten anhand von Schlüsseln gespeichert werden können. Dafür bilden alle Teilnehmer einen sogenannten Distributed Hash Table. Das gesamte Netz verhält sich wie eine Schlüssel/Wert Datenbank. Jeder Teilnehmer wählt eine zufällige Identifikationsnummer und verbindet sich mit beliebigen vertrauenswürdigen Peers.

Informationen im Freenet bestehen aus einem Schlüssel K und dem dazugehörigen Datensatz D. Um einen Datensatz im Freenet zu finden, sendet $\mathcal{A}$ eine Anfrage an den Nachbarn, dessen ID am nächsten zum gesuchten Schlüssel ist. Wenn $\mathcal{B}$ eine Anfrage erhält, durchsucht er seinen Zwischenspeicher und wenn dieser K nicht enthält wird er die Anfrage nach der gleichen Regel wie $\mathcal{A}$ weiterleiten. Wenn dieser Nachbar die Anfrage

als fehlgeschlagen zurückschickt, wird $\mathcal{B}$ den zweit Nächsten fragen. Wenn keiner seiner Nachbarn in der Lage ist, die Daten (bei sich selbst oder bei seinem Nachbarn) zu finden, wird $\mathcal{B}$ seinerseits eine Fehlermeldung an den Absender schicken. Damit wird eine Tiefensuche auf das gesamte Netz durchgeführt, ohne das ein Knoten mehr als seine unmittelbaren Nachbarn kennt (siehe Abb. 4.1). Diese Konfiguration von vielen „kleinen Welten“ wird im Freenet als small world Routing bezeichnet[1].

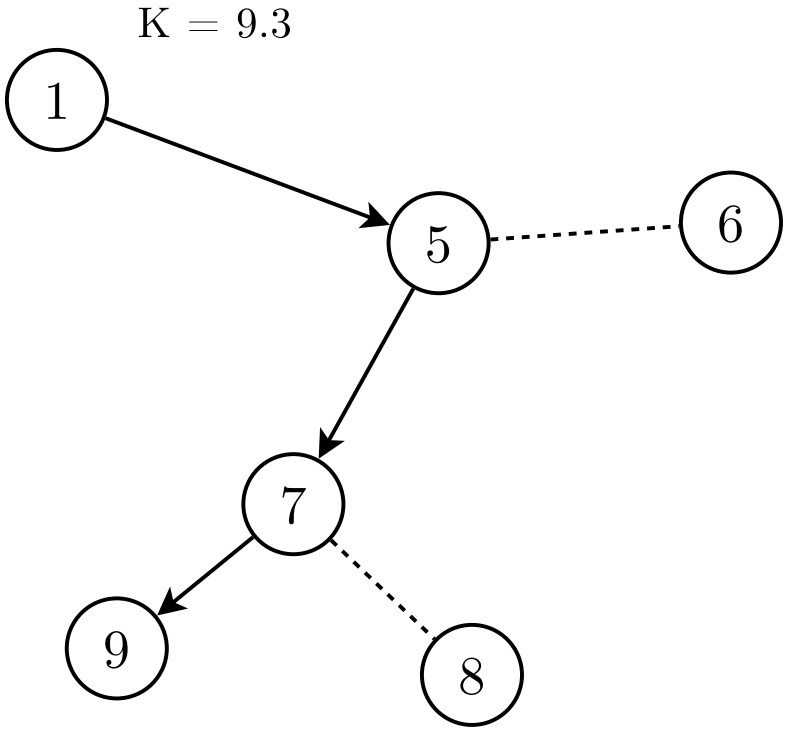

Abbildung 4.1: Suche nach einem Datensatz im Freenet.

Wenn der Datensatz gefunden wird, wird er auf dem Weg zurück zu $\mathcal{A}$ von jedem Teilnehmer in seinem Zwischenspeicher gesichert. Das Speichern eines Datensatzes funktioniert ähnlich. $\mathcal{A}$ sendet die Daten an den Nachbarn, dessen ID am nächsten zum Schlüssel ist. Die Daten werden weitergeleitet, bis sie an einen, Knoten ankommen, dessen eigene Identifikationsnummer näher am Schlüssel liegt als die aller seiner Nachbarn. Dieser Knoten wird die Daten dann in seinem Langzeitspeicher aufbewahren.

Unter der Voraussetzung, dass die Identifikationsnummern der Teilnehmer und Schlüssel gleich verteilt über alle möglichen Werten sind, wird das Netz Datensätze gleichmäßig verteilen und Engpässe vermeiden, wobei sich Daten mit ähnlichen Schlüsseln um Teilnehmer mit ähnlichen IDs gruppieren. Daten in diesem Netz zu zensieren ist extrem schwer. Ein Angreifer kennt nicht die IDs aller Teilnehmer (vorausgesetzt er überwacht nicht den gesamten Netzwerkverkehr des Darknets) und kann somit nicht Datensätze

[1] In Anlehnung an Stanley Milgrams six degrees of separation[18]

gezielt löschen. Selbst wenn der Knoten mit einem bestimmten Datensatz vom Netz getrennt wird, sind immer noch Kopien in den Zwischenspeichern der um ihn befindlichen Teilnehmern vorhanden. Die einzige Möglichkeit Informationen aus dem Freenet zu entfernen, ist es nicht nach ihnen zu fragen.

4.2 Crowds

Crowds[22] ist ein Onion Netzwerk, welches auf Anonymisierung von HTTP-Verkehr zugeschnitten ist. Es unterscheidet sich von Tor hauptsächlich durch das Fehlen von Hidden Services und einer dramatisch vereinfachten Struktur. Zudem ist Crowds ein Peer-to-Peer Netz, welches (im Gegensatz zu Freenet) jeden Teilnehmer aufnimmt. Crowds Ansatz Anonymität zu gewährleisten basiert auf der Abschwächung des Angreifermodells. Crowds schützt die Identität seiner Teilnehmer vor anderen Knoten im Crowds System und vor dem Endpunkt im öffentlichen Internet. Es geht nicht von einem global Agierenden, sondern von einem lokalen Angreifer aus, der nur einen Teil (wie ISP-Konzentrationsnetz oder Firmen-Intranet) kontrollieren kann. Crowds baut Kaskaden über möglichst weit verteilte (in verschieden Autonomous system befindlichen) Knoten um die Wahrscheinlichkeit, durch Korrelation des Verkehrs enttarnt zu werden zu verringern.

Da Crowds nur HTTP unterstützt, ist es in der Lage Protocol cleaning durchzuführen. Während bei Tor zusätzliche Software wie Privoxy[2] zu Einsatz kommt, um bösartige HTTP-Objekte zu filtern, macht Crowds dieses selbst. Hierbei werden Tracking-Cookies und HTTP-Header entfernt. Crowds ist dennoch (wie fast alle Dienste im Internet) verwundbar für Denial-of-Service Angriffe.

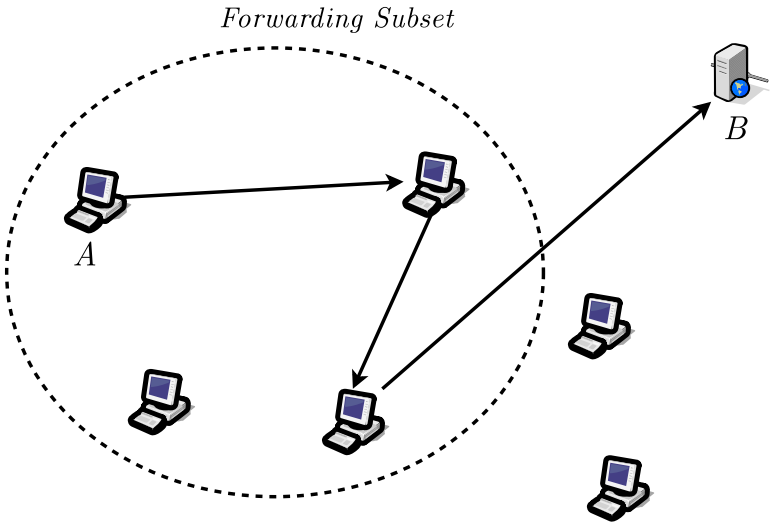

Abbildung 4.2: Funktionsweise von Crowds.

[2] http://www.privoxy.org

Alle Teilnehmer im Peer-to-Peer orientierten Crowds empfangen Anfragen von anderen Knoten zu Webseiten im öffentlichen Internet. Der einzelne Teilnehmer hat nun die Wahl diese an einen Anderen weiterzuleiten oder sie dem ultimativen Empfänger zuzustellen. Die Entscheidung wird anhand einer Zufallsvariable gefällt. Jede Anfrage besitzt eine zufällige Path-ID, die vom Initiator generiert wird. Diese erlaubt es, die Antwort auf demselben Weg wie die Anfrage zurückzuschicken (siehe Abb. 4.2). Kommunikation zwischen zwei Teilnehmern ist mit einem gemeinsamen, von einer zentralen Instanz generierten, Schlüssel geschützt (siehe unten). Damit ähnelt Crowds Art, Kaskaden zu generieren dem Leaky Pipe Modell von Tor: Der Initiator einer Transaktion hat keinen Einfluss auf die Route, diese entsteht zufällig. Allerdings wird diese Kaskade für alle weiteren Anfragen (genauso wie für die Antworten) verwendet. Obwohl es auf den ersten Blick so scheint, als ob das Wählen einer neuen Kaskaden für jede Anfrage die Sicherheit erhöhen würden, ist dem nicht so.

Damit ein böswilliger Knoten den Absender einer Nachricht erfahren kann, muss er wissen wann der unmittelbar vor ihm in der Kaskade sitzende Teilnehmer, der ursprüngliche Initiator der Anfrage ist. Aus der Sicht des Initiators stellt sich also die Frage: wie hoch ist die Wahrscheinlichkeit, dass falls sich ein böswilliger Knoten auf dem Pfad zum ultimativen Empfänger befindet, dieser die Position 1 (erster Knoten nach dem Initiator) einnimmt? Die Situation führt zum folgenden Modell.

n	Anz. Teilnehmer im System.
c	Anz. Teilnehmer unter feindlicher Kontrolle.
$p_f \in]0.5, 1]$	Ws., dass d. Knoten d. Nachricht weiterleitet.
H_k	Ereignis, dass der k-te Knoten böswillig ist.
$H_{k+} = H_k \vee H_{k+1} \vee \cdots \vee H_n$	
I	Erster böswilliger Knoten unmittelbar vor d. Initiator.

Gesucht ist nun $P(I|H_{1+})$, also die Wahrscheinlichkeit, gegeben einen böswilligen Knoten auf dem Pfad (H_{1+}), dass dieser der Erste ist (I). Zuerst ist die Wahrscheinlichkeit, dass der i-te Knoten auf dem Pfad der erste böswillige ist, ist

$$P(H_i) = \frac{c}{n} P(H_{i-1}) = \frac{c}{n} \prod_{j=0}^{i-1} \frac{p_f(n-c)}{n} = \frac{c}{n} \left(\frac{p_f(n-c)}{n} \right)^{i-1}$$

Dann ist

$$P(H_{i+}) = \frac{c}{n} \sum_{j=0}^{\infty} \left(\frac{p_f(n-c)}{n} \right)^j$$

und die geometrische Reihe liefert

$$P(H_{1+}) = \frac{c}{n - p_f(n-c)}$$

Die gesuchte Wahrscheinlichkeit ist dann

$$P(I|H_{1+}) = \frac{P(H_1)P(I|H_1) + p_f}{P(H_{1+})} = \frac{n - p_f(n-c-1)}{n}$$

Der hintere Teil im Nenner beschreibt die Möglichkeit, dass der Initiator die Nachricht an sich selbst schickt, was mit Wahrscheinlichkeit $\frac{1}{n}$ geschieht. Auflösen der Gleichung nach n ergibt

$$n \geq \frac{p_f(c+1)}{p_f - \frac{1}{2}} \Leftrightarrow P(I|H_{1+}) \leq \frac{1}{2}$$

Was eine Grenze für die Zahl an kollaborierenden, böswilligen Knoten gibt unter der das Netz noch glaubhafte Bestreitbarkeit ($P = \frac{1}{2}$) gewährleistet.

Letzte Probleme sind die Fragen, wie alle Teilnehmer des Systems voneinander erfahren, wie die Mitgliedschaft zum Crowds Netzwerk geregelt und wie die gemeinsamen Schlüssel generiert werden sollen. Die aktuelle 1.0 Version von Crowds verwendet ein einfaches, zentralisiertes Modell, mit sogenannten Blender Servern, die den Directory Server des Tor Netzwerks entsprechen. Jeder Teilnehmer teilt ein Benutzername/Passwort-Paar mit dieser zentralen Instanz. Wenn er sich mit dem Netz verbindet, authentifiziert er sich am Blender und erhält eine Liste von Crowds Knoten, inklusive der dazugehörigen symmetrischen Schlüssel, an die er Anfragen weiterleiten kann und welche ihrerseits Anfragen an ihn senden. Der Blender informiert alle ihm bekannten, aktiven Teilnehmer über Veränderungen wie Ein- und Austritt von Knoten vom Netz.

Den einzelnen Peers ist es aber erlaubt Teilnehmer von ihrer Liste zu streichen, falls sie nicht reagieren. Diese einfache Lösung birgt dieselbe Gefahr einer Partition Attack wie Tors Ansatz. Auch muss die Teilnahme an Crowds reguliert werden, da sonst ein Angreifer unter verschiedenen Identitäten das Netz betreten könnte und damit genug Knoten kontrollieren würde um mit hoher Wahrscheinlichkeit einzelne Teilnehmer zu enttarnen oder einzelne Crowds auf Netze zu beschränken, über die er die volle Kontrolle hat. Die dafür nötige Zentralisierung von Vertrauen im Blender ist aber schon aus praktischen Überlegungen problematisch. Die im Paper von Reiter und Rubin[22] vorgeschlagene Aufnahmeprozedur über notariell beglaubigte Formulare ist wohl als rein akademischer Vorschlag zu werten.

4.2.1 Multicast

Eine interessante Erweiterung zu Crowds ist Hordes[16], welches IP-Multicasting für den Rückkanal verwendet. Jeder Teilnehmer des Netzes besitzt ein Public-Key Paar, dessen öffentlicher Schlüssel allen anderen Teilnehmern bekannt ist (Hordes erreicht dies durch ein Blender-Server ähnliches, zentrales Register). Ein Teil dieser Rechner, das Forwarding Subset erhalten einen symmetrischen Schlüssel K_f von $\mathcal{A}$ (verschlüsselt mit ihrem öffentlichen Schlüssel). Eine Anfrage besteht nun aus einer zufälligen Path-ID, einer Multicast Adresse m und den zu transportierenden Daten. Diese Nachricht verschlüsselt $\mathcal{A}$ und sendet sie an einen zufällig ausgewählten Teilnehmer. Dieser entschlüsselt die Nachricht und verfährt genauso wie $\mathcal{A}$ mit Wahrscheinlichkeit $1 - p_f$. Falls $\mathcal{B}$ sich nun entscheidet (mit Ws. p_f) eine empfangene Anfrage zuzustellen, wird er die Antwort nicht wie bei Crowds über denselben Pfad zurücksenden, sondern die, mit K_f verschlüsselt, an m schicken. $\mathcal{A}$ wird über die, mit der Antwort übertragene, Path-ID erkennen können, dass es sich bei der, per Multicast empfangenen Nachricht um die Antwort auf ihre Anfrage handelt.

5 Angriffe

Nach der Beschreibung von Onion Routing, Tor und einigen Alternativen, wollen wir nun Angriffe auf diese Systeme besprechen.

Unter Anonymität werden wir im weiteren die Unbeobachtbarkeit der Kommunikation zwischen $\mathcal{A}$ und $\mathcal{B}$ und die Unverkettbarkeit einzelner Nachrichten oder Entitäten im System verstehen. Es geht also um die Vertraulichkeit der Quelle und des Ziels einer Nachricht. Auch muss der Inhalt der Nachrichten vertraulich übertragen werden, da dieser Informationen über Quelle/Ziel beinhalten könnte. Ohne den Einsatz von Steganografie ist für einen Einzelnen nur möglich sich in einer Gruppe Unverdächtiger zu verbergen. Im Weiteren werden wir diese Gruppe von Kommunikationsteilnehmern als Anonymity Set bezeichnen. Alle Anstrengung richtet sich auf das Ziel, einzelne Teilnehmer glaubhaft in der Masse deren Anonymity Sets untergehen zu lassen.

Bekannte Angriffspunkte bilden dabei einzelne oder Ströme von Paketen, die durch eine Mix-Kaskade fließen, die Zeit, welche Pakete dafür benötigen, die Anwendungsprotokolle, welche innerhalb eines anonymen Kanals verwendet werden und - etwas losgelöst von der technischen Umsetzung - juristische Fragen.

5.1 Verkehrsanalyse

Das Mittel um Anonymität zu gewährleisten ist, abgesehen von symmetrischer und asymmetrischer Kryptografie, die schon eingeführte Mix. Nach der Beschreibung des Konzepts ist es sinnvoll, die theoretischen Grenzen dieser Lösung anhand eines idealisierten Modells zu untersuchen. Wie in [14] gezeigt, lässt sich eine Mix durch folgende Parameter beschreiben.

N Gesamtzahl der Teilnehmer im System.

b Größe des Zwischenspeichers einer Mix, also Anzahl Nachrichten, die gesammelt und in zufälliger Reihenfolge weitergeleitet werden. Es gilt $b \leq N$.

n Größe des Anonymity Sets, also Menge der Teilnehmer, unter deren Nachrichten die Kommunikation von $\mathcal{A}$ gemischt wird. Wobei $n \leq b$ gilt.

$S_{i \in [N]}$ Quelle einer Nachricht im Mix Zwischenspeicher. $\mathcal{A}$ ist eine von ihnen.

$R_{i \in [N]}$ Empfänger einer Nachricht, nachdem die Mix sie gemischt weitergeleitet hat. $\mathcal{B}$ ist hier enthalten.

m Anzahl der Kommunikationspartner von $\mathcal{A}$. Darunter fällt $\mathcal{B}$ und Dummy-Partner, welche $\mathcal{A}$ zur Ablenkung (siehe unten) kontaktiert.

Unser Angreifer ist in der Lage den gesamten Netzwerkverkehr zu beobachten, allerdings nicht die Verschlüsselung der Zieladressen zu brechen, noch hat er Zugriff auf den internen Zustand (Zwischenspeicher) der Mix. Kann aus simpler Beobachtung ein Rückschluss auf die Partner von $\mathcal{A}$ gezogen werden?

In [14] wird eine Methode beschrieben, die in zwei Phasen und endlich vielen Beobachtungen alle Kommunikationspartner von $\mathcal{A}$ errechnet, solange $m \leq \lfloor N/n \rfloor$, also die Partner von $\mathcal{A}$ in disjunkten Anonymity Sets zu finden sind. Diese Disclosure Attack (siehe Abb. 5.1) funktioniert wie folgt. In einer ersten Phase, der Learning Phase, partitioniert ein Angreifer die Menge der Teilnehmer im System in m disjunkte Teilmengen. Danach, in der Excluding Phase (siehe Abb. 5.2), wird die Menge der Elemente in diesen Mengen sukzessive verringert. Jede neu beobachtete Menge von Empfängern, welche komplett in einer der disjunkten Teilmengen enthalten ist, ermöglicht den Ausschluss aller nicht in der Schnittmenge enthaltenen Empfänger.

$U' \leftarrow U^N$ {Menge der Teilnehmer}
$O \leftarrow \emptyset$ {Beobachtete Empfänger}
$R \leftarrow \emptyset$ {Gesuchte Partitionierung}
while $R \cap U \neq \emptyset$ do
  $O \leftarrow O \cup$ observe()
  $R \leftarrow$ try_partition(U',O)
end while
return R

Abbildung 5.1: Learning Phase einer Disclosure Attack.

Kern des Algorithmus ist die try_partiton() Funktion welche versucht U' in m disjunkte Teilmengen in R zu partitionieren. Diese Aufgabe lässt sich als binäres Constraint Satisfaction Problem beschreiben. Für kleine Parameter ist es möglich dieses $\mathcal{NP}$-Vollständige Problem praktisch zu lösen.

Da die Sender nicht koordiniert sind und damit jede Beobachtung eine zufällige Menge aus U' liefert, wird es in der Excluding Phase immer möglich sein, innerhalb endlich vieler Beobachtungen die Mengen in R auf ein Element zu reduzieren.

Experimente[14] mit den drei Parametern des Systems N, m und b zeigen, dass bei $N = 20000$ und $b = 50$ bis $m = 25$ die Zahl der benötigten Beobachtungen linear auf etwa 200 steigt, nach aber exponentiell zunimmt. Bei konstanten m explodiert die Zahl an Beobachtungen mit $b = 75$ bis $b = 85$ von 500 auf 2500. Es lässt sich also daraus schließen, dass bei einer rein theoretischen Betrachtung einer Mix die Parameter m und b so groß wie möglich gewählt werden sollten, um maximale Anonymität zu erreichen. Beide haben aber negativen Einfluss auf die Performance des Gesamtsystems. Eine große Menge an Blindnachrichten ($m \to \infty$) verringert die Bandbreite im Kanal, sowie an den einzelnen Mixes. Ein großer Zwischenspeicher ($b \to \infty$) erhöht die Latenz des System, da es bei gleicher Nachrichtenrate länger dauert, bis einzelne Nachrichten die Mix verlassen.

Die Disclosure Attack wurde in [2] als Dogan Attack noch einmal verbessert. Hierbei wird die Menge der Teilnehmer nicht partitioniert und dann verringert, sondern als

```
R ← learning_phase()
M ← ∅ {Gefundene Partner von 𝒜 }
while |R| > 0 do
  O ← observe()
  if ∃R_i : R_i ∩ O ≠ ∅ ∧ R_j ∩ O = ∅ ∀j ≠ i then
    R_i ← R_i ∩ O {Letzte beobachtete Empfänger ist in R_i}
    if |R_i| = 1 then
      M ← M ∪ R_i {Menge auf einen Teilnehmer eingegrenzt}
      R ← R ∩ {R_i}
    end if
  end if
end while
return M
```

Abbildung 5.2: Excluding Phase einer Disclosure Attack.

Vektor beschrieben. Seine Elemente sind die Wahrscheinlichkeiten das ein bestimmter Knoten der Kommunikationspartner von $\mathcal{A}$ ist. Das führt zu folgendem Modell.

$\vec{u}$ Vektor aller Teilnehmer im System. Die Komponenten, mögliche Kommunikationspartner von $\mathcal{A}$, werden auf $\sqrt{N}/N$ gesetzt.

$\vec{o_i}$ Beobachtung zum Zeitpunkt i. Die Komponenten welche Nachrichten aus der i-ten Mix-Runde erhalten haben werden auf $1/b$ gesetzt, der Rest auf 0.

$\vec{v}$ Gesuchter Vektor mit den Wahrscheinlichkeiten, dass ein Teilnehmer Kommunikationspartner von $\mathcal{A}$ ist.

Alle Vektoren haben $\dim(\vec{x}) = N$ und $|\vec{x}| = 1$. Mit jeder Beobachtung $\vec{o_i}$ wird der Vektor $\vec{v}$ verfeinert (siehe Abb. 5.3). Das entspricht der Learning Phase in der klassischen Disclosure Attack.

$$\vec{v} = b\frac{\sum_{i=0}^{t} \vec{o_i}}{t} - (b-1)\vec{u}$$

Abbildung 5.3: Konstruktion des Vektors aller Kommunikationspartner von $\mathcal{A}$ mit t Beobachtungen.

Wenn $\vec{v}$ berechnet wurde, kann aus jeder weiteren Beobachtung eine Wahrscheinlichkeitsverteilung über alle möglichen Teilnehmer konstruiert werden und der Endpunkt mit der höchsten als $\mathcal{B}$ identifiziert werden (siehe Abb. 5.4).

Der größte Vorteil einer Dogan Attack ist die Tatsache, dass keine $\mathcal{NP}$-Vollständigen Probleme gelöst werden müssen, was sie für praktische Anwendungen interessanter macht.

Wenn das Angreifermodell nun erweitert wird, um auch aktive Angriffe zuzulassen, wird es möglich durch Eingriff in den Strom aus Paketen, $\mathcal{B}$ zu identifizieren.

$$\mathcal{B}_i = \max(\frac{\vec{v} * \vec{o_{t+i}}}{|\vec{v} * \vec{o_{t+i}}|})$$

Abbildung 5.4: Endpunkt von der Nachricht von $\mathcal{A}$ in Runde i.

Eine erste Idee ist es die statische (und bekannte) Größe des Zwischenspeichers einer Mix auszunutzen. Eine ganze Klasse dieser aktiven Angriffe wird als Blending Attack in [24] beschrieben. Es werden entweder zusätzliche Nachrichten an die Mix von $\mathcal{A}$ geschickt, oder einzelne Pakete aus dem Strom entfernt, um die gesamte Mix-Kaskade nach $\mathcal{B}$ offen zu legen.

Angenommen eine Nachricht von $\mathcal{A}$ erreicht die Mix unmittelbar, nachdem ihr Zwischenspeicher geleert wurde, dann ist die Nachricht die Einzige im Stapel. Wenn nun ein aktiver Angreifer genug Nachrichten an diese Mix sendet, um den Zwischenspeicher zu füllen, wird er die Nachricht von $\mathcal{A}$ aus dem Strom der ihm bekannten isolieren können. Selbst wenn mehrere unbekannte Nachrichten im Zwischenspeicher sind, bevor dieser „geflutet“ wird, kommt das einer Verringerung des Parameters b gleich was passive Verkehrsanalyse erleichtert (siehe oben).

5.2 Timing Attacks

Nicht nur die Quelle und das Ziel eines Pakets kann genutzt werden um eine Mix-Kaskade zu „de-anonymisieren“, auch die Latenz einzelner Pakete oder die Bandbreite eines Datenstroms kann Informationen über die Kommunikationspartner enthalten.

Wei Dai zeigt in[6] eine simple aktive Attacke gegen Mix-Kaskaden, die Datenströme übertragen. Ein Angreifer generiert Blindverkehr durch die Mixes von denen er vermutet, dass sie zur Kaskade von $\mathcal{A}$ nach $\mathcal{B}$ gehören. Ist diese Vermutung richtig, wird sich die Bandbreite von $\mathcal{A}$ zur ersten Mix verringern, da einzelne Mixes nur eine begrenzte Kapazität haben. Diese Clogging Attack kann noch modifiziert werden, indem man nicht den gesamten Datenstrom durch eine Mix verlangsamt sondern nur einzelne Pakete verzögert. Diese muss sich dann durch die gesamte Kaskade bis $\mathcal{B}$ fortsetzen. Beide Techniken benötigen einen Angreifer der in der Lage ist den gesamten Netzwerkverkehr durch das Netzwerk zu verfolgen, was nicht immer praktikabel ist. In [19] wird eine verbesserte Version dieser Attacke beschrieben, welche nur einen böswilligen Server und eine Mix (in diesem Fall Tor) unter der Kontrolle des Angreifers benötigt. Die fundamentale Beobachtung ist, dass alle Datenströme durch eine Tor Mix um Ressourcen konkurrieren. Wenn die Datenrate eines Stroms steigt, steigt auch die Latenz der anderen Verbindungen. Ein Angreifer etabliert nun eine Verbindung mit den Mixes welche er in der Kaskade von $\mathcal{A}$ vermutet und misst die Latenz von Paketen, welche durch diese 1-Hop-Verbindung geleitet werden. Wenn nun $\mathcal{A}$ eine Anfrage an den böswilligen Server initiiert wird dieser die Datenrate der Antwort in einem erkennbaren Muster modulieren (z.B. Puls-förmig). Wenn $\mathcal{A}$ ihre Pakete durch die Mix erhält, zu welcher der böswillige Tor Knoten seine

Verbindung aufgebaut hat, wird dieser das Muster in Form von wechselten Latenzen wiedererkennen (siehe Abb. 5.5).

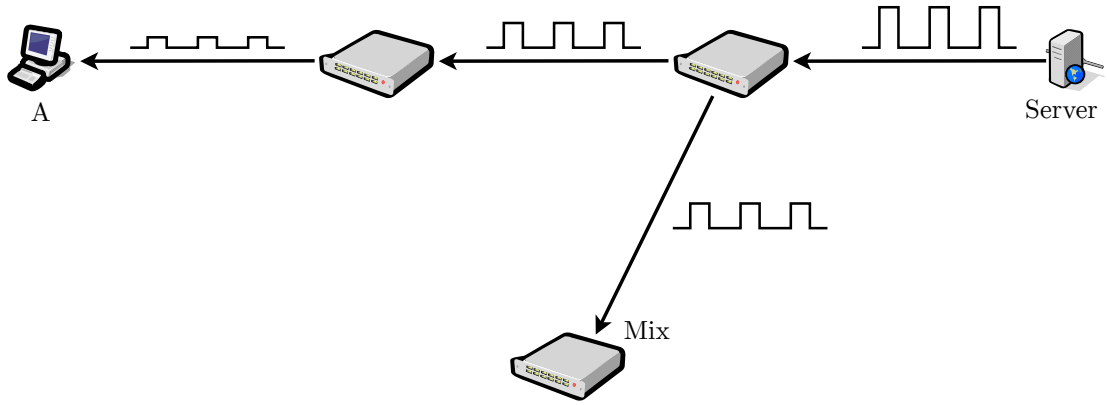

Abbildung 5.5: Das auf die Datenrate aufgeprägte Muster kann in Form der Latenz wiedererkannt werden.

Wie in [13] gezeigt ist die Latenz von einzelnen Paketen ein Indikator, von wo sie kommen. In einem hypothetischen Anonymisierungsnetzwerk, welches keine weitere Latenz verursacht, können die Absender von Paketen aus 14.000 auf zwei bis vier eingeschränkt werden, nur aufgrund der Round-Trip-Time. Obwohl dieser Angriff im ersten Augenblick theoretisch ist, sollte man bedenken, dass die Verringerung der Latenz in vielen praktischen Onion Routing Systemen ein Ziel ist. Daher ist auch Tor verwundbar für diese Angriffsform[25].

5.3 Anwendungsprotokolle

Die vergangene Diskussion zum Onion Routing drehte sich nur um die Netzwerk- und Transportprotokolle. Dieser Ansatz liegt nahe, da dies der Horizont der Mix Technik ist. In der Praxis werden aber komplexe Anwendungsprotokolle durch Mix-Kaskaden getunnelt, welche auch Möglichkeiten zum Identifizieren eines Benutzers bieten.

Für diesen Angriff prädestiniert ist HTTP. Laut [17] macht Port 80 16% aller Datenströme aus, die das Tor Netzwerk verlassen. HTTP eröffnet einem Angreifer die Möglichkeit, Programmcode in Form von aktiven Inhalten auf dem Computer von $\mathcal{A}$ auszuführen. Andrew Christensen von FortConsult zeigt in seinem Paper [4] einige Beispiele, wie mit Hilfe von Abobe Flash, Java Applets, Javascript und ActiveX Objekten die IP-Adresse von $\mathcal{A}$ an einen bösartigen Server geschickt werden kann. Die gezeigten Methoden lassen sich in zwei Gruppen gliedern. Die Erste versucht die IP-Adresse, mit Hilfe des dem Skript bereitgestellten API zu ermitteln und durch einen Seitenkanal an den Server zu übertragen.

Java Applets können über java.net.InetAddress.getLocalHost() ein Objekt konstruieren, welches die IP-Adresse des Rechners enthält. Ähnliches ist mit ActiveX Objekten

möglich. Diese Informationen können dann zum Server geschickt werden, indem z.B. Javascript eine Ressource vom Server anfragt, die die IP-Adresse in der URL enthält. Problematisch ist, dass ActiveX nur im Internet Explorer ausgeführt werden kann und selbst Microsoft nicht empfiehlt, ActiveX Objekte von unbekannter Quelle auszuführen. Außerdem ist die IP-Adresse des Rechners oft keine öffentliche, da inzwischen selbst im Heimbereich NAT-Gateways Standard sind.

Die zweite Kategorie versucht, über einen Seitenkanal eine Verbindung zu einem böswilligen Server aufzubauen. Diese würde dann nicht über ein Anonymisierungsnetzwerk geleitet werden und $\mathcal{A}$ verraten. In den meisten Konfigurationen wird nur HTTP-Verkehr über eine lokale Proxy an eine Mix-Kaskade geschickt. Adobe Flash, Java Applets und neuerdings auch Javascript erlauben es, TCP-Socket Verbindungen aufzubauen. Der vorgesehenen Schutzmechanismus, nämlich die Same-Origin Policy würde hier nicht greifen, da die nicht-anonymisierte Verbindung zum selben Server und über das selbe Protokoll erfolgen wird.

Abseits von HTTP erlauben auch noch andere Protokolle die Identifikation des Clients. FTP im aktiven Modus sendet die eigene IP-Adresse an den Server, damit dieser seinerseits eine Datenverbindung zum Client aufbauen kann. SSL Cipher suites[23] sind zwar nicht eindeutig, können aber die Menge der Kandidaten für $\mathcal{A}$ eingrenzen.

5.4 Tor-spezifische Angriffe

Gegen Tor selbst gibt es noch Angriffe, die sich nicht ohne Weiteres auf allgemeine Onion Routing Netze übertragen lassen. Erstens die Tatsache, dass Tor Protokoll Cleaning von HTTP über externe Software realisiert: die vor der HTTP-Anfrage vom Browser gesendete DNS-Nachricht wird ohne zusätzliche Konfiguration nicht anonymisiert, was dem DNS-Server erlaubt, alle besuchten Webseiten zu protokollieren, auch wenn alle TCP-Kommunikation über Tor abgewickelt wird.

Zwei weitere Angriffe werden aufgrund Tors Directory Server möglich: Entscheidend für die Sicherheit ist, dass alle Directory Server identische Listen verteilen. Ansonsten ließe sich das Tor Netzwerk in mehrere Teile trennen (eine s.g. Partition Attack) um dann Nutzer innerhalb kleinerer Teilmengen leichter über Verkehrsanalyse zu identifizieren. Da Hidden Services, um Timing Attacken zu erschweren, oft auf Onion Router betrieben werden und in der Regel keine 99,99% Verfügbarkeit haben, ist es möglich durch Beobachtung der Listen der Directory Server, IP-Adressen von Hidden Services zu ermitteln. Ein Angreifer vergleicht die Verfügbarkeit der Onion Router (deren IP-Adressen über die Directory Server veröffentlicht werden müssen) mit der des gesuchten Hidden Service. Sollte dieser gleichzeitig ein Onion Router sein, wird seine Verfügbarkeit mit der des Onion Routers Korrelieren.

5.5 Juristisches

Bei der reinen technischen Betrachtung von Onion Routing, welche bis hier hin stattgefunden hat (und noch weiter gehen wird), sollte eines nicht vergessen werden: Mix-

Kaskaden funktionieren nur innerhalb eines Rechtsstaates. Auch wenn es nirgendwo expliziert erwähnt wurde, basieren alle diese Techniken von Chaum Mixes bis hin zu Tor und Freenet auf der Annahme, dass glaubhafte Bestreitbarkeit ausreicht, um Anonymität zu gewährleisten. Das setzt voraus, dass die Unschuldsvermutung gilt.

Wie schon im ersten Kapitel erwähnt, wird bei keinem der hier beschriebenen Systeme Steganografie eingesetzt. Es wird zwar erschwert (bis unmöglich gemacht) den Nachrichtenfluss zwischen zwei Teilnehmern eindeutig nachzuvollziehen, dennoch kann nicht verschleiert werden, dass beide durch ein Anonymisierungsnetzwerk kommunizieren. Wenn z.B. die Nutzung von Onion Routing an sich als illegal erklärt wird, ist auch wieder die Zensur (durch Blockierung) seiner Teilnehmer möglich.

Laut der Electronic Frontier Foundation [9] gab es noch keinen Fall, bei dem die Nutzung von Tor rechtliche Konsequenzen hatte. Allerdings gab es in 2003 einen Zwischenfall mit dem JAP-Anonymisierungsdienst welcher an der TU Dresden entwickelt wurde. Nach einem richterlichen Beschluss erhielt die Mix Software eine Hintertür (siehe Abb. 5.6), die es ermöglicht Anfragen nach bestimmten URLs zu protokollieren.

```
#ifdef LOG_CRIME
if (checkCrime(pChainCell->firstCell.data, payloadLength)) {
  /* we've captured a stupid gangsta, who sent a suspected
   * webaddress completely in the first packet
   */

  UINT8 crimeBuff[MAX_FIRST_UPSTREAM_CHAINCELL_PAYLOAD + 1];
  /* ensure that there is a trailing 0 -> use one byte more
   * than necessary for the plain data
   */
  memset(crimeBuff, 0, MAX_FIRST_UPSTREAM_CHAINCELL_PAYLOAD + 1);
  memcpy(crimeBuff, pChainCell->firstCell.data, payloadLength);

  /* for compatibility with the default mix-implementation
   * we will send an extra-packet on the channel with a
   * crime-signal (without using the channel-cipher)
   */
  tQueueEntry oSigCrimeQueueEntry;
  memset(&oSigCrimeQueueEntry, 0, sizeof(tQueueEntry));
  UINT32 id = m_pMuxIn->sigCrime(currentMixPacket->channel,
                                 &oSigCrimeQueueEntry.packet);
  m_pQueueSendToMix->add(&oSigCrimeQueueEntry, sizeof(tQueueEntry));
  m_logDownloadedPackets++;
  int log = LOG_ENCRYPTED;

  if (!options.isEncryptedLogEnabled()) {
    log = LOG_CRIT;
    CAMsg::printMsg(log,"Crime detected -- ID: %u -- Content: \n%s\n",
                    id, crimeBuff);
  }
}
#endif
```

Abbildung 5.6: Hintertür in der JAP Mix.

6 Fazit

Wir haben gesehen, wie Onion Routing die Möglichkeit bietet, in IP-basierenden Netzen Daten auszutauschen und dabei trotzdem glaubhafte Bestreitbarkeit zu gewährleisten. Von den simplen Anfängen bei Chaum, über ISDN-Mixes und letzten Endes zu Tor sind die Konzepte der Mix und der iterativen Verschlüsselung des Datenstroms sicherer und performanter geworden. So kann ein Tor-Circuit in den Bereichen Latenz und Bandbreite mit langsamen TCP-Verbindungen mithalten. Damit und durch Hidden Services kann Tor fast schon als „Drop-In“ Ersatz für nicht-anonymes TCP verwendet werden.

Dennoch bleiben Probleme zu lösen. Tors zentrale Directory Server skalieren nicht beliebig, Timing Angriffen können nicht vollkommen abgewehrt werden, ohne Performanz einzubüßen und Tor – wie auch das gesamte Onion Routing Konzept – ist nicht steganografisch. Es kann (und wird) durch Blockieren einzelner Rechner lahmgelegt. Auch bieten Anwendungsprotokolle eine schier unendliche Fülle an Seitenkanälen, um Teilnehmer zu identifizieren.

Deswegen bleibt die Entwicklung von Tor, welches als „[...] test-bed for future research.“ begann interessant. Steganographie, niedrigere Latenz, stabilere Directory Server Infrastruktur und nicht zuletzt die Frage wie mehr Menschen für Tor begeistert werden können, damit sie Ressourcen für Onion Router bereitstellen, bedürfen weiterer Forschung.

Literaturverzeichnis

[1] Alessandro Acquisti. Essays on privacy, anonymity, and tracking in computer-mediated economic transactions. PhD thesis, University of California, Berkeley, 2003. AAI3105132.

[2] Dakshi Agrawal, Dogan Kesdogan, and Stefan Penz. Probabilistic treatment of mixes to hamper traffic analysis. In Proceedings of the 2003 IEEE Symposium on Security and Privacy, SP '03, pages 16–, Washington, DC, USA, 2003. IEEE Computer Society.

[3] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. Commun. ACM, 24(2):84–90, February 1981.

[4] Faerch Christensen. Peeling the onion: Unmasking tor users.

[5] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: a distributed anonymous information storage and retrieval system. In International workshop on Designing privacy enhancing technologies: design issues in anonymity and unobservability, pages 46–66, New York, NY, USA, 2001. Springer-Verlag New York, Inc.

[6] Dai. Attack 1. http://www.weidai.com/freedom-attacks.txt.

[7] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: the second-generation onion router. In Proceedings of the 13th conference on USENIX Security Symposium - Volume 13, SSYM'04, pages 21–21, Berkeley, CA, USA, 2004. USENIX Association.

[8] Roger Dingledine, Nick Mathewson, and Paul Syverson. Challenges in deploying low-latency anonymity. Technical report, 2005.

[9] Electronic Frontier Foundation. Tor legal faq. https://www.eff.org/torchallenge/legal-faq/.

[10] Sharad Goel, Mark Robson, Milo Polte, and Emin Sirer. Herbivore: A scalable and efficient protocol for anonymous communication. Technical report, Cornell University, 2003.

[11] David M. Goldschlag, Michael G. Reed, and Paul F. Syverson. Hiding routing information. In Proceedings of the First International Workshop on Information Hiding, pages 137–150, London, UK, UK, 1996. Springer-Verlag.

[12] Goodell. reconsidering default exit policy. http://archives.seul.org/or/talk/Mar-2005/msg00042.html.

[13] Nicholas Hopper, Eugene Y. Vasserman, and Eric Chan-Tin. How much anonymity does network latency leak? In Proceedings of the 14th ACM conference on Computer and communications security, CCS '07, pages 82–91, New York, NY, USA, 2007. ACM.

[14] Dogan Kedogan, Dakshi Agrawal, and Stefan Penz. Limits of anonymity in open environments. In Revised Papers from the 5th International Workshop on Information Hiding, IH '02, pages 53–69, London, UK, UK, 2003. Springer-Verlag.

[15] Marcus et al. Leech. Rfc 1928: Socks protocol version 5, Mar 1996.

[16] Brian Neil Levine and Clay Shields. Hordes: a multicast based protocol for anonymity. J. Comput. Secur., 10(3):213–240, September 2002.

[17] Karsten Loesing, Steven J. Murdoch, and Roger Dingledine. A case study on measuring statistical data in the tor anonymity network. In Proceedings of the 14th international conference on Financial cryptograpy and data security, FC'10, pages 203–215, Berlin, Heidelberg, 2010. Springer-Verlag.

[18] Stanley Milgram. The small world problem. Psychology Today, 1967.

[19] Steven J. Murdoch and George Danezis. Low-cost traffic analysis of tor. In Proceedings of the 2005 IEEE Symposium on Security and Privacy, SP '05, pages 183–195, Washington, DC, USA, 2005. IEEE Computer Society.

[20] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. http://bitcoin.org, 2009.

[21] Andreas Pfitzmann, Birgit Pfitzmann, and Michael Waidner. Isdn-mixes: Untraceable communication with small bandwidth overhead. In Kommunikation in Verteilten Systemen, Grundlagen, Anwendungen, Betrieb, GI/ITG-Fachtagung, pages 451–463, London, UK, UK, 1991. Springer-Verlag.

[22] Michael K. Reiter and Aviel D. Rubin. Crowds: Anonymity for web transactions. Technical report, 1997.

[23] Ristić. Http client fingerprinting using ssl handshake analysis. https://www.ssllabs.com/projects/client-fingerprinting/.

[24] Andrei Serjantov, Roger Dingledine, and Paul F. Syverson. From a trickle to a flood: Active attacks on several mix types. In Revised Papers from the 5th International Workshop on Information Hiding, IH '02, pages 36–52, London, UK, UK, 2003. Springer-Verlag.

[25] Lasse Øverlier and Paul Syverson. Locating hidden servers. In Proceedings of the 2006 IEEE Symposium on Security and Privacy, SP '06, pages 100–114, Washington, DC, USA, 2006. IEEE Computer Society.

[26] Lasse Øverlier and Paul Syverson. Valet services: improving hidden servers with a personal touch. In Proceedings of the 6th international conference on Privacy Enhancing Technologies, PET'06, pages 223–244, Berlin, Heidelberg, 2006. Springer-Verlag.

[27] Lasse Øverlier and Paul Syverson. Improving efficiency and simplicity of tor circuit establishment and hidden services. In Proceedings of the 7th international conference on Privacy enhancing technologies, PET'07, pages 134–152, Berlin, Heidelberg, 2007. Springer-Verlag.