

# **Konstruktion und Implementierung eines versteckten Datenkanals mit Hilfe der Steganographie oder Covert Channels**

**Bachelorarbeit**

**Wintersemester 2018/19**

im Studiengang Angewandte Informatik

an der Hochschule Ravensburg - Weingarten

von

Maximilian Nestle Matr.-Nr.: 27427

Abgabedatum : 25. April 2019

---

# Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit mit dem Titel

**Konstruktion und Implementierung eines versteckten Datenkanals mit Hilfe  
der Steganographie oder Covert-Channels**

selbständig angefertigt, nicht anderweitig zu Prüfungszwecken vorgelegt, keine anderen als die angegebenen Hilfsmittel benutzt und wörtliche sowie sinngemäße Zitate als solche gekennzeichnet habe.

Weingarten, 25. April 2019

Maximilian Nestle

---

# Inhaltsverzeichnis

<b>Kurzfassung</b>	<b>V</b>
<b>1 Einleitung</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Aufgabenstellung und Zielsetzung . . . . .	2
1.3 Aufbau . . . . .	2
1.4 Eigene Leistung . . . . .	3
<b>2 Grundlagen</b>	<b>4</b>
2.1 Internet Protokolle . . . . .	4
2.1.1 Sicherungsschicht/Data Link Layer (Schicht 2) . . . . .	4
2.1.2 Vermittlungsschicht/Network Layer (Schicht 3) . . . . .	5
2.1.3 Transportschicht/ Transport Layer (Schicht 4) . . . . .	6
2.2 Datensicherheit . . . . .	7
2.2.1 Vertraulichkeit . . . . .	7
2.2.2 Integrität . . . . .	8
2.2.3 Authentizität . . . . .	8
2.2.4 Verfügbarkeit . . . . .	8
2.3 Datenschutz . . . . .	9
2.4 Kryptographie . . . . .	9
2.4.1 Symmetrische Verschlüsselung . . . . .	10
2.4.2 Asymmetrische Verschlüsselung . . . . .	10
2.4.3 „Shamir’s Secret Sharing” Methode . . . . .	11
2.5 Information Hiding . . . . .	12
2.5.1 Steganographie . . . . .	12
2.5.2 Covert Channel . . . . .	13
<b>3 Anforderung an die Lösung</b>	<b>16</b>
<b>4 Lösungsansätze</b>	<b>17</b>
4.1 Covert Channel . . . . .	17
4.1.1 Zeitabhängige Covert Channel . . . . .	17
4.1.2 Storage Channel . . . . .	19

4.1.3	IPv6 . . . . .	21
4.1.4	TCP . . . . .	21
4.1.5	Traffic Normalizers . . . . .	22
4.1.6	Benutzung verschiedener Protokolle . . . . .	22
4.1.7	Verwendung der Nutzdatengröße . . . . .	24
4.2	Steganographie . . . . .	25
4.2.1	Klassische textbasierende Verfahren . . . . .	25
4.2.2	Basierend auf RGB Bilder . . . . .	25
4.2.3	Bildformate mit Alpha Kanal . . . . .	27
4.2.4	Verwendung von PDF Dateien . . . . .	28
<b>5</b>	<b>Bewertung der Lösungsansätze</b>	<b>29</b>
<b>6</b>	<b>Konstruktion und Aufbau des Covert Channels</b>	<b>31</b>
6.1	Konstruktion des Covert Channels . . . . .	31
6.1.1	Geheime Daten . . . . .	31
6.1.2	Schlüssel . . . . .	32
6.1.3	Trägerkanal . . . . .	32
6.2	Aufbau des Systems . . . . .	33
6.2.1	Anforderungen an den Aufbau . . . . .	33
6.2.2	Lösungsansatz 1: Aktiver Aufbau . . . . .	33
6.2.3	Lösungsansatz 2: Passiver Aufbau . . . . .	34
6.2.4	Bewertung des Aufbaus . . . . .	34
<b>7</b>	<b>Umsetzung der aktiven Variante</b>	<b>36</b>
7.1	Fehlerkorrektur . . . . .	36
7.1.1	Anforderung an die Fehlerkorrektur . . . . .	36
7.1.2	Lösungsansatz 1: Paritätsbits . . . . .	36
7.1.3	Bewertung der Fehlerkorektur . . . . .	37
7.2	Integrität . . . . .	38
7.2.1	Anfoderung an die Hashfunktion . . . . .	38
7.2.2	Lösungsansatz 1: MD5-Hash . . . . .	38
7.2.3	Lösungsansatz 2: Pearson-Hash . . . . .	38
7.2.4	Bewertung der Hash-Funktionen . . . . .	39
7.2.5	Pearson Hash Implementierung . . . . .	39
7.3	Authentizität . . . . .	40
7.4	Netzwerkprotokoll . . . . .	40
7.4.1	Anforderung an das Netzwerkprotokoll . . . . .	40
7.4.2	Lösungsansatz 1: HTTP . . . . .	41
7.4.3	Lösungsansatz 2: SMTP . . . . .	41
7.4.4	Lösungsansatz 3: FTP . . . . .	41

7.4.5	Bewertung der Netzwerkprotokolle . . . . .	41
7.5	Server . . . . .	42
7.5.1	Anforderungen an den Server . . . . .	42
7.5.2	Lösungsansatz 1: Java HttpServer . . . . .	42
7.5.3	Lösungsansatz 2: Node.js und Express . . . . .	43
7.5.4	Bewertung des Servers . . . . .	43
7.5.5	Express Implementierung . . . . .	44
7.6	Kommunikation des Covert Channel . . . . .	44
7.6.1	Anforderung an die Kommunikation des Covert Channels . . . . .	45
7.6.2	Lösungsansatz 1: Websockets . . . . .	45
7.6.3	Lösungsansatz 2: Socket.IO . . . . .	45
7.6.4	Bewertung der Kommunikation des Covert Channels . . . . .	45
7.6.5	Socket.IO Implementierung . . . . .	46
7.7	Kodierung und Dekodierung . . . . .	46
7.7.1	Anforderung an die Kodierung und Dekodierung . . . . .	47
7.7.2	Lösungsansatz 1: Kodierung mit festen Zeitrastern . . . . .	47
7.7.3	Lösungsansatz 2: Kodierung basierend auf den Paketabständen . . . . .	47
7.7.4	Bewertung der Kodierung . . . . .	49
7.7.5	Implementierung der Kodierung . . . . .	49
7.8	Geheime Daten . . . . .	50
7.9	Front-End . . . . .	51
7.9.1	HTML . . . . .	51
7.9.2	JQuery . . . . .	51
7.9.3	Socket.IO . . . . .	51
7.10	Clientseitige Auswertung des Covert Channel . . . . .	51
7.10.1	Anforderung an die Auswertung . . . . .	52
7.10.2	Lösungsansatz 1: Auswertung im Front-End . . . . .	52
7.10.3	Lösungsansatz 2: Auswertung mit einem externen Programm . . . . .	52
7.10.4	Bewertung der Covert Channel Auswertung . . . . .	52
7.11	Client . . . . .	53
7.11.1	Programmiersprache . . . . .	53
7.11.2	Mitschneiden der Datenpakete . . . . .	54
7.11.3	Interpretieren der Zeitstempel . . . . .	56
7.11.4	Verarbeiten der Erhaltenen Daten . . . . .	58
<b>8</b>	<b>Optimales Einstellen und Anwendung des aktiven Covert Channel</b>	<b>59</b>
8.0.1	Verhältnis nicht interpretierbar/ interpretierbar . . . . .	59
8.0.2	Anzahl korrekt übertragener Dateien . . . . .	60
8.0.3	Zeit bis zum ersten korrekten Eintreffen der Datei . . . . .	61
8.0.4	Padding zum Synchronisieren . . . . .	63

---

<b>9</b>	<b>Umsetzung der passiven Variante</b>	<b>64</b>
9.0.1	Anforderung an den Proxy . . . . .	64
9.0.2	Lösungsansatz 1: Veränderung eines Open-Source Proxys . . . . .	64
9.0.3	Lösungsansatz 2: Proxy selbst programmieren . . . . .	65
9.0.4	Bewertung des Proxys . . . . .	65
9.0.5	Implementierung . . . . .	65
<b>10</b>	<b>Anwendung des passiven Covert Channel</b>	<b>67</b>
10.1	Probleme bei realen Webseiten . . . . .	67
10.1.1	HTTPS . . . . .	67
10.1.2	Generierung einer ausreichenden Anzahl von Netzwerkpaketen . . . . .	67
10.1.3	Parallele Datenabfrage . . . . .	68
10.2	Test mit realen Servern . . . . .	69
<b>11</b>	<b>Evaluierung</b>	<b>71</b>
<b>12</b>	<b>Fazit und Ausblick</b>	<b>72</b>
12.1	Aktiv . . . . .	72
12.2	Passiv . . . . .	72
12.3	Ausblick . . . . .	73
	<b>Danksagung</b>	<b>75</b>
	<b>Literatur</b>	<b>76</b>

# Kurzfassung

In dieser Arbeit wird eine Methode entwickelt, die als Alternative zu oft verwendeten Verschlüsselungsverfahren dienen soll. Hierzu werden Anwendungen aus dem Bereich der Steganographie, aber auch so genannte Covert Channels analysiert. Die Verfahren werden evaluiert und die beste Alternative als Proof of Concept implementiert.

Bei dem ausgewählten Kommunikationsverfahren handelt es sich um einen zeitabhängigen Covert Channel. Die geheimen Daten werden durch die Manipulation der Zeitabstände zwischen den Datenpaketen übertragen. Dafür gibt es zwei Möglichkeiten: Zum einen kann der sendende Server die Manipulation aktiv vornehmen, zum anderen bedient man sich einer passiven Variante, bei der ein vorgeschalteter Proxy die Beeinflussung der Zeitintervalle übernimmt. Zusätzlich wird ein Client erstellt, der den Covert Channel auswertet.

Die entstandene Lösungsvariante, dient als Proof of Concept und bestätigt die Funktionalität eines zeitabhängigen Covert Channels. Hiermit können Daten ohne inhaltliche Veränderung der Netzwerkpakete übertragen werden.

Heutzutage konzentrieren sich Angreifer meist auf das Dechiffrieren von verschlüsselten Nachrichten. Methoden wie Covert Channels sind dagegen nicht im Fokus.

# 1 Einleitung

## 1.1 Motivation

In der heutigen Zeit wird die Datensicherheit immer wichtiger, da immer mehr Daten im Internet preisgegeben werden. Um die Datenübertragung zu sichern wird meistens ein asymmetrisches Verschlüsselungsverfahren verwendet.

Dieses Verfahren bietet zwar ein hohes Maß an Sicherheit, hat aber auch Nachteile wie zum Beispiel eine Erhöhung der Rechenzeit, die Verwaltung eines Key-Managers und die Bedrohung durch einen „Man-in-the-Middle“ Angriff.

Mögliche Alternativen sind die Steganographie oder Covert Channels. Beide Techniken nutzen legitimierte Datenkanäle, um darin Daten zu verstecken, ohne eine Verschlüsselung anzuwenden. Dabei verwendet die Steganographie zum Beispiel Bild-, Audio- oder PDF-Dateien um Informationen zu verbergen. Covert Channels nutzen hingegen die Netzwerkattribute für die Datenübertragung.

Eine konstruktive Anwendung könnte die Kommunikation zwischen einem Polizeipräsidium und ihren verdeckten Ermittlern sein. Da die Fahnder davon ausgehen müssen, dass die Kommunikation abgehört wird, würde eine verschlüsselte Kommunikation zu viel Aufsehen erregen. Zudem kann es sein, dass die Verschlüsselung schon geknackt wurde.

Hier kommt die Steganographie ins Spiel, die es möglich macht einen legitimen und unauffälligen Kanal für die Datenübertragung zu verwenden. So ein Kanal könnte der Stream beim Schauen eines Videos oder die Nachrichten einer Webseite sein.

Die Steganographie, aber auch die Covert Channels, bieten gerade deswegen, da sie oft hinter den großen Verschlüsselungsverfahren in Vergessenheit geraten, eine sehr gut Möglichkeit um hoch sensible Daten zu versenden.



## 1.2 Aufgabenstellung und Zielsetzung

Ziel der Bachelorarbeit ist es, dem in der Motivation bereits beschriebenen Polizeieinsatz eine Kommunikationsmöglichkeit zu schaffen. Dabei soll es möglich sein, dass Anweisungen, Treffpunkte aber auch Bilder an den verdeckten Ermittler übertragen werden können. Hier soll die Kommunikation über ein Netzwerk stattfinden und mit Hilfe der Steganographie oder von Covert Channels realisiert werden.

Die entstehende Anwendung soll als ein „Proof of Concept“ dienen und das Potential von Information Hiding veranschaulichen.

Die Daten sollen nicht mit einem kryptographischen Verfahren verschlüsselt werden, sondern in ein oder mehreren Protokollen „versteckt“ eingebettet und übertragen werden. Dazu soll ein optimales Verfahren zur Dateninfiltration und -exfiltration gefunden werden. Das Verfahren sollte unauffällig, für Dritte schwer zu interpretieren und mit größt möglicher Übertragungsrate senden. Optimal wäre, eine ähnliche Sicherheit zu gewährleisten, wie mit einer mathematischen Verschlüsselung.

Ziel ist es außerdem, jedes Dateiformat übertragen zu können.

## 1.3 Aufbau

In Kapitel 2 werden die thematischen Grundlagen erklärt und definiert. Kapitel 3 beschäftigt sich mit der Findung der Anforderungen an die Lösung. Hier werden alle Punkte definiert die bei der Umsetzung realisiert werden müssen.

Danach beschäftigt sich die Arbeit in Kapitel 4 mit der Findung eines optimalen Verfahrens zum Information Hiding, mit dem die Daten übertragen werden sollen. In Kapitel 5 werden die gefundenen Methoden bewertet und die gewünschte Übertragungsart für die Umsetzung festgelegt.

Das 6. Kapitel thematisiert die Konstruktion und den Aufbau des Covert Channels. Danach wird sich mit der Umsetzung der aktiven Variante beschäftigt, die im 7. Kapitel dokumentiert ist. Kapitel 8 thematisiert das optimale Einstellen der Covert Channel Parameter und zeigt, wie dieser angewendet wird. Die Umsetzung der passiven Variante

folgt in Kapitel 9 und dessen Anwendung und Test in Kapitel 10. Die entstandenen Ergebnisse werden im 11. Kapitel evaluiert. Im letzten und 12. Kapitel folgt das Fazit und ein Ausblick auf mögliche Verbesserungsmöglichkeiten.

## 1.4 Eigene Leistung

Es werden steganographische Verfahren und Covert Channels betrachtet. Diese werden bewertet und analysiert und so das Optimum bestimmt, um eine verdeckte Datenübertragung zu realisieren.

Das gefundene Ergebnis wird als „Proof of Concept“ Anwendung implementiert, die passend zur Zielsetzung die Datenkommunikation übernimmt. Um diese Kommunikation möglich zu machen, wird ein Webserver und ein Proxy erstellt, die jeweils als Sender fungieren können. Zum Empfangen der Daten wird eine entsprechende Client-Anwendung entwickelt.

Das Programm wird danach evaluiert und auf reale Anwendungen getestet.

## 2 Grundlagen

### 2.1 Internet Protokolle

In den folgenden Kapiteln soll kurz das OSI-Schichtenmodell, auf welches das heutige Internet aufbaut, erklärt werden. Dabei repräsentiert jede Schicht Protokolle, die für die Kommunikation im Internet nötig sind.

Es werden nur die Protokolle betrachtet, die für dieses Projekt relevant sind.

#### 2.1.1 Sicherungsschicht/Data Link Layer (Schicht 2)

Diese Schicht beinhaltet Protokolle, welche eine weitestgehend fehlerfreie Datenübertragung garantieren sollen. Außerdem wird der Zugriff auf das Übertragungsmedium ermöglicht. [Wik18]

#### Ethernet

Beim Ethernet-Protokoll werden die Daten in Pakete zerteilt. Diese können zwischen den Geräten im Netzwerk verschickt werden. Dabei ist das Ethernet-Protokoll immer nur im jeweiligen Netzwerksegment gültig. [Zis13] Die Adressierung wird mit Hilfe der MAC-Adressen realisiert. Diese Adressen sind eindeutig und werden jedem netzwerkfähigen Gerät vom Hersteller zugeordnet.

7 Byte	1 Byte	6 Byte	6 Byte	4 Byte	2 Byte	bis 1500 Byte	max. 42 Byte	4 Byte
Präam- bel,	SFD	MAC- Adresse Ziel	MAC- Adresse Quelle	VLAN-Tag	Typ	Nutzdaten	PAD	FCS

Bild 2.1: Erweiterter Ethernet-Frame nach IEEE 802.1Q [Zis13]

### 2.1.2 Vermittlungsschicht/Network Layer (Schicht 3)

Die Protokolle dieser Schicht werden verwendet, um über Netzwerkgrenzen hinaus Nachrichten zu versenden. [Zis13] Diese Protokolle liegen innerhalb der Nutzdaten des Ethernet-Pakets.

#### IPv4

Zur Adressierung werden IPv4 Adressen verwendet, die 32 bit (4 Byte) lang sind. Vergeben werden die Adressen von der IANA (Internet Assigned Numbers Authority). Jeder, der aus dem Internet erreichbar sein will, muss sich bei der IANA, oder einer untergeordneten Organisation, eine IP-Adresse oder Adressbereich geben lassen.

Das Internet Protokoll wird, wie in unten stehender Abbildung 2.2 gezeigt, in das Ethernet Paket eingebettet.

#### IPv6

Da die IPv4 Adressen langsam knapp werden, wurde das IPv6 Protokoll erstellt, welches über Adressen mit 6 Byte Länge verfügt. Dies bedeutet, dass deutlich mehr Adressen erstellt und vergeben werden können. Die Funktion ist aber mehr oder weniger die gleiche.

Der Header des IPv6 Protokolls ist in folgender Abbildung 2.3 gezeigt.

Version	IHL	TOS	Länge	
Identifikation			Flags	Fragment Offset
TTL	Protokoll		Header-Prüfsumme	
Sender-IP-Adresse				
Ziel-IP-Adresse				
Optionen			Padding	
Daten				

Bild 2.2: IPv4 Header [Zis13]

Version	Traffic Class	Flow Label	
Payload Length		Next Header	Hop Limit
Absender-Adresse (128 Bit)			
Ziel-Adresse (128 Bit)			

Bild 2.3: IPv6 Header [Zis13]

### 2.1.3 Transportschicht/ Transport Layer (Schicht 4)

Diese Schicht beinhaltet Protokolle, die festlegt wie Nachrichten transportiert werden.

#### TCP

Bei TCP (Transmission Control Protocol) ist die Übertragung verbindungsorientiert. Mit Hilfe der Ports können, auf TCP aufbauende Protokolle, von Anwendungen adressiert werden. Damit keine Pakete verloren gehen oder falsch zusammengesetzt werden, arbeitet TCP mit Sequenz-Nummern die jedes Paket identifizieren. Hat der Empfänger ein Paket

erhalten so sendet dieser eine Empfangsbestätigung, die gleichzeitig ein nächstes Paket anfordern kann. Hierzu wird die Acknowledgement Number verwendet. Kommt keine Bestätigung so wird von einem Verlust ausgegangen und erneut gesendet. [Wen12b]

In Bild 2.4 ist der TCP Header gezeigt, der den Daten vorangestellt ist.

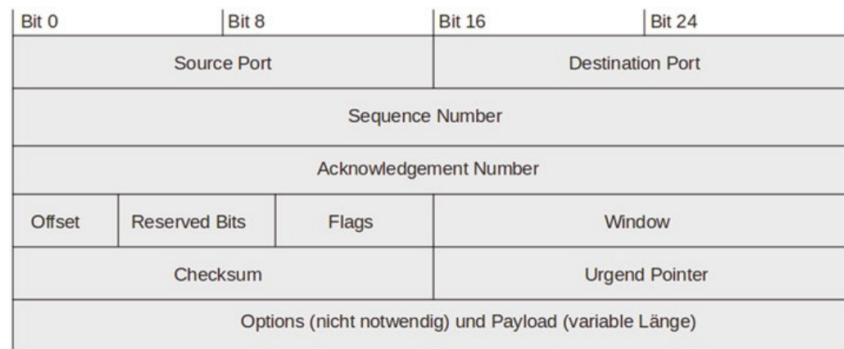


Bild 2.4: TCP Header [Wen12b]

## 2.2 Datensicherheit

Die Aufgaben der Datensicherheit werden durch das CIA-Prinzip beschrieben.

Zur Datensicherheit gehören nach diesem Prinzip alle Maßnahmen, die die **C**onfidentiality, **I**ntegrity und **A**vailability (Vertraulichkeit, Integrität, Verfügbarkeit) gewährleisten. [Nic18]

Eine zusätzliche Aufgabe ist die Sicherstellung der Authentizität. Viele dieser Aufgaben werden mit Hilfe der Kryptographie realisiert und umgesetzt.

### 2.2.1 Vertraulichkeit

Die Vertraulichkeit ist dann gewährleistet, wenn die Daten nicht von unbefugten Personen eingesehen werden können. Es muss also ein System verwendet werden, bei dem sich befugte Benutzer legitimieren können und unbefugte beim Interpretieren gehindert werden. In den meisten Fällen wird dies durch eine Verschlüsselung (symmetrisch oder asymmetrisch) umgesetzt. Alle legitimierten Benutzer erhalten den Schlüssel. Die Personen

ohne Schlüssel können die Informationen nicht entschlüsseln - die Vertraulichkeit ist so garantiert.

### **2.2.2 Integrität**

Die Integrität stellt sicher, dass Daten nicht unbemerkt verändert oder gefälscht werden. Eine Nachricht soll genau so beim Empfänger ankommen, wie sie abgesendet wurde. Hierzu können Hash-Funktionen verwendet werden, die beim Verändern der Nachricht andere Werte ergeben würden. Dabei müsste entweder die Hash-Funktion geheim sein oder der Hash-Wert verschlüsselt werden.

### **2.2.3 Authentizität**

Die Authentizität bestätigt, dass Daten von der angegebenen Informationsquelle stammen. Es ist ein Identitätsbeweis des Absenders gegenüber dem Empfängers.

Dies kann zum Beispiel mit einer Public-Key Verschlüsselung realisiert werden. Dafür verschlüsselt der Sender die Nachricht mit seinem Private Key. Der Empfänger kann einzig mit dem Public Key des Senders die Nachricht entschlüsseln. Ist das Entschlüsseln möglich, so weiß der Empfänger, dass die Nachricht genau von diesem Sender kommt.

### **2.2.4 Verfügbarkeit**

Der Dritte Punkt ist die Verfügbarkeit. Es soll immer sichergestellt sein, dass Daten aber auch Programme immer abrufbar sind. Hierzu gehören Mechanismen zur Vermeidung von DoS (Denial of Service) Angriffen. Diese Angriffe würden beispielsweise einen Server derart überfordern, dass dieser keine Dateien mehr ausliefern kann - die Verfügbarkeit ist dann nicht mehr gewährleistet.

## 2.3 Datenschutz

Der Datenschutz ist ein Überbegriff für das im Gesetz festgelegte Recht auf informationelle Selbstbestimmung. Was bedeutet, dass jede Person über die Preisgabe der personenbezogenen Daten bestimmen kann. Die Bundesbeauftragten für den Datenschutz und die Informationssicherheit (BfDI) definieren den Datenschutz wie folgt:

*„Datenschutz garantiert jedem Bürger Schutz vor missbräuchlicher Datenverarbeitung, das Recht auf informationelle Selbstbestimmung und den Schutz der Privatsphäre“* [Die18]

So kann jeder, der personenbezogene Daten, ohne Zustimmung des Betroffenen speichert oder weiterverarbeitet vor Gericht angeklagt werden. Damit dies nicht passiert, haben die meisten Institute, die mit personenbezogenen Daten umgehen, einen Datenschutzbeauftragten, der die Einhaltung dieser Gesetze überwacht.

## 2.4 Kryptographie

Kryptographie bedeutet wörtlich: „Die Lehre vom Geheimen schreiben“ [Hel18] und beschäftigt sich mit der mathematischen Verschlüsselung von Informationen. Dabei gibt es zwei große Verschlüsselungsarten - die symmetrischen und die asymmetrischen Verschlüsselungen.

Ein Grundprinzip der Kryptographie wurde bereits im 19. Jahrhundert von A.Kerkhoffs aufgestellt. Eine der wichtigsten Aussagen hierbei ist, dass die Sicherheit einer Verschlüsselung nicht von dem Verschlüsselungsalgorithmus, sondern allein von dem Schlüssel abhängig sei. [EL18] Das heißt, dass ein guter Verschlüsselungsalgorithmus öffentlich gemacht werden kann, ohne die Sicherheit zu gefährden. Ein Beispiel ist der RSA Algorithmus. Dies ist einer der heute verbreitetsten Algorithmen. Der Algorithmus ist für jeden öffentlich zugänglich, dies hat aber keine Auswirkung auf die Sicherheit, da die Sicherheit allein auf der Geheimhaltung des Passwortes basiert.

Dies hat zum Beispiel auch den Vorteil, dass in einer Firma bei einem Personalwechsel nicht der ganze Algorithmus ausgetauscht werden muss, sondern nur das Passwort.



### 2.4.1 Symmetrische Verschlüsselung

Bei der symmetrischen Verschlüsselung wird zum Verschlüsseln und Entschlüsseln der gleiche Schlüssel verwendet.

$$E_k(M) = C$$

$$D_k(C) = M$$

[Ert01]

Der Schlüssel  $k$  wird dazu verwendet die Nachricht zu ver- und entschlüsseln. Das Problem bei symmetrischen Verschlüsselungen ist die Schlüsselübertragung, die auf jeden Fall geheim stattfinden muss. Bekannte Beispiele sind der DES und AES.

### 2.4.2 Asymmetrische Verschlüsselung

Bei einer asymmetrischen Verschlüsselung hat man zum Verschlüsseln einen anderen Schlüssel wie zum Entschlüsseln. Dieses System wird „Public-Key-Kryptographie“ genannt, da es einen öffentlichen ( $k_1$ ) und einen privaten Schlüssel ( $k_2$ ) gibt. Dabei wird  $k_1$  zum Verschlüsseln verwendet und  $k_2$  zum Entschlüsseln.

$$E_{k_1}(M) = C$$

$$D_{k_2}(C) = M$$

[Ert01]

Dieses System löst das Problem der Schlüsselübergabe, da der öffentliche Schlüssel ohne Bedenken an den Kommunikationspartner übertragen werden kann. Bei einem möglichen Angriff kann der Angreifer mit dem Schlüssel nichts anfangen, da er mit ihm nicht entschlüsseln kann. Nur der private Schlüssel, der geheim bleibt und nicht versendet wird, kann die entschlüsselte Nachricht dechiffrieren.

Diese Art von Algorithmus kann so auch zur Authentifizierung eingesetzt werden. Bekannte asymmetrische Verschlüsselungen sind der RSA-Algorithmus, der Algorithmus von Diffi und Hellmann oder der Algorithmus von ElGamal. [Ert01]

### 2.4.3 „Shamir’s Secret Sharing” Methode

Die Shamir’s Secret Sharing Methode ist ein mathematisches Verfahren, das es möglich macht, ein Geheimnis auf mehrere Instanzen aufzuteilen. Dabei sind das Geheimnis und die dabei entstehenden „Shares” Integer-Werte.

Die einzelnen Instanzen lassen dabei keine Rückschlüsse auf das Geheimnis zu. Allein wenn man den Großteil der „Shares” besitzt, kann man das Geheimnis rekonstruieren.

Der folgende Abschnitt basiert auf dieser Literatur: [LT10]

Verschlüsseln:

Verwendet wird ein Geheimnis  $d$ , eine Anzahl von  $n$  Instanzen und eine Schwelle  $k < n$ .

1. Es wird eine Primzahl  $p$  zufällig gewählt.
2. Es werden  $k-1$  Werte  $c_1, c_2, \dots, c_{k-1}$  mit den Werten zwischen 0 und  $p-1$  vergeben.
3. Für  $x_1, x_2, \dots, x_n$  jeweils eine eindeutige reale Zahl wählen.
4. Mit Hilfe einer Polynomgleichung mit dem Grad  $k-1$  werden nun  $n$  Gleichungen und somit auch Werte berechnet.  $i = 1, 2, \dots, n$

$$F(x_i) = (d + c_1x_i + c_2x_i^2 + \dots + c_{k-1}x_i^{k-1}) \bmod(p)$$

5. Nun können die  $n$  Shares erstellt werden. Diese sind wie folgt aufgebaut:  $(x_i, F(x_i))$

Das Geheimnis ist nun in einer Funktion „abgespeichert”. Die Shares sind Punkte, die auf dieser Funktion liegen. Damit die Funktion wieder rekonstruiert werden kann, müssen mindestens  $k$  der  $n$  Shares vorhanden sein.

Entschlüsseln:

Um die Nachrichten zu entschlüsseln müssen  $k$  Shares in die Formel oben eingesetzt werden.

Das hierdurch entstandene Gleichungssystem mit den Unbekannten  $d$  und  $c_1$  bis  $c_{k-1}$ , kann zum Beispiel mit dem Gaußsches Eliminationsverfahren oder durch die Lagrangesche Interpolationsformel gelöst werden.

## 2.5 Information Hiding

### 2.5.1 Steganographie

Die Steganographie ist die Kunst vom verborgenen Schreiben. Je nachdem welche Literatur man verwendet, wird die Steganographie als Unterpunkt der Kryptographie oder als eine eigene Disziplin gesehen. In dieser Arbeit wird die Steganographie eigenständig betrachtet und als Alternative zur Kryptographie gesehen.

Beide, die Kryptographie und die Steganographie, sind Möglichkeiten Informationen geheim und von Dritten ungesehen zu übertragen. Wie bereits oben beschrieben, beschäftigt sich die Kryptographie mit dem verschlüsselten Schreiben. Die Steganographie hingegen benutzt keine Verschlüsselung, sondern versucht die geheime Information in einem unauffälligen oder legitimierten Informationskanal zu verstecken.

Wie von Peter Purgathofer [Pur10] beschrieben hat die Steganographie eine große Bedeutung in der Geschichte, denn die Menschen waren gerade in Kriegszeiten schon immer auf der Suche nach einem sicheren Weg, Informationen zu übertragen.

So hat zum Beispiel der griechische Spion Demaratos Wachstafeln dazu benutzt, um Informationen zu verschicken. Nur hat er diese nicht ins Wachs geschrieben, sondern in das darunterliegende Holz.

Ebenfalls soll Histiaeus, der Tyrann von Milet, seine geheimen Nachrichten auf die Schädel der Sklaven tätowiert haben. Die Haare wuchsen nach und die Nachricht war verborgen. Es gibt noch viele andere Beispiele, angefangen von unsichtbarer Tinte bis hin zu Morsezeichen in Gemälden. Vor allem Künstlern wurde oft vorgeworfen, mit Hilfe von Steganographie geheime Nachrichten zu verbreiten. So wurde zum Beispiel Mozart immer wieder beschuldigt freibeuterische Nachrichten in der „Zauberflöte“ versteckt zu haben.

Die Beispiele der Geschichte zeigen deutlich wie die Steganographie funktioniert: Es gibt immer eine unauffällige Trägernachricht (Wachstafel, Sklave, Gemälde, Musikstück... ). In dieser Trägernachricht wird die geheime Nachricht versteckt (Unter Wachs oder Haaren, Blickwinkel auf das Gemälde, Notenreihenfolge...).

Um die Nachricht zu entschlüsseln benötigt der Empfänger die Information wo sich die

Nachricht befindet, beziehungsweise wie sie versteckt wurde. Das Schema von Gary C. Kessler [Kes15] macht dieses Prinzip sehr anschaulich:

*Steganographisches Medium = Geheime Nachricht + Träger Nachricht + Steganografischer Schlüssel*

Dabei darf der steganografische Schlüssel nicht mit dem aus der Kryptographie verwechselt werden. Es handelt sich hier mehr um das Wissen, wo und wie die geheime Nachricht verborgen ist.

Dabei bedient sich die Steganographie der „Security by Obscurity“ (Sicherheit durch Unwissenheit), was bedeutet, dass die Sicherheit allein davon abhängt, ob das Geheimhaltungsverfahren unbekannt bleibt. Im übrigen gehören kryptographische Verfahren die nicht unter Kerkhoffs Prinzip fallen auch zu „Security by Obscurity“. Will man also ein solches System sicherer machen, muss man dafür sorgen, dass das Verfahren so abwegig, beziehungsweise so obskur gestalten wird, sodass nie jemand auf die Idee kommt, nach einer geheimen Nachricht zu suchen.

Die Steganographie hat in der Geschichte eine relativ einfache aber sichere Methode geboten Nachrichten zu übertragen. Aber auch heute im Internet sind wir nahezu immer von Datenkanälen umgeben, die sich für die steganographische Datenübertragung eignen. Der Vorteil hierbei ist, dass meistens unter den ganzen kryptographisch verschlüsselten Datenpaketen die Steganographie vergessen wird.

**Im heutigen informationstechnischen Kontext spricht man von Steganographie, wenn geheime Informationen in Texte, Bilddateien, PDFs oder ähnlichem eingebettet und verschickt werden.**

Methoden diese Daten zu manipulieren werden im Kapitel „Lösungsideen“ vorgestellt.

### 2.5.2 Covert Channel

Die Covert Channels (Verdeckte Kanäle) haben wie die Steganografie die Aufgabe, Daten in legitimen Kanälen zu verstecken. Der Unterschied zwischen der Steganographie und den Covert Channels, liegt allein in der Art, wie diese Daten versteckt werden.

**Covert Channels nutzen die Kommunikationsattribute der Netzwerkpakete, um die geheimen Informationen einzubetten. [Wen12b]**

Zur Erstellung eines Covert Channel wird ein legitimer Datenkanal benötigt. Dort werden die Kommunikationsattribute so manipuliert, dass zusätzliche geheime Daten übertragen werden können. Kommunikationsattribute sind hier alle veränderbaren Parameter in den Netzwerkprotokollen. Hierzu gehören die Protokoll Header, der Zeitpunkt des Absendens aber auch die Größe der Nutzdaten.

Das Kapitel der Lösungsansätze beschäftigt sich damit, wie die Kommunikationsattribute manipuliert werden können.

Der Aufbau eines Covert Channel lässt sich durch folgendes Schema darstellen:

*Covert Channel = Geheime Daten + Trägerkanal + Manipulation der Kommunikationsattribute*

Bei den weit verbreiteten mathematischen Verschlüsselungen, muss sich meistens keine Sorgen gemacht werden, ob der Datenaustausch von Dritten entdeckt werden kann, da hier die Daten ohne den richtigen Key nutzlos sind.

Bei den Covert Channels ist dies problematischer, da ein entdeckter Kanal in der Regel direkt interpretiert werden kann. Ein Covert Channel lebt, wie der Name schon verrät, davon wie gut dieser versteckt ist.

Dennoch besteht ein großer Vorteil: Ist das Verfahren zum Verstecken der Daten nicht bekannt, so ist es nahezu unmöglich, den Covert Channel zu finden, da nicht klar ist, wo und nach was gesucht werden muss (Security by Obscurity).

Ist hingegen klar, um welches Verfahren es sich handelt und besteht die Vermutung, dass eine Kommunikation über einen versteckten Kanal stattfindet, so sind die Informationen nicht lange sicher.

Um einen Covert Channel zu Bewerten müssen folgende Aspekte betrachtet werden:

Der erste ist die Fähigkeit, wie einfach sich ein Kanal verstecken lässt. Hier fließt die allgemeine Unauffälligkeit des Covert Channels ein, aber auch die Eigenschaften des bereits herrschenden Netzwerkverkehrs, in den der Channel eingebettet werden soll.

Der zweite Aspekt ist die Unbekanntheit des Verfahrens, sodass nicht nach einem möglichen versteckten Kanal gesucht wird. Die individuelle Gestaltung des Channels verbessert stets

den Grad der Sicherheit.

Natürlich muss auch die Datenübertragungsrate betrachtet werden. Diese ist meistens sehr gering. Es existieren jedoch große Unterschiede zwischen den einzelnen Verfahren.

Die Integrität der Daten müssen die Channels ebenfalls gewährleisten.

### **Aktive und Passive Covert Channels**

Covert Channels können in zwei Kategorien eingeteilt werden: In aktive und passive Covert Channels.

Die aktive Variante generiert den zu Verändernden Netzwerkverkehr selbst und sendet diese mit veränderten Attributen zum Empfänger.

Bei passiven Kanälen schleust sich der Sender in eine bereits bestehende Kommunikation ein und manipuliert dort die Netzwerkattribute. Die Nachrichten werden nicht vom Sender erzeugt, sondern nur weitergeleitet. [Wen12b]

### 3 Anforderung an die Lösung

Es soll eine Lösung gefunden werden, bei der mittels Methoden des Information-Hidding Daten übertragen werden können. Dabei sollen beliebige Formate versendet werden können. Zudem soll er unauffällig sein und mit höchst möglicher Übertragungsgeschwindigkeit funktionieren. Die Lösungsidee muss sich als Algorithmus umsetzen lassen und soll möglichst nicht von Netzwerkbedingungen abhängig sein. Die Lösung darf sich nicht von Netzwerk Normalisierungen beeinflussen lassen.

Eine Methode zur Sicherstellung der Integrität soll implementierbar sein und eine Umsetzung als passiven Covert Channel ist wünschenswert.

## 4 Lösungsansätze

Im folgenden Kapitel werden verschiedene, bereits existierende steganographische Covert Channel betrachtet, die für das Erreichen der Zielsetzung in Frage kommen.

### 4.1 Covert Channel

#### 4.1.1 Zeitabhängige Covert Channel

Bei zeitabhängigen Covert Channel werden die Daten so versendet, dass der Absen-  
dezeitpunkt oder der Abstand zwischen den Paketen die geheime Information enthält.  
Dabei ähnelt dieses Verfahren dem in der Vergangenheit oft eingesetzten Morse Code und  
beschränkt sich meistens auf Binärdaten.

Bei dem in Abbildung 4.1 dargestellten Covert Channel wird mit einem festen Zeitintervall  
gearbeitet. Dieses Zeitintervall muss sowohl dem Sender, sowie dem Empfänger bekannt  
sein. Wird ein Datenpaket innerhalb des Zeitintervalls gesendet, wird dies als binäre 1  
interpretiert, falls kein Paket gesendet wird als 0. So lassen sich beliebige Daten übertragen.  
[CBS04]

Da die Datenübertragung sehr stark von der Netzwerkgeschwindigkeit abhängig ist, muss  
man zusätzlich ein Verfahren zur Sicherstellung der Integrität implementieren.

Durch das Versenden des Hashwertes, der über einen bestimmten Anteil der Nachricht  
gebildet wird, kann die Integrität garantiert werden.

Eine andere Methode, bei der die Integrität jedoch nicht vollständig garantiert, aber  
simpler umzusetzen ist, ist die Verwendung eines Paritätsbits. [CBS04]



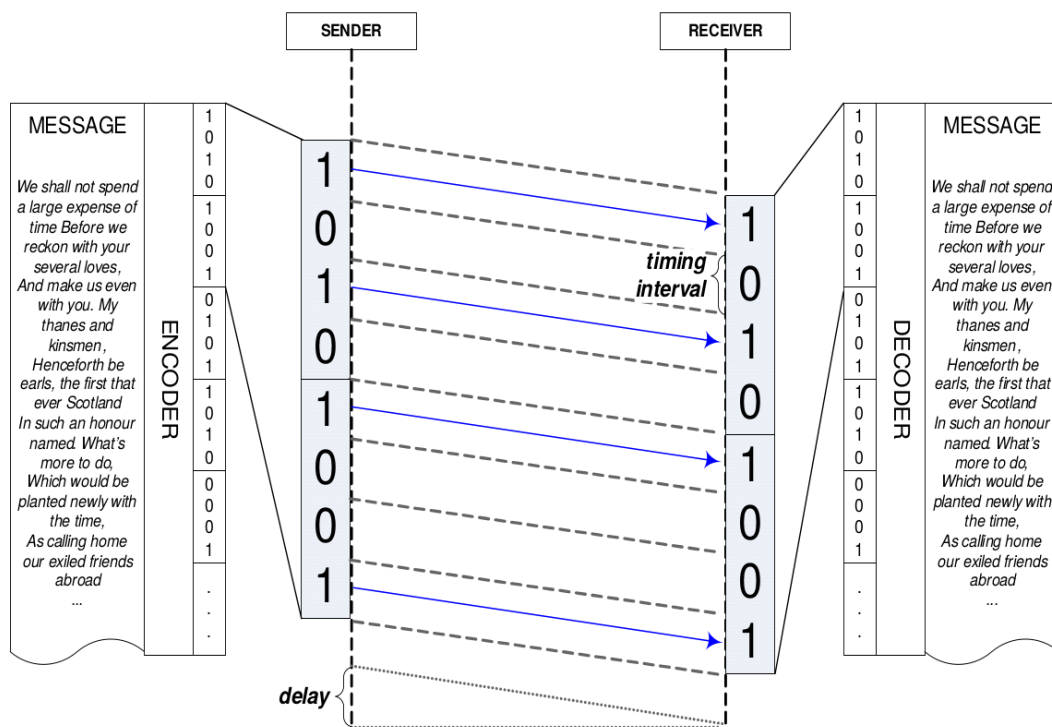


Bild 4.1: Kommunikation über einen Zeitabhängigen Kanal [CBS04]

Die Datenübertragung über diesen Channel ist mit einem Bit pro Paket relativ gering. Die Datenübertragung ist hier aber auch direkt vom verwendeten Zeitintervall abhängig. Deshalb gilt es, ein möglichst kleines Zeitintervall zu wählen und dieses optimal an die herrschenden Netzwerkbedingungen anzupassen.

Die Problematik besteht, dass mit der Reduzierung des Netzwerkintervalls die Fehleranfälligkeit ebenfalls zunimmt. Abbildung 4.2 zeigt, wie das gewählte Zeitintervall und die Genauigkeit zusammenhängen. So lässt sich bei einer langsamen Datenübertragung mit einem Intervall von 0.05 Sekunden eine Genauigkeit von ca. 100 Prozent garantieren. Diese Zahlen sind jedoch sehr stark vom jeweiligen Netzwerk abhängig. Der Ersteller dieser Grafik [CBS04] hat ebenfalls einen Wert  $k$  in seine Implementierung eingebaut, der die Länge einer Pause angibt, die bei einer Übertragungsverzögerung eingelegt werden kann. Diese verbessert zwar die Genauigkeit der Datenübertragung, die Geschwindigkeit wird aber erheblich reduziert.

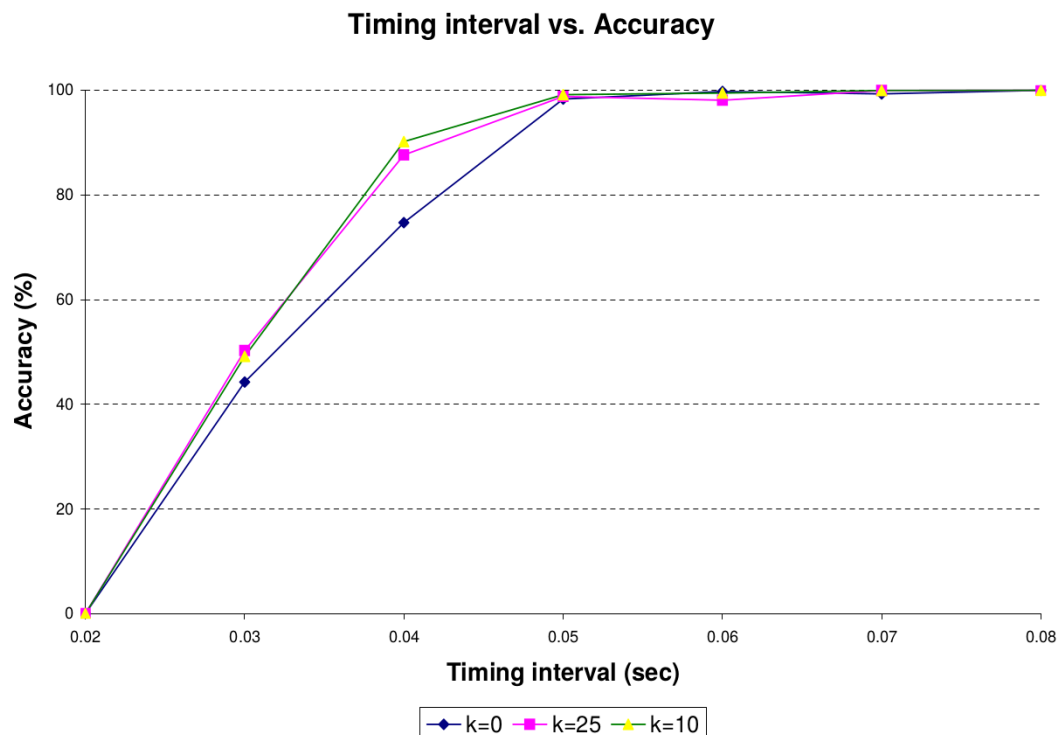


Bild 4.2: Zusammenhang zwischen Zeitintervall und Genauigkeit [CBS04]

Beim zeitabhängigen Covert Channel kann jedes beliebige Protokoll eingesetzt werden. Es bietet sich an, die Protokolle auf den umliegenden Netzwerkverkehr anzupassen, um ihn so unauffällig wie möglich zu machen.

### 4.1.2 Storage Channel

Bei Storage Channels benutzt man Speicherattribute [Wen12b] in Protokollheadern, um unbemerkt Daten zu übertragen. Da in dieser Arbeit über Netzwerkgrenzen hinaus kommuniziert werden soll, wird hier mit dem Internet Protokoll begonnen.

## IPv4

Der IPv4 Header bietet einige Möglichkeiten Daten in Speicherattribute (vergleiche Kapitel Grundlagen) zu verstecken. Folgend werden die Attribute begutachtet, die Potential zur Manipulation bietet.

### **Type of Service:**

Die letzten beiden Bits dieses Feldes sind unbenutzt und können so zur Datenübertragung verwendet werden. (2 Bits)

### **Identification:**

Bietet 16 Bits die theoretisch frei wählbar sind. (16 Bits)

### **Reserved Flag:**

Das erste Bit der Flags ist für zukünftige Benutzung reserviert und ist derzeit noch unbenutzt. (1 Bit)

### **Fragment Offset:**

Der Fragment Offset wird dazu verwendet, um Pakete nach einer Fragmentierung wieder zusammenzusetzen. Geht man davon aus, dass die Paket Fragmente sich frei konfigurieren lassen bietet sich die Chance 13 Bit zu verwenden. In Realität liegt die Nutzbarkeit der Bits deutlich tiefer. ( $< 13$  Bit)

### **Time to Live:**

Dieses 8 Bit Feld lässt sich frei wählen. Jedoch muss man bei der Benutzung wissen, wie viele Netzwerkstationen, die dieses Feld herunterzählt, auf dem Weg liegen. Ein zu kleiner Wert kann dazu führen, dass die Nachricht nicht ankommt. ( $< 8$  Bit)

### **Total Length:**

Die Gesamtlänge des Pakets lässt sich auch manipulieren. Diese Länge wird mit einem 16 Bit Wert angegeben. Jedoch ist dieser Wert durch die Mindestgröße eingeschränkt. Durch Fragmentierung des Pakets oder das Hinzufügen von Optionen ändert sich dieser Wert. ( $< 16$  Bit)

**Options:**

Dem Ip-Header können Optionen hinzugefügt werden, in die sich ebenfalls Daten einbetten lassen.

**Padding:**

Die durch das IHL Feld angegebene Headerlänge muss ein Vielfaches von 4 Byte erreicht werden. Durch die Verwendung von Options wird diese Länge nicht immer erreicht und wird deshalb mit Padding aufgefüllt. Dieses Padding lässt sich theoretisch auch umwandeln und zur Datenübertragung verwenden.

### 4.1.3 IPv6

Bei IPv6 kann man in der Regel die äquivalenten Speicherattribute verwenden, wie bei IPv4. Hier unterscheidet sich meistens nur die Namensgebung. [Wen12b]

### 4.1.4 TCP

Im TCP Protokoll bieten sich ebenfalls Möglichkeiten Daten unbemerkt zu transportieren. So kann der Source Port zur Codierung verwendet werden. Bei der Benutzung des optionalen TCP Timestamps, können die letzten Bits manipuliert werden. [Wen12b]

Eine weitere Möglichkeit ist das Manipulieren der Sequence Number. [Wen12b] Diese Nummer gibt die Reihenfolge der Datenpakete an. Dabei wird sie am Anfang der Datenübertragung vom Sender errechnet und anschließend alle 4 Mikrosekunden um eins hochgezählt. Sollte diese 32-Bit-Nummer überlaufen, so wird sie wieder auf Null zurückgesetzt. So wird sichergestellt, dass jedes Paket eine einzigartige Sequence Number bekommt [Inf81]

Um diese Nummer zu verändern, muss eine Übersetzungsschicht eingebaut werden, die die übertragenen Geheimdaten abfängt und wieder mit der richtigen Sequence Number ersetzt. [Wen12b]

### 4.1.5 Traffic Normalizers

Storage Channels haben einen großen Schwachpunkt: Traffic Normalizer schreiben die oben beschriebenen Headerattribute gezielt um, oder werfen diese, wenn auffällige Werte gesetzt sind.

Ein Traffic Normalizer auf IP Ebene könnte zum Beispiel das TTL Feld manipulieren, die Flags „Don't Fragment“ und „Reserved“ auf 0 setzen, die Options löschen, oder Pakete bei denen das IHL Feld größer als 5 ist werfen. [Wen12a]

Das Padding und die ungenutzten Bits des „Type of Service“ Feldes auf 0 können gesetzt werden.

Ein solches System, in ähnlicher Weise auf alle Netzwerkschichten angewendet, ist eine sehr effektive Methode um gegen Storage Channels vorzugehen.

### 4.1.6 Benutzung verschiedener Protokolle

Covert Channels können durch die Verwendung verschiedener Protokolle realisiert werden. Hier kann man zwischen Protocol Hopping Covert Channels und Protocol Channels unterscheiden. [Wen12b]

#### Protocol Channels

Beim Protocol Channel werden die Daten mit Hilfe mehrerer Protokolle kodiert. Eine mögliche Kodierung könnte folgende sein:

HTTP -> 00	DNS -> 01
ICMP -> 10	POP3 -> 11

[Wen12b]

So ist man in der Lage binäre Daten mit vier Protokollen zu versenden. In Abbildung 4.3 ist dieses Prinzip veranschaulicht.

Die Wahl der Protokolle ist hier abhängig von den im Netzwerk verwendeten Protokollen. Natürlich funktioniert dieses Prinzip auch mit zwei Protokollen. Denkbar wäre hier die

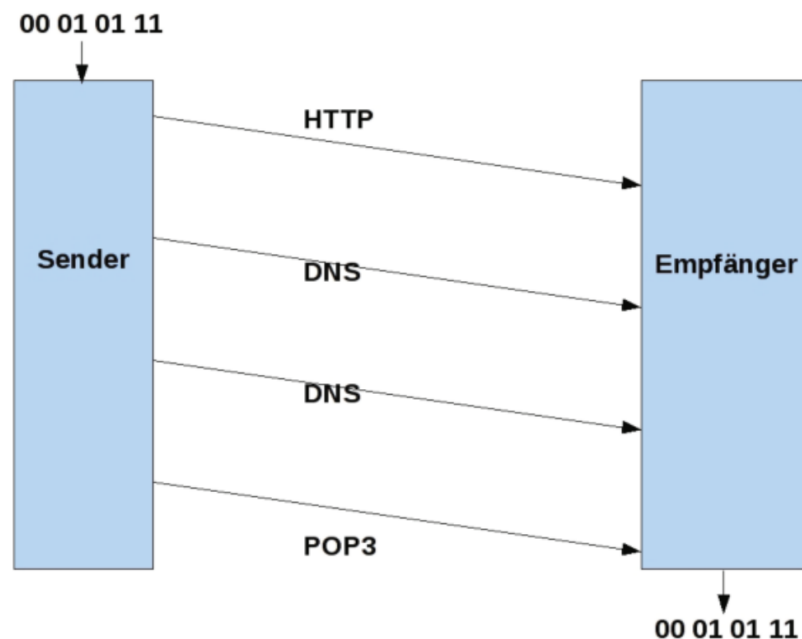


Bild 4.3: Protokol Channel [Wen12b]

Verwendung von IPv4 und IPv6, da diese Protokolle unter Umständen unterschiedliche Wege durchs Internet nehmen und so noch unauffälliger werden. Wie auch bei den zeitabhängigen Covert Channel beträgt hier die Übertragungsrate 1 oder 2 Bit pro Paket. Die Übertragungsgeschwindigkeit wird durch die Netzwerkgeschwindigkeit eingeschränkt.

Eine denkbare Unterart dieses Kanals ist die Verwendung verschiedener Options im Header, wodurch die Codierung realisiert wird.

### Protocol Hopping Covert Channel

Dieser Kanal ist eine Mischung des Protocol Channel und des Storage Channel. Dabei werden verschiedene Storage-Channels zu einem zusammengefasst und abwechselnd Daten übertragen.

In Abbildung 4.4 wird dargestellt, wie dies realisiert werden kann. Die Binärdaten werden

hier über zufällig gewählte Storage Channel übertragen. Dadurch wird bewirkt, dass der Kanal unauffälliger wird, da sich ein reales Netzwerk simulieren lässt.

Im Beispiel unten werden jeweils 4 Bit an den Storage Channel übergeben und an den Empfänger weitergeleitet.

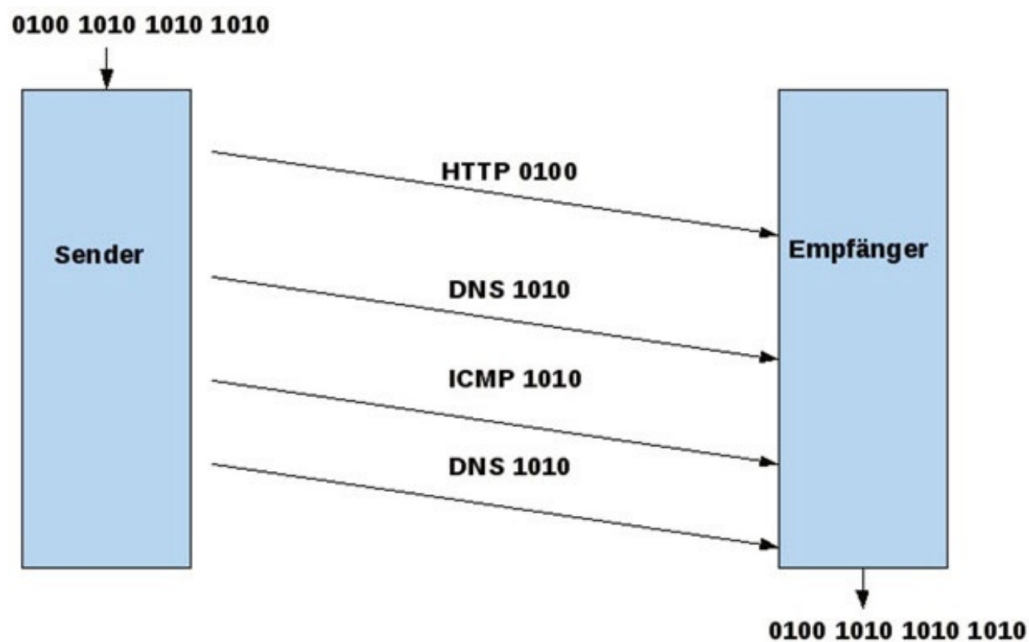


Bild 4.4: Protokoll Hoping Covert Channel [Wen12b]

#### 4.1.7 Verwendung der Nutzdatengröße

Die Größe des Pakets lässt sich über die Versendung unterschiedlicher Daten einfach manipulieren. Daraus ergeben sich etliche Varianten, um einen Covert Channel zu erschaffen. Beispielsweise kann zwischen großen/kleinen Paketen oder gerade/ungerade Datenanzahl unterschieden und so binär Daten übertragen werden.

Aber auch direkt in der Größe der Nutzdaten können Informationen versteckt werden: Will man 8 Bit pro Paket übertragen, benötigt man die Werte von 0 bis 255. Die Größe der Nutzdaten muss so angepasst werden, dass sie jeweils den zu übertragenden Werten entspricht. Da Nutzdaten von null oder einem Byte relativ selten sind, empfiehlt sich die

Verwendung eines statischen oder flexiblen Offsets, der addiert wird, um die Paketgröße anzuheben.

Dieser Kanal lässt sich auf alle Protokolle, die zur Datenübertragung fähig sind, anwenden.

## 4.2 Steganographie

### 4.2.1 Klassische textbasierende Verfahren

Botschaften in Texten teilweise einzubetten ist mit der Steganographie möglich. Gängig unter Textmanipulatoren ist die gezielte Wahl des ersten Buchstaben des Wortes, wobei die Aneinanderreihung dieser Buchstaben ein neues Wort ergibt. Bei einem wissenschaftlich erarbeiteten Rückblick treten einige, nicht zu unterschätzende, Sicherheitslücken auf, wenn diese Art der Verschlüsselung angewendet wird.

(Im oberen Text ist zur Veranschaulichung eine Nachricht an einen potentiellen Prüfer eingebettet. Nach jedem sinnvollen deutschem Wort muss mit dem nächsten Satz begonnen werden)

Eine weitere Methode ist, die Satzzeichen zu verwenden, um Informationen zu kodieren. So kann zum Beispiel ein Punkt 00, ein Komma 01, ein Fragezeichen 10 und ein Ausrufezeichen 11 bedeuten. [LC17]

Durch diese Verfahren lassen sich versteckte Datenkanäle konstruieren indem ein solcher Text beispielsweise per E-Mail versendet wird.

### 4.2.2 Basierend auf RGB Bilder

Bildformate, die auf dem RGB Formaten basieren, sind zum Beispiel BMP (Windows Bitmap) oder auch GIF (Graphics Interchange Format). Diese Formate zeigen die Pixelfarbe basierend auf dem **R**ot-, **G**rün- und **B**lauwert. In diesen Werten können Daten versteckt werden. [KP00] So können beispielsweise die letzten beiden Bits manipuliert werden. Diese



Veränderung ist für das menschliche Auge nicht zu erkennen, da die letzten Bits kaum eine Auswirkung auf die Höhe der Zahl haben.

Bei einer Farbtiefe von 24 Bit beträgt die maximale Änderung der Farbwerte nur 3 Farbstufen von insgesamt 256 möglichen.

00000000 (0) -> 00000011 (3)

11111111 (255) -> 11111100 (252)

Auf diese Weise lassen sich 6 Bit pro Pixel übertragen. In einem unkomprimierten HD Bild mit einer Auflösung von 1280x720 Pixeln (ca. 22 MByte) lassen sich 0,6912 Mbyte Daten übertragen.

In Abbildung 4.5 wird dieses System veranschaulicht:

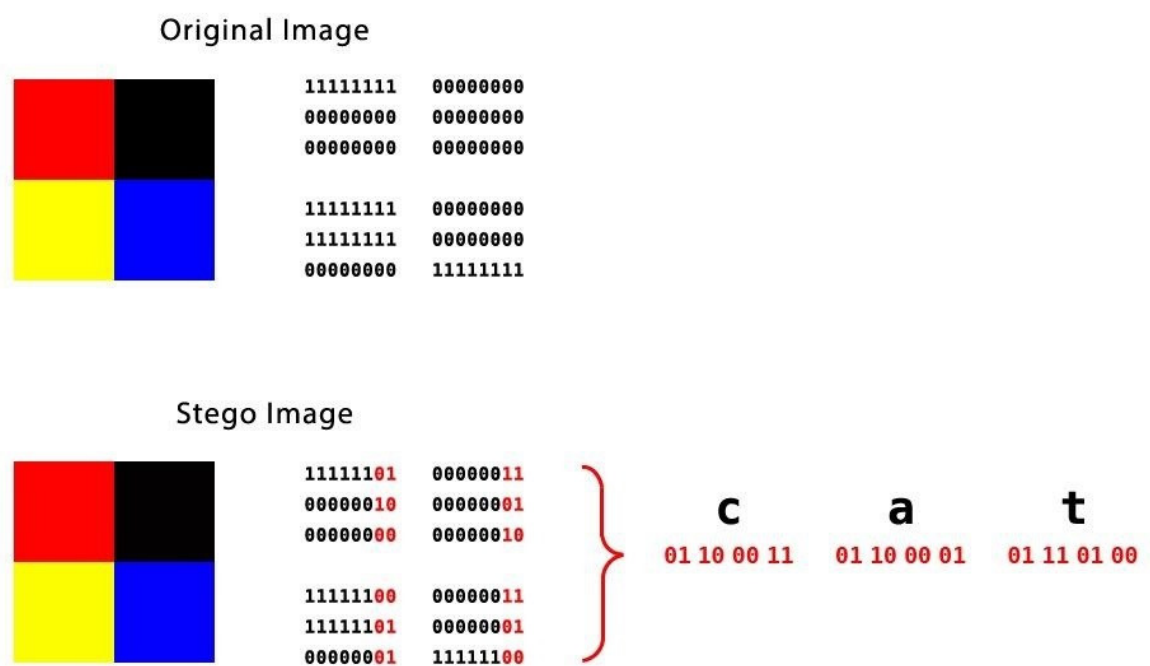


Bild 4.5: Codierung in den 2 letzten Bits [Use18]

### 4.2.3 Bildformate mit Alpha Kanal

Bildformate wie PNG (Portable Network Graphics) können zusätzlich zu den RGB Kanälen auch einen Alpha Kanal besitzen. Dies ist ein zusätzlicher Wert der die Transparenz des jeweiligen Pixel angibt. Bei einem Wert von 0 ist der Pixel „unsichtbar“ und bei 255 ist der Pixel komplett sichtbar. Hier könnte man die Daten wie oben in den letzten beiden Werten speichern.

Da die Transparenz keine direkte Auswirkung auf die Farbe der Pixel hat, kann man unbemerkt auch mehr Daten speichern.

#### Verwendung von „Shamir’s Secret Sharing“ Methode

Das in Kapitel 2.4.3 beschriebene Verfahren kann man dazu verwenden, um Nachrichten unauffällig in den Alpha Kanal eines PNG Bildes zu integrieren.

Die zu übertragende Nachricht  $M$  wird hierzu in Segmente von  $t$  Bit, mit  $t = 3$  unterteilt. Bei der Umwandlung der Segmente in Dezimalzahlen entsteht so ein neues Array  $M' = d_1, d_2, \dots$  bei dem die Werte zwischen 0 und 7 liegen.

Im Gegensatz zu dem original Algorithmus greift man hier zusätzlich auf die Werte  $c_1, c_2$  und  $c_3$  zurück, um hier ebenfalls ein „Geheimnis“ einzubetten. Zusammen mit  $d$  können nun  $k$  Werte integriert werden.

So ergibt sich:

$$d = m_1, c_1 = m_2, c_2 = m_3, c_3 = m_4$$

Folgende Werte werden definiert:

$$p = 11$$

$$x_1 = 1, x_2 = 2, x_3 = 3, x_4 = 4$$

$x$  kann hier eine beliebige Zahl annehmen und so als eine Art Verschlüsselung dienen. Fraglich ist, ob es sich dann noch um reine Steganographie handelt.

Die Werte von  $q_1$  bis  $q_4$  ergeben sich dann durch Einsetzen in die Polynomgleichung.

$$q_1 = F(x_1) = (m_1 + m_2x_1 + m_3x_1^2 + m_4x_1^3) \bmod(p)$$

$$q_2 = F(x_2) = (m_1 + m_2x_2 + m_3x_2^2 + m_4x_2^3) \bmod(p)$$

$$q_3 = F(x_3) = (m_1 + m_2x_3 + m_3x_3^2 + m_4x_3^3) \bmod(p)$$

$$q_4 = F(x_4) = (m_1 + m_2x_4 + m_3x_4^2 + m_4x_4^3) \bmod(p)$$

Die Werte  $q_1$  bis  $q_4$  können nun in den Alpha Kanal eines PNG Bildes eingefügt werden. Durch die Modulo Funktion entstehen so Werte zwischen 0 und 10. Damit das Bild eine möglichst niedrige Transparenz bekommt, muss der alpha Wert so groß wie möglich sein. Deshalb wird zu  $q$  jeweils der Wert 245 addiert, um so nah wie möglich an 255 zu kommen. Die neu entstandenen Werte  $q'_1$  bis  $q'_4$  kann man nun in den Alpha Kanal einfügen und über eine sensible Datenverbindung übertragen.

Zum Entschlüsseln muss man nun wieder 245 von den Werten des Alpha Kanal subtrahieren. Durch das Erstellen und Lösen des Gleichungssystems, mit den oben gezeigten Formeln, können die Werte  $m_1$  bis  $m_4$  wieder berechnet werden.

[LT10]

#### 4.2.4 Verwendung von PDF Dateien

Auch in PDF Dateien können Daten versteckt eingebettet werden. Das PDF Format setzt sich aus einer Reihe von Befehlen zusammen, in der die Formatierung der Seite angegeben wird. So können Elemente zur Positionierung von Texten eingesetzt werden, um Informationen einzubetten. [ZCC07] Open Source Programme machen es für jeden möglich auf diese Weise Daten zu verstecken.

## 5 Bewertung der Lösungsansätze

In diesem Kapitel wird sich damit beschäftigt, welcher der im vorhergehenden Kapitel vorgestellten Covert Channel für die Problemstellung geeignet ist. Es soll nun der optimale Kanal gefunden werden, der das Problem der Kommunikation zwischen Polizeipräsidium und Informant lösen kann.

Betrachtet man die **Textbasierenden Verfahren** etwas genauer wird schnell klar, dass diese nicht für größere Datenmengen geeignet sind und sich auch sehr schwierig als Algorithmen darstellen lassen.

Die **Zeitabhängigen Verfahren** sind sehr unauffällig, da die Datenpakete nicht direkt manipuliert werden müssen. Es muss sich jedoch um die Integrität der Daten gekümmert werden, da der Kanal sehr stark von den Netzwerkbedingungen abhängt. Die Übertragungsgeschwindigkeit ist mit einem Bit pro Paket sehr gering. Jedoch lässt sich dieser Kanal sehr gut als passiver Covert Channel realisieren.

**Storage Channels** sind relativ auffällig, vor allem wenn die Pakete genauer angeschaut werden. Zudem gibt es das Problem der Netzwerk Normalisierung, die den Storage Channel stark einschränken würde. Die Methode, bei der die TCP Sequence Number manipuliert wird, ist hingegen für dieses Projekt denkbar, da sie nicht durch die Normalisierung verändert werden kann und pro Paket 32 Bit übertragen kann. Bei einer sehr genauen Analyse kann hier aber auffallen, dass die Nummern nicht in der richtigen Reihenfolge versendet werden.

**Protocol Channels** sind im Gegensatz zu zeitabhängigen Verfahren nicht von den Netzwerkbedingungen abhängig und machen so ein Verfahren gegen Integritätsverlust überflüssig.

Es ist schwierig einen realen Netzwerkkanal zu realisieren, da die Reihenfolge der Pakete/Protokolle zufällig ist. Dies ist bei realen Netzwerken nicht der Fall. So kommen zum Beispiel DNS Anfragen viel seltener vor als TCP Pakete.

Die Veränderung der Pakete ist bei diesem Covert Channel nicht nötig. Eine Implementierung als passiver Channel ist aber nicht möglich, da von einem beliebigen Server nicht die benötigte Vielzahl an Protokollen geliefert wird.

**Projekt Hopping Channels** haben die gleichen negativen Eigenschaften wie die Storage Channels und kommen deshalb nicht für dieses Projekt in Frage.

Die Verwendung der **Nutzdatengröße** für die Datenübertragung ist sehr unauffällig. Die Pakete bleiben bis auf die Nutzdaten regulär und ziehen so fast keine Aufmerksamkeit auf sich. Der Kanal ist nicht von Netzwerkbedingungen abhängig und die Datenübertragung kann 8 Bit pro Paket betragen. Es muss jedoch eine unauffällige Methode ausgearbeitet werden, bei der es möglich ist, variable Datenpakete zu versenden. Es ist hier ebenfalls nicht möglich, den Kanal passiv zu gestalten.

Werden **Bilddateien** zum Einbetten der Daten verwendet, muss sich überlegt werden, wie die Übertragung der Bilder stattfinden soll. Hier muss eine unauffällige Möglichkeit gefunden werden, um diesen Austausch zu realisieren. Hier lassen sich viele Daten auf einmal übertagen. Hingegen ist diese Methode vor allem durch die Medien sehr bekannt geworden. Ein aktuelles Beispiel ist eine Sicherheitslücke in Android, indem durch das Öffnen eines PNG Bildes, Schadcode mit den Rechten des Benutzers ausgeführt werden kann. [ De19].

Da der zeitabhängige Covert-Channel die Pakete nicht manipulieren muss und beim Netzwerkverkehr, bis auf den zeitlichen Offset, keine Veränderung vorgenommen werden muss, soll dieses Projekt mit diesem Kanal durchgeführt werden. Zudem ist die Möglichkeit gegeben, den Channel passiv zu realisieren. Die allgemeine Unbekanntheit des Channels ist ein weiterer positiver Aspekt.

## 6 Konstruktion und Aufbau des Covert Channels

### 6.1 Konstruktion des Covert Channels

Wie in den Grundlagen schon beschrieben, kann man einen Covert Channel mit folgender Formel veranschaulichen:

*Covert Channel = Geheime Daten + Trägerkanal + Schlüssel (Manipulation der Kommunikationsattribute)*

#### 6.1.1 Geheime Daten

Die geheimen Daten beinhalten die Informationen, welche versteckt übertragen werden. Diese sollen jedes beliebige Format annehmen können. Zum Senden werden die Daten in ihre binäre Form übertragen. So muss man sich keine Sorge um das Datenformat machen. Da eine sehr geringe Übertragungsgeschwindigkeit erwartet wird, ist es hilfreich, wenn diese Daten so klein wie möglich ausfallen.

Zum Entwickeln wird hierzu die Datei *test.txt* verwendet, die als Inhalt den String „Hallo Welt“ besitzt.

### 6.1.2 Schlüssel

Der Schlüssel bildet sich aus dem, im vorhergehenden Kapitel gewählten, Covert Channel. Er bestimmt auf welche Art die Daten in den Träger infiltriert und später exfiltriert werden.

Aufgrund der Wahl des zeitabhängigen Covert Channel können folgende Schlüssel definiert werden.

Schlüssel zur Dateninfiltration:

*Manipuliere den Datenstrom des Trägerkanals so, dass die geheimen Daten kodiert werden.*

Schlüssel zur Datenexfiltration:

*Lese die Zeitabstände der Datenpakete im Trägerkanal und dekodiere diese.*

### 6.1.3 Trägerkanal

Es wird ein Trägerkanal benötigt in diesen die geheimen Nachrichten eingebettet werden sollen.

Für den gewählten Covert Channel kann prinzipiell jedes Netzwerkprotokoll verwendet werden. Da die Übertragung über Netzwerkgrenzen hinaus stattfinden soll muss jedoch mindestens das Internet Protokoll verwendet werden. Die Verwendung von ICMP Paketen ist durch die Filterung von Firewalls nicht geeignet. Die verbleibenden und sinnvollen Möglichkeiten sind demnach entweder TCP oder UDP.

Da immer nur ein Bit durch ein Datenpaket übertragen wird, wird ein Datenstrom (Stream) mit vielen Datenpaketen benötigt.

Die Entscheidung beschränkt sich nun auf einen UDP oder TCP Stream. UDP hat das Problem, dass es sich hierbei um ein verbindungsloses Protokoll handelt und man nicht sicher sein kann, dass die Daten in der richtigen Reihenfolge oder überhaupt ankommen. Durch die Verwendung eines zeitlichen Covert Channel, muss aber sowieso ein System zur Sicherstellung der Integrität implementiert werden.

Auf der anderen Seite sind UDP Pakete selten geworden, da HTTP und somit die Webserver auf TCP aufbauen. Auch die Streams von Firmen wie Netflix oder YouTube basieren heute alle auf dem TCP/IP Stack.

So ist die Verwendung eines TCP Streams die beste Lösung, wobei sich die verbindungsorientierte Datenübertragung von TCP zusätzlich positiv auf die Zuverlässigkeit des Datenkanals auswirkt wird.

## 6.2 Aufbau des Systems

In diesem Kapitel sollen die einzelnen Bausteine, die im vorhergehenden Abschnitt beschrieben wurden, zu einem kompletten System zusammengefügt werden. Der Aufbau dieses Systems dient später als Struktur bei der Programmierung.

Im Allgemeinen können hier zwei Systeme entstehen, entweder ein System mit einem aktiven oder passiven Covert Channel.

### 6.2.1 Anforderungen an den Aufbau

Der Aufbau soll eine Grundlage bieten, um die Funktion eines zeitabhängigen Covert Channel zu beweisen. Es muss die volle Kontrolle über die Netzwerkpakete möglich sein. Die Benutzung von Netzwerkpaketen, von beliebigen Webservern, soll angestrebt werden.

### 6.2.2 Lösungsansatz 1: Aktiver Aufbau

Soll ein aktiver Covert Channel erstellt werden, so ist der Sender gleichzeitig als Server realisiert. Dieser sendet aktiv Datenpakete an den Empfänger. Durch die Codierung der geheimen Nachricht in die Zeitabstände zwischen den Paketen, gelangt die Information zum Empfänger. In *Abbildung 6.1* wird dieses System vereinfacht dargestellt.



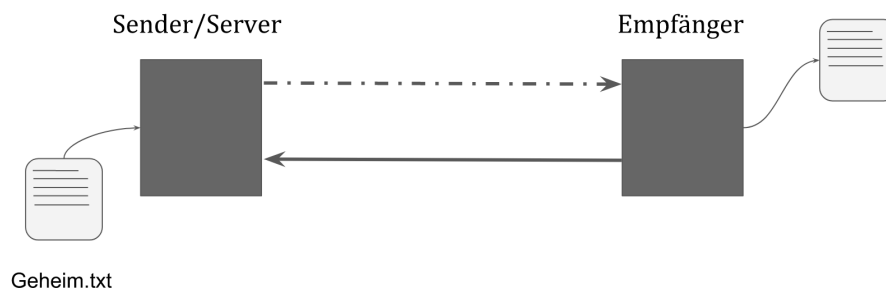


Bild 6.1: Aufbau eines aktiven Systems

### 6.2.3 Lösungsansatz 2: Passiver Aufbau

Bei der passiven Alternative (Abbildung 6.2) verbindet sich der Empfänger über einen Proxy mit einem beliebigen Webserver, der einen konstanten Datenstrom generiert. Dies könnte zum Beispiel ein Video- oder Audiostream sein.

Dieser Datenstrom läuft über den Proxy, der nun die Möglichkeit hat, den Datenstrom zu manipulieren. Der Sender der geheimen Nachrichten nimmt in diesem System die Rolle des Proxys ein. Er muss die Daten nicht verändern, sondern nur verzögern.

### 6.2.4 Bewertung des Aufbaus

Die aktive Variante bietet eine gute Grundlage, um die Methode eines zeitabhängigen Covert Channel zu beweisen und die Funktion zu testen. Außerdem hat man bei dieser Variante die volle Kontrolle über die versendeten Datenpakete und kann so wichtige Parameter genau bestimmen.

Wird die passive Umsetzung realisiert, bekommt man als Ergebnis eine flexible Lösung die jede Webseite nutzen kann. Da große Teile des aktiven Lösungsansatzes beim passiven

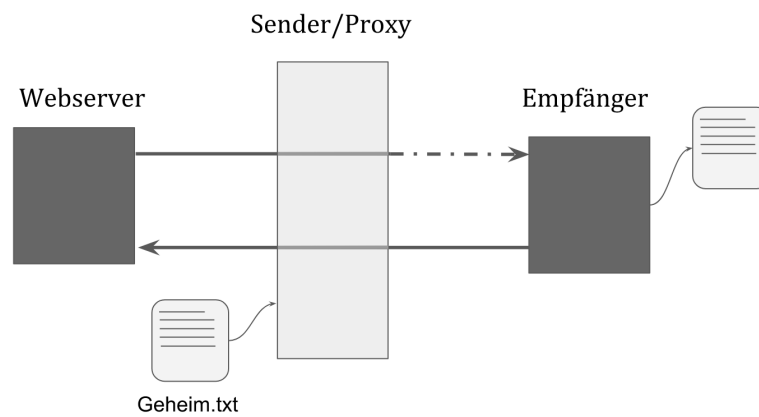


Bild 6.2: Aufbau eines passiven Systems

wiederverwendet werden können, soll in diesem Projekt zunächst der aktive und als Erweiterung der passive Covert Channel realisiert werden.

## 7 Umsetzung der aktiven Variante

### 7.1 Fehlerkorrektur

#### 7.1.1 Anforderung an die Fehlerkorrektur

Eine Fehlerkorrektur soll in der Lage sein in 8 oder 16 Bit Worten fehlerhafte Bits zu finden und diese zu korrigieren. Die extra Bits zur Fehlerkorrektur sollen direkt an die Datenworte angehängt werden, dürfen aber maximal 10% des Datenwortes betragen.

#### 7.1.2 Lösungsansatz 1: Paritätsbits

Die Fehlerkorrektur kann mit einem Paritätsbit realisiert werden. Dieses Bit gibt an, ob die Anzahl der Einsen in einer binären Zahlenfolge eine gerade oder ungerade Zahl ist. Wird bei der Datenübertragung ein Bit verändert, so ändert sich auch die Anzahl der Einsen und stimmt nicht mehr mit dem Paritätsbit überein.

Mit nur einem Paritätsbit lässt sich feststellen, dass ein Fehler aufgetreten ist. Der Fehlerort ist aber nicht bekannt. Um auch dies herauszufinden, kann der so genannte Hammingcode angewendet werden.

Bei dieser Codierungsform werden den Datenbits mehrere Paritätsbits hinzugefügt. Mit Hilfe dieser Bits kann dann ein Fehler in den Datenbits gefunden und gezielt korrigiert werden. [LMU17]

Die folgende Tabelle zeigt wie viele Paritätsbits zu den Datenbits hinzugefügt werden müssen, um ein falsches Bit in  $n$  Datenbits zu korrigieren.

Daten Bits:	8	16	32	64	128	
Paritäts-Bits:	4	5	6	7	8	[LMU17]
Codewort:	12	21	38	71	136	

Zum Verschlüsseln prüft jedes Paritätsbit mehrere genau festgelegte Bits des Codeworts. Das Paritätsbit wird dann so gesetzt, dass die Summe der geprüften Bits (sich selbst eingeschlossen) gerade ist.

Um einen Fehler zu erkennen, wird jetzt wieder die Summe ausgerechnet. Ist die Summe gerade, so ist kein Fehler aufgetreten und die Paritätsbits können aus dem Codewort gelöscht werden. Ist die Summe ungerade ist ein Fehler aufgetreten. Nun müssen alle fehlerhaften Paritätsbits ausfindig gemacht werden.

Bildet man nun die Schnittmenge aller, von nicht korrekten Paritätsbits geprüften, Bits und eliminiert alle Bits, die auch von korrekten Paritätsbits geprüft wurden, so bleibt das falsche Bit übrig. [LMU17]

Da die Datenübertragung über den Covert Channel auf Grund von Netzwerkschwankungen fehleranfällig ist, empfiehlt es sich den Hammingcode auf jeweils 8 Bits anzuwenden. Dies bedeutet pro Byte 4 zusätzliche Paritätsbits was ein Codewort von 12 Bits Länge ergibt. In diesem Codewort kann ein potentiell falsches Bit ausfindig gemacht und korrigiert werden.

### 7.1.3 Bewertung der Fehlerkorrektur

Da die Fehleranfälligkeit bei schlechten Netzwerkbedingungen und hoher Übertragungsgeschwindigkeit sehr hoch ist, sollte die Fehlerkorrektur auf einen möglichst kleinen Anteil von Bits angewendet werden. Dies hat den Nachteil, dass der Anteil von Paritätsbits bei Verwendung von 8 Bits 50% und bei 16 Bits 31% ausmachen würden.

Ein weiterer Nachteil ist, dass Fehler oft nacheinander auftreten, da sie von der gleichen Netzwerkverzögerung ausgelöst wurden. Mit dem Hammingcode kann bei mehreren Fehlern ein Fehler detektiert werden, jedoch ist eine Fehlerkorrektur nicht mehr möglich.

Bei einer Verwendung von großen Datenpaketen und sehr verstreut auftretenden Fehlern würde die Implementierung des Hamming-Codes Sinn machen, aber in dieser Arbeit wird aus den oben genannten Gründen darauf verzichtet.

## 7.2 Integrität

Um die Integrität der gesendeten Daten zu gewährleisten, soll am Ende der Daten ein Hashwert mitgesendet werden.

Um die Integrität zu garantieren, bildet der Sender den Hash über die zu sendende Nachricht. Die Nachricht und der Hashwert werden nun versendet. Der Empfänger bildet nun ebenfalls den Hash über die empfangene Nachricht. Stimmt dieser Hashwert mit dem überein, den er vom Sender bekommen hat, so ist die Nachricht unverändert versendet worden. Die Integrität ist sichergestellt.

Stimmt der Hash nicht überein, so müssen die Daten erneut gesendet werden. Es ist auch möglich, den Hash nicht erst am Ende der Datenübertragung zu senden, sondern nach einer festgelegten Datenmenge. Dies hätte den Vorteil, dass nicht die komplette Nachricht wiederholt werden muss.

### 7.2.1 Anforderung an die Hashfunktion

Die Hashfunktion soll eine Nachricht auf einen eindeutigen Hashwert abbilden. Er muss aber nicht zur Authentifizierung verwendet werden. Der Wert sollte so klein wie möglich sein, um die Datenmenge gering zu halten.

### 7.2.2 Lösungsansatz 1: MD5-Hash

Die MD5 Hashfunktion wurde 1991 als Weiterentwicklung der MD4 Hashfunktion veröffentlicht. Diese bildet eine beliebige Nachricht auf einen 128-Bit-Hashwert ab. Zu erwähnen ist, dass der MD5 aus Sicherheitsgründen nicht mehr empfohlen wird. [Wät18]

### 7.2.3 Lösungsansatz 2: Pearson-Hash

Der Pearson-Algorithmus verwendet eine zufällig initialisierte statische Mapping-Tabelle, um jedes Byte von jedem Hash-Wert auf einen neuen Hash-Wert abzubilden. [Dav10]

Mit diesem simplen Algorithmus lassen sich Hash-Werte mit der Länge von einem Byte erzeugen. Die Mapping-Taballe muss jedoch auf allen beteiligten Systemen gleich sein.

## 7.2.4 Bewertung der Hash-Funktionen

Beide Algorithmen verändern sich maßgeblich, wenn ein Bit geändert wird. Der md5 ist auf Grund seiner Länge aber auch komplexeren Algorithmus erheblich sicherer. In dieser Arbeit soll die Sicherheit jedoch allein von der Unauffälligkeit des Covert Channel abhängen. Die Hashfunktion soll nur Sicherstellen, dass die Nachricht unverändert beim Empfänger angekommen ist.

Dazu ist der Pearson Hash ebenfalls in der Lage und hat den Vorteil, dass er nur ein sechzehntel an Datenmenge in Anspruch nimmt.

Aus diesen Gründen wird in dieser Arbeit mit dem Pearson Hash gearbeitet.

## 7.2.5 Pearson Hash Implementierung

Um die Pearson Hash-Funktion auf die Daten anzuwenden, muss eine Mapping-Tabelle mit den Werten von 0-255 in zufälliger Reihenfolge generiert werden. Diese lässt sich beispielsweise mit einem Python Script unter Verwendung der *shuffel* Funktion erstellen. Die hier generierte Tabelle muss beim Server, sowie beim Client bekannt sein, um bei gleicher Hash-Funktion und gleichen Daten den identischen Hashwert zu generieren.

Den Algorithmus zur Berechnung des Hashs ist in Listing 7.1 gezeigt. Hier wird am Anfang ein Hash-Wert  $h$  generiert. Dieser wird bitweise XOR mit dem Datenwert verknüpft. Der entstandene Wert wird als Index in der Mapping-Tabelle verwendet. Der dortige Wert in der Mapping-Tabelle, wird der neue Wert  $h$ . Dies wird so lange wiederholt, bis alle Datenwerte verwendet worden sind.

```
for (var j = 0; j < hashLength; j++){
    var h = table[(parseInt(data.charAt(0)) + j) % 256];
    for (var i = 1; i < data.length; i++){
        h = tabel[(h ^ data[i])];           // XOR
    }
```

```
    hash[j] = h;  
}
```

Listing 7.1: Erstellen der Mapping-Table

Da hier mit einer *hashLength* von 1 gearbeitet wird, ist das Ergebnis ein 8 Bit-Wert, der nun an die Daten angehängt werden kann.

## 7.3 Authentizität

Die Authentizität wird durch den Pearson-Hash teilweise abgedeckt. Dies gilt aber nur solange die Mapping-Tabelle geheim bleibt. Ist ein Angreifer jedoch in besitz einer kompletten dekodierten Nachricht, kann er selbst mit Brute-Force in kürzester Zeit die Tabelle rekonstruieren.

Es könnte ein Public-Key Verfahren eingesetzt werden mit dem zum Beispiel der Hash-Wert verschlüsselt wird. Jedoch würde so eine Methode nicht zu einem auf *Security by obscurity* basierenden Verfahren passen. Auch steht es in Konfrontation mit der Anforderung eine alternative zu verschlüsselten Kommunikationen zu erstellen.

## 7.4 Netzwerkprotokoll

### 7.4.1 Anforderung an das Netzwerkprotokoll

Die Anforderungen hierfür wurden bereits zum Großteil bei der Wahl des Trägerkanals formuliert. Dabei wird die Nutzung eines TCP basierenden Protokolls festgelegt.

Zusätzlich kommt der Aspekt hinzu, dass sich mit diesem Protokoll legitim große Datenmengen verschicken lassen sollen, ohne Aufmerksamkeit zu erregen. Um der Philosophie von Covert Channels und der Steganographie gerecht zu werden, wird von einer Verschlüsselung abgesehen.

### 7.4.2 Lösungsansatz 1: HTTP

Das Hypertext Transfer Protocol (HTTP) ist ein Protokoll auf Anwendungsebene. Es ist ein generisches, zustandsloses Protokoll, das für viele Aufgaben verwendet werden kann. HTTP wird zur Versendung von Webseiten und Informationen seit dem Jahr 1990 verwendet. [FGM<sup>+</sup>99]

Auch heute stellt es, zusammen mit der verschlüsselten Variante HTTPS, einen elementaren Bestandteil des Internets dar. Dabei besteht die Hauptaufgabe darin, Daten von Webservern in den Browser zu laden.

### 7.4.3 Lösungsansatz 2: SMTP

Das Simple Mail Transfer Protocol (SMTP) hat die Aufgabe Mail zuverlässig und effizient weiterzuleiten. SMTP Nachrichten und die in ihnen enthaltenen Mails, werden von SMTP Servern entgegengenommen, um sie an den Empfänger weiterzuleiten. Auch die Kommunikation zwischen den SMTP Servern wird mit Hilfe dieses Protokolls realisiert. [Kle01]

### 7.4.4 Lösungsansatz 3: FTP

Mit dem File Transfer Protocol (FTP) können Dateien über ein Netzwerk zu einem Server hoch- und heruntergeladen werden. Ebenfalls ist es möglich das Dateisystem auszulesen und auf entfernten Rechnern Dateien zu erstellen, aber auch zu löschen. Die hierzu nötige Kommunikation, wird durch das FTP definiert. [PR85]

### 7.4.5 Bewertung der Netzwerkprotokolle

Alle Protokolle basieren auf TCP und verzichten auf eine Verschlüsselung. FTP wird heute in der Regeln aber nicht mehr angewendet, da es von Protokollen wie SFTP und SSH abgelöst wurde. Daher wäre eine Verwendung in dieser Arbeit nicht geeignet.

Auch bei SMTP wird heute vermehrt auf die SSL verschlüsselte Variante zurückgegriffen. Ein weiteres Problem bei SMTP ist, dass die Nachrichten bei großen Paketmengen als



Spam interpretiert werden können.

HTTP ist trotz der Einführung von HTTPS im Internet immer noch weit verbreitet und zieht sehr wenig Aufmerksamkeit auf sich. Zusätzlich lässt sich durch die Bereitstellung einer unauffälligen Webseite die wahre Aufgabe des Webservers - die geheime Datenübertragung - verbergen.

Aus diesen Gründen soll in dieser Arbeit ein HTTP Server realisiert werden, der als Sender der geheimen Daten dient.

## 7.5 Server

### 7.5.1 Anforderungen an den Server

Der Server muss in der Lage sein, einen HTTP Request entgegenzunehmen und im Gegenzug eine Webseite auszuliefern. Da die geheimen Daten vom Server an den Client übertragen werden sollen, muss der Server in der Lage sein, aktiv, ohne Requests, HTTP Nachrichten an den Client zu senden.

Zusätzlich ist es für diese Anwendung essentiell, dass sich das Senden der Nachrichten verzögern lässt, um die nötigen zeitlichen Abstände für den Covert Channel zu realisieren. Aus diesem Grund soll der Server frei programmierbar sein und auch Daten verarbeiten können, um beispielsweise Binärdaten zu erstellen, oder einen Hashwert zu generieren.

Der Server soll leichtgewichtig, einfach zu bedienen und auf einem Linux Betriebssystem lauffähig sein.

### 7.5.2 Lösungsansatz 1: Java HttpServer

HttpServer ist eine Java Klasse, die es ermöglicht, einen einfachen HTTP Server zu erstellen. Die von Oracle angebotene Klasse implementiert einen Webserver, der an eine IP-Adresse und an einen Port gebunden ist und dort auf eingehende TCP Verbindungen hört.

Um den Server nutzen zu können, müssen ihm ein oder mehrere `HttpHandler` hinzugefügt werden. Diese bearbeiten, die Anfragen auf verschiedene URL Pfade.

Der `HttpServer` kann bei der Verwendung der Unterklasse `HttpsServer` auch verschlüsselte Verbindungen realisieren. [Ora18]

### 7.5.3 Lösungsansatz 2: Node.js und Express

Node.js ist eine Plattform, ausgerichtet um Netzerkennungen zu erstellen. Dabei ist Node.js eine asynchrone und ereignisgesteuerte JavaScript Runtime. Node.js wird in JavaScript programmiert und kann mit Paketen von npm (Node Paket Manager) erweitert werden.

Als asynchrone Laufzeitumgebung arbeitet Node.js sehr viel mit Callback-Funktionen, die beim Erfüllen von Events ausgeführt werden. Durch die asynchrone Abarbeitung des Programmcodes entsteht eine sehr gut skalierbare Anwendung, die keine Deadlocks generieren kann. [Nod19]

Um mit Node.js einen Webserver zu erstellen, kann das Web-Framework Express verwendet werden. Express lässt sich mit Hilfe von npm in das Projekt eingliedern. Das Express Objekt ist in der Lage, auf einen Port auf TCP Verbindungen zu warten und diese entgegenzunehmen. Jedem Pfad ist eine Callback-Funktion zugeordnet, die bei dessen Aufrufen die Abarbeitung der Anfrage übernimmt. [Str19]

### 7.5.4 Bewertung des Servers

Da der Server in der Lage sein muss HTTP Nachrichten zeitlich verzögert abzuschicken, sind fertige und schwergewichtige Server, wie der Apache oder NGINX ungeeignet.

Bei beiden oben vorgestellten Servern, hat man die Möglichkeit den Datenfluss zu manipulieren und zu verzögern. Da es sich um normalen Java oder JavaScript Code handelt, kann man beliebige Funktionen eigenhändig implementieren und auch aus Dateien lesen. So ist der Java `HttpServer` und auch Express in der Lage den gewünschten Covert Channel zu realisieren.

Der sehr simple Aufbau und das einfache Hinzufügen und Verwenden von Paketen sprechen für den Node.js Server, weshalb dieser hier verwendet wird. Ein weiterer Vorteil von Node.js ist das einfache Installieren auf einem Linux System. Zusätzlich wird keine IDE benötigt.

### 7.5.5 Express Implementierung

Das importierte Express Paket wird als *app* Objekt in den Code eingebunden. Mit Hilfe dieses Objekts kann definiert werden, wie auf einen Request an eine URL-Route reagiert werden soll.

Eine Route wird wie in Listing 7.2 gezeigt hinzugefügt. Die mitgegebene Callback-Funktion wird ausgeführt, falls ein GET Request an die „Wurzel-Route“ erfolgt. Ist dies der Fall, so wird hier die *index.html* Seite ausgeliefert.

```
app.get('/', function(req, res){  
    res.sendFile(__dirname + '/index.html');  
});
```

Listing 7.2: Hinzufügen der Stamm-Route

Ebenfalls von Express stammt das *http* Objekt, welches den Http Server darstellt. Der Server Port kann wie in Listing 7.3 gewählt werden.

```
http.listen(80, function(){  
    console.log('listening on port:80');  
});
```

Listing 7.3: Wählen des Server Ports

## 7.6 Kommunikation des Covert Channel

Bei der Express Implementierung wird die *index.html* Datei mit Hilfe von REST (Representational State Transfer) ausgeliefert. REST ist ein Architekturstil, der die Kommunikation zwischen Server und Client regelt. Dieses System beruht darauf, dass ein Server eine beliebige Ressource, wie beispielsweise *index.html*, anbietet. Die wichtigsten Methoden um mit diesen Ressourcen umzugehen sind GET, PUT, POST und DELETE. [BB10]

Diese Methoden, auf die mit *http* zugegriffen wird, sehen nicht vor, dass ein Server selbständig und ohne danach gefragt zu werden Nachrichten an den Client sendet.

### 7.6.1 Anforderung an die Kommunikation des Covert Channels

Da die Hauptaufgabe des Server das Senden manipulierter Pakete an den Client sein wird, muss eine alternative Kommunikation gefunden werden, die nach dem Ausliefern der Webseite den Nachrichtentransport übernimmt. Dabei muss eine bidirektionale Kommunikation möglich sein. Auch eine einfache Anwendung soll angestrebt werden.

### 7.6.2 Lösungsansatz 1: Websockets

Das WebSocket Protokoll baut auf HTTP auf. Bei HTTP wird, nachdem der Client die Antwort vom Server erhalten hat, meistens die Verbindung geschlossen. Bei der Verwendung von Websockets wird die darunterliegende TCP/IP Verbindung weiterverwendet und kann wie ein normaler Netzwerksocket benutzt werden. Dies ermöglicht dem Client und dem Server jeder Zeit Daten zu senden. [Abt15]

### 7.6.3 Lösungsansatz 2: Socket.IO

Socket.IO ist eine JavaScript-Bibliothek, mit der sich eine bidirektionale Echtzeitkommunikation realisieren lässt. Die Funktion ähnelt einem WebSocket und lässt eine ereignisgesteuerte Kommunikation zwischen Browser und Server zu. [Soc19]

Um die Verbindung zwischen Server und Client auf jeden Fall sicherzustellen und um so viele Browser wie möglich zu unterstützen, setzt Socket.IO auf mehrere Technologien, wie Websockets, Flash-Sockets oder Comet. [Ull12]

### 7.6.4 Bewertung der Kommunikation des Covert Channels

Beide Methoden sind in der Lage den Covert Channel zu realisieren. Jedoch gibt es bei Websockets Probleme mit der Verwendung von Proxys.

Socket.IO stellt eine Erweiterung der WebSockets dar und verfügt über weitere Funktionen, wie zum Beispiel Broadcasting. Aus diesen Gründen und der Tatsache, dass eine sehr

komfortable node.js Schnittstelle vorhanden ist, soll hier die Kommunikation mit Socket.IO realisiert werden.

### 7.6.5 Socket.IO Implementierung

Socket.IO wird in Form des *io* Objekts in den Code eingebunden. Der Verbindungsaufbau ist, wie in Listing 7.4 gezeigt, aufgebaut. Den Events werden ihre jeweiligen Callback-Funktionen zugeordnet.

Wurde eine Verbindung aufgebaut, so sendet der Server eine *test* Nachricht an den Client. Hat der Client diese empfangen, so antwortet dieser mit einem *ClientHello*. Wird diese vom Server empfangen, so wird der jeweilige Socket in ein Array gespeichert und später zum Senden von weiteren Daten verwendet.

```
io.on('connection', (socket) => {  
  var address = socket.handshake.address.replace(/^.*/, '');  
  timestamps.push({ip: address ,time: getMinute()});  
  
  socket.emit('test', 'test');  
  
  socket.on('ClientHello', function (data) {  
    console.log("Client connected");  
    socken.push(socket);  
  });  
});
```

Listing 7.4: Verbindungsaufbau mit Socket.IO

## 7.7 Kodierung und Dekodierung

Der folgende Abschnitt beschäftigt sich damit, wie die Daten in den Trägerkanal codiert werden können, indem nur der Zeitpunkt des Versendens manipuliert wird.

Folgend werden zwei Methoden vorgestellt, die zur Kodierung der Binärdaten in Frage kommen.

### 7.7.1 Anforderung an die Kodierung und Dekodierung

Die Kodierung muss binäre Dateien, in Abstände zwischen Netzwerkpaketen umwandeln. Dabei sollen so wenig Pakete wie möglich benötigt werden und der Synchronisierungsaufwand möglichst gering sein. Ebenfalls soll die Kodierung resistent gegenüber Netzwerkschwankungen sein. Bei der Datenübertragung muss eine Pause eingelegt werden können.

### 7.7.2 Lösungsansatz 1: Kodierung mit festen Zeitrastern

Das in [CBS04] vorgestellte Verfahren basiert auf einer Generierung von Zeitintervallen. Diese werden sowohl beim Sender und Empfänger erstellt. Dabei haben die Intervalle des Empfängers einen zeitlichen Offset, der ungefähr der Übertragungsdauer entspricht. Dies dient dazu, dass Pakete die vom Sender abgesendet werden, beim Empfänger im gleichen Intervall ankommen. Nachdem Sender und Empfänger synchronisiert sind, kann man mit der Datenübertragung starten. Dabei wird ein Paket, dass in einem Zeitintervall ankommt als binäre 1 interpretiert. Kommt kein Paket so wird eine 0 geschrieben. Durch die Wahl längerer Zeitintervalle kann die Fehleranfälligkeit verringert werden, wobei jedoch die Übertragungsgeschwindigkeit ebenfalls abnimmt.

Für diese Art der Codierung ist es unabdingbar, dass die Uhren von Sender und Empfänger möglichst genau übereinstimmen.

Zur Veranschaulichung ist die Kodierung in *Abbildung 7.1* schematisch dargestellt.

### 7.7.3 Lösungsansatz 2: Kodierung basierend auf den Paketabständen

Zur Kodierung kann ebenso die Veränderung der Paketabstände benutzt werden. Hierzu wird zwischen einer kleinen oder großen Pause zwischen zwei Nachrichten unterschieden. Eine große Pause wird als binäre 1 interpretiert, eine kleine Pause als 0.

Je nachdem, wie gut die Netzwerkbedingungen sind, kann hier durch eine gezielte Verkleinerung der Pausen eine Erhöhung der Übertragungsgeschwindigkeit generiert werden.

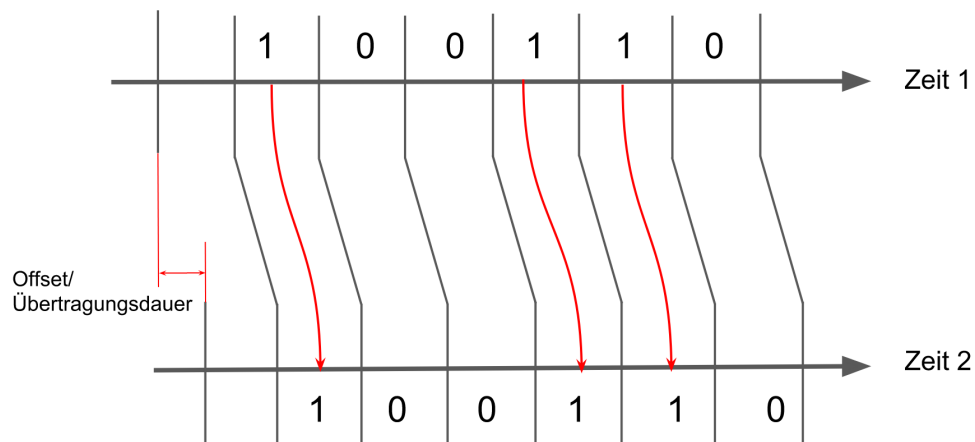


Bild 7.1: Kodierung mit festen Zeitrastern

Durch eine Verlängerung dieser Pausen sinkt jedoch die Fehleranfälligkeit.

Zu sehen ist diese Art der Kodierung in *Abbildung 7.2*

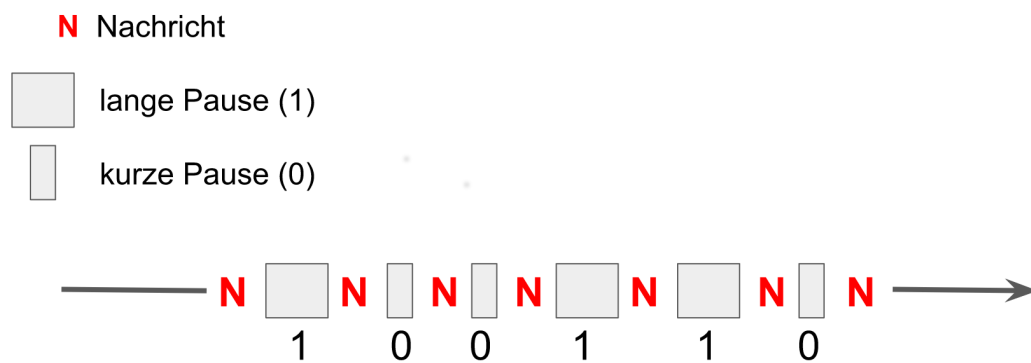


Bild 7.2: Kodierung anhand der Paketabstände

### 7.7.4 Bewertung der Kodierung

Bei der Kodierung mit Zeitrastern ist die Synchronisierung von entscheidender Bedeutung. Außerdem gilt es den Offset passend zu wählen. Diese Methode hat den Vorteil, dass zum Senden einer 0 keine Nachricht benötigt wird. Kommt es jedoch zu Fehlern im Offset, oder zu plötzlichen Übertragungsschwankungen ist dieses System sehr anfällig, da die Nachrichten in andere Zeitintervalle hineingeraten.

Werden die Paketabstände zu Kodierung verwendet, so wird zum Senden einer 0 eine Nachricht benötigt. Dadurch werden mehr Nachrichten benötigt, die kontinuierlich zu Verfügung stehen müssen.

Dabei besteht der Vorteil darin, dass sich nicht um die Synchronisierung gekümmert werden muss. Bei schlechten Netzwerkbedingungen lässt sich die Übertragung pausieren. Hierzu kann einfach die Datenübertragung gestoppt werden. Bei der Codierung mit Rastern würde dieses abrupte Stoppen der Datenübertragung als 0-Werte interpretiert werden. Bei einer reinen Betrachtung der Differenz zwischen den langen und kurzen Pausen, ist der Sender in der Lage die Sendegeschwindigkeit beliebig anzupassen.

Aufgrund der Vorteile durch die Codierung mit Hilfe der Paketabstände, soll diese Methode in dieser Arbeit verwendet werden.

### 7.7.5 Implementierung der Kodierung

Um den Covert Channel zu implementieren, werden Nachrichten zeitlich verzögert an den Client gesendet. Grundsätzlich kann jede beliebige Nachricht an den Client gesendet werden. Hier wird zum Testen die aktuelle Uhrzeit verwendet.

Die Länge der Pause zwischen den Daten hängt davon ab, ob eine 1 oder eine 0 übertragen werden soll. Eine 1 wird durch eine lange, eine 0 durch eine kurze Pause festgelegt.

Zur Definition der Pausen wird die lange Pause in Millisekunden angegeben. Zusätzlich wird ein Faktor festgelegt, der das Verhältnis zwischen der langen und kurzen Pause angibt. Durch die Variation von Pausenlänge und Faktor kann der Covert Channel eingestellt und optimiert werden. Ist eine komplette Datei übertragen, wird eine vorab definierte Pause



von einer Sekunde eingelegt. Danach beginnt die Datenübertragung erneut. In Listing 7.5 ist der Code gezeigt, der diese Kodierung umsetzt.

```
async function covertChannel(){
  while (true) {
    if (fileLoad == true) {
      for(i = 0; i < dataBits.length; i++){
        if(socken.length != 0){
          socken[0].emit('time', getTimeString());
        }
        if(dataBits[i] == "1"){
          await sleep(longBreak);
        }
        else {
          await sleep(shortBreak);
        }
      }
      if(socken.length != 0){
        socken[0].emit('time', getTimeString());
      }
    }
    await sleep(breakBetweenTransmit);
  }
}
```

Listing 7.5: Covert Channel

## 7.8 Geheime Daten

Die geheimen Daten werden vom Dateisystem gelesen und anschließend mit Hilfe des npm Paketes *buffer-bits* in einen Bit-String umgewandelt.

## 7.9 Front-End

### 7.9.1 HTML

Das Front-End wird durch eine einfache HTML-Seite ausgeliefert. Die Funktion der Webseite ist vorerst das Anzeigen der aktuellen Uhrzeit. Diese Uhrzeit wird vom Server empfangen und danach im Browser angezeigt.

### 7.9.2 JQuery

JQuery ist eine JavaScript Bibliothek, die es wesentlich einfacher macht, das HTML-Dokument und den DOM-Baum zu manipulieren und zu verändern. In diesem wird es dazu verwendet, um jeweils die aktuelle Uhrzeit dynamisch auf der Seite anzuzeigen. Zudem kommt es zum Einsatz, um die Eventhandler für Socket.IO einzubinden.

### 7.9.3 Socket.IO

Der nötige Code wird für das Paket über ein Script Tag heruntergeladen und entsprechend hinzugefügt. Wie auch serverseitig, kann hier ein SocketIO Objekt generiert werden. Diesem werden die Events zum Empfangen der Nachrichten, dem Verbindungsaufbau und dem Empfangen der Zeitpakete hinzugefügt.

## 7.10 Clientseitige Auswertung des Covert Channel

Die vom Server erhaltenen Pakete und vor allem die zeitlichen Abstände zwischen den Paketen müssen ausgewertet werden, um die geheime Nachricht zu rekonstruieren.

### 7.10.1 Anforderung an die Auswertung

Für die Auswertung muss eine Anwendung geschrieben werden, die die eingehenden Nachrichten vom Server analysiert. Dazu muss der exakte Zeitpunkt des Eintreffens der Pakete so genau wie möglich aufgezeichnet werden. Nach dem Rekonstruieren der geheimen Nachricht, muss diese auf das Dateisystem abgespeichert werden. Ein weiterer, wichtiger Aspekt ist, dass es für Dritte möglichst schwierig sein soll, die Analyse des Covert Channels zu bemerken.

### 7.10.2 Lösungsansatz 1: Auswertung im Front-End

Bei der Auswertung im Front-End werden die Daten direkt nach dem Empfangen im Browser ausgewertet. Hier sind die nötigen Funktionen mit JavaScript geschrieben. Sobald die Nachricht beim Browser ankommt, wird die Event Callback-Funktion ausgeführt, die den aktuellen Zeitstempel abspeichert.

### 7.10.3 Lösungsansatz 2: Auswertung mit einem externen Programm

Bei der Auswertung mit einem externen Programm, hat eine zweite, browserunabhängige Anwendung die Aufgabe, die eingehenden Nachrichten zu interpretieren. Dazu muss die Anwendung in der Lage sein, den Netzwerkverkehr mitzulesen und den Zeitpunkt des Eintreffens abzuspeichern und zu interpretieren.

### 7.10.4 Bewertung der Covert Channel Auswertung

Die Auswertung direkt im Front-End hat den Vorteil, dass kein zweites Programm benötigt wird. So ist diese Variante ressourcenschonender und vereinfacht die tatsächliche Anwendung.

Die Auswertung direkt im Front-End hat jedoch einen großen Nachteil. Der Front-End Code wird an jeden versendet, der die Seite aufruft. So kommen Dritte, die möglicherweise die

Kommunikation abhören, an den Sourcecode, der den Covert Channel auswertet. Zusätzlich wäre dadurch der Algorithmus der Hash-Funktion bekannt und auch die zugehörige Mapping-Tabelle. Dies würde einem Dritten, der als Man-in-the-Middle zwischen dem Server und Client steht, die Möglichkeit geben, selbst Nachrichten zu verfassen oder diese zu manipulieren.

Aus diesen Gründen wird die Auswertung des Covert Channel durch ein externes Programm realisiert.

## 7.11 Client

Der Client stellt hier den Empfänger des Covert Channels dar. Wie im vorhergehenden Kapitel festgelegt, soll dieser als externes Programm realisiert werden, das unabhängig vom Browser ist.

### 7.11.1 Programmiersprache

#### Anforderung an die Programmiersprache

Die zu verwendende Programmiersprache muss in der Lage sein, Daten effizient zu verarbeiten. Eine harte Echtzeit ist jedoch nicht nötig. Es soll ein Programm entstehen, das auf allen Unix Systemen lauffähig ist und das sich über die Konsole oder ein Shell-Script öffnen lässt.

Es soll möglichst einfach sein, ein anderes Konsolenprogramm zu öffnen und dessen Output zu empfangen und auszuwerten. Was in Kapitel 6.12.2 näher erläutert wird.

#### Lösungsansatz 1: Java

Java ist eine objektorientierte Programmiersprache, dessen Code auf mehr als 3 Milliarden Geräten ausgeführt wird. Der Java Code orientiert sich an C++, aber auch an anderen Skript Sprachen.

Der Java Code wird von einem Compiler in Bytecode umgewandelt. Dieser kann auf jedem

Gerät, das die Java Runtime besitzt, ausgeführt werden. Dadurch muss der Code nicht mehr für jedes Gerät kompiliert werden. [Ull04]

## **Lösungsansatz 2: Python**

Python ist eine objektorientierte Skript Sprache. Der Code wird als lesbares Skript an den Anwender übergeben und von einem Python Interpreter ausgewertet. So wird kein Compiler und auch nicht unbedingt eine IDE benötigt. Die Python Syntax ist so entworfen, dass die Skripte sehr gut lesbar und auch wiederverwendbar sind.

Trotzdem ist der Code sehr kompakt und hat in der Regel ein Drittel bis ein Fünftel der Codelänge von traditionellen Programmiersprachen wie Java oder C++. [Wei06]

## **Bewertung der Programmiersprache**

Beide Programmiersprachen sind in der Lage das Clientprogramm zu realisieren. Java zeigt im direkten Vergleich mit Python eine bessere Performance. Python hingegen ist besser dafür geeignet, um mit zusätzlichen Konsolenprogrammen zu arbeiten und bringt von Haus aus viele Funktionen mit, die bei der Datenverarbeitung helfen. Da es sich hier nicht um ein kommerzielles Projekt handelt, kann das Skript einfach verbreitet und nach Bedarf angepasst werden. Aus diesen Gründen wird in diesem Projekt Python als Programmiersprache verwendet.

### **7.11.2 Mitschneiden der Datenpakete**

Paket-Sniffer sind Programme, die in der Lage sind, den Netzwerkverkehr auf den verschiedenen Interfaces aufzuzeichnen und für den Benutzer zu veranschaulichen. Ein solches Programm soll hier verwendet werden, um die Datenpakete mitzuschneiden.

## Anforderungen an den Paket-Sniffer

Der Paket-Sniffer soll sich über die Konsole öffnen lassen und die Ergebnisse in eine Pipe schreiben. Das Tool soll frei für alle UNIX Systeme erhältlich sein. Die Pakete sollen möglichst korrekt mitgeschnitten und der Eintreffzeitpunkt des Pakets ausgegeben werden. Zusätzlich soll das Filtern der Pakete möglich sein, um die mitgeschnittenen Pakete auf die vom sendenden Server einzugrenzen.

### Lösungsansatz 1: tcpdump

tcpdump ist eine Konsolenanwendung für UNIX Systeme, die die empfangenen Netzwerkpakete an einer Netzwerkschnittstelle ausgibt. Dabei basiert tcpdump auf der Betriebssystemschnittstelle *libpcap*. Es lassen sich ebenfalls Filter einstellen, die beispielsweise nur Pakete von einem bestimmten Host aufzeichnen. [JLM03]

### Lösungsansatz 2: tshark

tshark ist eine Version von Wireshark, die dessen volle Funktion in der Konsole aufrufbar macht. tshark ist ein sehr mächtiges Tool um Netzwerkpakete nicht nur aufzuzeichnen, sondern auch zu Decodieren. Es lassen sich ebenfalls unzählige Filter realisieren. [Wir19]

## Bewertung der Paket-Sniffer

tshark hat einen größeren Funktionsumfang. In diesem Projekt wird aber nur der Zeitstempel der Nachricht benötigt. tcpdump ist hierzu in der Lage und hat ebenfalls den Vorteil, dass er bei vielen UNIX Systemen wie zum Beispiel Ubuntu bereits vorhanden ist und nicht installiert werden muss.

## Implementierung des Paket-Sniffer

Für das Öffnen und anschließende Mitschneiden der Daten wird ein eigener Thread gestartet. Hier wird zuerst mit *os.popen* ein neuer Prozess erstellt in dem tcpdump läuft. Die nötigen Parameter für die Einstellung des Filters werden ebenfalls übergeben. Der Rückgabewert ist hier ein *open file object*, in welches die Pipe zu tcpdump die Ergebnisse schreibt. Aus diesem *open file object* kann die tcpdump Ausgabe gelesen und der Empfangszeitpunkt in einen Buffer gespeichert werden.

Die Zeitstempel werden dann kontinuierlich in ein globales Datenarray geschrieben. Um Kollisionen zu vermeiden, wird ein Mutex verwendet.

Der Programmcode, der den Paket-Sniffer handhabt, ist in Listing 7.6 gezeigt.

```
pipe = os.popen("tcpdump -s 0 host "+host+" and src port "+
    port+" -q -i any -l")
for line in pipe:
    buffer.append(line[0:15])

    if len(buffer) > bufferzize:
        mutex.acquire()
        data = data + buffer
        buffer = []
        mutex.release()
```

Listing 7.6: Paket-Sniffing

### 7.11.3 Interpretieren der Zeitstempel

Die Zeitstempel vom tcpdump-Thread sollen ausgewertet werden. Dafür werden die als String abgespeicherten Zeitstempel in *datetime* Objekte geparkt, um mit ihnen Rechnen zu können.

Die hiermit berechneten zeitlichen Differenzen werden entweder als eine binäre 1, oder 0 interpretiert. Wie auch schon serverseitig, wird die Geschwindigkeit des Covert Channels durch die Angabe der langen Pause und dem Faktor, der den Unterschied zwischen der langen und der kurzen Pause beschreibt, definiert.

So kann zum Beispiel eine lange Pause als 50 Millisekunden und der Faktor als 0.5 definiert werden. Daraus ergibt sich eine kurze Pause von 25 Millisekunden.

## Toleranz

Da die Zeiten mit sechs Nachkommastellen angegeben werden, ist es unmöglich, dass die Pausen exakt der Angabe entsprechen. Durch Verzögerungen, Schwankungen in der Netzwerkgeschwindigkeit, aber auch durch Prozess-Scheduling, kann es zu Abweichungen kommen.

Deshalb muss die Pause nicht als ein fester Zeitpunkt, sondern als Zeitfenster definiert werden. Ist die Länge der Differenz zwischen den Paketen im jeweiligen Zeitfenster enthalten, kann sie entsprechend interpretiert werden. Falls nicht, wird dieses Paket ignoriert.

Die Zeitfenster werden durch die prozentuale Angabe, relativ zur Pausenlänge, in positiver und negativer Richtung angegeben.

Wird die Pause mit 50 Millisekunden, einer positiven Toleranz von 30% und einer negativen Toleranz von 10% angegeben, ergibt sich ein Zeitfenster zwischen 45 und 65 Millisekunden. Da nur die Differenz der Nachrichten betrachtet wird, können Nachrichten auch zu früh kommen, wenn die vorhergehende Nachricht erheblich zu spät war. Aus diesem Grund muss das Zeitfenster auch in negativer Richtung erweitert werden.

Von der Einstellung dieser Toleranzfenster hängt erheblich die Qualität der empfangenen Daten ab.

Der nachstehende Code zeigt, wie eine Klassifizierung der Differenz, hier *f1* genannt, in die Zeitfenster realisiert ist.

```
if write == True:
    if sBigBreakTolerance < f1 < bBigBreakTolerance:
        codedata.append("1")
        print(str(f1) + " \t=> 1 ")
    else:
        if sSmallBreakTolerance < f1 < bSmallBreakTolerance:
            codedata.append("0")
            print(str(f1) + " \t=> 0 ")
        else:
```



```
print(str(f1) + " \t=> undefind: will be ignored")
```

Listing 7.7: Interpretation mit Zeitfenstern

#### 7.11.4 Verarbeiten der Erhaltenen Daten

Von den erhaltenen Daten werden die letzten 8 Bit abgeschnitten. Dies ist der Hash-Wert vom Server. Um die Daten zu validieren, wird auf diese ebenfalls die Hash-Funktion angewendet. Stimmt der Hash-Wert mit dem vom Server überein, so wird die Datei ins Dateisystem geschrieben.

Damit die Daten nicht UTF8-Codiert auf die Datei geschrieben werden, wird das Paket *BitArray* verwendet, welches es möglich macht, die binären Daten zu schreiben.

Die Datei oder Nachricht ist hiermit erfolgreich übertragen.

## 8 Optimales Einstellen und Anwendung des aktiven Covert Channel

Es gilt nun den Covert Channel so optimal wie möglich einzustellen. Dazu muss die beste Kombination aus Pausenlänge und Zeitfenster gefunden werden.

Gezielte Test-Datenübertragungen sollen Aufschluss zu diesen Werten geben. Um reale Netzwerkbedingungen zu schaffen, wird der Server auf einem 1&1 Cloud Server in Karlsruhe gehostet. Das Ergebnis eines TCP-Pings ergibt eine Übertragungsdauer von 30 ms zum Server. Bei allen Tests wird eine Testdatei mit 12 Byte (Hallo Welt) verwendet.

### 8.0.1 Verhältnis nicht interpretierbar/ interpretierbar

Der erste Test ist die Ermittlung des Verhältnisses zwischen interpretierbaren und nicht interpretierbaren Zeitstempeln. Interpretierbar ist ein Zeitstempel, wenn er sich entweder im Zeitfenster für Nullen, oder Einsen befindet.

Die Ergebnisse dieses Tests sind in Tabelle 8.1 zu sehen. Die Spalten zeigen, jeweils die Länge einer langen Pause in Sekunden. Die Zeilen stehen für den prozentualen Anteil an der Pausenlänge, die das Zeitfenster definiert.

Die Ergebnisse sind in Prozent angegeben. Die Länge der kurzen Pausen beträgt die Hälfte der langen.

Ein Wert von 0 Prozent spiegelt eine Übertragung wieder, bei der alle Zeitstempel interpretiert werden konnten. Liegt der Wert bei 100 Prozent waren die Anzahl interpretierter und nicht interpretierter Zeitstempel gleich.

Aus der Tabelle lässt sich ableiten, dass ein Zeitfenster mit 20% nicht zum Optimum führt,

da hier die Werte für eine stabile Datenübertragung zu hoch sind. Auch die Übertragung mit einer Pausenlänge von 0,03 ms ist nicht geeignet.

Da dieser Test jedoch keine Fehlinterpretationen betrachtet, muss noch ein weiterer Test hinzugefügt werden.

	<b>0,08</b>	<b>0,05</b>	<b>0,03</b>
20%	1,26	66,44	96,53
30%	0,62	1,74	82,87
40%	0,26	0,31	2,49
50%	0,20	0,1	0,95

Tabelle 8.1: Verhältniss nicht interpretierbar/ interpretierbar

Diese Erkenntnisse decken sich auch mit dem auf Bild 4.2 gezeigten Ergebnissen von [CBS04]

### 8.0.2 Anzahl korrekt übertragener Dateien

Bei diesem Test soll die Korrektheit der ankommenden Daten begutachtet werden. Dazu werden für jede mögliche Kombination 20 Datenübertragungen durchgeführt und die Anzahl der korrekt übertragenen Dateien gezählt. Dadurch ergibt sich folgende Tabelle:

	<b>0,08</b>	<b>0,05</b>	<b>0,03</b>
20%	11	0	0
30%	14	10	0
40%	15	9	0
50%	0	0	0

Tabelle 8.2: Korrekt übertragene Dateien

Zusätzlich zu den im ersten Test ausgeschlossenen Zeilen und Spalten, kann nun auch die Zeile mit einem 50%igem Zeitfenster ausgeschlossen werden. Den guten Werten dieses Zeitfensters im ersten Test steht eine nahezu 100 %-ige Fehlinterpretation gegenüber.

### 8.0.3 Zeit bis zum ersten korrekten Eintreffen der Datei

Nachdem durch die ersten beiden Tests Kombinationen ausgeschlossen werden konnten, sind nun noch die Kombinationen 30%/0,08s, 40%/0,08s, 30%/0,05s und 40%/0,05s übrig. Um hieraus das Optimum zu ermitteln, wird ein Test durchgeführt, der einer realen Anwendung am nächsten kommt. Es wird die Zeit bis zum ersten korrekten Eintreffen einer gesendeten Datei gemessen.

Diese Messung wird jeweils für alle Kombinationen 10 mal ausgeführt. Die Ergebnisse sind in Bild 8.1 zu sehen.

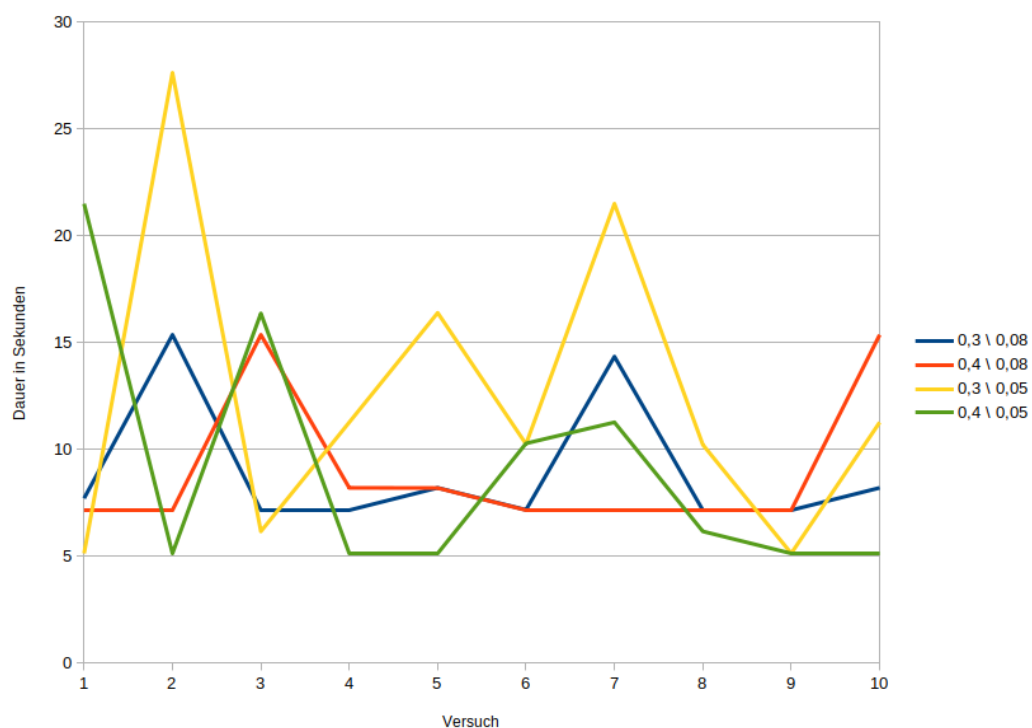


Bild 8.1: Zeit bis zum ersten korrekten Eintreffen

In Tabelle 8.3 sind die Durchschnittswerte gezeigt, die sich aus diesem Test ergeben.

Aus der Graphik in Abbildung 8.1 lässt sich erkennen, dass die Werte der Kombination aus 0,3%/0,05s (gelb) enorme Schwankungen beinhalten. Auch der höchste Durchschnittswert mit 12,418 Sekunden spricht gegen die Verwendung dieser Kombination. Bei den Werten

	0,08	0,05
30%	8,956	12,418
40%	9,004	9,114

Tabelle 8.3: Durchschnittliche Zeiten

30%/0,08s sind die besten Durchschnittswerte aufgetreten, weshalb diese auf jeden Fall als Einstellungswerte zugelassen werden. Ebenfalls sehr vielversprechend ist die Kombination aus 40%/0,05s (grün). Hier zeigt die Grafik, bis auf einen Ausschlag bei der ersten Messung, sehr niedrige Werte.

Um ein eindeutigeres Ergebnis zu schaffen, werden noch weitere Messungen mit den Kombinationen 30%/0,08s und 40%/0,05s durchgeführt.

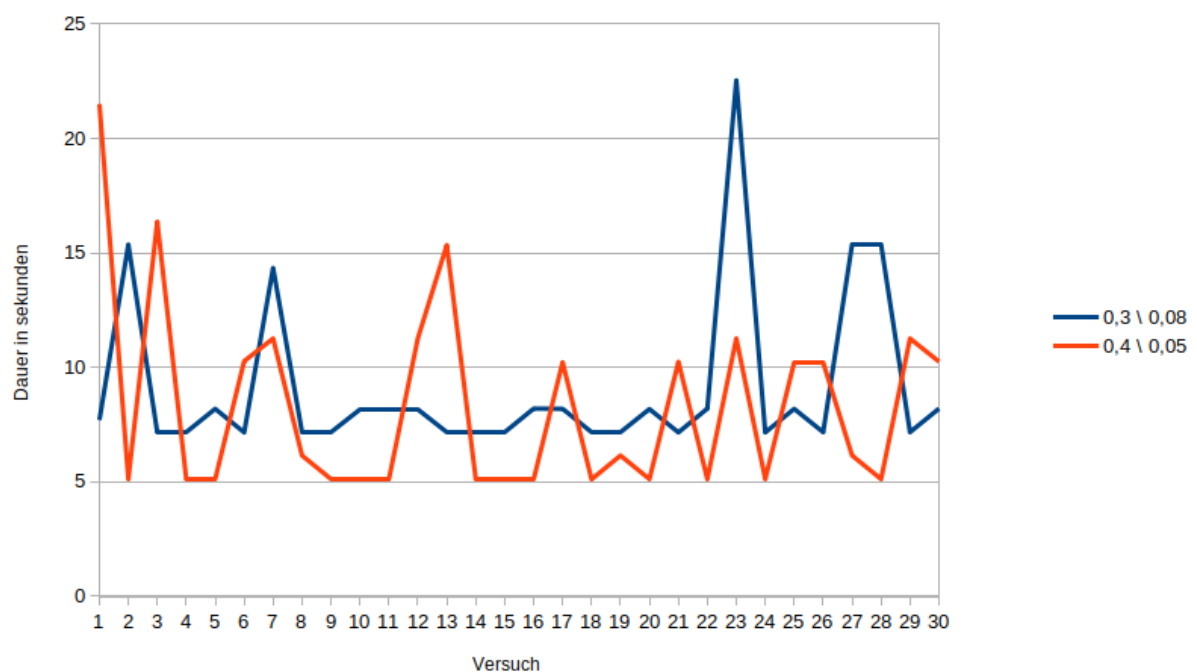


Bild 8.2: Zeit bis zum ersten korrekten Eintreffen

Die Grafik in Bild 8.2 zeigt die Zeiten der eingesetzten Werte. Bei der Verwendung der

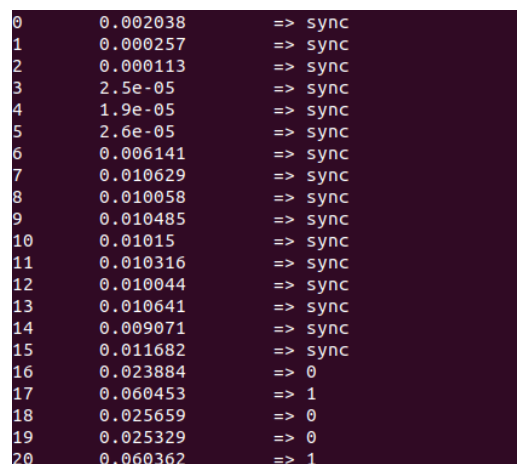
Werte 30%/0,08s (blau) lässt sich eine höhere Konstanz erkennen. Hier werden deutlich mehr Daten auf den ersten Versuch korrekt übertragen. Das ist bei 40%/0,05s seltener der Fall, jedoch ist hier die mögliche Übertragungsgeschwindigkeit auf Grund der kleineren Pausen größer. Trotz größerer Schwankungen, ist die durchschnittliche Zeit von 40%/0,05s mit 8,327s kleiner als die von 30%/0,08s mit 9,095s.

Die Kombination aus einem Zeitfenster von 40% und einer große Pause von 0,05 Sekunden ist die optimale Lösung für diesen Covert Channel.

### 8.0.4 Padding zum Synchronisieren

Bei der Anwendung in einem realen Netzwerk treten am Anfang des Sendevorgangs erhebliche Schwankungen in der Übertragungszeit auf. Diese Abweichungen könnten dem allgemeinen Routing oder dem Verbindungsaufbau von Socket.IO entspringen.

Um dieses Problem zu umgehen, werden 2 Byte Padding, zum Synchronisieren, an den Anfang der Dateien gestellt. Der Abstand zwischen diesen Paketen ist auf 0.01 Sekunden festgelegt. In Abbildung 8.3 ist zu sehen, dass mindestens 6 Datenpakete benötigt werden, bis sich der Korrekte Paketabstand einstellt.



0	0.002038	=> sync
1	0.000257	=> sync
2	0.000113	=> sync
3	2.5e-05	=> sync
4	1.9e-05	=> sync
5	2.6e-05	=> sync
6	0.006141	=> sync
7	0.010629	=> sync
8	0.010058	=> sync
9	0.010485	=> sync
10	0.01015	=> sync
11	0.010316	=> sync
12	0.010044	=> sync
13	0.010641	=> sync
14	0.009071	=> sync
15	0.011682	=> sync
16	0.023884	=> 0
17	0.060453	=> 1
18	0.025659	=> 0
19	0.025329	=> 0
20	0.060362	=> 1

Bild 8.3: Fehler in den ersten Paketen

## 9 Umsetzung der passiven Variante

Die in Abbildung 6.2 dargestellte passive Lösungsvariante des Covert Channels soll in diesem Kapitel betrachtet werden. Diese Variante hat den Vorteil, dass als Server jeder beliebige Webserver verwendet werden kann. Die Codierung wird von einem Proxy realisiert.

Für die Proof of Concept Implementierung wird hier der Client der aktiven Variante wiederverwendet. Im Browser des Clients muss die Verwendung eines Proxys eingetragen werden.

### 9.0.1 Anforderung an den Proxy

Der Proxy hat die Aufgabe, TCP Nachrichten von einem Client an einen Server zu senden. Die Nachrichten sollen bidirektional durch den Proxy geschleust werden. Nachrichten vom Client werden an den Server weitergeleitet, ebenso wie umgekehrt. Dabei soll der Inhalt der Nachrichten nicht verändert werden. Wie bei der aktiven Variante, soll der Proxy in der Lage sein, die TCP Pakete zeitlich zu verzögern, um so die geheimen Daten zu übertragen.

Der Proxy soll auf einem Linux System lauffähig sein.

### 9.0.2 Lösungsansatz 1: Veränderung eines Open-Source Proxys

Ein Lösungsansatz ist die Verwendung eines fertigen Proxys, wie zum Beispiel eines Squid oder Tinyproxy. Für diese Open Source Programme ist der Code frei erhältlich. Hier gilt es die Funktion zum Senden der Nachrichten so zu verändern, dass die Nachrichten gezielt verzögert werden können.

### 9.0.3 Lösungsansatz 2: Proxy selbst programmieren

Eine alternative Lösung ist die eigenständige Programmierung eines Proxys. Dazu soll ein Python-Skript geschrieben werden, das jeweils ein Netzwerksocket zum Client und zum Server verwaltet. Werden die TCP Pakete von einem an den anderen Socket weitergegeben, erhält man einen Proxy. Die Pakete können hier frei verändert werden.

### 9.0.4 Bewertung des Proxys

Beide Lösungsansätze sind in der Lage die Anforderungen umzusetzen. Der selbst programmierte Proxy bietet die volle Kontrolle, bei gleichzeitig mehr Spielraum für diese Proof of Concept Implementierung. Zudem ist diese sehr leichtgewichtige Software Variante komfortabler zu debuggen und zu optimieren. Aus diesen Gründen soll hier der Proxy eigenhändig erstellt werden.

Sollte sich der Proof of Concept als Erfolg herausstellen, ist die erste Lösung für eine reale Anwendung besser geeignet, da ein großer Funktionsumfang mitgenutzt werden kann.

### 9.0.5 Implementierung

#### Sockets

Es wird ein Listen-Socket erstellt, der auf eingehende Nachrichten auf Port *80* hört. Dieser Port muss später in den Proxy-Einstellungen des Browsers hinterlegt werden.

Sendet der Browser einen HTTP-Request an den Proxy, wird ein Client-Socket erstellt. Der Proxy erstellt ebenfalls einen Server-Socket, indem er sich mit dem, im HTTP-Request stehenden Webserver verbindet. Da oft externe Ressourcen verwendet werden und Verbindungen nach dem Senden einer Datei geschlossen werden, entstehen pro Webseite mehrere solcher Verbindungen.

Um die hierdurch entstehenden Sockets zu verwalten, wird das Betriebssystemmodul *select* verwendet. *select* stellt fest, ob die angegebenen Sockets zum Lesen oder Schreiben bereit sind. Ist dies der Fall wird eine Liste mit beschreibbaren und eine mit lesbaren Sockets zurückgegeben. Dadurch müssen nicht alle Sockets dauerhaft mit Polling abgefragt werden



müssen.

In Listing 7.1 ist der Code zu sehen, der die oben beschriebene Handhabung der Sockets übernimmt. Die Liste aller Sockets wird zur Kontrolle an *select* übergeben.

```
while True:
    readable, writable, exceptional = select.select(self.lsock, self.
        lsock, self.lsock)

    self.read(readable)
    self.send(writable)
```

Listing 9.1: Socket Handling

Anhand ihres Indexes in der Socket-Liste, können die Sockets identifiziert werden. Dabei gehören immer ein Client- und ein Server-Socket zusammen. Die Nachrichten die zwischen diesen zwei ausgetauscht werden sollen, werden in einem Array gespeichert.

## Geheime Daten

Vergleichbar mit der aktiven Variante, werden die zu übertragenden Daten von einer angegebenen Datei gelesen und in eine binäre Liste umgewandelt. Hieraus wird ein 8 Bit Pearson-Hash berechnet und an das Ende der binären Liste angehängt.

## Covert Channel

Bei der Erstellung des Covert Channels müssen die Nachrichten an den Client zeitlich verzögert werden. Dabei besteht allerdings das Problem, dass bei dieser Anwendung nur ein Thread verwendet wird, der gleichzeitig für das Senden und Empfangen zuständig ist. Auf Grund dessen kann der Lösungsansatz der aktiven Variante, wobei der Thread für die benötigte Zeit in *sleep* gesetzt wird, nicht verwendet werden.

Daher wird nach jedem Senden an den Client der Zeitstempel abgespeichert. Bevor ein neues Paket gesendet wird, wird kontrolliert, ob die Differenz der aktuellen Zeit und dem letzten Sendezeitpunkt der gewünschten Pause entspricht.

Die Pausen werden äquivalent zu der aktiven Variante definiert.

# **10 Anwendung des passiven Covert Channel**

## **10.1 Probleme bei realen Webseiten**

### **10.1.1 HTTPS**

Der Proxy ist nicht für die Verwendung von HTTPS ausgelegt, wodurch hier nur HTTP Webseiten abgerufen werden können.

### **10.1.2 Generierung einer ausreichenden Anzahl von Netzwerkpaketen**

Für den Covert Channel wird ein kontinuierlicher Datenstrom benötigt. Durch den verbindungsorientierten Aufbau von TCP ist es schwierig, die Daten zu puffern und später auf einmal abzusenden, da auf das ACK-Flag gewartet wird.

Um künstlich einen hohen Netzwerkverkehr zu generieren, kann beim Empfangen der Daten eine geringe Paketgröße gesetzt werden.

In diesem Fall wird eine maximale Paketgröße von 256 Byte gewählt. Dadurch wird gerade beim Empfangen von Bildern ein enormer Netzwerkverkehr erzeugt, welcher für den Covert Channel genutzt werden kann.

### 10.1.3 Parallele Datenabfrage

Wie vorhergehend bereits beschrieben, öffnet ein Webserver gleich mehrerer Sockets, um parallel Daten zu empfangen. Diese Nachrichtenpakete, der einzelnen Sockets können nicht vom Client auseinander gehalten werden, da dieser nur den Source-Port betrachtet, der jedoch bei allen Paketen gleich.

In Bild 10.1 ist ein Ausschnitt aus den Firefox Development Tool zu sehen. Hier wird die Übertragungsdauer der einzelnen Dateien gezeigt. Dabei fällt auf, dass bei der hier verwendeten Webseite der *PH-Weingarten*, 5 CSS Dateien gleichzeitig übertragen werden. Diese 5 daraus entstehenden Covert Channels überschneiden sich bei der Interpretation im Client und sind so nicht mehr auswertbar.

Der Covert Channel funktioniert nur, wenn genau eine Datei übertragen wird. Oft dauert die Übertragung großer Bilder jedoch länger als die von Quellcode. Dies bietet die Chance, dass am Ende der Übertragung nur noch ein Socket für die Bildübertragung aktiv ist. Dieser kann für die Datenübertragung im Covert Channel genutzt werden kann.





















Datei	Urspr...	Typ	Übertragen	Größe	0 ms	10,24 s	20,48 s
 /	document	html	5,43 KB	21,45 KB	 7532 ms		
 22b408405...	stylesheet	css	504 B	214 B	 5881 ms		
 9697ca18ff...	stylesheet	css	7,16 KB	6,88 KB	 6845 ms		
 magnific-p...	stylesheet	css	7,14 KB	6,86 KB	 6841 ms		
 ph.css?151...	stylesheet	css	19,56 KB	19,27 KB	 8690 ms		
 ph_print.cs...	stylesheet	css	2,24 KB	1,95 KB	 6095 ms		
 jquery-ui.css	stylesheet	css	6,19 KB	30,61 KB	 59 ms		
 ui.theme.css	stylesheet	css	2,38 KB	9,59 KB	 88 ms		
 nLByrxPyE...	subdocumenthtml		17,23 KB	46,73 KB		 128 ms	
 www-playe...	stylesheet	css	51,64 KB	303,68 KB		 167 ms	

Bild 10.1: Parallele Datenabfrage

Als mögliche Lösung kann der Client zur Berücksichtigung des Destination Ports erweitert werden, um die verschiedenen Covert Channel zu unterscheiden.

## 10.2 Test mit realen Servern

Um die Funktion des Proxys zu testen und reale Netzwerkbedingungen zu schaffen, wird er auf einem 1&1 Cloud Server installiert. Die zu übertragende Datei ist eine Textdatei mit 12 Byte. Zur Datenübertragung werden die gleichen zeitlichen Einstellungen verwendet, die bereits im Kapitel 7 evaluiert wurden. Das bedeutet, die Lange Pause hat eine Länge von 0,08 Sekunden und das Zeitfenster entspricht jeweils 30%. Als Client wird das Programm der aktiven Variante verwendet. Um genügend Netzwerkverkehr zu generieren, wird die maximale Paketgröße vom Server zum Client auf 256 Byte festgelegt.

Um den Proxy nutzen zu können, muss dieser im Browser eingetragen werden. Dadurch werden alle HTTP Nachrichten zuerst an den Proxy weitergeleitet.

Nun wird eine beliebige Webseite gewählt, die als Paketquelle dient. Diese Seite muss jedoch über HTTP ausgeliefert werden. Hier wird die Webseite der Pädagogischen-Hochschule Weingarten mit der Adresse *http://www.ph-weingarten.de/* verwendet. Wird die Webseite geladen, so sendet der Webserver der PH-Weingarten die Webseitendateien über den Proxy an den Client.

Jedoch besteht anfangs, das im vorhergehenden Kapitel beschriebene, Problem mit der parallelen Datenübertragung. Das Ergebnis ist in Bild 10.2 zu sehen. Bei einer optimalen Funktion werden den zeitlichen Differenzen links binären Werte zugeordnet. Daher gelingt die Zuordnung in den meisten Fällen nicht.

Ist nur noch ein Socket aktiv, der für die Dateiübertragung eines größeren Bilds zuständig ist, so findet die Datenübertragung wie in Bild 10.3 gezeigt statt. Dies ist der beste Zeitpunkt, die Datenübertragung via Covert Channel zu starten. Fehler treten von diesem Zeitpunkt an nur noch aufgrund von Netzwerk- oder Serveranomalien auf.

Die Datei kann erfolgreich übertragen werden.

```

113 0.217937 => undefind: will be ignored
114 0.040102 => 0
115 6.09174 => undefind: will be ignored
116 0.039757 => 0
117 6.127644 => undefind: will be ignored
118 0.080082 => 1
119 6.104416 => undefind: will be ignored
120 0.080401 => 1
121 6.091236 => undefind: will be ignored
122 0.040008 => 0
123 1.685666 => undefind: will be ignored
124 0.038596 => 0
125 0.081227 => 1
126 0.087133 => 1
127 0.067917 => 1
128 0.123142 => undefind: will be ignored
129 0.080827 => 1
130 2.083262 => undefind: will be ignored
131 0.086003 => 1
132 1.669796 => undefind: will be ignored
133 0.039838 => 0

```

Bild 10.2: Fehlinterpretation des Clients

```

89 0.078208 => 1
90 0.040631 => 0
91 0.080264 => 1
92 0.040214 => 0
93 0.039708 => 0
94 0.039927 => 0
95 0.041007 => 0
96 0.040095 => 0
97 0.080147 => 1
98 0.040025 => 0
99 0.079626 => 1
100 0.040867 => 0
101 0.039591 => 0
102 0.080694 => 1
103 0.079372 => 1
104 0.080304 => 1
105 0.041033 => 0
106 0.079489 => 1
107 0.040031 => 0
108 0.041041 => 0
1.082022 => Start of File
Hash from serer: 116
Hash from client: 116

Data: 01001000011000010110110001101100011011110010000001010
1110110010101101100011101000000101000001010
Data Length: 96

```

Bild 10.3: Korrekte Interpretation des Clients

## 11 Evaluierung

Das Ergebnis dieser Arbeit benutzt einen Covert Channel, als eine Methode des Information-Hiding, um Daten zu übertragen. Dabei können die Daten jedes beliebige Format annehmen, da diese in ihrer binären Form übertragen werden. Die für diesen Covert Channel höchst mögliche Übertragungsgeschwindigkeit wurde ermittelt und kann auch angewendet werden. Es gibt jedoch andere Covert Channels, wie beispielsweise Storage Channels, die zu deutlich höheren Übertragungsgeschwindigkeiten fähig sind. Der Kanal ist als Algorithmus umsetzbar, was durch die Realisierung mittels eines Computerprogramms gezeigt worden ist. Dadurch, dass der Covert Channel unabhängig vom Paketinhalt ist, lässt er sich nicht von Netzwerk-Normalisierungen beeinflussen. Er ist jedoch stark von Netzwerkbedingungen abhängig, was mit der Manipulation der Zeitabstände einhergeht.

Die Umsetzung des Covert Channels ist sowohl als aktive, als auch passive Variante möglich, was durch die erfolgreichen Implementierungen gezeigt worden ist.

Auf Probleme stößt man jedoch bei der passiven Lösung, wenn vom Webserver Dateien parallel übertragen werden. Für die aktive Variante stellte sich die Schwankungen beim Übertragungsstart als problematisch dar. Abhilfe konnte durch das Voranstellen von Synchronisationsbits geschaffen werden.

## 12 Fazit und Ausblick

### 12.1 Aktiv

Der Server und sowie Client sind in der Lage einen Covert Channel zu erstellen, der in der Lage ist, Daten zu übertragen. Die Kommunikation geschieht verdeckt, kommt aber ohne Verschlüsselung aus.

Das in der Motivation genannte Beispiel lässt sich mit dieser Lösung umsetzen. Es können Daten jeglicher Größe übertragen werden. Jedoch macht eine Größe über 25 Byte keinen Sinn, da sonst die Chance auf falsch interpretierte Bits zu groß wird. Die entstandene Softwarelösung bietet jedoch eine gute Basis, um entsprechende Erweiterungen zu implementieren.

Mit einer durchschnittlichen Datenübertragungsrate von 11,53 Bits pro Sekunde ist der Covert Channel nicht sehr schnell, aber ausreichend, um kleinere Datenmengen zu übertragen.

Wird der Covert Channel zur Kommunikation verwendet, ist es für Dritte nahezu unmöglich eine verdeckte Kommunikation zu erkennen. Wird der Server so getarnt, dass er nicht mit einem Sender, wie zum Beispiel der Polizei, in Verbindung gebracht werden kann, so stellt diese Lösung eine sehr sichere Variante dar, um kurze Nachrichten wie Treffpunkte, Anweisungen oder Telefonnummern auszutauschen.

### 12.2 Passiv

Die passive Variante des Covert Channels ist eine *Proof of Cocept* Implementierung und noch nicht geeignet, um als reale Anwendung verwendet zu werden. Das Konzept eines

passiven, zeitlichen Cover Channel wurde bestätigt und dessen Funktion bewiesen.

Wie auch bei der aktiven Variante ist eine Datengröße über 25 Byte nicht sinnvoll. Es ist aber eine gute Basis entstanden, die erweitert werden kann. Durch die Einschränkungen ist es nicht einfach, eine geeignete Webseite für den Covert Channel zu finden. Bei einer realen Umsetzung sollte die Verwendung des ersten Lösungsansatzes in Betracht gezogen werden. Dabei kann man auf einer funktionierenden Proxylösung aufbauen, bei der alle Webseiten unterstützt werden.

Dennoch kann man mit der Kombination aus Proxy und Client, Netzwerkpakete eines fremden Servers verwenden, um Informationen ohne Veränderung des Paketinhalts zu versenden.

## 12.3 Ausblick

In diesem Kapitel werden einige mögliche Erweiterungen vorgestellt, die in Zukunft implementiert werden könnten, beziehungsweise sollten, um eine reale Anwendung zu schaffen.

### **Dateien aufteilen**

Um auch große Dateien zu verschicken, können diese in mehrere kleine aufgeteilt werden. Es müsste jedoch eine Methode gefunden werden, um die Datei Fragmente wieder zusammenzusetzen.

### **Automatische Pausenerkennung**

Um die Bedienung komfortabler zu machen, kann eine automatische Erkennung der Pausenlänge implementiert werden. So wird eine variable Anpassung der Pausenlänge möglich. Diese Anpassung kann vom Server oder Proxy als Reaktion auf eine Netzwerkanomalie veranlasst werden.



### **Beachtung des Destination Ports beim Client**

Um das derzeitige Problem bei der parallelen Datenübertragung zu lösen, kann zusätzlich zur Erkennung der Pakete der TCP Destination Port beachtet werden. So könnten die verschiedenen Übertragungen der Dateien unterschieden werden. Damit wäre sogar eine parallele Datenübertragung möglich.

### **Server mit realistischer Webseite**

Die derzeitige Webseite zeigt die aktuelle Uhrzeit an. Um den Covert Channel noch sicherer und unauffälliger zu machen, kann eine Webseite implementiert werden, die einen plausiblen Einsatzzweck hat.

### **Verwendung von HTTPS**

Um wirklich alle Webseiten verwenden zu können, muss der Proxy in der Lage sein HTTPS Pakete weiterzuleiten. Es macht ebenfalls Sinn den Server mit einer verschlüsselten Verbindung auszustatten.

## Danksagung

Am Ende dieser Bachelorarbeit bedanke ich mich bei allen, die mir bei der Fertigstellung dieser Arbeit geholfen haben.

Einen besonderen Dank geht an Herr Prof. Dr. Eggendorfer, der es möglich gemacht hat, dass ich mein Wunschthema realisieren konnte.

Abschließend möchte ich mich bei meinen Eltern bedanken, die mir während der Schulzeit und dem Studium immer zur Seite standen und mich unterstützt haben.

# Literatur

- [De19] DENNIS SCHIRRMACHER : *Patchday: Das Öffnen von PNG-Bildern kann Android-Geräte kompromittieren*. <https://www.heise.de/security/meldung/Patchday-Das-oeffnen-von-PNG-Bildern-kann-Android-Geraete-kompromittieren-43.html>, 2019. Abrufdatum: 12.02.2019.
- [Abt15] ABTS, DIETMAR: *Bidirektionale Kommunikation mit WebSocket. Masterkurs Client/Server-Programmierung mit Java*, 189–205. Springer, 2015.
- [BB10] BARTON, THOMAS HARRIET BACH: *Modellierung eines Anwendungssystems zur Behälterlokalisierung und Behälterreservierung auf Basis des Architekturstils REST*. MKWI, 2411–2418, 2010.
- [CBS04] CABUK, SERDAR, CARLA E BRODLEY CLAY SHIELDS: *IP covert timing channels: design and detection*. *Proceedings of the 11th ACM conference on Computer and communications security*, 178–187. ACM, 2004.
- [Dav10] DAVIES, MICHAEL: *Traffic distribution techniques utilizing initial and scrambled hash values*, 26 2010. US Patent 7,821,925.
- [Die18] DIE BUNDESBEAUFTRAGTEN FÜR DEN DATENSCHUTZ UND DIE INFORMATIONSSICHERHEIT (BFDI): *Was ist Datenschutz?* <https://www.bfdi.bund.de/DE/Datenschutz/Ueberblick/ueberblick-node.html>, 2018. Abrufdatum: 16.10.2018.
- [EL18] ERTEL, WOLFGANG EKKEHARD LÖHMANN: *Angewandte Kryptographie*. Carl Hanser Verlag GmbH Co KG, 2018.
- [Ert01] ERTEL, WOLFGANG: *Angewandte Kryptographie. Fachbuchverlag Leipzig*. Carl Hanser Verlag), ISBN, 2001.
- [Fab99] FABIEN A. P. PETITCOLAS, ROSS J. ANDERSON AND MARKUS G. KUHN: *Information Hiding A Survey*. <https://www.petitcolas.net/fabien/publications/ieee99-infohiding.pdf>, 1999. Abrufdatum: 22.02.2019.

- [FGM<sup>+</sup>99] FIELDING, ROY, JIM GETTYS, JEFFREY MOGUL, HENRIK FRYSTYK, LARRY MASINTER, PAUL LEACH TIM BERNERS-LEE: *Hypertext transfer protocol-HTTP/1.1.* , 1999.
- [Gol03] GOLTZ, JAMES P.: *Under the radar:A look at three covert communications channels.* 2003.
- [Hel18] HELLMANN, ROLAND: *IT-Sicherheit: eine Einführung.* Walter de Gruyter GmbH & Co KG, 2018.
- [HS96] HANDEL, THEODORE G MAXWELL T SANDFORD: *Hiding data in the OSI network model.* *International Workshop on Information Hiding*, 23–38. Springer, 1996.
- [Inf81] INFORMATION SCIENCES INSTITUTE UNIVERSITY OF SOUTHERN CALIFORNIA: *TRANSMISSION CONTROL PROTOCOL DARPA INTERNET PROGRAM PROTOCOL SPECIFICATION.* <https://tools.ietf.org/html/rfc793>, 1981. Abrufdatum: 01.02.2019.
- [JLM03] JACOBSON, VAN, CRAIG LERES STEVEN MCCANNE: *TCPDUMP public repository.* Web page at <http://www.tcpdump.org>, 2003.
- [Kes15] KESSLER, GARY C.: *An Overview of Steganography for the Computer Forensics Examiner (Updated Version, February 2015)*, 2015.
- [Kle01] KLENSIN, JOHN: *RFC 2821: Simple mail transfer protocol.* Request For Comment, Network Working Group, 2001.
- [KP00] KATZENBEISSER, STEFAN FABIEN PETITCOLAS: *Information hiding techniques for steganography and digital watermarking.* Artech house, 2000.
- [LC17] LOCKWOOD, ROBERT KEVIN CURRAN: *Text based steganography.* *International Journal of Information Privacy, Security and Integrity*, 3(2):134–153, 2017.
- [LMU17] LMU MÜNSCHEN: *Erklärung Hamming Codes.* <http://www.mobile.ifi.lmu.de/wp-content/uploads/lehrveranstaltungen/rechnerarchitektur-rose17/Erkl%C3%A4rungHammingCodes.pdf>, 2017. Abrufdatum: 27.02.2019.
- [LT10] LEE, CHE-WEI WEN-HSIANG TSAI: *A new steganographic method based on information sharing via PNG images.* *2nd International Conference on Computer and Automation Engineering (ICCAE)*, 2010.

- [LT13] LEE, CHE-WEI WEN-HSIANG TSAI: *A data hiding method based on information sharing via PNG images for applications of color image authentication and metadata embedding*. Signal Processing, 93(7), 2013.
- [Nic18] NICO GRUNDMEIER: *Sicherheitslücken im Internet*. <http://www.informatik.uni-oldenburg.de/~iug10/sli/indexd917.html?q=node/19>, 2018. Abrufdatum: 16.10.2018.
- [Nod19] NODE.JS FOUNDATION: *About Node.js®*. <https://nodejs.org/en/about/>, 2019. Abrufdatum: 01.04.2019.
- [Ora18] ORACLE, 2005,2018.
- [PR85] POSTEL, JON JOYCE REYNOLDS: *Rfc 959: File transfer protocol (ftp)*. InterNet Network Working Group, 1985.
- [Pur10] PURGATHOFER, PETER: *Eine kurze Geschichte der Steganographie*. Die Funktion verdeckter Kommunikation: Impulse für eine Technikfolgenabschätzung zur Steganographie, 9:65, 2010.
- [Soc19] SOCKET.IO: *What Socket.IO is*. <https://socket.io/docs/>, 2019. Abrufdatum: 02.04.2019.
- [Str19] STRONGLOOP, INC.: *Beispiel ?Hello World?* <https://expressjs.com/de/starter/hello-world.html>, 2019. Abrufdatum: 01.04.2019.
- [Ull04] ULLENBOOM, CHRISTIAN: *Java ist auch eine Insel*, 8. Galileo Press, 2004.
- [Ull12] ULLRICH, BENJAMIN: *WebSockets: spezifikation/implementierung*. Innovative Internet Technologies and Mobile Communications (IITM), 53, 2012.
- [Use18] USER: BLACK SLASH: *How to Hide Secret Data Inside an Image or Audio File in Seconds*. <https://null-byte.wonderhowto.com/how-to/steganography-hide-secret-data-inside-image-audio-file-seconds-0180936/>, 2018. Abrufdatum: 22.01.2019.
- [Wät18] WÄTJEN, DIETMAR: *Hashfunktionen*. Kryptographie, 93–112. Springer, 2018.
- [Wei06] WEIGEND, MICHAEL: *Objektorientierte Programmierung mit Python*. mitp Bonn, 2006.
- [Wen12a] WENDZEL, STEFFEN: *The Problem of Traffic Normalization Within a Covert Channel's Network Environment Learning Phase*. Sicherheit, 12, 149–161, 2012.

- 
- [Wen12b] WENDZEL, STEFFEN: *Tunnel und verdeckte Kanäle im Netz: Grundlagen, Protokolle, Sicherheit und Methoden*. Springer-Verlag, 2012.
- [Wik18] WIKIPEDIA CONTRIBUTORS: *OSI-Modell* — *Wikipedia, The Free Encyclopedia*. <https://de.wikipedia.org/wiki/OSI-Modell>, 2018. Abrufdatum: 03.01.2019.
- [Wir19] WIRESHARK: *tshark man-page*. <https://www.wireshark.org/docs/man-pages/tshark.html>, 2019. Abrufdatum: 06.04.2019.
- [ZCC07] ZHONG, SHANGPING, XUEQI CHENG TIERUI CHEN: *Data Hiding in a Kind of PDF Texts for Secret Communication*. *IJ Network Security*, 4(1):17–26, 2007.
- [Zis13] ZISLER, HARALD: *Computer-Netzwerke*, 2013.