Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorthesis

Bernd Juris

Methoden zur computergestützten Detektion steganographischer Inhalte in Bilddaten

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Bernd Juris

Methoden zur computergestützten Detektion steganographischer Inhalte in Bilddaten

Bachelorthesis eingereicht im Rahmen der Bachelorprüfung
im Studiengang Informations- und Elektrotechnik
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr.-Ing. Robert Fitz
Zweitgutachter : Prof. Dr. rer. nat. Annabella Rauscher-Scheibe

Abgegeben am 5. Mai 2017

Bernd Juris

Thema der Bachelorthesis

Methoden zur computergestützten Detektion steganographischer Inhalte in Bilddaten

Stichworte

Kryptographie, Steganographie, Bildverarbeitung, künstliche Intelligenz, Python

Kurzzusammenfassung

Diese Arbeit befasst sich mit der Erarbeitung und Analyse ausgewählter Methoden, um Steganogramme zu erkennen. Sie zeigt die Möglichkeiten und Grenzen einer statistischen Analysemethode und diskutiert die Anwendbarkeit von künstlicher Intelligenz in Hinblick auf steganographische Systeme.

Bernd Juris

Title of the paper

Methods for computer assisted detection of steganographic content in image data

Keywords

cryptography, steganography, image processing, artificial intelligence, Python

Abstract

This report describes and analyses selective methods, which can be used for detection of steganographic contents in image data. It shows the possibilities and the limits of a statistical method of analysis and discusses the application of artificial intelligence for steganographical systems.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Glossar

AES Advanced Encryption Standard, ein Verschlüsselungsverfahren basierend auf Substitutionsboxen.

AI artificial intelligence

Chiffretext Verschlüsselte Nachricht in Textform

Glitch Hier: ein durch den Angreifer verursachter Spannungsabfall am Prozessor zu einem bestimmten Rechenzeitpunkt, um Fehler zu induzieren und so die Verschlüsselung zu brechen

IT Informationstechnik

KI Künstliche Intelligenz

LSB Least significant Bit mit der Wertigkeit 2^0 eines Bytes

Nibble „Halbbyte“, ein aus 4 Bits bestehendes Datum

OTP One-Time-Pad ist ein symmetrisches Verschlüsselungsverfahren, bei dem ein zufällig generierter Schlüssel der Länge der zu verschlüsselnden Nachricht einmalig verwendet wird.

Pixel Bildpunkt in der digitalen Bildverarbeitung

PNG Portable Network Graphics

Python Python ist eine vielseitige open-source Programmiersprache.

RAW verlustfreies Aufnahmeformat von digitalen Kameras (inkl. Metadaten)

RSA Ein sehr starkes, asymmetrisches Verschlüsselungsverfahren benannt nach seinen Entwicklern: Rivest, Shamir, Adleman.

WEP Wired equivalent privacy, ein veraltetes als unsicher geltendes Verschlüsselungsverfahren für Drahtlosnetzwerke.

WLAN wireless local area network, Kurzbezeichnung für ein Drahtlosnetzwerk

1 Einführung

1.1 Motivation

Die Problematik der Daten- und Informationssicherheit wird politisch und gesellschaftlich zunehmend präsenter. Dabei ist vorrangig die Kryptographie im Gespräch. Es wird über Verschlüsselungsverfahren, das Recht auf Privatsphäre und auch über die Gefahrenabwehr diskutiert.
Waren in weiter Vergangenheit Verschlüsselungsverfahren nur für die Militärs von Bedeutung, sind sie heute schon Alltag geworden. Neben den Bankkarten, die schon sehr früh Sicherheitsmechanismen auf kryptografischer Ebene bekommen haben, sind es heutzutage der Personalausweis, der Reisepass, E-Mails, Chats, Online-Konten, Cloud-Systeme und viele weitere Dienste, die wir täglich nutzen.
Um die Sicherheit solcher Systeme zu verbessern hat sich im Verlauf der Geschichte das Kerckhoffs'sche Prinzip durchgesetzt. Auguste Kerckhoffs postulierte schon im Jahr 1883 mehrere Grundsätze zur Verschlüsselung. Das Bekannteste besagt, dass die Sicherheit einer Verschlüsselung nicht von der Geheimhaltung des Algorithmus abhängen darf [Kerckhoffs, 1883].
Ein passendes Beispiel für die Wirksamkeit dieses Grundsatzes ist der AES[1]. Er wurde im Rahmen einer öffentlichen Ausschreibung für einen im Jahr 2000 neuen Standard ausgewählt und so einem breiten Spektrum an Experten zur Verfügung gestellt. Seit dieser Zeit konnten sich Fachleute aus aller Welt mit der Sicherheit dieses Algorithmus beschäftigen und diese stetig verbessern. Trotzdem gibt es Schwachstellen, die ausgenutzt werden können, um auch sichere Algorithmen auszuhebeln. Daraus ergibt sich die Fragestellung, ob es nicht sinnvoll wäre, die verschlüsselte Information vor eventuellen Angreifern zu verstecken. Denn wenn Informationen ohne das Mitwissen anderer übertragen werden, sinkt auch die Wahrscheinlichkeit eines Angriffs auf diese. Aber auch solche steganographischen Algorithmen haben Schwachstellen. Denn das Verstecken von Information verändert das Containermedium. Je besser der steganographische Algorithmus ist, desto schwieriger ist es, diese Veränderungen zu erkennen. Um die Weiterentwicklung der Steganographie nach dem Kerckhoffs'schen Prinzip voranzutreiben, sollten auch in dieser Disziplin Methoden erforscht werden, die zur Detektion versteckter Inhalte dienen.

[1] Advanced Encryption Standard

1.2 Zielsetzung

Basierend auf einem von D.I. Dauch implementierten Verfahren, werden Möglichkeiten zur computergestützten Steganalyse seines Systems gefunden. Dazu entsteht eine Programmbibliothek, die verschiedene Analyse- und Auswertungsmöglichkeiten bietet. Konkret sollen diese Tools erkennen, ob und gegebenenfalls in welchem Umfang Informationen übertragen werden. Besonderes Augenmerk liegt dabei darauf, dass die Schwachstellen der Analysemethoden, die Dauch durch seine Implementierung umgehen konnte, ausgebessert werden. Um dieses Ziel zu erreichen, wird das gegebene System analysiert und nachgebildet. Die entstehende Bibliothek soll möglichst universell, transparent und effizient bleiben, weswegen sie mit Hilfe der Programmiersprache Python erarbeitet wird.
Die Analysemethoden sollen anschließend noch auf ihre praktische Anwendbarkeit sowie ihre Effizienz und Zuverlässigkeit untersucht werden.

2 Grundlagen

In diesem Kapitel werden die Grundlagen der Kryptographie, Steganographie, der Bildverarbeitung, der statistischen Bildanalyse sowie der künstlichen Intelligenz soweit erläutert, wie es zum Verständnis dieser Arbeit notwendig ist.

2.1 Kryptologie

Die Kryptologie ist der Oberbegriff für die Teilgebiete Kryptographie und Kryptoanalyse. Dabei ist die Kryptographie die Wissenschaft der Verschlüsselung von Informationen und die Kryptoanalyse bezeichnet die Wissenschaft der Entschlüsselung von verschlüsselten Daten, ohne vorherige Kenntnis des Schlüssels.

2.1.1 Kryptographie

Die Kryptographie ist im Allgemeinen die Wissenschaft des Verschlüsselns bzw. der Geheimhaltung von Informationen. Diese Geheimhaltung wird seit jeher umgesetzt, indem man den Klartext nach einem bestimmten Schema oder Algorithmus ändert und ihn so für andere unlesbar macht. Nur Menschen, denen das Verfahren und der Schlüssel bekannt war, konnten die Nachricht wieder entschlüsseln. Dieses Grundprinzip wird auch heute noch verfolgt.
Um die Kryptographie einzuordnen, müssen zunächst einige Definitionen getroffen werden. In der Informationstechnik (IT) gibt es eine Vielzahl an Teilgebieten. Ein sehr wichtiges davon ist die IT-Sicherheit oder auch Computersicherheit. Dabei bietet der deutsche Wortschatz nur den Begriff Sicherheit, der viel bedeuten kann. Die englische Sprache erlaubt eine konkretere Unterteilung. Hier werden zwei Begriffe unterschieden: Security und Safety.

- **Safety** ist dabei im Allgemeinen der Schutz von Anlage, Mensch und Umwelt vor Unfällen, Fehlfunktionen oder Ähnlichem. In diese Kategorie fallen alle Ursachen, die unbeabsichtigt oder auf natürlichem Wege geschehen, wie Naturkatastrophen, Fehlbedienungen oder Hardwaredefekte.
- **Security** hingegen bezeichnet den Schutz vor äußeren Einflüssen, die beabsichtigt Schäden anrichten sollen. Beispiele dafür sind Sabotage, Hacking, Einbrüche oder auch Diebstahl.

In diesem Zusammenhang wird das deutsche Wort Sicherheit in dieser Arbeit mit dem englischen Begriff Security gleichgesetzt. Im Speziellen ist die IT-Sicherheit hier die Sicherheit von Informationen, d.h. deren Schutz vor unbefugtem Entschlüsseln, Mitlesen oder Verändern. Die Kryptographie wiederum kann als Teilgebiet der IT-Sicherheit eingeordnet werden. Sie beschäftigt sich mit der Sicherung von Daten durch Verschlüsselungsverfahren, sodass diese geschützt übertragen oder gespeichert werden können. In manchen Fachbüchern wird die Kryptographie speziell der Netzwerksicherheit zugeordnet [Schmeh, 2009]. Diese Zuordnung verschwimmt jedoch zunehmend. Denn auch unvernetzte Rechner nutzen die Kryptographie, zum Beispiel in militärischen oder industriellen Systemen bei denen die Speichermodule verschlüsselt werden.
Weiterhin ist die Kryptographie eine mathematische Disziplin, da zum Verständnis und zur Umsetzung kryptographischer Prozesse oft die mathematische Darstellung genutzt wird. Es hat sich ebenfalls gezeigt, dass die bekanntesten Algorithmen der heutigen Zeit (AES, RSA[1]) mathematisch verhältnismäßig sehr einfach dargestellt werden können und daher für Entwickler gut implementierbar sind.
Im Allgemeinen werden Ver- und Entschlüsselung als Funktionen mit den Variablen Nachricht (m), verschlüsselte Nachricht (c) und Schlüssel (k) dargestellt. Die Funktion zur Verschlüsselung (Encryption) ist definiert als:

$$c = E(k_{enc}, m) \tag{2.1}$$

Die Entschlüsselung (Decryption) zur Rekonstruktion des Klartextes m lautet:

$$m = D(k_{dec}, c) \tag{2.2}$$

Hierbei ist es möglich, aber nicht notwendig, dass die beiden Schlüssel k_{enc} und k_{dec} gleich sind. Man unterscheidet hier symmetrische ($k_{enc} = k_{dec}$) und asymmetrische ($k_{enc} \neq k_{dec}$) Verfahren.

- **Symmetrische** Verschlüsselungsverfahren nutzen denselben Schlüssel, um die Nachricht zu ver- und zu entschlüsseln. Folglich muss der Schlüssel beiden Seiten bekannt sein. Solche Verfahren bringen die Problematik des sogenannten Schlüsselaustauschproblems mit sich. Ein solcher Schlüsselaustausch muss entweder unverschlüsselt ausgeführt werden oder ist auf ein anderes Verschlüsselungsverfahren zur sicheren Schlüsselübertragung angewiesen. Ist dieses Problem der Schlüsselverteilung überwunden, kann eine Kommunikation auf Basis von symmetrischer Verschlüsselung auch nur bedingt als sicher eingestuft werden. Beispielsweise kann die verschlüsselte Übertragung trivialer Informationen dazu führen, dass potentielle Angreifer die Originalinformation erraten und damit auf den verwendeten Schlüssel schließen können. Dazu muss der Angreifer lediglich im Besitz des originalen Nachrichteninhalts und der verschlüsselten Nachricht sein sowie Kenntnis über das verwendete Verfah-

[1] asymmetrisches Verschlüsselungsverfahren benannt nach seinen Entwicklern: Rivest, Shamir und Adleman.

ren haben. Dies war einer der Gründe, warum der WLAN-Standard WEP[2] gebrochen werden konnte [Tews et al., 2007].
Deswegen sind symmetrische Verfahren per se nicht unsicher. Ein Beispiel für einen sicheren Algorithmus dieser Art ist der AES.

- **Asymmetrische** Verschlüsselungsverfahren hingegen nutzen Schlüsselpaare. Mit dem ersten, öffentlichen Schlüssel wird verschlüsselt und mit dem zweiten, privaten entschlüsselt. Dazu muss der Empfänger einer Nachricht lediglich dem Sender seinen öffentlichen Schlüssel mitteilen und kann dann ausschließlich mit seinem privaten Schlüssel die Nachricht entschlüsseln. Eine Entschlüsselung mit dem öffentlichen Schlüssel ist nicht möglich. Man kann sich das in etwa so vorstellen, dass der Empfänger dem Sender ein offenes Vorhängeschloss übergibt. Mit diesem Schloss verschließt der Sender eine Truhe mit den geheimen Informationen und schickt sie dem Empfänger. Dieser wiederum kann mit dem Schlüssel für sein Vorhängeschloss die Truhe öffnen. Schlüssel und Vorhängeschloss müssen auf jeden Fall zueinander passen, damit die Nachricht nach der Übertragung entschlüsselt werden kann. Das bedeutet für die digitale Schlüsselgeneration, dass der öffentliche und der private Schlüssel in einen mathematischen Zusammenhang gesetzt werden müssen. Im besten Fall ist ein Errechnen des öffentlichen Schlüssels sehr einfach und umgekehrt, das Errechnen des privaten Schlüssels sehr schwer.
  Ein bekanntes Beispiel für solch ein Verfahren ist der RSA. Dieser Algorithmus basiert auf der Multiplikation von zwei (großen) Primzahlen zur Errechnung des öffentlichen Schlüssels. Würde ein Angreifer eine der beiden Primzahlen kennen, so könnte er die andere berechnen und so auf den privaten Schlüssel schließen, der ebenfalls aus den beiden Primzahlen berechnet wird. Doch dazu müsste er unter anderem auf den öffentlichen Schlüssel eine Primfaktorzerlegung anwenden. Diese Primfaktorzerlegung ist aufwendig und die Komplexität des Problems steigt mit der Größe der multiplizierten Faktoren. Die Sicherheit des RSA basiert also auf der Größe der Primzahlen und der damit aufwendigen Primfaktorzerlegung. Eine Methode zur Errechnung des privaten Schlüssels ist dabei immer bekannt. Jedoch kostet die Durchführung dieser Rechnung bei entsprechend großen Zahlen sehr viel Zeit, selbst bei sehr hoher Rechenleistung. Außerdem besteht für den Angreifer das Risiko, dass nach erfolgreicher Berechnung des Schlüssels die Daten obsolet sind oder bereits mit einem neuen Schlüssel verschlüsselt wurden.
  Dennoch kann theoretisch jederzeit eine effektivere Methode entdeckt werden, die eine solche Primfaktorzerlegung auch bei großen Zahlen schnell durchführen kann. Damit wäre der RSA gebrochen. Dies ist einer der Gründe, warum Kryptographen mit großer Sorge auf den RSA schauen und er deswegen nur begrenzte Anwendung findet.

[2] Wired Equivalent Privacy

2.1.2 Kryptoanalyse

Die Kryptoanalyse ist der Gegenpol zur Kryptographie. Hier wird nach Schwachstellen gesucht und es werden Krypto-Systeme angegriffen. Diese Angriffe haben das Ziel aus verschlüsselten Daten die Klartextnachricht zu gewinnen. Dazu ist es in den meisten Fällen nötig, den Schlüssel, mit dem der Klartext chiffriert worden ist, zu berechnen. Das Kerckhoffs'sche Prinzip bietet hier einen großen Vorteil für die Kryptoanalytiker, da den Angreifern Ver- und Entschlüsselungsverfahren schon bekannt sind. Im Allgemeinen unterscheidet man bei Angriffen auf verschlüsselte Daten verschiedene Szenarien:

- **Ciphertext Only** beschreibt ein Angriffsszenario bei dem lediglich eine oder mehrere verschlüsselte Nachrichten bekannt sind. Selbst bei einem bekannten Verschlüsselungsalgorithmus ist es in diesem Fall sehr schwer den Schlüssel zu berechnen.

- **Known Plaintext** bedeutet, der Angreifer verfügt über den Klartextinhalt sowie über die verschlüsselte Nachricht selbst, um daraus den Schlüssel abzuleiten. Etwas weniger Informationen stehen beim Probable-Plaintext-Szenario zur Verfügung. Hier wird angenommen, dass Teile der Klartextnachricht bekannt sind. In beiden Fällen ist es leichter möglich den Schlüssel zu berechnen als im Ciphertext-Only-Fall.

- **Chosen Plaintext** heißt, beliebig viele selbstgewählte Klartextnachrichten können vom Angreifer verschlüsselt werden und somit sind ihm Klartext und Chiffretext bekannt. Dadurch sind weitere Analysen möglich, da der Zusammenhang von Klartextveränderungen und Chiffretextveränderungen untersucht werden kann. Dieses Szenario kann auf asymmetrische Verfahren zutreffen. Hier können, wenn der öffentliche Schlüssel bekannt ist, Nachrichten beliebig verschlüsselt werden.

- **Chosen Ciphertext** beschreibt die Möglichkeit des Angreifers für eine begrenzte Zeit verschlüsselte Nachrichten zu entschlüsseln.

  (Quellen: [Wikipedia, 2017] und [Schneier, 2000])

Je mehr Informationen zur Verfügung stehen, desto leichter ist es herauszufinden, wie man die Nachricht entschlüsseln kann. Der RSA zeigt jedoch, eine Verschlüsselung kann sicher sein, obwohl bereits ein Algorithmus zum Entschlüsseln (Primfaktorzerlegung) bekannt ist. Durch solche Ansätze hilft die Kryptoanalyse Verschlüsselungsverfahren weiterzuentwickeln und zu optimieren.
Abhängig vom Untersuchungsszenario gibt es noch viele weitere Möglichkeiten, wie beispielsweise die Seitenkanalanalyse oder auch das gezielte Stören von Prozessoren (Glitch), um Verschlüsselungen zu brechen. Für diese Arbeit sind nur die oben genannten Grundprinzipien von Bedeutung, um die Ansätze der im Folgekapitel beschriebenen Steganalyse ebenfalls einordnen zu können.

2.2 Steganographie

Die Steganographie ist die Wissenschaft des Versteckens von Informationen. Auch sie hat die Geheimhaltung zum Ziel. Allerdings steht hier keine starke Verschlüsselung von Nachrichten im Mittelpunkt, sondern deren Übermittlung in unauffälligen Informationen, sogenannten Containermedien. Wurden in einem solchen Containermedium Informationen eingebettet, so spricht man von einem Steganogramm.
Anders als bei der Kryptologie, die die Kryptographie und die Kryptoanalyse unter einem Oberbegriff zusammenfasst, gibt es für die Steganographie keinen gängigen Oberbegriff. Dennoch stehen die beiden Themengebiete Steganographie und Steganalyse in ähnlich engem Zusammenhang wie Kryptographie und Kryptoanalyse.

2.2.1 Übersicht und Einordnung

Auch bei der Einordnung der Steganographie finden mehrere Teilgebiete zusammen. Historisch gesehen ist die Steganographie eine Kunstform. Weit vor der Erfindung des ersten Computers wurden von Künstlern Informationen auf Papier, Holz oder Leinwänden versteckt. Sei es durch Wasserzeichen oder kleine Textnachrichten in Gemälden. Auch in Texten können Inhalte nach einem bestimmten Schema versteckt werden, die dann nur von Personen entschlüsselt werden können, denen die Methodik bekannt ist. Hier verschwimmt allerdings die Grenze zur Kryptographie etwas, da auch Klartexte in eine Art von Chiffretexten überführt werden. Auch das vielen bekannte Schreiben mit unsichtbarer Tinte ist eine Form der Steganographie.
In der modernen Steganographie werden Informationen hauptsächlich in von Menschen konsumierten Trägermedien versteckt. Dies hat den Hintergrund, dass die Veränderung von computergenutzten Daten auch Veränderungen in deren Verhalten verursachen würde. Wohingegen der Mensch solche Veränderungen gegebenenfalls nicht wahrnimmt. Ändert man zum Beispiel in einer Bilddatei jede Farbe nur ganz leicht oder fügt ein wenig mehr Grundrauschen in ein Audio-File ein, so wird dies vom Menschen kaum bis gar nicht wahrgenommen.
Um solche Verfahren umzusetzen sind Computerprogramme nötig, die entsprechende Dateien bearbeiten können. Außerdem müssen auch hier die Vorgänge dargestellt und analysiert werden. Dafür bietet sich analog zur Kryptographie wieder die Mathematik an. Daher kann die Steganographie als mathematische und informationstechnische Disziplin angesehen werden.
Eine Grundidee für die Verwendung von steganographischen Verfahren in der heutigen Zeit ist deren Kombination mit Verschlüsselungen. Denn wenn niemand eine Datenübertragung vermutet, gibt es auch keinen Angreifer, der versuchen kann, die Nachrichten zu entschlüsseln. Anders ausgedrückt, die Wahrscheinlichkeit der Datenentschlüsselung sinkt, wenn weniger Angreifer versuchen sie zu entschlüsseln.
In dieser Arbeit ist lediglich die Steganographie im Bereich der Bildverarbeitung von Bedeutung. Im Folgenden ist das Prinzip kurz erläutert:

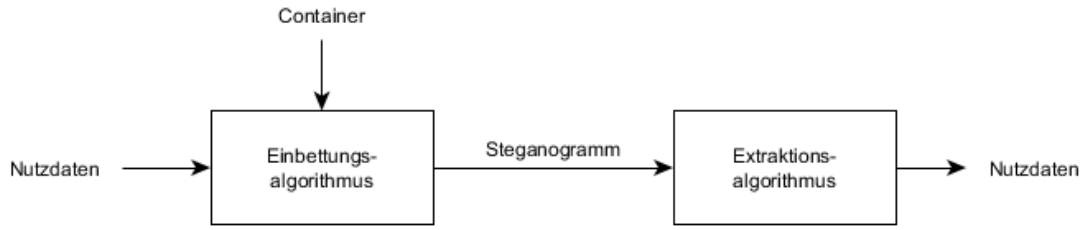


Abbildung 2.1: Blockdiagramm; allgemeines steganographisches System

Abbildung 2.1 beschreibt vereinfacht den allgemeinen Ablauf beim computergestützten Verstecken von Informationen. Nutzdaten und ein passender Container werden einem Algorithmus zur Verfügung gestellt. Dieser erstellt ein Steganogramm, das nun für den Betrachter wie das originale Trägermedium aussieht, jedoch alle (geheimen) Nutzdaten enthält. Bevor die Daten vom Extraktionsalgorithmus empfangen werden, passieren sie im Realfall noch einen Übertragungskanal. Da dieser jedoch keine Veränderung des Steganogramms hervorruft, kann er vernachlässigt werden. Nach der Extraktion können nun die Nutzdaten weiterverarbeitet werden. Bei dieser vereinfachten Darstellung ist zu beachten, dass Einbettungs- und Extraktionsalgorithmus in Zusammenhang stehen, da zu einer Einbettungsmethode eine passende Extraktionsmethode genutzt werden muss.
Beschränkt man nun dieses Modell auf Bilddateien als Container und textuelle Nachrichten als Nutzdaten, so kann man beides mathematisch darstellen.

2.2.2 Bildverarbeitung in der Steganographie

Ausgehend von einem einfachen digitalen Bild im Portable Network Graphics-Format (PNG) ohne Vorfilterung[3], besteht dieser Container aus einer Matrix von Pixeln. Ein Pixel repräsentiert den Farbwert an einer bestimmten Stelle in der Matrix. Bei einem Graustufenbild mit 8 Bit Farbtiefe ist es beispielsweise ein ganzzahliger Skalar im Bereich von 0 bis 255. Das heißt es sind 256 ($\widehat{=}2^8$) Graustufen darstellbar. Möchte man ein farbiges Bild in diesem Format darstellen, so besteht jedes Pixel aus 3 Farbkanälen. Ein Pixel p ist somit mathematisch nichts anderes als ein Vektor mit drei ganzzahligen Werten, welche die Farben Rot, Grün und Blau (RGB) darstellen.

$$p = [p_0 \quad p_1 \quad p_2] \qquad \text{mit } p_i \in \mathbb{N}_0,\ p_i < 256,\ i = 0,\ 1,\ 2 \tag{2.3}$$

[3] Vorfilterung bei PNGs bedeutet, nicht alle Pixel müssen abgespeichert werden. Fehlende Pixel werden vom Anzeigeprogramm berechnet. Auch PNGs mit Vorfilterung können zur Verarbeitung genutzt werden, da sie verlustfrei komprimiert sind.

Eine m×n Bild-Matrix P bestehend aus m·n Pixeln $p_{i,j}$ hat dann folgende Form:

$$P = \begin{bmatrix} p_{0,0} & p_{0,1} & \cdots & p_{0,n-1} \\ p_{1,0} & p_{1,1} & & \\ \vdots & & \ddots & \vdots \\ & & & \\ p_{m-1,0} & & \cdots & p_{m-1,n-1} \end{bmatrix} \tag{2.4}$$

Setzt man für jedes Pixel $p_{i,j}$ den entsprechenden Pixelvektor aus Gleichung 2.3 ein sieht die Bild-Matrix P folgendermaßen aus:

$$P = \begin{bmatrix} \begin{bmatrix} p_{0,0,0} & p_{0,0,1} & p_{0,0,2} \end{bmatrix} & \cdots & \begin{bmatrix} p_{0,n-1,0} & p_{0,n-1,1} & p_{0,n-1,2} \end{bmatrix} \\ \vdots & \ddots & \vdots \\ \begin{bmatrix} p_{m-1,0,0} & p_{m-1,0,1} & p_{m-1,0,2} \end{bmatrix} & \cdots & \begin{bmatrix} p_{m-1,n-1,0} & p_{m-1,n-1,1} & p_{m-1,n-1,2} \end{bmatrix} \end{bmatrix} \tag{2.5}$$

Um nun Informationen verstecken zu können, muss man zunächst überlegen, wo dies sinnvoll wäre. Das Containermedium sollte, wie in der Einführung beschrieben, so wenig wie möglich verändert werden. In erster Linie soll der Betrachter des Bildes keinen Verdacht schöpfen. Daher liegt es nahe nur kleine Eingriffe vorzunehmen. Zum Beispiel ist in einem diskreten Wertebereich von 0 bis 255 die kleinstmögliche Änderung ein Addieren oder Subtrahieren mit 1. Mathematisch ist die Veränderung sofort erkennbar, wenn der Ausgangswert bekannt ist. Bewirkt diese Rechnung allerdings nur einen Farbumschlag, der eine Farbe eines Pixels nur um $\frac{1}{256}$ ändert, dann wird das für einen Menschen nur schwer zu erkennen sein, selbst wenn alle Kanäle aller Pixel manipuliert sind und er das originale Bild kennt. Auf dieser Basis können nun Algorithmen implementiert werden, die Bildcontainer mit Nutzdaten füllen (siehe Abschnitt 3.1.3).

2.3 Steganalyse

2.3.1 Allgemeines

Analog zur Kryptoanalyse aus Abschnitt 2.1.2 beschäftigt sich die Steganalyse mit den Schwachstellen steganographischer Algorithmen und deren Implementierungen. Dabei besteht eine klare Abgrenzung zur Kryptoanalyse, da es hier nicht das Ziel ist, die Information zu entschlüsseln. In der Steganographie ist es, bei Kenntnis der verwendeten Methode, keine Schwierigkeit die Daten aus dem Container zu extrahieren. Neben der Extraktion der Daten ist die eigentliche Herausforderung, herauszufinden, ob Informationen eingebettet wurden. Die Erkennung von Steganogrammen ist die wichtigste Aufgabe in der Steganalyse. Nur so

kann man aus den Containern gewonnene Informationen sinnvoll weiterverarbeiten und auswerten. Denn wie eingangs erwähnt, ist es durchaus möglich, dass die versteckten Nutzinformationen verschlüsselt sind und dann aufwendige Kryptoanalyse betrieben werden muss. Würden hier die falschen Datensätze aus Containern ohne eingebettete Informationen ausgewertet, wäre die Analyse hinfällig.
Zur Detektion im digitalen Bereich können hauptsächlich zwei verschiedene Ansätze verfolgt werden. Auf der einen Seite können Filter Veränderungen bemerkbar (sichtbar/hörbar) machen. Allerdings ist es dabei ohne den Originalcontainer sehr schwierig zu erkennen, ob der Container manipuliert wurde. Im Abschnitt 3.1.5 sind visuelle Filter für Bildcontainer erklärt.
Eine andere Möglichkeit bietet die statistische Analyse. Oft haben Medien einen gewissen Zusammenhang. Wenn Bilder Motive zeigen oder Audiodateien bestimmte Muster aufweisen, so stecken schon oft große Teile der Bildinformation in kleinen Teilen der Nutzdaten. Auf diese Abhängigkeit hin kann untersucht und es können Aussagen getroffen werden, mit welcher Wahrscheinlichkeit es sich um ein bzw. um kein Steganogramm handelt. Eine solche statistische Methode wird im Abschnitt 3.1.4 für die Anwendung auf Bilddateien implementiert.

2.3.2 Statistische Analyse in der Steganographie

Die statistische Analyse beschäftigt sich damit, Anomalien zu finden, die im Vergleich zu unmanipulierten Containern auftreten. Die stochastischen Eigenschaften der Containermedien werden hierbei untersucht. Dabei ist die Kenntnis der Eigenschaften unmanipulierter Container ebenso wichtig, wie das Wissen darum, was eine Einbettung bestimmter Informationen für Auswirkungen auf diese Eigenschaften hat. Je mehr Informationen dem Analysten bekannt sind, desto sicherer ist seine Vorhersage, ob es sich um ein Steganogramm handelt. Besitzt er zum Beispiel ein Steganogramm und den zugehörigen Originalcontainer, so kann er mit hundertprozentiger Sicherheit sagen, ob Veränderungen am Container vorliegen und welche das gegebenenfalls sind. Diese Informationen sind aber nicht hinreichend, um die Nutzinformation ohne Kenntnis des Implementierungsalgorithmus zu extrahieren. Denn es besteht die Möglichkeit, dass manche Veränderungen im Trägermedium zur Verschleierung getätigt wurden. Diese würden dem Analysten zwar als Differenzen zum ihm bekannten Original auffallen, jedoch tragen sie nicht zur Zusammensetzung der Nutzinformationen bei. Deshalb ist es bei dieser Betrachtung wichtig, das Szenario einzugrenzen. Im Speziellen werden hier die sogenannten Known-Steg-Fälle (Bezeichnung analog zur Kryptoanalyse aus Abschnitt 2.1.2) untersucht, bei denen die Art der Informationseinbettung bekannt ist und daher auch beliebige Steganogramme erzeugt werden können.
Konkrete Vorgehensweisen bei der Untersuchung von Containermedien sind von der Art des Containers und der Art der Einbettung abhängig. Am Beispiel der Bildverarbeitung soll das allgemeine Prinzip einer solchen statistischen Analyse kurz verdeutlicht werden.
Orientiert man sich am Ansatz aus Abschnitt 2.2.2, die kleinstmögliche Veränderung am

Containerbild vorzunehmen, so liegt eine LSB[4]-Substitution nahe. Hierbei werden die niederwertigsten Bits jedes ausgewählten Bytes erst gelöscht und anschließend durch die Einzelbits der Nachricht ersetzt (Beschreibung siehe Abschnitt 3.1.3). Um die maximale Informationsmenge zu verstecken und das Beispiel zu veranschaulichen, werden sämtliche Bytes des Bildes manipuliert. Damit stehen der Analyse zwei Container zur Verfügung, die sich nur in den LSBs der einzelnen Pixel-Bytes unterscheiden können.
Nach „Angriffe auf steganographische Systeme“ von A. Westfeld [Westfeld] können die Bytes in verschiedene Wertepaare (engl. Pairs of Values bzw. PoVs) eingeordnet werden. Ein Wertepaar repräsentiert zwei benachbarte Zahlenwerte, die ein Pixel annehmen kann. Die Menge B von diskreten Werten (Bytes) kann also in eine Menge W von Wertepaaren projiziert werden.

$$B = \{0,\ 1,\ 2,\ \ldots,\ 254,255\} \tag{2.6}$$

$$W = \{0,\ 1,\ 2,\ \ldots,\ 126,\ 127\} \tag{2.7}$$

Wobei B in W projiziert wird:

$$P : B \longrightarrow W, \qquad \text{mit } W \subset B \tag{2.8}$$

Die Projektion folgt dem Schema in Abbildung 2.2.

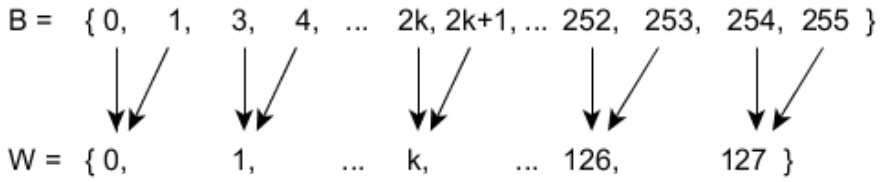


Abbildung 2.2: Bildung der Wertepaare für eine 8-Bit Farbdarstellung

Mit Hilfe dieser Ereignismengen wird nun die jeweilige Häufigkeit des Auftretens der Ereignisse gezählt. Dazu wird eine Menge von Zählwerten Z definiert, die die Auftrittshäufigkeit gültiger Ereignisse aus B zählt. Dabei entspricht der Index in Z dem Ereignis aus B.

$$Z = \{z_0,\ z_1,\ \ldots,\ z_{254},\ z_{255}\} \tag{2.9}$$

So zählt z_0 die Auftrittshäufigkeit von Nullbytes, z_1, die von Bytes mit dem Wert 1 usw. Bildet man nun Z in W ab, so muss man lediglich die Häufigkeiten der zusammengehörigen Wertepaare addieren und das Ergebnis dem Wert mit dem zugehörigen Index aus W

[4]Least Significant Bit

zuordnen.

$$Z_W \quad = \quad \{\overline{z}_0,\ \overline{z}_1,\ \ldots\ ,\ \overline{z}_{126},\ \overline{z}_{127}\} \tag{2.10}$$

Wobei

$$\overline{z}_k \quad = \quad z_{2k} + z_{2k+1} \qquad \text{mit } k \in \mathbb{N}_0, \quad 0 \leq k \leq 127 \tag{2.11}$$

gilt.
Sind Z_W und Z bekannt, was mit Vorliegen des Containerbildes zutrifft, kann aus Z_W ein Erwartungswert z^* berechnet werden. Ausgehend von einer nahezu Gleichverteilung der LSBs nach dem Verstecken von Informationen, sollte die Auftrittshäufigkeit einer Seite der Summe aus Gleichung 2.11 der Hälfte der Gesamtauftrittshäufigkeit entsprechen:

$$z^*_{2k} = z^*_{2k+1} \quad = \quad \frac{\overline{z}_k}{2} \tag{2.12}$$

Ist diese Art von Verteilung im Containerbild vorhanden, so ist es wahrscheinlich ein Steganogramm.
Ein Nachweis kann nach der Arbeit von A. Westfeld (siehe: [Westfeld]) mit dem Chi^2-Test erfolgen:
Die Chi^2-Verteilung (χ^2-Verteilung) nach Pearson ist die Verteilungsfunktion der Quadratsumme von normalverteilten Zufallsvariablen:

$$\chi^2_f \quad = \quad X_1^2 + X_2^2 \ \ldots\ X_f^2 \qquad \text{für} \quad f = 1, 2, \ldots \tag{2.13}$$

Dabei heißt χ^2_f: *Chi-Quadrat-Verteilung mit f Freiheitsgraden*. Über die Dichtefunktion dieser Verteilung trifft er folgende Aussage:

$$d_f(x) \quad = \quad \begin{cases} 0 & \text{für } x \leq 0 \\ \dfrac{e^{-\frac{x}{2}} x^{\frac{f}{2}-1}}{2^{\frac{f}{2}}\Gamma(\frac{f}{2})} & \text{für } x > 0 \end{cases} \tag{2.14}$$

Wobei $\Gamma(r)$ für die *Eulersche Gammafunktion* steht.
Mit diesen Hilfsmitteln werden Prüfwerte ermittelt und deren Abweichung zu den theoretisch zu erwartenden Werten festgestellt. Nach Pearson ist dafür im Allgemeinen folgender Zusammenhang geeignet (Indizes an Gleichungen 2.9 und 2.12 für gerade Farbwerte angepasst):

$$\chi^2_{k-1} \quad = \quad \sum_{i=0}^{k-1} \frac{(z_{2i} - z^*_{2i})^2}{z^*_{2i}} \tag{2.15}$$

z_i - Beobachtungswert aus 2.9

z_i^* - Erwartungswert nach Gleichung 2.12

k - Freiheitsgrade, wobei die Freiheitsgrade $f = k - m - 1$ mit der Anzahl m der geschätzten Werte abnimmt

Der letzte Schritt ist die Ermittlung des Wahrscheinlichkeitswertes (p-Wertes nach [Westfeld]) durch Integration der Dichtefunktion (Gleichung 2.14):

$$p \;=\; 1 - \int_0^{\chi^2_{k-1}} d_{k-1}(x)\,\mathrm{d}x \tag{2.16}$$

Dieser p-Wert erlaubt eine (prozentuale) Aussage darüber, wie nahe die ausgewertete Stichprobe an einer Gleichverteilung der einzelnen Wertepaare liegt, also mit welcher Wahrscheinlichkeit ein Steganogramm vorliegt.

2.4 Neuronale Netze

2.4.1 Allgemeines

Künstliche Intelligenz (KI) oder auch artificial intelligence (AI) ist ein Teilgebiet der Informatik. Der Schwerpunkt dieses Gebietes liegt bei der Erforschung des menschlichen Verhaltens, bzw. der Automatisierung von Computern, Probleme eigenständig zu lösen. Da „Intelligenz“ bzw. „intelligentes menschliches Verhalten“ noch nicht eindeutig definiert ist, ist es schwer, eine genaue Definition für die KI festzulegen. Die künstliche Intelligenz stellt aber auch ein Werkzeug dar, mit dem man Theorien der Intelligenz empirisch testen kann. Vorreiter auf diesem Gebiet ist A.M. Turing, der bereits 1950 einen Aufsatz verfasste, in dem er sich die Frage stellte, wie man feststellen kann, ob ein Programm intelligent handelt. Bei dem inzwischen sehr bekannten Turing-Test kommuniziert eine Testperson über einen Computer mit zwei nicht sichtbaren Partnern. Der eine Partner ist dabei ein Mensch, der andere ein Programm. Das Programm gilt als intelligent, wenn die Testperson nicht unterscheiden kann mit welchem Partner sie gerade kommuniziert. Die dabei gestellten Fragen können aus beliebigen Gebieten stammen. Schachcomputer oder Computer zur medizinischen Diagnose spiegeln einen eingeschränkten Turing-Test wider. Die Wahrnehmung bleibt beim Turing-Test jedoch unberücksichtigt [Wichert, 2000].
Die Schwächen eines Computers liegen im emotionalen Bereich sowie beim logischen Schließen und Lernen. Die künstliche Intelligenz soll im allgemeinen den Alltag und die Arbeit erleichtern sowie Zeit und Kosten sparen .
In bildverarbeitenden Systemen soll es der KI in der Regel gelingen, durch Kameras mit ihrer Umwelt zu agieren. Durch Algorithmen werden die Bilder für künstliche Intelligenz vereinfacht. Die wesentlichen Dinge, wie das Erkennen einer Tür oder Treppe stehen im Vordergrund [Dorn u. Gottlob, 1999].

Die Anwendbarkeit solcher Musteranalysen soll in Hinblick auf die Erkennung von Steganogrammen untersucht werden.

2.4.2 Funktionsweise

Neuronale Netze sind dem menschlichen Gehirn nachempfunden. Sie bestehen aus Neuronen (auch Units genannt), die in Schichten (Layern) angeordnet sind. Die Units eines Layers sind mit den Units des angrenzenden Layers verbunden. Je nachdem, wo sich eine Unit im Netz befindet trägt sie eine andere Bezeichnung (siehe Abb. 2.3).

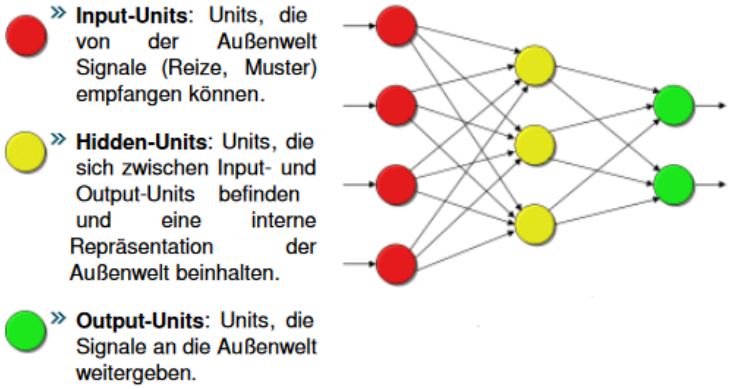


Abbildung 2.3: Schematischer Aufbau eines neuronalen Netzes [neuronalesnetz.de]

Neben den Input- und Output-Units gibt es noch die Hidden-Units. Von diesen Neuronen kann es beliebig viele in solchen Netzen geben. Ein neuronales Netz kann so zum Beispiel auch aus mehreren Hidden-Layers bestehen.
Jeder Unit hat als Eigenschaft ein bestimmtes Gewicht. Diese Gewichte beschreiben, wie stark eine Unit die mit ihr verbundenen Units beeinflusst. Ein höheres Gewicht bedeutet größeren Einfluss. Das „Lernen“ eines solchen Netzes ist also im Prinzip die Gewichtsveränderung der Units abhängig von Input-Reizen. Wie sich die Gewichte verändern ist dabei von der Lernregel und der Netzart abhängig. Die große Schwierigkeit bei der Entwicklung eines solchen Netzes ist also die Auswahl und Struktur des Netzes und die (mathematische) Formulierung der Lernregeln [neuronalesnetz.de].

3 Analyse

Dieses Kapitel beschäftigt sich mit der Arbeit „Implementierung eines Steganographie- und Kryptographie-Verfahrens auf einem FPGA mit Betriebssystemanbindung“ von D. I. Dauch [Dauch, 2015]. Insbesondere sollen seine steganographischen Implementierungen untersucht werden. Da es sich um ein komplexes System handelt, werden die zu untersuchenden Teile entsprechend isoliert. Für jeweilige Analysen wurden Teile der kryptographischen und steganographischen Funktionen seines Verfahrens in der Programmiersprache Python nachgebildet. Auch von ihm angesprochene analytische Methoden wurden anhand der Literaturquellen programmiert und angewendet. Auf Basis dieser Informationen werden in Abschnitt 3.2 Möglichkeiten erarbeitet, Steganogramme solcher Art zu erkennen.

3.1 Anforderungen

Das beschriebene System soll einen sicheren Nachrichtenaustausch mit Hilfe von HTTP oder FTP zwischen mehreren Teilnehmern ermöglichen. Dabei soll ein beliebiges Dateiformat verwendet werden können.
Um diese Sicherheit zu gewährleisten, werden verschiedene Stufen der Verschlüsselung bzw. Verschleierung (Steganographie) durchlaufen. Die steganographischen Stufen sollen insbesondere untersucht werden. Abbildung 3.1 zeigt ein Blockdiagramm der zu untersuchenden Stufen.

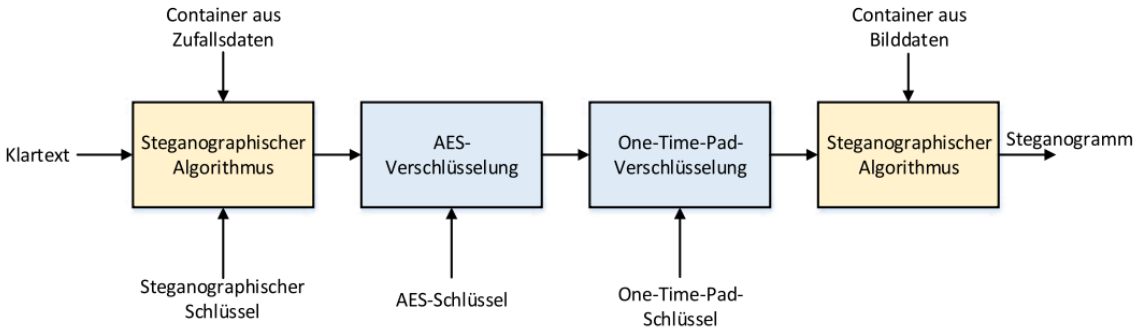


Abbildung 3.1: Systembeschreibung der Klartextverschlüsselung (gelb: Softwarekomponente, grau: Hardwarekomponente, Quelle: [Dauch, 2015])

Die Nutzinformationen werden, bevor sie das System mittels AES und OTP (One-Time-Pad) verschlüsselt, zunächst in einen Container aus Zufallsdaten eingebettet. Ziel dabei ist

es, den AES sicherer zu machen, da die Klartextinhalte damit schwerer zu erraten sind und ein *known plaintext*-Angriff somit ebenfalls erschwert wird. In den nächsten Abschnitten werden die Verfahren kurz erläutert und diskutiert.

3.1.1 Nutzinformationen

Die Nutzinformationen, hier als Klartext bezeichnet, sollen gesichert übertragen werden. Dazu müssen zunächst die Eigenschaften des Klartextes definiert werden. Mit Berücksichtigung der AES-Verschlüsselung wurde dieser in 128-Bit-Blöcke unterteilt. Weiterhin arbeitet das System mit einer Webcam, die farbige (3-kanalige) PNG-Bilder der Auflösung 640x480 als Container erzeugt, womit diese jeweils Platz für 7200 Blöcke bieten. Die inhaltliche Aufteilung zeigt Abbildung 3.2.

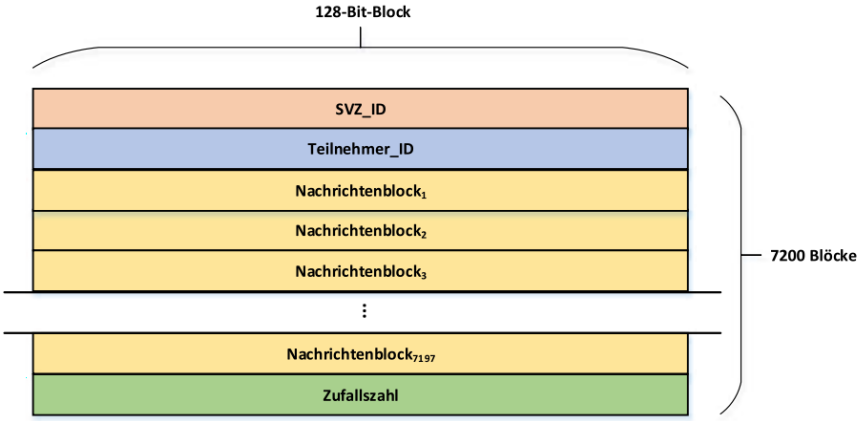


Abbildung 3.2: Nachrichtenaufbau des Klartextes (Quelle: [Dauch, 2015])

Man beachte, dass die SVZ-ID (Schlüsselverteilungszentrale) sowie die Teilnehmer-ID aus vordefinierten ID-Mengen stammen und sich in den Nachrichten oft wiederholen werden. Die Nutzinformationen der einzelnen Blöcke sowie die Zufallszahl am Ende sind in ihrem Inhalt beliebig bzw. zufällig. Um diese Klartextdaten, wie in Abbildung 3.1 gezeigt, in einen Container aus Zufallsdaten einzubetten, ist zunächst ein steganographischer Schlüssel nötig. Dieser bestimmt an welcher Stelle des Containers die Nachricht bzw. Teile der Nachricht eingebettet werden. Dauch generiert dazu aus dem 128-Bit-AES-Schlüssel 125 verschiedene Halbbytes, sogenannte Nibble,. Diese setzen sich wie folgt zusammen:

$$
\begin{array}{lrcl}
\text{AES - Schlüssel:} & q_{AES} & = & \{b_0,\ b_1,\ \ldots\ ,\ b_{127}\} \\
\text{Nibble 0:} & N_0 & = & \{b_0,\ b_1,\ b_2,\ b_3\} \\
\text{Nibble 1:} & N_1 & = & \{b_1,\ b_2,\ b_3,\ b_4\} \\
 & & \vdots & \\
\text{Nibble 124:} & N_{124} & = & \{b_{124},\ b_{125},\ b_{126},\ b_{127}\}
\end{array}
$$

Das Programm generiert zur Einbettung einen Container aus Pseudozufallszahlen, der das 16-fache der Größe der Klartextnachricht aufweist. Der Wert des ersten Nibbles $N_0 \in \{0,\ 1,\ \ldots,\ 15\}$ bestimmt nun, welches der 16 ersten Containerbytes mit dem ersten Nachrichtenbyte überschrieben wird. Die übrige Einbettung folgt demselben Muster. Da nicht alle Nachrichtenbytes mit 125 Nibbles eingefügt werden können, werden die Nibble wiederverwendet.
Die Hauptproblematik dieses Verfahrens wurde bereits vom Autor selbst diskutiert. Der Informationsgehalt der Nachricht leidet stark, da nur noch 1/16 der Nachricht aus Nutzinformationen besteht. Im Sinne der effizienten Informationsübertragung ist dies ein hoher Preis. Insbesondere, da dieses Verfahren Teil eines übergeordneten steganographischen Algorithmus ist, bei dem nochmals zusätzliche Datenmengen verursacht werden. Aus diesem Grund wurde dieses Verfahren auch bei Dauch als abschaltbar implementiert.
Weiterhin nennt er die Verbesserung der Sicherheit des AES als Grund für die Anwendung dieser Methode. Grundlegend stimmt diese Aussage, da der AES anfällig für *known-plaintext*-Angriffe ist (vgl. [Tews et al., 2007]). Die Nutzung der Zufallscontainer macht es dem Angreifer also nahezu unmöglich, den genauen Klartext zu erraten bzw. zu berechnen. Dennoch gibt es in den Nachrichten sich wiederholende Muster in Form der IDs und möglicherweise auch in den Nachrichtenblöcken. Sollten also genügend Informationen vom Angreifer gesammelt werden können, die den gleichen AES-Schlüssel verwenden, ist ein Brechen dieses Systems denkbar.
Da allerdings das Brechen des AES nicht Gegenstand dieser Arbeit sein soll und der genaue Inhalt der Klartextnachricht wenig Einfluss auf die allgemeine Beschaffenheit der AES-verschlüsselten Nachricht hat, wurde auf die Nachbildung dieses steganographischen Verfahrens verzichtet. Im Sinne der maximalen Nutzdatenübertragungsrate wurde die Nachrichtenstruktur aus Abbildung 3.2 zur Weiterverarbeitung der Daten beibehalten.

3.1.2 Kryptographische Methoden

AES

Der AES ist ein aktuelles, international bekanntes sowie standardisiertes Verschlüsselungsverfahren. Es ist, wie in der Einführung angesprochen, ein sicheres Verfahren, wenn einige Randbedingungen erfüllt sind. Diese beziehen sich auf dessen Implementierung und Anwendung. Python bietet im Modul `pycrypto` (Python Cryptography Toolkit)[1] eine sehr gut dokumentierte und leicht zu implementierende Lösung. Da diese Module öffentlich zu-

[1] siehe: https://pypi.python.org/pypi/pycrypto

gänglich sind und schon oft eingesetzt wurden, wird davon ausgegangen, dass die Implementierung hinsichtlich der Effizienz und Sicherheit hinreichend gut ist. Daher wird dieses Modul zur Verschlüsselung des Klartextes genutzt.
Die größte Bedeutung für die Detektion von AES-verschlüsselten Nachrichten hat die Informationsbeschaffenheit einer solchen Nachricht. Ist es bei Klartext noch so, dass abhängig von der Sprache verschiedene Zeichen häufiger vorkommen als andere, verursacht eine Verschlüsselung eine annähernde Gleichverteilung von Symbolen. Im Klartext ließe sich also durch Ermitteln der Häufigkeit jedes Zeichens erkennen, dass es sich um eine (natürliche) Sprache handelt. Meist kann sogar festgestellt oder eingegrenzt werden, um welche Sprache es sich handeln könnte[2]. Eine Häufigkeitsermittlung von Symbolen in AES-verschlüsselten Texten lässt dies nicht zu.

One-Time-Pad

Das One-Time-Pad-Verfahren (OTP) ist eine polyalphabetische Substitutionschiffre. Verwendet man ein einfaches Substitutionsverfahren, tauscht zum Beispiel alle Buchstaben einer Nachricht mit seinem Nachfolger aus (Cäsar-Chiffre [Schmeh, 2009]), so deutet das Vorkommen jedes Zeichens auf einen eindeutigen Klartextbuchstaben (B$\rightarrow$ A, C$\rightarrow$ B, usw.). Diese Verfahren nennt man monoalphabetisch. Bei einem polyalphabetischen Verfahren trifft diese eindeutige Abbildung nicht zu. Per Definition besteht der Schlüssel des OTP aus einer Zufallsfolge, die die gleiche Länge wie die Nachricht aufweist und wird nur einmalig verwendet. In der modernen Kryptographie werden Nachricht und Schlüssel in ihrer binären Darstellung Exklusiv-Oder-verknüpft. Auf der Zufälligkeit und Einmaligkeit dieses Schlüssels basieren die Sicherheit, aber auch die Nachteile des OTP-Verfahrens. Hauptsächlich entstehen Schwierigkeiten bei der Erzeugung und dem Umgang mit dem Schlüssel. In erster Linie ist es in der Computertechnik nicht leicht, echte Zufallsfolgen zu erzeugen. Zusätzlich muss der Schlüssel für jede Nachricht neu erzeugt werden und das in vollständiger Länge der Nachricht, was einen großen Rechenaufwand bedeutet. Zuletzt muss dieser lange, unhandliche Schlüssel auch noch übertragen werden, was eine potentielle Gefahr darstellt, da der Schlüssel abgehört werden könnte. All diese Probleme machen das Verfahren für die Praxis unbeliebt. Um es trotzdem anwenden zu können, wird es oft vereinfacht, was die Sicherheit erheblich beeinflussen kann. So wird in manchen Anwendungen aus einem kurzen, zufälligen Schlüssel ein langer Schlüssel erzeugt, der dann wie im OTP eingesetzt wird, womit die totale Sicherheit, die das Verfahren gewährleistet, nicht mehr gegeben ist.
In dem vorliegendem System findet ebenfalls eine Vereinfachung statt. Das Schlüsselaustauschproblem wird von Dauch gelöst, indem der Schlüssel aus dem Containerbild bzw. dem Steganogramm heraus bestimmt wird. Er nutzt die Bits b_1 bis b_3, um das Schlüsselbit

[2]Die Häufigkeiten gewisser Symbole in verschlüsselten Texten waren oft Schwachpunkte älterer Verschlüsselungsverfahren, da man durch sie auf häufig verwendete Buchstaben im Sprachgebrauch und somit auf das Verfahren und dessen Schlüssel selbst schließen konnte.

im vorliegenden Pixel auszuwählen.

$$n_{key} = (b_3 b_2 b_1)_2 \qquad \text{mit} \quad n_{key} \in \{1, \ldots, 7\} \tag{3.1}$$

Mit diesem Schlüsselbit wird dann das einzubettende Nachrichtenbit Exklusiv-Oder-verknüpft und anschließend eingefügt.

$$b_{cyph} = b_m \oplus b_{n_{key}} \tag{3.2}$$

b_{cyph} - Verknüpfungsergebnis, Bit des Chiffretextes

b_m - aktuelles einzubettendes Bit der zu verschlüsselnden Nachricht

b_{n_key} - Bit des Containers, dass zur Verschlüsselung der Nachricht ausgewählt wurde

Der Vorteil besteht darin, dass kein gesonderter Schlüsselkanal benötigt wird und die verschlüsselten Nutzdaten, die eingebettet werden, selbst bei gleichem Klartext und gleichem AES-Schlüssel immer eine andere Struktur aufweisen. Dies erschwert eine statistische Analyse (vgl. Abschnitt 3.1.4).
Wenn allerdings dem Angreifer im Sinne des Kerckhoffs'schen Prinzips das Verfahren bekannt ist, dann bietet das System keine Sicherheit mehr, da der Schlüssel sozusagen als Klartext im Container enthalten ist. Weiterhin stellen manche steganographische Algorithmen, die die Bits im Container einbetten, Hürden für die Anwendung dieser Methode dar. Wendet man das in Abschnitt 3.1.3 beschriebene erweiterte LSB-Verfahren an, dann kann sich nach der Einbettung von b_{cyph} gegebenenfalls das gesamte Containerbyte ändern, womit eine Schlüsselextraktion auf die eben beschriebene Art nicht mehr möglich ist, da sich die Bits zur Schlüsselbitberechnung geändert haben. Möchte man dieses Verfahren dennoch anwenden, müssten die Schlüssel eventuell aus vorherigen Steganogrammen berechnet werden oder alternativ doch über einen gesonderten Kanal, was die Handlichkeit und Anwendbarkeit seines Systems einschränkt.
Für den weiteren Verlauf der Analyse muss aufgrund dieser Erkenntnisse beachtet werden, dass die Datenbeschaffenheit nicht oder nur sehr eingeschränkt zur Steganalyse genutzt werden kann. Eine Nachbildung des OTP-Verfahrens ist jedoch zur Analyse und Detektion von Steganogrammen nicht zwingend notwendig, da die Datenstrukturen der versteckten Nachrichten sich gegenüber den AES-verschlüsselten Daten kaum ändern.

3.1.3 Steganographische Methoden

D. I. Dauch beschreibt in seiner Arbeit zwei Methoden, um Steganogramme aus den Containerbildern der Webcam zu erzeugen. Das erste Verfahren ist die einfachste Art, Daten zu implementieren, die LSB-Substitution oder auch LSB-Replacement. Die zweite Methode, die hier als *erweiterte LSB-Methode* bezeichnet werden soll, basiert auf einem LSB-

Matching-Verfahren (vgl. [Böhme, 2010]), bei dem abhängig vom Nachrichtenbit die Pixel-Bytes inkrementiert bzw. dekrementiert werden. Diese beiden Methoden wurden implementiert (vgl. Abschnitt 4.1), um die steganalystischen Ansätze aus Abschnitt 3.2 anwenden und evaluieren zu können. Im Folgenden sind beide Methoden kurz erläutert.

LSB-Substitution

Die LSB-Substitution ist ein sehr einfaches Verfahren. Die Nutzdaten einer beliebigen Nachricht m mit der Länge i werden in ihre binäre Darstellung überführt. Jedes Bit dieser Folge wird nun in ein Byte des Containers C^{mxn} mit eingebettet. Dazu formt man, unter der Annahme $i = m \cdot n$ diesen Bitvektor m so um, dass er der Struktur der $m \times n$ Containermatrix entspricht:

$$m = [b_0\, b_1\, \ldots\, b_{i-1}] \tag{3.3}$$

$$M^{mxn} = \begin{bmatrix} [b_0 & b_1 & b_2] & \ldots & [b_{n-3} & b_{n-2} & p_{n-1}] \\ \vdots & & \ddots & & \vdots & & \\ & & \ldots & & [b_{i-3} & b_{i-2} & b_{i-1}] \end{bmatrix} \tag{3.4}$$

Im nächsten Schritt werden die LSBs des Containers gelöscht. Dazu wird die Containermatrix mit einer entsprechenden Bitmaske D, welche an allen Postionen mit der Zahl 254 ($\widehat{=}(11111110)_2$) belegt ist, elementweise Und-verknüpft (hier $\&$)[3].

$$C_{del} = C \,\&\, D \tag{3.5}$$

Um das Steganogramm S zu erzeugen, muss lediglich der bearbeitete Container C_{del} mit der Nachricht-Matrix M Oder-verknüpft (hier $||$)[4] werden.

$$S = C_{del} \,||\, M \tag{3.6}$$

Nun wurden alle LSBs des Containers mit den Bits der Nachricht ausgetauscht. Sollte die Nachricht nicht lang genug sein, um den Container vollständig zu füllen, können die ungenutzten LSBs in ihrem Ursprungszustand belassen werden. Eine Einbettung von mehr Nutzbits als es Bytes im Container gibt, ist nicht vorgesehen. Bei längeren Nachrichten müssen diese also auf mehrere Steganogramme verteilt werden.

[3] hier erzeugt der $\&$-Operator aus den beiden Operanden-Matrizen eine Matrix, bei der jedes Element dem Und-Verknüpfungsergebnis der jeweiligen Einzelelemente der Operanden entspricht

[4] hier erzeugt der $||$-Operator aus den beiden Operanden-Matrizen eine Matrix, bei der jedes Element dem Oder-Verknüpfungsergebnis der jeweiligen Einzelelemente der Operanden entspricht

Erweiterte LSB-Methode

Die im System von Dauch angewandte erweiterte LSB-Methode verursacht zwar größere Veränderungen am Containerbild, erschwert jedoch eine statistische Detektion des Steganogramms. Grund für diese erschwerte Detektion ist eben diese größere Veränderungen der Pixelbytes. Bei dem einfachen LSB Verfahren nutzt der in Abschnitt 2.3.2 beschriebene χ^2-Test die Tatsache aus, dass die Häufigkeiten nebeneinanderliegender Farbwerte plötzlich gleichmäßig verteilt sind, anstatt, wie in unbearbeiteten Containern, unabhängige Häufigkeiten aufzuweisen. Dieser Effekt bleibt aus, wenn man das gesamte Byte durch In-/Dekrementieren verändert. Dadurch führt die Einbettung einer 0 oder 1 nicht zwingend dazu, dass sich nur ein Bit ändert. Folgendes Beispiel soll diesen Effekt verdeutlichen:

Beispiel: Folgende Binärzahlen sollen bei Auftritt einer einzubettenden 1 genau dann um 1 dekrementiert werden, wenn das LSB nicht mit dem Nachrichtenbit übereinstimmt.

Binärzahl	**1er-Dekrement**
001	001
010	001
011	010
⋮	⋮
100	011
⋮	⋮
100000000	011111111

Tabelle 3.1: Bedingtes 1er-Dekrement verschiedener Binärzahlen

Das Beispiel zeigt wie sich, abhängig vom Ausgangsbyte, verschieden viele Bits ändern. Womit die Häufigkeit der Entstehung von Wertepaaren des χ^2-Tests, die sich nur im LSB unterscheiden, vermindert wird.
Es soll wieder eine Nachricht der Länge i eingebettet werden. Das Verfahren selbst beginnt, wie die LSB-Substitution, mit dem Erstellen der Nachricht-Matrix M (siehe Gleichung 3.4). Bei der Maskierung des Containers C müssen hier allerdings drei Fälle unterschieden werden.

Fall 1: Das LSB des Containerbytes ist ungleich dem LSB der Bytes der Matrix M.

$$LSB(c_k) \neq m_k \qquad \text{mit } k = 0,\ 1,\ 2,\ \ldots,\ i$$

Hierbei müssen alle betreffenden Bytes um 1 dekrementiert werden.

Fall 2: Das LSB des Containerbytes ist gleich dem LSB des Bytes der Matrix M.

$$LSB(c_k) = m_k \qquad \text{mit } k = 0,\ 1,\ 2,\ \ldots,\ i$$

Hierbei bleiben die Bytes, auf die diese Eigenschaft zutrifft, unverändert.

Fall 3: Das Containerbyte ist 0 und das LSB des Containerbytes ist ungleich dem Nachrichtenbyte.

$$c_k = 0, \quad LSB(c_k) \neq m_k \qquad \text{mit } k = 0,\ 1,\ 2,\ \ldots,\ i$$

In diesem Fall müssen die Containerbytes um 1 inkrementiert werden.

Die Funktion $LSB(\cdot)$ auf ein Element einer Matrix gibt das LSB aus der binären Darstellung der Zahl zurück. Weiterhin sind c_k und m_k Elemente der Matrizen C bzw. M. Um diese Fälle mit Matrizenoperationen behandeln zu können, sind mehrere Masken nötig. Zunächst wird die Maske zur Filterung der Elemente, auf die Fall 1 zutrifft, erstellt. Dazu wird eine Matrix O benötigt, die die gleichen Dimensionen wie der Container aufweist und vollständig mit Einsen gefüllt ist. Mit dieser kann man aus dem Container C eine Matrix C_{LSB} berechnen, die aus den LSBs des Containers besteht.

$$C_{LSB} \ = \ O \,\&\, C \tag{3.7}$$

Mit Hilfe der Exklusiv-Oder-Verknüpfung ($\oplus$) entsteht eine Maske $D_{\neq}$, die an den Stellen, an denen sich die LSBs von C und M unterscheiden mit 1 belegt ist.

$$D_{\neq} \ = \ C_{LSB} \ \oplus \ M \tag{3.8}$$

Man beachte, dass bei Gleichung 3.8 streng genommen Bytes mit Bits verknüpft werden. Dies stellt hier kein Problem dar, da die Matrix M zwar sinngemäß die Bits der einzubettenden Nachricht m enthält, diese jedoch in der Praxis auch als 8-Bit-Zahlen (*uint8*) abgespeichert werden. Somit ist eine mathematische Verknüpfung der beiden Matrizen zulässig. Da in diesem Verfahren allerdings ganze Bytes betrachtet werden, wird eine Maske benötigt, die nicht mit 0 und 1 belegt ist, sondern mit 0 und 255, sodass bei Maskierung des Containers alle Bits der zu bearbeitenden Bytes übernommen werden. Dazu nutzt man eine *clip*-Funktion mit folgenden Eigenschaften:

$$clip(x) \ = \ \begin{cases} 0 & \text{für } x = 0 \\ 255 & \text{für } x > 0 \end{cases} \tag{3.9}$$

Die Anwendung von *clip(·)* auf eine Matrix bedeutet elementweises clippen in der Matrix, sodass alle Nullen 0 bleiben und alle Elemente >0 den Wert 255 annehmen.

Mit

$$D_{\neq,clipped} = clip(D_{\neq}) \tag{3.10}$$

ist die Grundmaske zur Behandlung der Fälle 1 und 3 berechnet.
Mit diesem Ergebnis ist die Maske für den zweiten Fall ebenfalls schnell berechnet. Es muss lediglich die binäre Inverse (*not(·)*)[5]

$$D_{=,clipped} = not(D_{\neq,clipped}) \tag{3.11}$$

gebildet werden.
Für den dritten Fall bietet sich die Nutzung der bisher berechneten Masken an. Anhand der Differenzenmaske $D_{\neq,clipped}$ können die zu bearbeitenden Bytes des Containers extrahiert werden.

$$C_{\text{diff}} = (D_{\neq,clipped} \,\&\, (C - O)) + O \tag{3.12}$$

Hierbei wird bei der Addition der Einser-Matrix O ein Überlauf des Zahlenbereiches[6] genutzt, um genau die Bytes Null zu setzen, die im Container schon Null sind und geändert werden sollen.
Wendet man nun die *clip*-Funktion auf diesen Differenzencontainer an

$$C_{\text{diff},clipped} = clip(C_{\text{diff}}) \tag{3.13}$$

und addiert wieder die O-Matrix

$$C_{inc} = C_{\text{diff},clipped} + O \tag{3.14}$$

so erhält man, unter Berücksichtigung des Überlaufs, den Teilcontainer C_{inc}, dessen Elemente 1 sind, für die der Fall 3 gilt, und dessen übrige Elemente 0 sind.
Um nun auch für den Fall 1 einen Teilcontainer C_{dec} erstellen zu können, sind noch zwei weitere Schritte nötig.
Zunächst werden der Differenzencontainer aus Gleichung 3.13 und die Differenzenmaske $D_{\neq,clipped}$ miteinander Und-verknüpft.

$$D_{dec} = C_{\text{diff},clipped} \,\&\, D_{\neq,clipped} \tag{3.15}$$

Man erhält dadurch eine Maske D_{dec}, in der alle Bytes für, die Fall 1 zutrifft, mit 255 belegt sind (alle anderen Bytes sind 0).
Mit dieser Maske erstellt man den Teilcontainer C_{dec} indem der originale Container C um

[5] Binäre Inverse bedeutet, dass alle Bits jedes Elements invertiert werden (nicht zu verwechseln mit der Inversen einer Matrix)

[6] Beim Datentyp uint8, der hier genutzt wird, handelt es sich, aufgrund informationstechnischer Gegebenheiten, um einen Zahlenkreis, d.h. $0 - 1 = 255$

1 dekrementiert wird und anschließend mit D_{dec} Und-verknüpft wird.

$$C_{dec} \quad = \quad C \,\&\, D_{dec} \tag{3.16}$$

Zum Schluss müssen die beiden Teilcontainer sowie der unveränderte Teil des originalen Containers zum Steganogramm S zusammengefügt werden.

$$S \quad = \quad (C_{dec} \,||\, C_{inc}) \,||\, (C \,\&\, D_{=,clipped}) \tag{3.17}$$

3.1.4 Statistische Analyse

Bei der statistischen Analyse, die hier betrieben werden soll, handelt es sich, um bei den Begrifflichkeiten aus Abschnitt 2.1.2 zu bleiben, um eine Art *chosen-plaintext*-Angriff. Nur das dieser auf den steganographischen Algorithmus angewandt wird. Aufgrund der Nachbildung oben beschriebener Implementierungsalgorithmen können für diese Analysen beliebige Steganogramme erzeugt werden. Allerdings müssen einige Einschränkungen beachtet werden, die hier zur Vollständigkeit nochmals zusammengetragen werden.

1. Es dürfen lediglich die statistischen Eigenschaften des Containers betrachtet werden und in Bezug darauf die Veränderungen, die die Einbettung einer Nachricht verursacht. Denn es wird davon ausgegangen, dass die Nachrichten selbst mit immer anderen AES-Schlüsseln verschlüsselt werden und das OTP-Verfahren aus Abschnitt 3.1.2 angewendet wurde. Daher sind Ansätze, die sich auf eine konkrete wiederkehrende Struktur der LSBs (also der Nachricht) beziehen nicht zulässig. Dennoch ist die Nachricht ähnlich einer Zufallsfolge von Bits beschaffen und dieser Aspekt soll bei der Analyse genutzt werden.

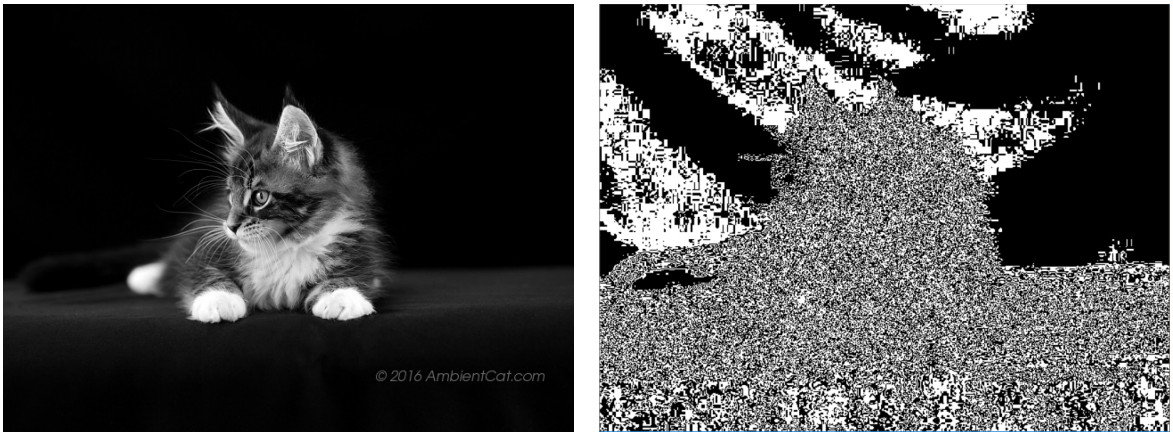
2. Inhalt der Nachricht ist irrelevant, da sich selbst bei immer gleichem Nachrichteninhalt die verschlüsselten Chiffretexte unterscheiden (siehe Punkt 1).

3. Im Allgemeinen sind alle Methoden zur Kryptographie und Steganographie bekannt, die verwendeten Schlüssel allerdings nicht (Kerckhoffs'sches Prinzip).

Ziel dieser statistischen Analysen ist, für jedes mutmaßliche Steganogramm eine Aussage zu erhalten, ob es sich wirklich um ein Steganogramm handelt und bestenfalls noch die Wahrscheinlichkeit auszugeben, mit der diese Aussage zutrifft (vgl. Abschnitt 2.3.2).

3.1.5 Visuelle Filter

Die Idee die hinter visueller Filterung steckt ist die, dass in den LSBs der Bilder viele Bildinformationen stecken und sich so eine subjektive Ähnlichkeit zum Originalbild feststellen lässt. Abbildung 3.3 verdeutlicht diese Idee anhand eines Beispiels.



Quelle des Originalcontainers (links): http://www.ambientcat.com [2016]

Abbildung 3.3: Originalcontainer (links) und zugehöriges gefiltertes Bild (rechts)

Hier wurden die LSBs des Bildes untersucht und mit der *clip*-Funktion bearbeitet.

$$C_{filt} \quad = \quad clip(LSB(C)) \tag{3.18}$$

Obwohl das gefilterte Bild C_{filt} lediglich aus den Informationen der LSBs gewonnen wurde, lassen sich noch im Originalcontainer dagewesene Muster und Konturen erahnen. Besonders bei einfarbigen Flächen, wie sie der obere Hintergrund im linken Bild aufweist, sind bei dieser Art Filter von Vorteil. Bei Farbübergängen oder Flächen mit Mustern, wie dem Fell der Katze oder dem Boden auf dem sie sitzt, beginnt das gefilterte Bild stark zu rauschen. Solche Effekte treten auch bei Farbbildern auf, wie Abbildung 3.4 zeigt.

Abbildung 3.4: Farbiger Originalcontainer (links) und zugehöriges gefiltertes Bild (rechts)

Auch hier ist es günstig für diesen Effekt, wenn das Containerbild kräftige Farben auf großen Flächen aufweist. Denn der weiße Tisch sowie einige dunkle Flächen im Hinter-

grund sind eindeutig als Muster auch nach der Filterung zu erkennen. Selbst zwei der Stifte auf dem Tisch stechen noch heraus.

3.2 Angriffsszenarien

Nachdem alle notwendigen Methoden, die in Dauchs System auftreten, beschrieben und diskutiert wurden, bleibt nur noch die Frage der Angreifbarkeit offen. Im Wesentlichen sollen in dieser Arbeit zwei Szenarien untersucht werden, in denen man mit Hilfe der angesprochenen Analyseverfahren Steganogramme erkennen können soll. Streng genommen, kann allerdings nicht mit Sicherheit gesagt werden, dass es sich um ein Steganogramm handelt. Die umgekehrte Aussage allerdings ist schon eher möglich. Deswegen wird bei diesen Verfahren oft das Augenmerk darauf gelegt, Container auszuschließen, die wahrscheinlich keine Steganogramme sind. Die Implementierungen sind in Kapitel 4 beschrieben. Eine Diskussion über Qualität und Anwendbarkeit dieser Angriffe folgt in Kapitel 5.

3.2.1 Visuelle Angriffe

Der Hauptgedanke hinter dem visuellen Angriff besteht darin, dass ein Mensch mit seinem subjektiven Empfinden eine Aussage darüber treffen kann, ob es sich möglicherweise um ein Steganogramm handelt oder nicht. Er muss dazu lediglich wissen, wie ein Originalcontainer und wie ein Steganogramm jeweils aussehen. Hier stößt man auf das erste Problem: Steganogramm und Container sind ohne Weiteres für einen Menschen selbst im direkten Vergleich nicht zu unterscheiden, wie Abbildung 3.5 zeigt.

Abbildung 3.5: Farbiger Originalcontainer (links) und zugehöriges Steganogramm (rechts)

Um dieses Problem zu lösen, setzt man die im Vorfeld beschriebenen visuellen Filter ein. Hier werden die Unterschiede zwischen Steganogramm und Container plötzlich sehr offensichtlich. Allerdings sind auch hier Einschränkungen zu beachten. Unterschiede werden am besten sichtbar, wenn kontrastreiche Motive und große einfarbige Flächen im Containerbild vorhanden sind, so dass nach der Filterung immer noch Muster zu erkennen sind (siehe

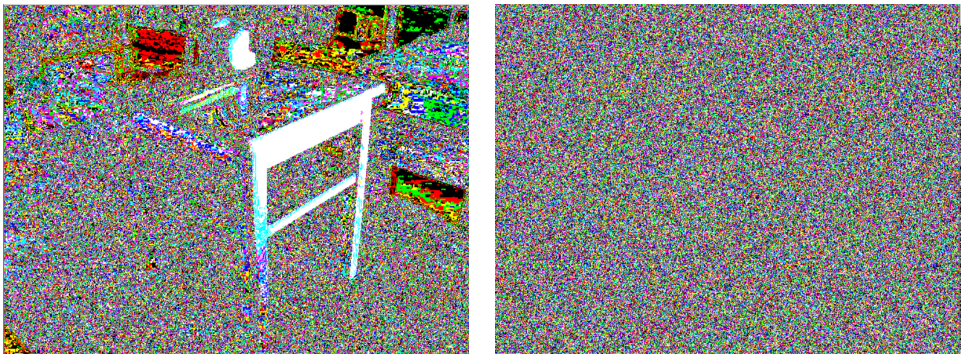


Abbildung 3.6: Gefilterter farbiger Originalcontainer (links) und zugehöriges gefiltertes Steganogramm (rechts)

Abbildung 3.6).
Wenn der Betrachter weiß, dass der Container eigentlich nach der Filterung noch Überreste von Formen aufweisen soll und ein Bild, wie rechts zu sehen vorgelegt bekommt, so handelt es sich vermutlich um ein Steganogramm.
Doch für den Analysten ungünstige Containerbilder, wie in Abbildung 3.7 zu sehen, können zu fehlerhaften Vermutungen führen. In diesem Container sind keine Informationen versteckt, dennoch sieht das gefilterte Bild aus, als könnte es sich um ein Steganogramm handeln. Warum ist das gefilterte Bild so verrauscht? Der Grund hierfür liegt unter anderem in der Qualität der Aufnahme. Die Abbildung 3.5 ist mit einer Standardwebcam mit der Auflösung 640x480 aufgenommen worden. Die Webcam bietet kein besonders scharfes Motiv und verfügt auch nicht über sehr hohe Kontrastwerte. Weiterhin wurden die Bilddaten direkt aus der Webcam ausgelesen und verarbeitet. Somit werden eigentlich sehr kontrastreiche Bereiche mit weniger Werten dargestellt. So gehen feine Farbunterschiede verloren und der weiße Tisch ist auch nach dem Filtern größtenteils weiß geblieben.

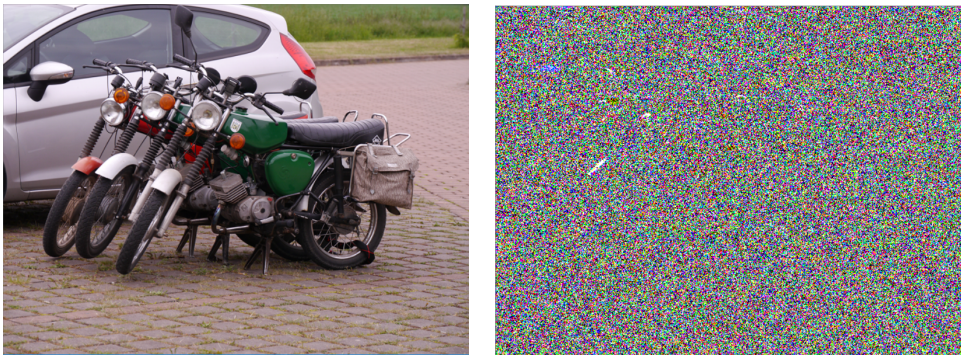


Abbildung 3.7: Farbiger Originalcontainer (links) und zugehöriger gefilterter Container (rechts)

Abbildung 3.7 wurde hingegen mit einer Lumix dmc G5 Systemkamera in einer Auflösung von 4608x3456 aufgenommen und zusätzlich noch von einem anderen verlustbehafteten Bildformat in das PNG-Format konvertiert. Durch diese Operationen auf dem Containerbild und der deutlich höheren Qualität des Foto-Sensors der Systemkamera gehen die Strukturen nach der Filterung verloren. Es sind weniger einfarbige Flächen vorhanden. Nähme man selbiges Motiv jedoch mit der Webcam auf oder würde es aus dem RAW-Format[7] direkt in das PNG-Format konvertieren, entstünde wiederum ein ähnliches Ergebnis, wie in Abbildung 3.6 (links).
In Dauchs Arbeit wird ebenfalls eine Webcam als Bildquelle verwendet, weswegen die Anwendung eines solchen Filterverfahrens dennoch zu richtigen Ergebnissen führen kann. Dazu müssen solche gefilterten Bilder lediglich computergestützt ausgewertet werden können. Die Idee ist hier, die entstehenden einfarbigen Flächen zu markieren und ein Bild zu generieren, das nur noch die eingezeichneten Muster enthält. Die Entscheidung, ob es sich um ein Steganogramm handelt, hängt dann nur noch vom Vorhandensein und gegebenenfalls von der Anzahl der erkannten Muster ab.

3.2.2 Statistische Angriffe

Ein statistischer Ansatz entsteht aus der Überlegung, dass das Rauschen einer Art Gleichverteilung von Farbpunkten über das gesamte Bild entspricht. Während unbearbeitete Container diese Gleichverteilung nicht aufweisen. Setzt man nun einen visuellen Filter, äquivalent zu Abschnitt 3.2.1, ein, so entsteht entweder ein verrauschtes Bild (Steganogramm) oder ein verrauschtes Bild mit partiell einfarbigen Flächen (unbearbeiteter Container). Durch das Erstellen eines Farbhistogramms und dessen Untersuchung, sollte es möglich sein, eine Aussage zu treffen, ob es ein Steganogramm sein könnte oder nicht.
Dieses Prinzip basiert demzufolge auch darauf, solche Flächen zu erkennen. Hier geht es jedoch weniger um die Form, die eine solche Fläche aufweist, sondern eher, um die aus solchen Stellen entstehende Farbverteilung.

[7] RAW - ist im Allgemeinen ein Rohdatenformat (englisch *raw* „roh“)

4 Entwurf

4.1 Nachbildung

Das System von D. I. Dauch wird auf folgende Komponenten reduziert (siehe Abbildung 4.1).

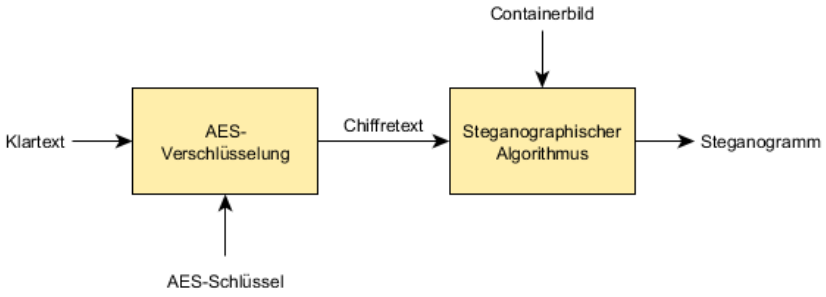


Abbildung 4.1: Blockdiagramm der Nachbildung von Dauchs steganographischem System (gelb: Softwarekomponenten)

Zu Analysezwecken wird eine weitere Softwarekomponente eingefügt, die in diesem Kapitel beschriebene Methoden auf das Steganogramm anwendet.

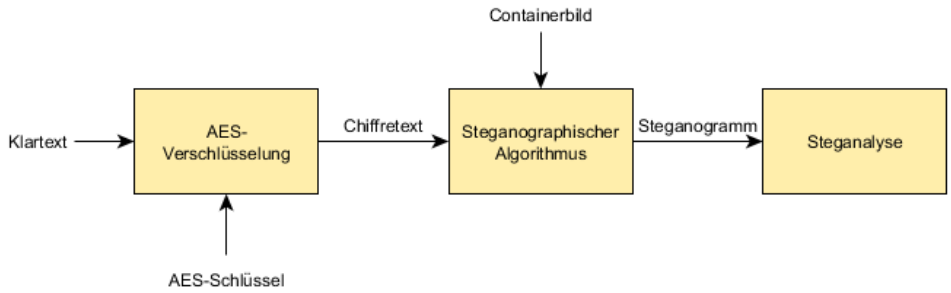


Abbildung 4.2: Blockdiagramm der Nachbildung von Dauchs steganographischem System inklusive der Steganalyse-Komponente (gelb: Softwarekomponenten)

Folgender Programmablaufplan soll den Softwareablauf der Nachbildung verdeutlichen (Abbildung 4.3). Hier ist die Funktion `encryptSingleFrame()` des Moduls `SteganographicSuite` dargestellt.

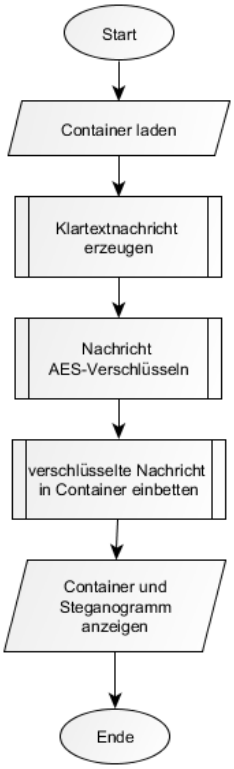


Abbildung 4.3: Programmablaufplan der vereinfachten Nachbildung von Dauch steganographischen System

Da festgestellt wurde, dass der genaue Inhalt der Klartextnachricht irrelevant ist, wird diese einmalig über die Funktion `getBlockMessage()` generiert. Dies geschieht in dieser Funktion abhängig von der Größe des Containers, daher muss dieser im Vorfeld bekannt sein und geladen werden können. Mögliche Quellen für Container sind PNG-Bilder oder auch Bilder, die von der Webcam eingelesen wurden. Das Einlesen des Containers erfolgt unter Verwendung des Pythonmoduls `openCV`[1]. Die AES-Verschlüsselung erfolgt über das Modul `pycrypto`. Das Einbetten der Nachricht ist im Modul `enc` implementiert (siehe Anhang A). Die Implementierung ist äquivalent zu den Beschreibungen aus Abschnitt 3.1.3 umgesetzt worden. Die dazu nötigen Matrizen- und Bit-Operationen stellen die Module `openCV` und `numpy`[2] bereit.

[1] siehe: http://opencv.org/opencv-3-0.html
[2] siehe https://docs.scipy.org/doc/numpy/

Da im Originalsystem Webcambilder verarbeitet werden, wurde auch diese Funktionalität implementiert. Hierzu gehört der Programmablaufplan der Funktion `filterCam()` aus Abbildung 4.4.

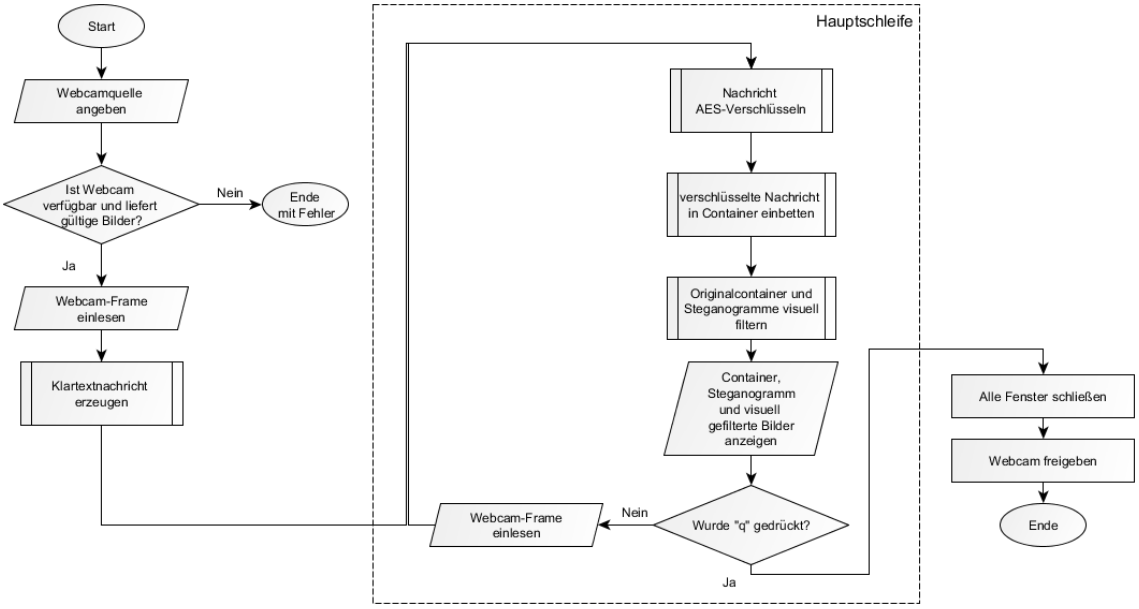


Abbildung 4.4: Programmablaufplan der vereinfachten Nachbildung von Dauch steganographischen System mit Livebildern einer Webcam als Quelle

Im Prinzip ist der Ablauf aus Abbildung 4.3 übernommen worden. Es wird im Vorfeld die Webcam eingehängt und es erfolgt eine kurze Prüfung, ob die Hardware verfügbar und funktionsfähig ist. Wenn diese Prüfung ohne Fehler abläuft, so kann der erste Frame aus der Webcam ausgelesen werden. Auch für diese Vorgänge wird das Modul `openCV` verwendet. Nach der Verschlüsselung und Einbettung der Klartextnachricht werden zusätzlich visuell gefilterte Bilder des Containers und des Steganogramms erzeugt, um direkte Auswirkungen des Motivs auf die Beschaffenheit des gefilterten Bildes zu zeigen. Es erfolgt eine Ausgabe aller Bilder.
Wird während des Vorgangs die Taste „q“ gedrückt, so wird der Prozess unterbrochen, alle Fenster werden geschlossen und die Webcam wird freigegeben.
Wird kein Tastendruck auf „q“ erkannt, liest das Programm den nächsten Frame ein.

4.2 Statistischer Angriff

Der in Abschnitt 2.3.2 beschriebene χ^2-Angriff zeigt eine sehr rechenaufwendige Art potentielle Steganogramme zu untersuchen. Besonders die Integration kann die Berechnungszeit

in die Höhe treiben.
Daher ist es das Ziel, ein Verfahren zu entwickeln, dass ohne solche Schritte auskommt und trotzdem ein verwertbares Ergebnis liefert. Dazu werden zwei Gegebenheiten des Originalsystems von Dauch ausgenutzt. Die erste Gegebenheit besteht in der Nutzung einer einfachen, handelsüblichen Webcam, die in 640x480 aufgelöste Bilder liefert. Solche Webcams sind mit einfachen Fotosensoren bestückt, die Filter-Effekte, wie sie in Abschnitt 3.2.1 beschrieben sind, hervorrufen können.
Der zweite Aspekt, der die Analyse vereinfacht, ist die Beschaffenheit der Daten in Kombination mit dem LSB-Verfahren. In seiner Masterarbeit beschreibt Dauch, wie er mit Hilfe des erweiterten LSB-Verfahrens (vgl. Abschnitt 3.1.3) sein System gegen den χ^2-Angriff immunisiert. Das ändert allerdings nichts an der Tatsache, dass die Informationen ausschließlich in den LSBs des Containers zu finden sind. Weiterhin unterliegen diese Informationen, aufgrund ihrer Verschlüsselung, einer Art Gleichverteilung (vgl. Abschnitt 3.1.2).

4.2.1 Beschreibung

Das Verfahren zur Erkennung von Steganogrammen basiert auf den Farbhistogrammen der visuell gefilterten Container. Aufgrund dieser Filterung können die Pixel in ihren jeweiligen Farbkanälen nur die Werte 0 oder 255 annehmen. Somit entsteht eine Menge von 8 verschiedenen Farbwerten, die die x-Achse eines solchen Histogramms bilden.

$$P_{Farbwerte} = \{[0,\ 0,\ 0]\,,\ [0,\ 0,\ 255]\,,\ \ldots,\ [255,\ 255,\ 255]\} \tag{4.1}$$

Diese Farbwerte sind im folgenden mit den Indizes 0 bis 7 bezeichnet. Die y-Achse ist die Häufigkeit des Vorkommens dieser „Farbe“. Ein solches Histogramm ist in Abbildung 4.5 für eine Zeile eines Containers und des dazugehörigen Steganogramms dargestellt.

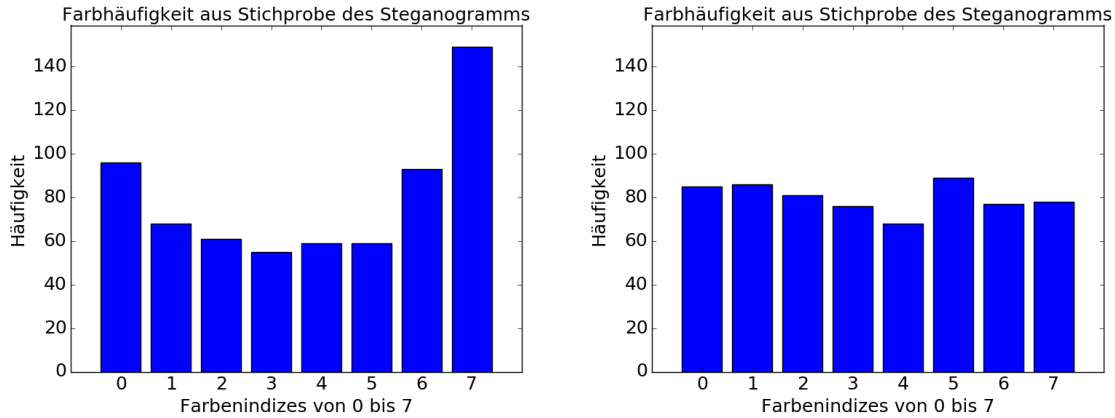


Abbildung 4.5: Farbhistogramme links von einem unbearbeiteten Container, rechts von einem Steganogramm (beide visuell gefiltert)

Dieses Beispiel verdeutlicht, wie stark die Einbettung den Verlauf des Histogramms beein-

flusst.
Um diesen Verlauf auch mathematisch festhalten zu können, werden die Histogrammdaten, wie in Abbildung 4.6 gezeigt wird, analysiert. Das Ergebnis ist ein Indikatorwert, der eine mögliche Einbettung anzeigt.

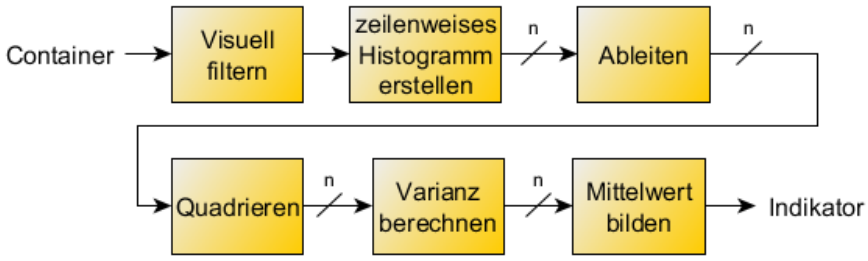


Abbildung 4.6: Blockdiagramm zum Ablauf der Histogrammanalyse

Nachdem das Containerbild mit n Zeilen visuell gefiltert wurde, wird für jede Zeile ein Histogramm erstellt. Es entstehen n Histogramme, bei denen festgestellt werden soll, wie stark die Häufigkeiten der auftretenden Farben (vgl. Abb. 4.5) variieren. Dazu wird die numerische Ableitung für jedes Histogramm gebildet. Diese Ableitungen werden in eine n×7-Matrix ΔH eingetragen.

$$\Delta H = \begin{bmatrix} \Delta h_{1,1}, & \dots, & \Delta h_{1,7} \\ \vdots, & \ddots, & \vdots \\ \Delta h_{n,1}, & \dots, & \Delta h_{n,7} \end{bmatrix} \tag{4.2}$$

Da nach der Differenziation im Allgemeinen auch negative Werte entstehen können, müssen diese durch elementweises Quadrieren beseitigt werden. Die Matrix mit den quadrierten Werten ist im Folgenden ΔH^*.

$$\Delta H^* = \begin{bmatrix} \Delta h^*_{1,1}, & \dots, & \Delta h^*_{1,7} \\ \vdots, & \ddots, & \vdots \\ \Delta h^*_{n,1}, & \dots, & \Delta h^*_{n,7} \end{bmatrix} = \begin{bmatrix} \Delta h^2_{1,1}, & \dots, & \Delta h^2_{1,7} \\ \vdots, & \ddots, & \vdots \\ \Delta h^2_{n,1}, & \dots, & \Delta h^2_{n,7} \end{bmatrix} \tag{4.3}$$

Die Varianz dieser quadrierten Ableitung gibt nun Auskunft darüber, ob die LSBs eher gleichverteilt sind oder ob sie stark variieren. Hierfür wird die empirische Varianz genutzt, die wie folgt definiert ist:

$$s^2 = \frac{1}{n} \sum_{1}^{n} (m_i - \overline{m})^2 \tag{4.4}$$

Hierbei ist m_i eine von $i = 1, ..., n$ Merkmalsausprägungen und $\overline{m}$ der Mittelwert über alle m_i.
Passt man diese allgemeine Varianz an den vorliegenden Fall an, berechnet sich der Indikatorwert I aus den Mittelwert aller Zeilenvarianzen der Matrix ΔH^*.

$$I = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{1}{7}\sum_{j=1}^{7}\left(\Delta h_{i,j}^{*} - \overline{\Delta h_i}^{*}\right)^2\right) \tag{4.5}$$

Dabei deutet ein kleiner Indikatorwert auf eine höhere Wahrscheinlichkeit hin, ein Steganogramm vorliegen zu haben. Je größer der Wert wird, umso sicherer ist es ein unbearbeiteter Container.
Eine genaue Analyse findet sich in Kapitel 5.1.1.

4.2.2 Implementierung

Die Implementierung erfolgt hier entsprechend der Beschreibung aus dem vorangegangenem Abschnitt. Folgender Programmablaufplan soll den Prozess verdeutlichen:

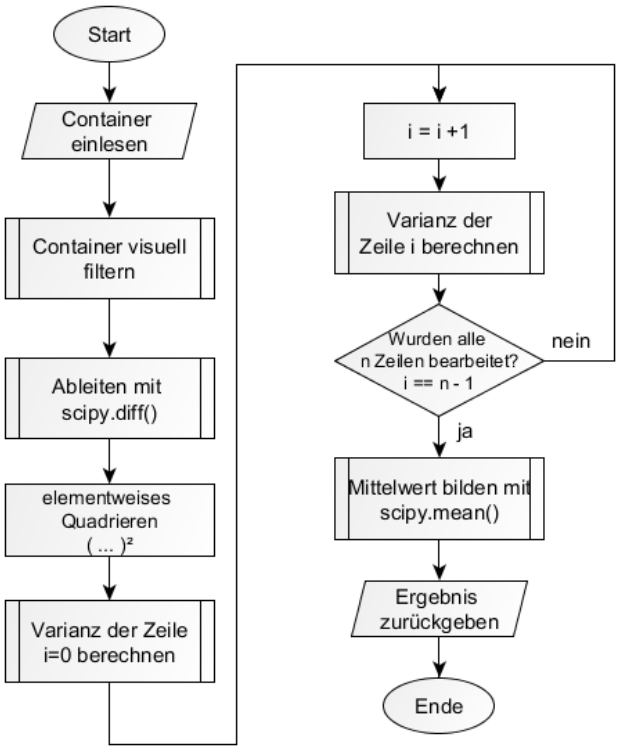


Abbildung 4.7: Programmablaufplan zum Zeilen-Varianz-Verfahren

Zum Einlesen des Containers und für den visuellen Filter wurden die Funktionen

`imread()` und `clip()` aus dem Modul `openCV` genutzt. Die numerische Ableitung, die durch die Wertedifferenz zweier benachbarter Werte einer Zeile beschrieben ist, wurde mit Hilfe des Moduls `scipy`[3] umgesetzt. Die Differenziation erfolgt mit der Funktion `diff()`. Die im Folgeschritt durchgeführte elementweise Quadrierung erfolgt durch einfache Matrixmultiplikation in `Python`. Die Multiplikation zweier Matrizen gleicher Dimension in der Form `Matrix1*Matrix2` ist hier als elementweise Multiplikation implementiert.
Zur anschließenden Berechnung der Varianz wurde eine selbst geschriebene Unterfunktion `evalHist()` genutzt, die die Funktion aus Gleichung 4.4 für beliebig lange Arrays umsetzen kann.

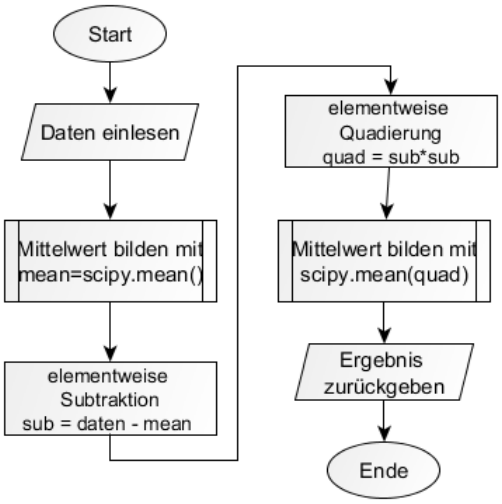


Abbildung 4.8: Programmablaufplan zur Funktion `evalHist()`

Die Abarbeitung der Varianzberechnung jeder Zeile wurde in einer For-Schleife mit dem Zählindex i umgesetzt. Sind alle Zeilen bearbeitet worden, wird aus den Ergebnissen, den Zeilenvarianzen, erneut ein Mittelwert gebildet, der Indikatorwert I.

4.3 Visueller Angriff

Der reine visuelle Angriff auf das Containerbild ist eine theoretische Betrachtung, wie man die visuell gefilterten Bilder (vgl. Abschnitt 3.1.5) ohne menschliches Eingreifen auswerten kann. Dabei soll die Einsetzbarkeit von künstlicher Intelligenz untersucht werden.
Um die Bilddaten kompatibel für das Training eines neuronalen Netzes zu machen, sollten möglichst Muster in den Bildern erkennbar gemacht werden. Eine einfache Mustererkennung bietet die Kantenerkennung nach dem Canny-Algorithmus von `openCV`.

[3] siehe: https://www.scipy.org/

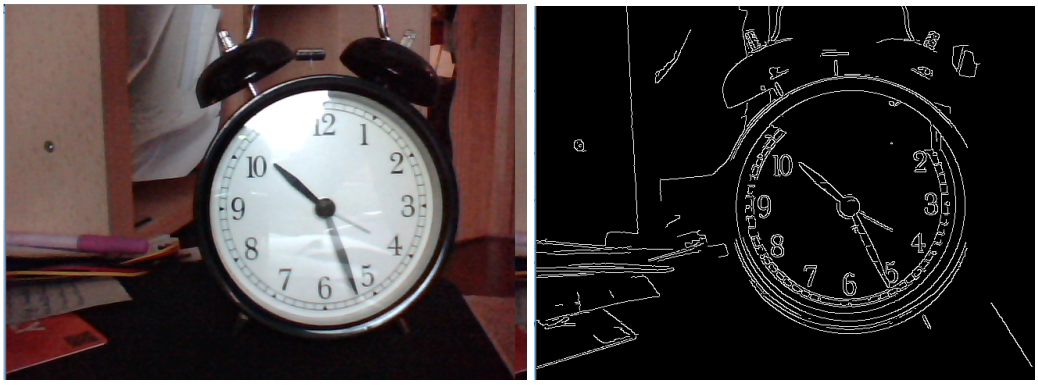


Abbildung 4.9: Anwendung des Canny-Algorithmus auf einen unbearbeiteten Container

Die Kanten und Muster des Motivs sind klar zu erkennen. Ziel dieser Idee ist es, solche Muster in den gefilterten Containern wiederzuerkennen. Wie bereits diskutiert wurde, stecken in den gefilterten Bildern viele Informationen des eigentlichen Containers. Dazu gehören auch Teile dieser Muster, die die Kantenerkennung liefert. Würde das neuronale Netz Muster des ungefilterten Bildes im gefilterten wiedererkennen, ist der Container definitiv kein Steganogramm. Die Kantenerkennung eines gefilterten Bildes zeigt folgendes Ergebnis:

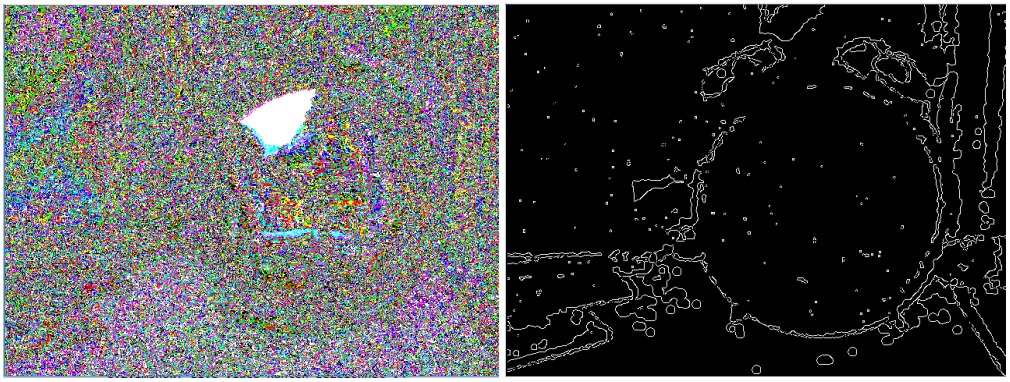


Abbildung 4.10: Anwendung des Canny-Algorithmus auf einen gefilterten Container ohne eingebettete Informationen

Um eine solche Kantenerkennung nach der visuellen Filterung zu ermöglichen sind Zwischenschritte notwendig, die das Rauschen im Bild reduzieren. Dazu wird das gefilterte Bild (links in Abb. 4.10) dem Canny-Algorithmus unterzogen, dann wird ein Weichzeichner (`GaussianBlur` aus `openCV`) angewandt und anschließend ein Threshold-Algorithmus, der nach dem Weichzeichnen die Farben auffrischt. Anschließend wird erneut der Canny-Algorithmus angewandt. Die Parametrierung der jeweiligen Funktionen wurde empirisch ermittelt. Eine analytische Ermittlung ist hier sehr schwierig, da das System äußerst empfindlich auf Motivänderungen reagiert.

Theoretisch sollte es mit einem solchen Ergebnis möglich sein, eine künstliche Intelligenz darauf zu trainieren solche Muster, wie sie in Abbildung 4.9 rechts gezeigt sind, in den gefilterten Container (Abb. 4.10) wiederzuerkennen.
Jedoch wurde auf eine weitere Umsetzung verzichtet, da die durch die Kantenerkennung entstandenen Muster, selbst aus den Originalbildern heraus, nicht optimal sind, aufgrund ihrer teilweisen Unvollständigkeit. In den gefilterten Containern wird der Effekt noch stärker und eine kleine Veränderung in den Lichtbedingungen kann schon zum vollständigen Versagen der Kantenerkennung führen. Da also kein gut reproduzierbarer Input für das Netz unter Laborbedingungen hergestellt werden konnte, wird eine praktische Anwendung dieses Verfahrens wahrscheinlich keinen Nutzen haben.

5 Ergebnis

5.1 Angriffsszenarien

5.1.1 Statistischer Angriff

Im Allgemeinen haben statistische Angriffe den Nachteil, dass sie ressourcenintensiv sind. Besonders der χ^2-Angriff hat aufgrund der mehrfach zu berechnenden Integration schon bei kleinen auszuwertenden Containern eine spürbare Rechenzeit. Eine weitere Problematik besteht in der oft zeilenweisen Auswertung der Container. Diese Methode ist nötig, um Steganogramme, die nur teilweise mit Informationen gefüllt sind, ebenfalls erkennbar zu machen. Die Folge ist eine steigende Rechenzeit mit steigender Containergröße.
Die Komplexität des Zeilen-Varianz-Verfahren aus Abschnitt 4.2 wurde mit Hilfe einer Testreihe untersucht, bei der ein in N-Schritten wachsender Teilcontainer immer wieder dem Analysealgorithmus unterzogen wurde. Für diese Untersuchung ist ausschließlich die Größe des Containerbildes relevant.
Zur besseren Vergleichbarkeit wurden in der Abbildung 5.1 die Kurven, die andere gängige Komplexitäten von Algorithmen beschreiben normiert und eingeblendet.

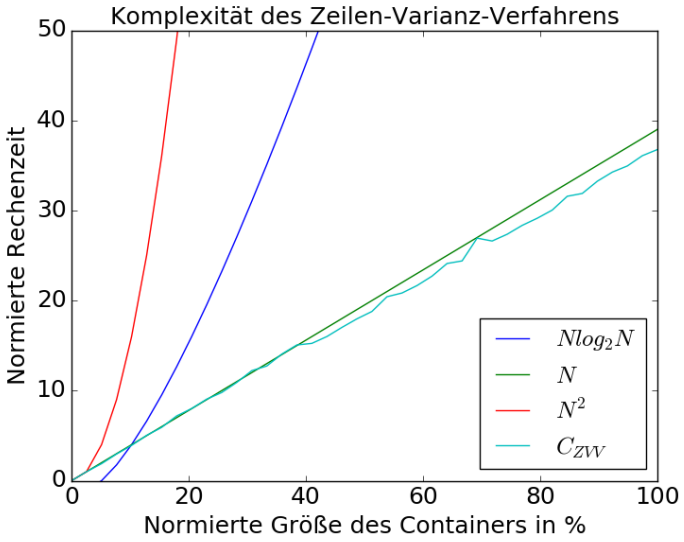


Abbildung 5.1: Komplexitätsuntersuchung des Zeilen-Varianz-Verfahrens

Die Grafik zeigt eine Komplexität C_{ZVV} des Zeilen-Varianz-Verfahrens (ZVV) die annähernd linear ist. Das bedeutet mit Vergrößerung des Containers um das n-fache, verlängert sich die theoretische Rechenzeit ebenfalls um den Faktor n. In der Praxis ist es dennoch denkbar, dass sich längere Rechenzeiten ergeben, da mit wachsender Containergröße irgendwann die Grenzen von Systemkomponenten, wie bspw. dem Arbeitsspeicher erreicht werden.
Als nächstes soll die Zuverlässigkeit des Verfahrens analysiert werden. Auch hier liegt wieder ein Vergleich mit dem bekannten χ^2-Verfahren nahe, da durch die erweiterte LSB-Methode der bisher zuverlässige χ^2-Test ausgehebelt wurde.
Dieses Problem wurde mit dem Zeilen-Varianz-Verfahren schon dadurch umgangen, dass lediglich die LSBs eines Containers untersucht werden. Damit ist das Verfahren unabhängig von den anderen Bits jedes Pixelbytes und kann somit mit einfacher LSB-Substitution genau so umgehen, wie mit der erweiterten Methode. Dennoch hat das Verfahren seine Grenzen.
Die Messung wurde mit verschiedenen Motiven durchgeführt. Das Bildmaterial wurde dabei mit zwei Geräten aufgenommen und untersucht.
In der ersten Messung wurden mit Hilfe der Webcam 100 Bilder aufgenommen und untersucht. Die Untersuchung erfolgte dabei für den jeweiligen Container und das dazugehörige Steganogramm. Die Motive erzeugen dabei ein gefiltertes Bild, dass vom subjektiven Eindruck recht nahe an gefilterten Steganogrammen liegen soll. Abbildung 5.2 zeigt ein solches Bild.

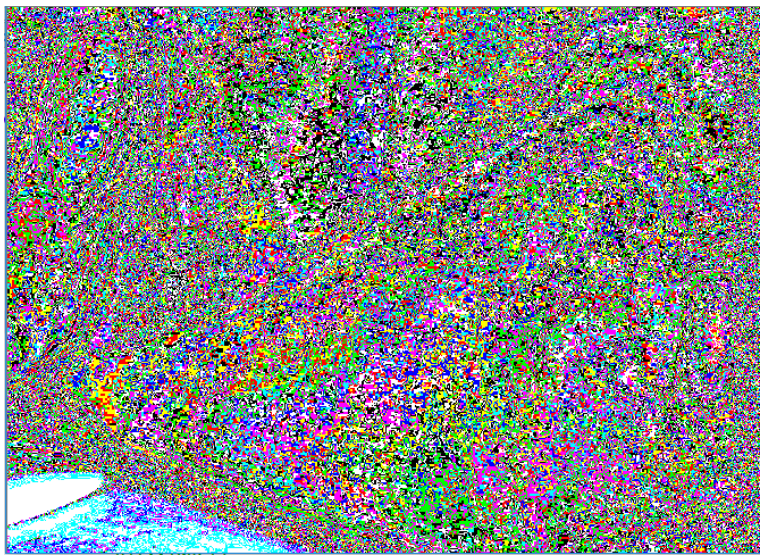


Abbildung 5.2: Beispiel eines für die Messung genutzten, gefilterten Containers ohne eingebettete Nachricht

Während der Messung wurden die Motive variiert, um auch stärker und schwächer rauschende Bilder in die Untersuchung mit einzubeziehen.

Die ersten 55 Bilder der Messung sind steganogrammähnlich und in Abbildung 5.3 dargestellt.

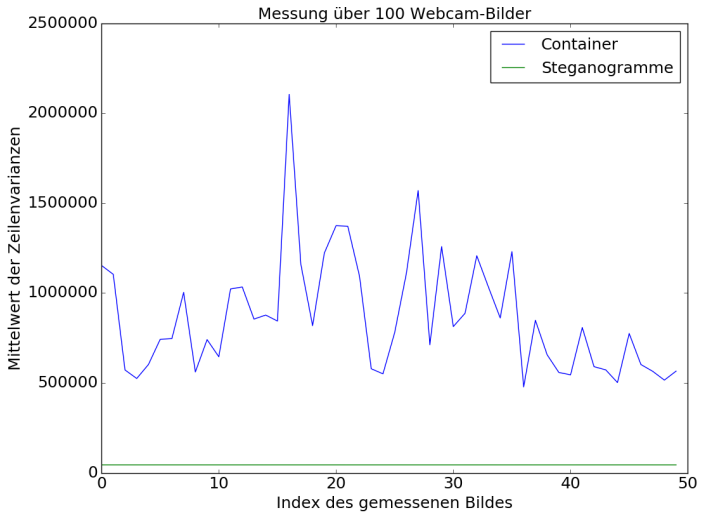


Abbildung 5.3: Messung der ersten 50 steganogrammähnlichen Container (blau) gegenüber 50 wirklichen Steganogrammen (grün)

In dieser Messung ergab sich ein Mittelwert für die grüne Kurve von ca. 44100.
Mit Hilfe dieses Mittelwertes kann nun ein besser einzuordnender Wert im Bereich von 0 bis 100 errechnet werden. Dazu nutzt man die deutlich höheren Zeilenvarianzen der nicht bearbeiteten Container. Der Kehrwert dieser Werte ergibt Werte nahe 0. Multipliziert man diesen Kehrwert mit dem Minimum der Zeilenvarianzen von Steganogrammen (grüne Kurve) dieser Messung ($I_{steg,min} = 44087$), so erhält man einen normierten Indikatorwert I_{norm}. Für Messwerte nahe dem Minimum ergeben sich normierte Werte nahe 1 und für Messerwerte mit hohen Zeilenvarianzen Werte, die sehr nahe an 0 liegen. Dieser hierfür genutzte Minimalwert wurde empirisch aus den vorliegenden und möglichen Messungen ermittelt. Deswegen kann und sollte er für andere Systeme neu ermittelt und verifiziert werden. Die anschließende Multiplikation mit 100 bringt das Ergebnis in den gewünschten Wertebereich 0 bis 100.

$$I_{norm} = \frac{1}{I} \cdot I_{steg,min} \cdot 100 \tag{5.1}$$

Normiert man die gesamte Messung mit dieser Formel ergibt sich Abbildung 5.4.
Mit Hilfe dieser Grafik ist sehr deutlich zu erkennen, wann Bilder Steganogramme sind und wann nicht. Besonders die Leercontainer mit den Indizes 56 bis 96 liegen sehr nahe an der 0-Grenze. Der Grund dafür liegt in der Motivänderung, die an dieser Stelle der Messung

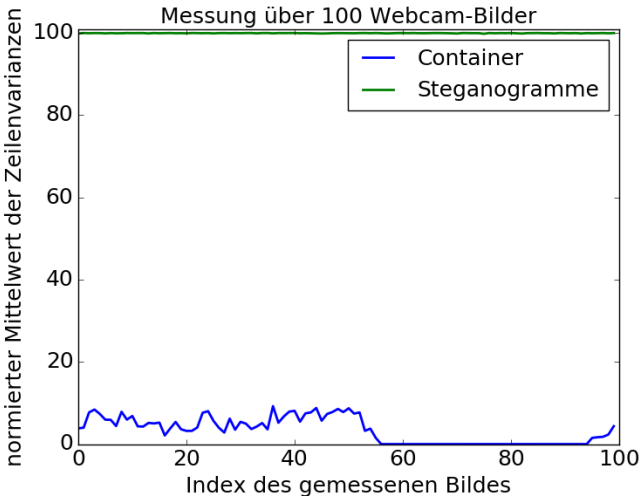


Abbildung 5.4: Graphen der normierten Messung, blau: Container, grün Steganogramme

stattgefunden hat. Statt Motive zu verwenden, die nach dem Filtern steganogrammähnlich aussehen, wurden hier bewusst Container mit großen einfarbigen Flächen erstellt und verwendet. Die Steganogramme, die in diese Container eingebettet wurden, werden nach wie vor sicher erkannt. Wurden die Container jedoch nicht bearbeitet so haben sie extrem hohe Zeilenvarianzen. In welchen Zahlenbereichen sich die Messwerte in einem solchen Fall bewegen zeigt Abbildung 5.5.

Abbildung 5.5: Messung der Container 50-100. Blau: Originalcontainer mit großem Anteil an einfarbigen Flächen. Grün: jeweilige Container als Steganogramme

Hier sind Messwerte möglich die sich teilweise in Bereichen zwischen 4×10^9 und 5×10^9 befinden. Deswegen und aus Gründen der Anwendbarkeit und Übersichtlichkeit wurde die Normierung aus Gleichung 5.1 angewandt.
Um die Funktion der Messung anhand anderer Bildquellen zu untersuchen, werden im Folgenden beispielhaft zwei Bilder einer Lumix dmc G5 Systemkamera vom RAW in das PNG Format (verlustfrei) konvertiert und auf die gleiche Weise, wie oben, untersucht.
Beide Motive weisen große einfarbige Flächen auf. Das linke Bild ist dabei eher kontrastreich gehalten und das rechte Bild eher kontrastarm.

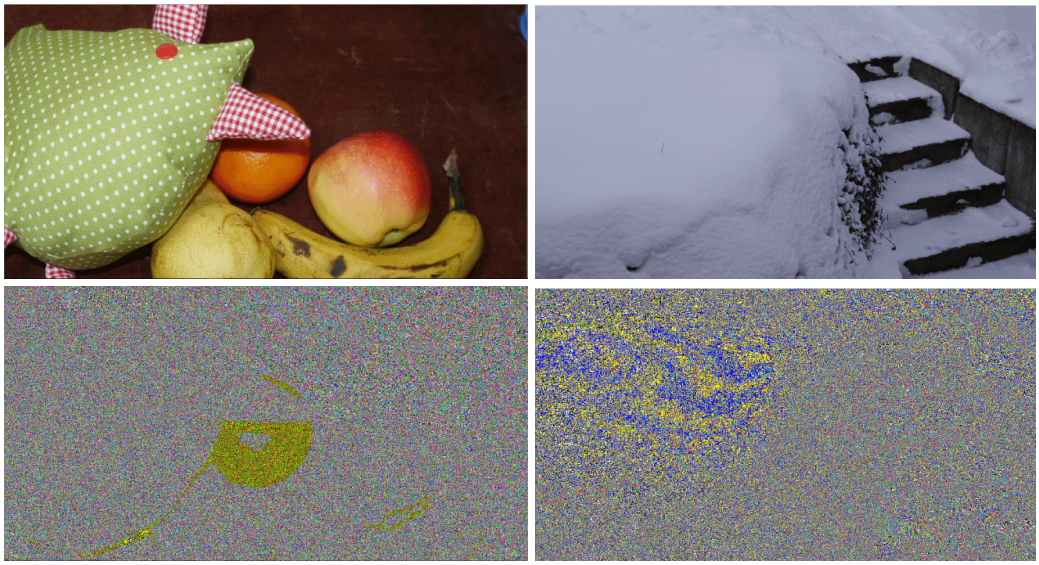


Abbildung 5.6: Links: Kontrastreiches Motiv, Rechts: Kontrastarmes Motiv

Abbildung 5.6 zeigt zwei Motive, die mit der Lumix dmc G5 Systemkamera im RAW-Format aufgenommen wurden. Die Filterung der Motive (untere Bilder) zeigt: auch hier bei diesen verlustfrei konvertierten Bildern treten solche „Muster“-Effekte auf, mit deren Hilfe das Zeilen-Varianz-Verfahren Steganogramme entdeckt. Die Effekte sind aufgrund der hohen Fotoqualität zwar geringer, aber eine Evaluation mit dem ZVV-Algorithmus zeigt trotzdem verwendbare Ergebnisse (siehe Tabelle 5.1).

Bildquelle:	kontrastreiches Motiv	kontrastarmes Motiv	kontrastreiches Steganogramm	kontrastarmes Steganogramm
Mittelwert d. Zeilenvarianz:	8.957×10^6	14.525×10^9	379.683×10^3	379.699×10^3

Tabelle 5.1: Messung der Bilder aus Abb. 5.6 mit dem Zeilen-Varianz-Verfahren

Zuerst fallen die Unterschiede zwischen den unbearbeiteten Motiv-Containern und den Steganogrammen auf. Beide Steganogramme befinden sich sehr nahe beieinander, wohingegen

die Originalcontainer in deutlich anderen Zehnerpotenzen liegen. Dass Steganogramm und Container einen so großen numerischen Unterschied aufweisen, ist sehr gut. Die Problematik besteht nun in der Normierung. Offensichtlich kann der für die Webcammessung bestimmte Mittelwert hier nicht eingesetzt werden, da damit die Steganogramme nicht erkannt würden. Für diese Bildquelle wäre eine Neubestimmung dieses Normierungsfaktors nötig, um sichere Ergebnisse zu erzielen, was einen Nachteil für die Anwendung des Verfahrens darstellt. Bei den Untersuchungen fiel auf, dass dieser Normierungsfaktor zum großen Teil von Bildformat und -größe abhängt. Es müssen also für jedes zu untersuchende System erst Daten erhoben werden, die die unterschiedlichen Bereiche in denen sich die Messwerte befinden können, bestimmen. Dies ist aber keine Hürde, da das Steganographieverfahren bekannt ist. So kann mit Hilfe eines beliebigen Containers, unabhängig davon, ob Daten eingebettet wurden oder nicht, ein Referenzsteganogramm erstellt und analysiert werden. Aus der Analyse dieses Steganogramms kann dann ein Normierungsfaktor berechnet werden.
Ein weiterer Anhaltspunkt für die Untersuchung kann ein Vergleich der Mittelwerte der Zeilenvarianzen sein. Diese Mittelwerte variieren bei unbearbeiteten Containern immer stark (über mehrere Zehnerpotenzen). Sind die Bilder Steganogramme, dann schwanken die Messwerte zwar auch, werden aber aus der bisherigen Erfahrung heraus immer in dem Bereich einer Zehnerpotenz bleiben. Man kann also Container, deren Mittelwert der Zeilenvarianzen stark von den Mitteln der Zeilenvarianzen anderer Container abweicht, als unbearbeitete Bilder einordnen.
Es ist also sicher, dass die Containerbeschaffenheit das Bild beeinflusst. Ein weiterer Extremfall soll die Grenzen dieses Algorithmus aufzeigen. In Abschnitt 3.2.1 in Abbildung 3.7 ist ein Container gezeigt, der ohne vom steganographischen Algorithmus bearbeitet worden zu sein aussieht, als wäre er ein Steganogramm. Es wurde bereits festgestellt, wie solche Effekte entstehen (z.B. durch Konvertierung aus verlustbehafteten Formaten). Untersucht man diesen Container mit dem Zeilenvarianzverfahren, so erhält man folgende normierte Indikatorwerte:

Art des Bildes:	Container	Steganogramm
Mittelwert d. Zeilenvarianzen:	44754.03	44100.04
normierter Indikatorwert:	98.51	99.97

Tabelle 5.2: Auswertung des Verfahrens für einen Extremfall

Da es sich hier wieder um einen Container der Größe 640x480 handelt, liegt die mittlere Zeilen-Varianz des Steganogramms wieder bei ca. 44100, weswegen der empirisch ermittelte Normierungsfaktor aus der ersten Messung wiederverwendet wurde. Der normierte Indikatorwert des Steganogramms liegt zwar noch über dem des unbearbeiteten Containers, jedoch ist der Unterschied zu gering, um hier den Container aus der Menge der potentiellen

Steganogramme auszuschließen.
Alle Steganogramme, in denen verschlüsselte Informationen in Bildcontainern eingebettet wurden, können mit diesem Verfahren erkannt werden. Lediglich das Einbettungsverfahren muss bekannt sein. Das heißt, es können nach der Anpassung der Filterfunktion auch Steganogramme anderer Verfahren und theoretisch sogar anderer Containerarten erkannt werden. Trotzdem können Fälle auftreten, in denen Container ohne versteckte Nachricht fälschlicherweise als Steganogramme erkannt werden. Es trat jedoch bei den bisherigen Untersuchungen nie auf, dass ein Steganogramm als unbearbeiteter Container eingeordnet wurde.

5.1.2 KI-gestützter visueller Angriff

Der KI-gestützte visuelle Angriff hat sich als schwer umzusetzen erwiesen. Auch dessen Anwendbarkeit, Effizienz und Zuverlässigkeit wären fragwürdig gewesen, da, neben den in Abschnitt 4.3 genannten Punkten, noch weitere Probleme auftreten.
Es ist nicht trivial ein neuronales Netz auf die Erkennung von Mustern zu trainieren. Schon für einfache Fälle kann es sehr aufwendig sein, Lernregeln mathematisch zu formulieren. Da bei diesem Szenario sehr viele verschiedene, auch abstrakte Muster auftreten können, wäre ein Training also sehr umfangreich gewesen. Eine andere Möglichkeit wäre, ein Netz jeweils auf ein bestimmtes Szenario (bzw. System) zu trainieren. Das ist jedoch aufwendig und da es einfachere und zuverlässigere Methoden gibt, wäre eine künstliche Intelligenz nicht das Mittel der Wahl.
Ein weiterer Punkt ist die Zuverlässigkeit. Bei den Ansätzen, dieses Verfahren umzusetzen hat sich bereits gezeigt, kleine Einflüsse im Bild können große Veränderungen in der Kantenerkennung bewirken, was zu Fehlern und Falscherkennungen führt. Da man als Stenalyst auch keinen Einfluss auf die Motive, die ein steganographisches System liefert, hat, ist die praktische Anwendung noch weiter eingeschränkt.

5.2 Fazit

Es wurden zwei Ansätze zur Detektion von Steganogrammen untersucht. Wie im letzten Abschnitt angedeutet, ist der Aufwand, ein neuronales Netz zur Erkennung von Steganogrammen zu programmieren, viel zu groß, gegenüber einem zuverlässigeren und solideren statistischen Verfahren, welches auf der gleichen Grundlage basiert.
Das Zeilen-Varianz-Verfahren hat aufgrund seiner Einfachheit und Schnelligkeit einige Vorteile. So können ohne Probleme viele Daten analysiert werden und kein Steganogramm würde unerkannt bleiben, wenn das Verfahren bekannt und die Nutzdaten verschlüsselt sind. Allerdings ist auch hier die Anwendbarkeit begrenzt. Sobald ein System ein Verfahren, wie die *erweiterte LSB-Methode* zur Steganogrammerzeugung nutzt und die Bildbeschaffenheit, wie im Extremfall (siehe Abb. 3.7) ist, so kann ein Steganogramm nicht mehr vom unbearbeiteten Container unterschieden werden. Im Gegensatz zum χ^2-Test würde das Zeilen-

Varianz-Verfahren jedoch die Daten immer als Steganogramme anzeigen. Außerdem hat man bei der Untersuchung eines solchen Falls, den Hinweis darauf, dass es sich um ein verdächtiges System handelt und man kann die Steganalyse mit anderen Mitteln betreiben.

5.3 Ausblick

Die Ergebnisse dieser Analyse schließen eine Eignung von künstlicher Intelligenz zur Detektion von Steganogrammen nicht aus. Lediglich der Ansatz, der gewählt wurde, ist ungeeignet.
Das statistische Verfahren ist, um die Entwicklung der Idee zu verdeutlichen, sehr einfach und damit einhergehend auch ineffizient programmiert. Der visuelle Filter ist beispielsweise nicht nötig, um die darauffolgenden Schritte durchzuführen. Stattdessen müssten lediglich die Wertigkeiten der LSBs untersucht werden. Daher sind hier noch Optimierungen möglich. Weiterhin ist das Verfahren bisher an zwei Containerquellen getestet worden. Um die Zuverlässigkeit des Verfahrens besser einschätzen zu können, müssen Erfahrungswerte aus vielen anderen Quellen gesammelt werden. Besonders interessant wäre hierbei, ob durch den Algorithmus unbekannte Systeme untersucht und richtig eingeordnet werden können. Theoretisch könnte das Verfahren genutzt werden, um nicht nur einzelne Steganogramme aus bekannten Systemen zu erkennen, sondern auch, um Systeme daraufhin zu untersuchen, ob sie Steganographie nutzen.
Weiterhin kann dieser Algorithmus auch auf andere Containerarten übertragen werden. Sobald ein Angreifer in der Lage ist, die Daten aus den richtigen Stellen zu extrahieren, kann er sie auf ihre Varianzen untersuchen. Auch hier wäre eine Untersuchung hinsichtlich Containern der Formate MP3 oder GIF aufschlussreich.

6 Zusammenfassung

Für diese Arbeit wurden zwei Ansätze zur Detektion von Steganogrammen untersucht. Dafür wurde, nach der Erläuterung der Grundlagen in Kapitel 2, das vorliegende Beispielsystem von Dauch nachgebildet und für die Testzwecke angepasst (siehe Kapitel 3 und 4). Unter diesen Testbedingungen wurden die erstellten Methoden analysiert.
Der Ansatz zur KI-gestützten visuellen Analyse, wurde während der Erarbeitung nicht mehr weiterverfolgt, da die zu erwartenden Ergebnisse nicht zuverlässig waren.
Das erarbeitete statistische Verfahren dagegen, bietet in gewissen Grenzen eine Zuverlässige Detektion von Steganogrammen am Testsystem an. Darüber hinaus, wie Kapitel 5 beschreibt, wächst die Rechenzeit des entwickelten Zeilen-Varianz-Verfahren linear mit der Größe des Problems. Im direkten Vergleich zu anderen statistischen Methoden ist dieser Algorithmus nur von den eingebetteten Nutzdaten abhängig und ist so immun gegen Änderungen des Einbettungsalgorithmus, was beispielsweise beim χ^2-Verfahren nicht der Fall ist. Auch wenn die Anwendbarkeit des Zeilen-Varianz-Verfahrens von der Containerbeschaffenheit begrenzt ist, werden keine Steganogramme fälschlicherweise als leere Container angezeigt. Der gegenteilige Fall, bei dem Container ohne eingebettete Daten als Steganogramme angezeigt werden, ist jedoch möglich, erfordert aber ein bestimmtes Muster der Ausgangscontainer.
Beim Erstellen dieser Arbeit ist eine Toolbox entstanden, mit der speziell das eingangs erwähnte System von D.I. Dauch untersucht werden kann, aber auch allgemeine Möglichkeiten zur Erstellung und Analyse von Steganogrammen zur Verfügung gestellt werden.

Literaturverzeichnis

[Böhme 2010] BÖHME, Dr. Rainer ; MAURER, Prof Dr. Ueli (Hrsg.) ; BASIN, Prof Dr. David (Hrsg.): *Advanced Statistical Steganalysis*. Springer-Verlag Berlin Heidelberg, 2010. – ISBN 978–3–642–14312–0.

[Dauch 2015] DAUCH, Dominique I.: *Implementierung eines Steganographie- und Kryptographieverfahrens auf einem FPGA mit Betriebssystemanbindung*, Hochschule für Angewandte Wissenschaften Hamburg sowie Fachhochschule Westküste, Masterarbeit, 2015. – .

[Dorn u. Gottlob 1999] DORN, J. ; GOTTLOB, G.: *Künstliche Intelligenz*. http://www.ec.tuwien.ac.at/~dorn/Papers/hi00.pdf. Version: 1999. – abgerufen am 12.04.2017.

[Kerckhoffs 1883] KERCKHOFFS, Auguste: La cryptographie militaire. In: *Journal des sciences militaires* (1883), Januar, Nr. 9, 5-38. http://www.petitcolas.net/kerckhoffs/crypto_militaire_1.pdf. – abgerufen am 12.04.2017.

[neuronalesnetz.de] NEURONALESNETZ.DE: *Neuronale Netze, Eine Einführung*. http://www.neuronalesnetz.de/downloads/neuronalesnetz_de.pdf. – abgerufen am 05.03.2017.

[Nischwitz et al. 2011a] NISCHWITZ, Alfred ; FISCHER, Max ; HABERÄCKER, Peter ; SOCHER, Gudrun: *Computergrafik und Bildverarbeitung, Band I: Computergrafik*. Vieweg + Teubner, 2011. – ISBN 978–3–8348–1304–6.

[Nischwitz et al. 2011b] NISCHWITZ, Alfred ; FISCHER, Max ; HABERÄCKER, Peter ; SOCHER, Gudrun: *Computergrafik und Bildverarbeitung, Band II: Bildverarbeitung*. Vieweg + Teubner, 2011. – ISBN 978–3–8348–1712–9.

[Schmeh 2009] SCHMEH, Klaus: *Kryptografie*. dpunkt.verlag, 2009. – ISBN 978–3–89864–602–4.

[Schneier 2000] SCHNEIER, Bruce: *Angewandte Kryptographie: Protokolle, Algorithmen und Sourcecode in C*. München [u.a.], 2000. – ISBN 3–89319–854–7.

[Tews et al. 2007] TEWS, Erik ; WEINMANN, Ralf-Philipp ; PYSHKIN, Andrei: Breaking 104 bit WEP in less than 60 seconds. (2007). https://eprint.iacr.org/2007/120.pdf. – abgerufen am 12.04.2017.

[Westfeld] WESTFELD, Andreas: Angriffe auf steganographische Systeme. https://www2.htw-dresden.de/~westfeld/publikationen/vis99.pdf. – abgerufen am 12.04.2017.

[Wichert 2000] WICHERT, Andreas: *Künstliche Intelligenz*. Spektrum der Wissenschaft, Lexikon der Neurowissenschaft. http://www.spektrum.de/lexikon/neurowissenschaft/kuenstliche-intelligenz/6810. Version: 2000. – abgerufen am 12.04.2017.

[Wikipedia 2017] WIKIPEDIA: *Kryptoanalyse*. https://de.wikipedia.org/wiki/Kryptoanalyse. Version: April 2017. – abgerufen am 12.04.2017.

A Anhang Python Module

Die vollständigen Python Quellen, inklusive der Kommentierung, die zur Nutzung der Programme hilfreich sind, werden auf der zur Arbeit gehörigen CD-ROM mitgeliefert. Hier erfolgt eine Aufstellung der Inhalte der CD.
Alle in Tabelle A.1 angegebenen Untermodule sind im Modul `StegangraphySuite` zusammengefasst und können über die dort hinterlegten Methoden genutzt werden.

Modulname	**Beinhaltete Funktionen**

`SteganographySuite`	`analyzeImage()`
	`openImage()`
	`encryptSingleFrame()`
	`decryptSingleFrame()`
	`filterImage()`
	`filterCam()`
	`print2HistBars()`
	`histSingleWebcamFrame()`
	`histSingleImage()`

Untermodulname	**Beinhaltete Funktionen**

`enc`	`openPNG()`
	`getBlockMessage()`
	`simpleLSB()`
	`extendedLSB()`

`dec`	`extract_LSB()`

`attacks`	`visualFilter()`
	`visualFilter_gray()`
	`makeHistogram()`
	`evalHist()`
	`evalVar()`
	`chiSquaredAnalysis()`

Tabelle A.1: Aufstellung der programmierten Python-Module

B Anhang Messreihen

Container		**Steganogramm**	
Zeilenvarianz	normierter Indikator	Zeilenvarianz	normierter Indikator
1.15275355e+06	3.82	4.41444740e+04	99.87
1.10343457e+06	3.99	4.40899271e+04	99.99
5.71754225e+05	7.71	4.41109179e+04	99.94
5.24407299e+05	8.40	4.40970839e+04	99.97
6.02188151e+05	7.32	4.40985258e+04	99.97
7.42016634e+05	5.94	4.41269067e+04	99.91
7.46314526e+05	5.90	4.40985740e+04	99.97
1.00320793e+06	4.39	4.41207395e+04	99.92
5.60616249e+05	7.86	4.41133533e+04	99.94
7.40614871e+05	5.95	4.40910931e+04	99.99
6.45212484e+05	6.83	4.41004310e+04	99.97
1.02222418e+06	4.31	4.40961720e+04	99.98
1.03330072e+06	4.26	4.40945658e+04	99.98
8.54674193e+05	5.15	4.41344401e+04	99.89
8.77097733e+05	5.02	4.40916067e+04	99.99
8.43853240e+05	5.22	4.41152741e+04	99.93
2.10429142e+06	2.09	4.41038890e+04	99.96
1.15819642e+06	3.80	4.40913488e+04	99.99
8.18106970e+05	5.38	4.41141863e+04	99.94
1.22170735e+06	3.60	4.41056843e+04	99.95
1.37416710e+06	3.20	4.41212924e+04	99.92
1.37061981e+06	3.21	4.40891472e+04	99.99
1.09323469e+06	4.03	4.40986720e+04	99.97
5.78491021e+05	7.62	4.41094884e+04	99.95
5.50260733e+05	8.01	4.40976839e+04	99.97
7.79960633e+05	5.65	4.41297611e+04	99.90
1.10988253e+06	3.97	4.40975881e+04	99.97
1.56926645e+06	2.80	4.40935964e+04	99.98
7.11893631e+05	6.19	4.41068482e+04	99.95
1.25772023e+06	3.50	4.40945128e+04	99.98
8.13135897e+05	5.42	4.41068496e+04	99.95

Container		**Steganogramm**	
Zeilenvarianz	normierter Indikator	Zeilenvarianz	normierter Indikator
8.87015004e+05	4.97	4.40938210e+04	99.98
1.20699304e+06	3.65	4.40904384e+04	99.99
1.03183264e+06	4.27	4.41241651e+04	99.91
8.60944640e+05	5.12	4.40920931e+04	99.99
1.22967376e+06	3.58	4.40879768e+04	100.0
4.77597844e+05	9.23	4.41250319e+04	99.91
8.48081147e+05	5.19	4.40926710e+04	99.98
6.56506434e+05	6.71	4.40990607e+04	99.97
5.57896883e+05	7.90	4.40931431e+04	99.98
5.44798941e+05	8.09	4.40904503e+04	99.99
8.07724179e+05	5.45	4.41141723e+04	99.94
5.90109722e+05	7.47	4.41037258e+04	99.96
5.72323908e+05	7.70	4.41150595e+04	99.93
5.01855663e+05	8.78	4.41306577e+04	99.90
7.74598840e+05	5.69	4.41430539e+04	99.87
6.01963770e+05	7.32	4.41304639e+04	99.90
5.64532791e+05	7.80	4.41054577e+04	99.96
5.15430018e+05	8.55	4.40975432e+04	99.97
5.65410771e+05	7.79	4.41030937e+04	99.96
5.05691972e+05	8.71	4.41080335e+04	99.95
5.94195161e+05	7.41	4.41155964e+04	99.93
5.72272157e+05	7.70	4.40941026e+04	99.98
1.36413619e+06	3.23	4.40903771e+04	99.99
1.17224395e+06	3.76	4.41077214e+04	99.95
2.91377693e+06	1.51	4.40913835e+04	99.99
3.56034238e+09	1.23e-03	4.41082962e+04	99.95
4.74878190e+09	9.28e-04	4.41330781e+04	99.89
4.72036961e+09	9.33e-04	4.41149044e+04	99.93
4.71643614e+09	9.34e-04	4.40918006e+04	99.99
4.61599090e+09	9.55e-04	4.40901339e+04	99.99
4.45690624e+09	9.89e-04	4.41039121e+04	99.96
4.38961947e+09	1.00e-03	4.40904896e+04	99.99
3.90588858e+09	1.12e-03	4.41100049e+04	99.95
4.09804635e+09	1.07e-03	4.41100503e+04	99.94
3.58966327e+09	1.22e-03	4.41006080e+04	99.97
3.52701748e+09	1.25e-03	4.40889651e+04	99.99
3.07384305e+09	1.43e-03	4.40938485e+04	99.98
2.86112159e+09	1.54e-03	4.41061806e+04	99.95

Container		**Steganogramm**	
Zeilenvarianz	normierter Indikator	Zeilenvarianz	normierter Indikator
3.34374851e+09	1.31e-03	4.41168292e+04	99.93
3.23294888e+09	1.36e-03	4.41430893e+04	99.87
3.24751792e+09	1.35e-03	4.40886781e+04	99.99
2.88713920e+09	1.52e-03	4.40959829e+04	99.98
2.05509041e+09	2.14e-03	4.41041645e+04	99.96
2.08859976e+09	2.11e-03	4.41038611e+04	99.96
1.82656814e+09	2.41e-03	4.41707741e+04	99.81
2.01938058e+09	2.18e-03	4.40898455e+04	99.99
1.97268127e+09	2.23e-03	4.41152268e+04	99.93
1.88048445e+09	2.34e-03	4.40964937e+04	99.98
1.96886055e+09	2.23e-03	4.41046986e+04	99.96
1.82414294e+09	2.41e-03	4.40896288e+04	99.99
1.98694258e+09	2.21e-03	4.41154026e+04	99.93
2.04902915e+09	2.15e-03	4.41453598e+04	99.87
2.46246018e+09	1.79e-03	4.41015223e+04	99.96
2.73573652e+09	1.61e-03	4.40961567e+04	99.98
2.97715166e+09	1.48e-03	4.40883788e+04	99.99
3.01752817e+09	1.46e-03	4.41119881e+04	99.94
2.82709426e+09	1.55e-03	4.41059015e+04	99.95
2.92969181e+09	1.50e-03	4.41306621e+04	99.90
2.88835552e+09	1.52e-03	4.40915319e+04	99.99
2.91557656e+09	1.51e-03	4.41002771e+04	99.97
2.80127771e+09	1.57e-03	4.41141937e+04	99.94
2.84286079e+09	1.55e-03	4.40960087e+04	99.98
2.87924062e+09	1.53e-03	4.41393509e+04	99.88
2.83194897e+09	1.55e-03	4.40969488e+04	99.97
2.91387492e+06	1.51	4.41161686e+04	99.93
2.63604939e+06	1.67	4.41058444e+04	99.95
2.48687548e+06	1.77	4.40904986e+04	99.99
1.89911252e+06	2.32	4.41191526e+04	99.92
1.01134550e+06	4.35	4.40923210e+04	99.99

Tabelle B.1: Messreihe zur Ermittlung von Erfahrungswerten für das Zeilen-Varianz-Verfahren

Größe des untersuchten Containers in %	normiert Rechenzeit
0.0	0.0
2.5	1.0
5.0	1.888783
7.5	2.96269515
10.0	3.91724551
12.5	5.03219595
15.0	5.91880766
17.5	7.19119106
20.0	8.01253648
22.5	9.05081399
25.0	9.78917894
27.5	10.90664453
30.0	12.21740369
32.5	12.73269603
35.0	14.05546113
37.5	15.06771269
40.0	15.2461413
42.5	16.00357624
45.0	17.01721309
47.5	17.96781388
50.0	18.78829471
52.5	20.42172072
55.0	20.83679986
57.5	21.6634212
60.0	22.70875294
62.5	24.11392866
65.0	24.39977207
67.5	26.926304
70.0	26.60813692
72.5	27.36009943
75.0	28.34391795
77.5	29.13495378
80.0	30.04078285
82.5	31.5748013
85.0	31.8876335
87.5	33.24475601
90.0	34.25571068
92.5	34.94163073
95.0	36.08026881
97.5	36.76275017

Tabelle B.2: Messung zur Komplexität des Zeilen-Varianz-Algorithmus

Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §16(5) APSO-TI-BM ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Hamburg, 4. Mai 2017	
Ort, Datum	Unterschrift