
DIPLOMARBEIT

Einsatz von Steganographie

im Projekt GeocachingTools

Ausgeführt im Schuljahr 2016/17 von:

Simon Lehner-Dittenberger, 5AHIF-10

Betreuer/Betreuerin:

OSTR Mag. Otto Reichel

St.Pölten, am March 30, 2017

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche erkenntlich gemacht habe.

Simon Lehner-Dittenberger

St.Pölten, am 24.04.20XX

Diplomandenvorstellung



Max MUSTERMANN

Geburtsdaten:
06.02.1996 in Musterort

Wohnhaft in:
Musterstraße 13/1
3100 Musterstadt

Werdegang:
2010 - 2015:
HTBLuVA St.Pölten, Abteilung für Informatik
2006 - 2010:
Bundesrealgymnasium Wieselburg a. d. Erlauf

Kontakt:
max.mustermann@gmx.at

Danksagungen

Danke

Zusammenfassung

Abstract

Contents

Vorwort	i
Diplomandenvorstellung	ii
Danksagungen	iii
Zusammenfassung	iv
Abstract	v
Inhaltsverzeichnis	vi
1 Steganographie	1
1.0.1 Steganographie Übersicht	1
1.0.2 Grundlagen Steganographie	1
1.0.3 Abgrenzung zur Kryptographie	2
1.0.4 Einsatzgebiete	3
1.0.5 Steganographie als "Wicked Problem"	4
1.1 Klassische Verfahren der Steganographie	6
1.1.1 Spreu und Weizen Verfahren	6
Sender-Seite	6
Die Spreu	7

Empfänger-Seite	8
Sicherheitsaspekte	8
1.1.2 Semagramme	9
1.2 Moderne Steganographie	12
1.2.1 Whitening	12
Pseudozufallsgenerator mit bitweiser Verknüpfung	14
AES-256 Verschlüsselung mit Key	16
1.2.2 Fehlerkorrekturverfahren	17
Redundante Datenspeicherung	17
Paritätsbit	18
Zyklische Redundanzprüfung (z.B. CRC-32)	18
1.2.3 Least Significant Bit Verfahren	19
Varianten des Least Significant Bit Verfahren	25
1.2.4 Barcode Verfahren	26
1.2.5 Stereoskopische Verfahren	30
Erzeugen von Stereogrammen mithilfe einer Depth Map	33
Steganographie mithilfe von Stereogrammen	36
Grafisches Hilfsmittel zum Entschlüsseln von Stereogrammen	36
Automatische Erkennung von Stereogrammen	37
Bildfehler erkennen	40
1.2.6 Odd-Pixel Verfahren	41
Anhang	46
Tabellenverzeichnis	48

Verzeichnis der Listings	49
Literaturverzeichnis	50

Chapter 1

Steganographie

1.0.1 Steganographie Übersicht

1.0.2 Grundlagen Steganographie

Die Steganographie ist eine Methode, die sich mit dem Verstecken von zu übermittelnden Nachrichten beschäftigt und kam schon in der Antike zum Einsatz. Das Wort kommt aus den griechischen Wörtern "stegano" und "graphein", was übersetzt "bedeckt schreiben" bedeutet [L: StegoGeschichte]. Dabei wird meist ein Text, aber auch andere Arten von Informationen, in einem Trägermedium versteckt. Diese Kombination wird als Steganogramm bezeichnet. Das Medium sollte so gewählt sein, dass sich die einzubettenden Daten leicht integrieren lassen. Außerdem benötigt es ein gewisses Maß an Entropie¹, damit Unregelmäßigkeiten nicht so stark auffallen, denn eine Blume ist in einer bunten Blumenwiese schwerer zu finden, als auf einem asphaltierten Parkplatz. Ziel ist es immer, die Wahrnehmungsschwelle eines Menschen so weit zu unterschreiten, dass man gar nicht auf die Idee kommt überhaupt nach einer versteckten Nachricht zu suchen.²

Die Möglichkeiten für Steganogramme haben sich mit der Entwicklung von Computer und elektronischer Datenverarbeitung sehr stark verändert, die Idee dahinter ist jedoch die Gleiche: Man versteckt Informationen. Früher hat man noch beispielsweise mit unsichtbarer Tinte geschrieben, welche erst mit Hitze sichtbar wird (z.B. Zitronensaft). Auch wurden Techniken wie etwa die monoalphabetische Substitution benutzt, bei welcher Buchstaben des zu versteckenden Wortes über eine Tabelle durch Wörter

¹Entropie ist ein Begriff zur Beschreibung der Unregelmäßigkeit von Daten oder Dingen

²TODO Welche Techniken wofür gut sind und welche Trägermaterialien man braucht wird in den späteren Abschnitten behandelt

ersetzt werden. Diese Wortfolge wird dann mit weiteren nicht in der Tabelle vorkommenden Worten ergänzt um vollständige, grammatisch korrekte Sätze bilden zu können. Eine solche Tabelle findet man zum Beispiel in dem Buch I der Polygraphia von Johannes Trithemius (Siehe: Figure 1.1). Heute werden vor allem Verfahren eingesetzt, die Bilder und Videos nutzen, denn man kann die große Menge an Daten verwenden, welche jeden Tag millionenfach versendet werden. Diese können mit den richtigen Programmen auch sehr leicht manipuliert und bearbeitet werden, um sie als Steganogramme einzusetzen.

A	Deus	A	clemens
B	Creator	B	clementissimus
C	Conditor	C	pius
D	Opifex	D	piaissimus
E	Dominus	E	magnus
F	Dominator	F	excelsus
G	Consolator	G	maximus
H	Arbiter	H	optimus
I	Index	I	sapientissimus
K	Illuminator	K	iruissibilis
L	Illustrator	L	immortalis
M	Rector	M	eternus
N	Rex	N	semipernus
O	Imperator	O	gloriosus
P	Gubernator	P	fortissimus
Q	Factor	Q	sanctissimus
R	Fabricator	R	incomprehensibilis
S	Conseruator	S	omnipotens
T	Redemptor	T	pacificus
V	Auctor	V	misericors
X	Principes	X	misericordissimus
Y	Pastor	Y	cunctipotens
Z	Moderator	Z	magnificus
W	Saluator	W	excellens

Figure 1.1: Buchstaben-Wort-Substitutionstabelle von Buch I der Polygraphia von Johannes Trithemius, Quelle: http://daten.digitale-sammlungen.de/bsb00026190/image_71

1.0.3 Abgrenzung zur Kryptographie

Kryptographie und Steganographie werden oft gemeinsam verwendet, wodurch meist nicht genau zwischen diesen beiden Verfahren unterschieden wird. Wie man in Table 1.1 ³ sehen kann, wirken beide Techniken auf den ersten Blick sehr ähnlich, sind

³ TODO Wie bekommt man die richtige Bezeichnung hier in den Text(unterschied zwischen label und caption)

aber bei genauerer Betrachtung zwei komplett unterschiedliche Verfahren. Wichtig ist hier vor allem zu beachten: Steganographie schützt Daten nicht vor Dritten, wenn diese gezielt danach Suchen und wenn sie sich sicher sind, dass in den Informationen, die Ihnen vorliegen weitere Nachrichten versteckt wurden. Des Weiteren haben Steganogramme die Eigenschaft von Menschen schlecht erkannt werden zu können. Computer auf der anderen Seite sind jedoch in der Lage verdeckte Nachrichten meist schnell und zuverlässig sichtbar zu machen. Bei dem Erfolg eines computergestützten Verfahren kommt es aber sehr stark auf die verwendete Technik und die verfügbare Rechenleistung an.

Steganographie	Kryptographie
stegano = verdeckt graphein = scrheiben	kryptē = geheim graphein = schreiben
Die Nachricht wird verborgen, nicht verschlüsselt	Die Nachricht wird verschlüsselt nicht verborgen
Scheinbar existiert gar keine Nachricht	Die Nachricht existiert, kann aber nicht gelesen werden

Table 1.1: Vergleich zwischen Steganographie und Kryptographie, Quelle: [L: Stego VS Crypto]

Am sichersten ist es, wenn man beide Verfahren kombiniert. Dadurch hat man nicht nur die Vorteile der Kryptographie (Vertraulichkeit, Integrität und Authentizität), sondern auch die der Steganographie. Interessant ist hier vor allem die Eigenschaft von Verschlüsselungen: Diese gelten dann als sicher, wenn sie den Klartext derart verändern, dass er keine statistischen Merkmale des ursprünglichen Text mehr aufweist. Der Geheimtext kann also bei guten Verschlüsselungsverfahren statistisch nicht mehr von Rauschen unterscheiden werden. Wenn man dieses "Rauschen" dann mit Hilfe von Steganographie in ein unauffälliges Trägermedium einbettet, ist es selbst mit elektromischer Datenverarbeitung nicht mehr möglich, eine Nachricht im Steganogramm zu entdecken. Die einzige Möglichkeit für Dritte hier noch etwas herauszufinden, ist es das Steganogramm mit dem originalen Trägermaterial zu vergleichen. Hier fallen dann Unterschiede auf. Diese Technik ist aber in der Praxis selten anwendbar, denn einzigartige Trägermaterialien können sehr leicht hergestellt werden (z.B. Digitalfotografie) und weil das Trägermedium nicht zum Dekodieren benötigt wird, kann das Original nach der Erstellung des Steganogramm gelöscht werden.

1.0.4 Einsatzgebiete

Steganographie schützt Daten nicht vor Missbrauch, warum sollte man sie dann überhaupt verwenden, wenn Kryptographie viel sicherer ist? In der westlichen Welt ist Ver-

schlüsselung durch das Internet so weit verbreitet, dass es als selbstverständlich erscheint, seine Daten und Konversationen verschlüsselt zu speichern. Doch in vielen Ländern ist es auch heute noch illegal solche Techniken einzusetzen. Selbst in den Vereinigten Staaten von Amerika gab es noch bis in das Jahr 2000 sehr restriktive Gesetze was Verschlüsselung anbelangt. Das führte soweit, dass sogar ein T-Shirt auf welches der Source-Code für RSA-Encryption gedruckt wurde, unter das Waffengesetz fiel, als "export-restricted munition" deklariert und für den Export verboten wurde.

Hier kommt Steganographie zum Einsatz. Sie bietet eine Möglichkeit seine Daten und sich selber trotz der lokal geltenden Gesetze zu schützen. Nicht nur dass es schwer zu erkennen ist, ob sich überhaupt versteckte Daten auf einem Laufwerk befinden, steganographische Verfahren werden meist gar nicht von den Gesetzten verboten. Man befindet sich hier oft in einer Grauzone, was einem einen gewissen Verhandlungsspielraum verschafft.

Steganographie bietet auch Schutz vor potentiellen Hackern. Denn während bei verschlüsselten Daten ein sich lohnendes Ziel auf den Angreifer wartet, ist es bei Steganographie sehr unwahrscheinlich etwas Verwertbares zu finden, falls überhaupt etwas vorhanden ist. Dadurch macht man sich als Opfer sehr unattraktiv.

[L: StegoVersteck]

1.0.5 Steganographie als "Wicked Problem"

Die Steganographie besitzt viele Eigenschaften von sogenannten "Wicked Problems".

- Es gibt keine genaue Definition des Problems
- Sie haben keine Stopp-Regel ("Hat man auch wirklich nichts übersehen?")
- Es gibt keinen ultimativen und sofortigen Test für die Richtigkeit von Lösungen des Problems
- Wicked Problems haben weder eine abzählbare Lösungsmenge, noch gibt es eine gut beschriebene Gruppe an gültigen Operatoren

Diese Eigenschaften und die Tatsache dass Kommunikation schwer zu definieren ist, führen dazu, dass paranoide oder phantasievolle Menschen glauben, Nachrichten zu empfangen, obwohl keine vorhanden sind. Da können selbst kleine unbedeutende

Handlungen von Mitmenschen als geheime Nachrichtenübertragung interpretiert werden, was unter anderem zu Problemen führen kann, wo eigentlich keine sind.

Doch auch in der Verbrechensaufklärung kann die Wicked-Problem Eigenschaft von Steganographie zum Problem werden. Wird hier denn wirklich neben der offensichtlich übertragenen Information noch eine versteckte Nachricht mitgesendet? Eine verdächtige Person kauft jeden Tag einen Kaffee auf dem Weg zur Arbeit. Die Verkäuferin ist die Freundin von dem Steuerberater des Chefs des Verdächtigen. Plötzlich kauft er aber einen Kräutertee. Hat er jetzt den Steuerberater vor irgendetwas gewarnt oder hat er heute nur Halsweh und möchte seinen Hals schonen? Das ist eben die Natur von Wicked Problems. Man kann sich nie sicher sein, denn es gibt weder eine genaue Fragestellung, noch ein eindeutiges Erfolgskriterium oder eine klar definierte Ausgangslage.

1.1 Klassische Verfahren der Steganographie

Mit klassischen Steganographie-Verfahren sind Techniken gemeint, welche größtenteils oder gänzlich ohne Computersysteme funktionieren und somit nicht zwingend auf das "digitale Zeitalter" angewiesen sind. Das nachfolgenden Kapitel soll mit praxisnahen Beispielen einen Überblick darüber verschaffen, was genau der Grundgedanke von Steganographie ist.

1.1.1 Spreu und Weizen Verfahren

Das Spreu und Weizen-Verfahren⁴ stellt eine Mischform zwischen Steganographie und Kryptographie da und können daher nicht wirklich eindeutig zugeordnet werden. Trotzdem enthält es sehr viele Elemente der Steganographie und eignet sich gut als Einstieg in diesen Thema. Die Idee des Verfahrens ist es, die zu versteckenden Daten in einem Haufen nicht-relevanter Information zu verstecken, wie die Nadel im Heuhaufen.

⁵ Es handelt sich bei diesem Verfahren deswegen um eine Mischform, weil es sich genau genommen lediglich um das Authentifizieren von gesendeten Paketen handelt. Das Hinzufügen der Spreu kann auch von einer dritten unwissenden Person geschehen. Dadurch können sowohl Sender als auch Empfänger sämtliche Verantwortung abstreiten und argumentieren, sie wollen nur die Authentizität ihrer Nachrichten sicherstellen. Deshalb können auch Gesetze, welche Kryptographie beschränken oder sogar verbieten, nicht auf das Spreu und Weizen Verfahren angewandt werden.

The power to authenticate is in many cases the power to control, and handing all authentication power to the government is beyond all reason

— Ronald L. Rivest, 1998

[L-Confidentiality without Encryption]

Sender-Seite

Der Sender muss seine Nachricht in Pakete unterteilen. Ihre Größe kann beliebig gewählt werden. Er muss die Pakete außerdem in irgendeiner Weise durchnum-

⁴eng.: Chaffing and Winnowing

⁵TODO Nachfolgenden Absatz und Zitat ans Ende der Section schieben?

merieren, um sie auf der Empfängerseite wieder in der richtigen Reihenfolge zusammensetzen zu können.

An jedes Paket wird nun ein Message Authentication Code (kurz: MAC) angehängt. Dieser dient, wie der Name schon vermuten lässt, zur Authentifizierung der Nachrichten. Einen MAC zu verwenden ist ein vernünftiger Schritt, welcher oft verwendet wird, und erregt somit wenig Aufmerksamkeit.

Die Nachricht "Hallo Hans, wir treffen uns am 24. Jan um 18 Uhr am Hauptbahnhof" könnte etwa so aufgeteilt werden:

ID	Nachrichtenfragment	MAC
1	Hallo Hans,	9192
2	wir treffen uns am 24. Jan	3766
3	um 18 Uhr	2816
4	am Hauptbahnhof	8370

Als MAC wird hier eine vier stellige Zahl eingesetzt, welche als gültig angesehen wird, wenn sie durch zwei teilbar - also eine gerade Zahl - ist.

Die Spreu

Dieser Schritt kann auch von einer dritten unwissenden Person erfolgen. Vorteilhaft ist hier, wenn Nachrichten erzeugt werden, welche ...

- ... ähnlichen Inhalt mit den oben angeführten Nachrichten haben
- ... auf jeden Fall einen ungültigen MAC besitzen.

Die in diesem Beispiel verwendete Methode führt sehr leicht zu zufällig gültigen MAC's, ist also in einem realen Anwendungsfall nicht ausreichend. Hier könnten zum Beispiel Codes aus der HMAC⁶ Familie zur Anwendung kommen, wie der recht bekannte HMAC_SHA256.

Spreu Nachrichten könnten etwa so aufgebaut sein:

⁶Hash-based Message Authentication Code

ID	Nachrichtenfragment	MAC
1	Hallo Alice,	2373
1	Hallo Bob,	5323
2	wir telefonieren am 27. Jan	5847
3	um 10 Uhr	7881
4	am Westbahnhof	9821
4	am Bahnhof Meidling	1155

Hier deutlich zu erkennen die ungültigen MAC's, nämlich ungerade Zahlen.

Empfänger-Seite

Der Empfänger sortiert nun die Pakete nach der ID und überprüft deren MAC's. Für ihn ist nun gut ersichtlich, welche Pakete die ursprüngliche Nachricht enthalten. Wie man in dem Beispiel gut sehen kann, ist es für einen eingeweihten Empfänger sehr leicht, die ursprüngliche Nachricht zu erkennen. Für einen unwissenden Dritten kann es sich jedoch äußerst schwierig gestalten, die richtigen Nachrichtenfragmente zusammenzusetzen.

Es könnten durchaus auch Texte wie "Hallo Alice, wir telefonieren am 27. Jan um 18 Uhr am Westbahnhof" oder "Hallo Bob, wir treffen uns am 24. Jan um 18 Uhr am Bahnhof Meidling" als mögliche Lösungen gesehen werden.

ID	Nachrichtenfragment	MAC
1	Hallo Alice,	2373
1	Hallo Hans,	9192
1	Hallo Bob,	5323
2	wir treffen uns am 24. Jan	3766
2	wir telefonieren am 27. Jan	5847 ⁷
3	um 10 Uhr	7881
3	um 18 Uhr	2816
4	am Hauptbahnhof	8370
4	am Westbahnhof	9821
4	am Bahnhof Meidling	1155

⁷TODO Statt einer langen unübersichtlichen Tabelle ein Diagramm mit den möglichen Lösungswegen welches man selber durchgehen kann.

Sicherheitsaspekte

In dem obigen Beispiel gibt es $3 * 2 * 2 * 3 = 36$ verschiedene Lösungswege, eine Nachricht zusammenzusetzen. Wenn man den Text verlängert und in noch mehr Pakete aufspaltet, ergeben sich dadurch auch mehr Kombinationen. Angenommen man sendet für jedes Paket ein ungültiges alternatives Spreupaket, hätte man nach n Paketen 2^n mögliche Ergebnisse.

Bereits nach 10 Fragmenten ergeben sich dadurch 1024 verschiedene Nachrichten, nach 30 sogar schon mehr als 1 Milliarde. Wie man sehen kann, entstehen sogar bei geringer Menge an Spreu sehr schnell eine riesige Menge an Kombinationen und man kann durchaus auch hunderte Spreupakete mitsenden. Dies fällt vor allem einfach in "packet-switched network environments" wie dem Internet.

1.1.2 Semagramme

Eine einfache Variante für klassische Steganographie in Bildern ist das Kodieren von Text in Form von Morsecode oder ähnlichem. Dieser kann zum Beispiel in der Länge von Grashalmen auf einem gemalten Bild kodiert sein, wie es in dem bekannten Beispiel 1.2 gemacht wurde.

Der Fantasie sind hier keine Grenzen gesetzt. Um ein bisschen einen Eindruck zu vermitteln, wie kreativ man beim Verstecken von Nachrichten sein kann, sind hier einige Beispiele angeführt.

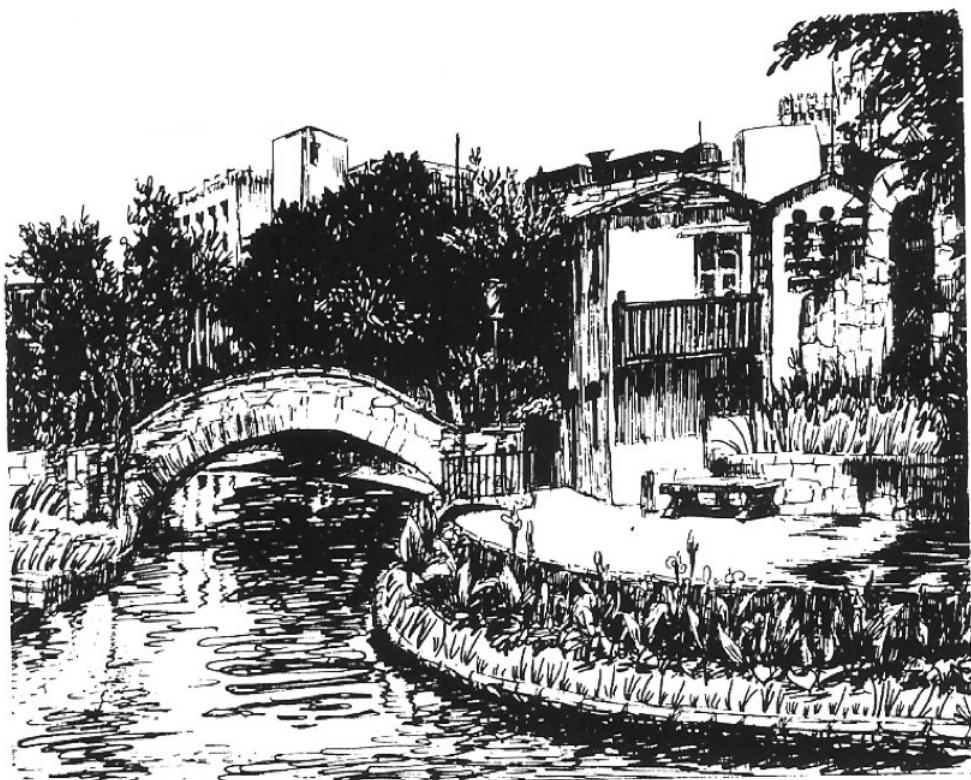


Figure 1.2: Hier wurde mit den Grashalmen links von der Brücke, auf der kleinen Mauer und entlang des Wasser Morsecode hinzugefügt



Figure 1.3: Wenn das grüne Band zum dekodieren verwendet wird ergibt sich die Nachricht "Renseignements arrivent", auf Deutsch: "Informationen angekommen"



Figure 1.4: In diesem Beispiel wurde Morsecode in Form der Größe der Vögel, der Zaunstangen und der Blumen kodiert. Dekodiert ergeben sich die Namen der drei Automarken VW, Buick und Volvo

1.2 Moderne Steganographie

Mit moderner Steganographie sind Verfahren gemeint, welche nur mit Hilfsmittel der elektronischen Datenverarbeitung funktionieren. Sie verlassen sich meist darauf, dass in riesigen Zahlenmengen kleine Hinweise versteckt sind. Solch große Datenmengen lassen sich per Hand nicht mehr berechnen, wie etwa die vielen Millionen Bildpunkte auf einer digitalen Fotografie.

Bilder auf einem Computer sind nichts anderes als Matrizen, welche Farbeinformationen für die einzelnen Pixel des Bildes beinhalten. Es gibt zahlreiche Verfahren in diesen Matrizen Daten zu verstecken. Im den folgenden Kapitel werden einige dieser Verfahren vorgestellt. Dadurch, dass alle Daten im Computer in Binärform als Zahl gespeichert sind, ist es vollkommen egal, welche Datei als Payload eingebettet wird.

In dem nachfolgenden Abschnitten werden einige dieser Verfahren erklärt und etwaige Fehler und Schwierigkeiten, die damit verbunden sind, aufgezeigt. Außerdem werden einige Implementierungen vorgestellt.

1.2.1 Whitening

Als Whitening werden Prozesse und Techniken mit dem Ziel bezeichnet, Daten jegliche Ordnung und Regelmäßigkeit zu entziehen. Je besser ein Whitening-Verfahren funktioniert, desto weniger ist der Datenstrom von generierten Zufallswerten unterscheidbar.

Dies hat den Vorteil, dass die Daten nach der Transformation in dem einzubettenden Medium nicht so stark auffallen wie zuvor.

Whitening-Verfahren wandeln auf deterministische Weise Datenströme, welche in etwa wie 1.5 aussehen können, um, damit sie dem Profil von 1.6 ähneln. Dies muss auf eine Art und Weise geschehen, dass beim Extrahieren der Daten die angewandte Operation wieder rückgängig gemacht werden kann. Es können also keine wirklich zufälligen generierten Daten zum Einsatz kommen.

Man verwendet hier vor allem sogenannte Pseudozufallsgeneratoren, welche lediglich die Eigenschaften von echten zufälligen Ereignissen nachahmen, bei Bedarf aber beliebig oft wiederholt werden können. Dadurch ist sichergestellt, dass sich die angewandten Operationen auch wieder rückgängig machen lassen. Es werden auch oft Methoden aus der Kryptologie verwendet, welche nicht nur herausragende Ergeb-

nisse in Hinsicht auf Zufälligkeit erzielen, sondern zugleich auch die Daten mit Hilfe eines Schlüssel schützen.

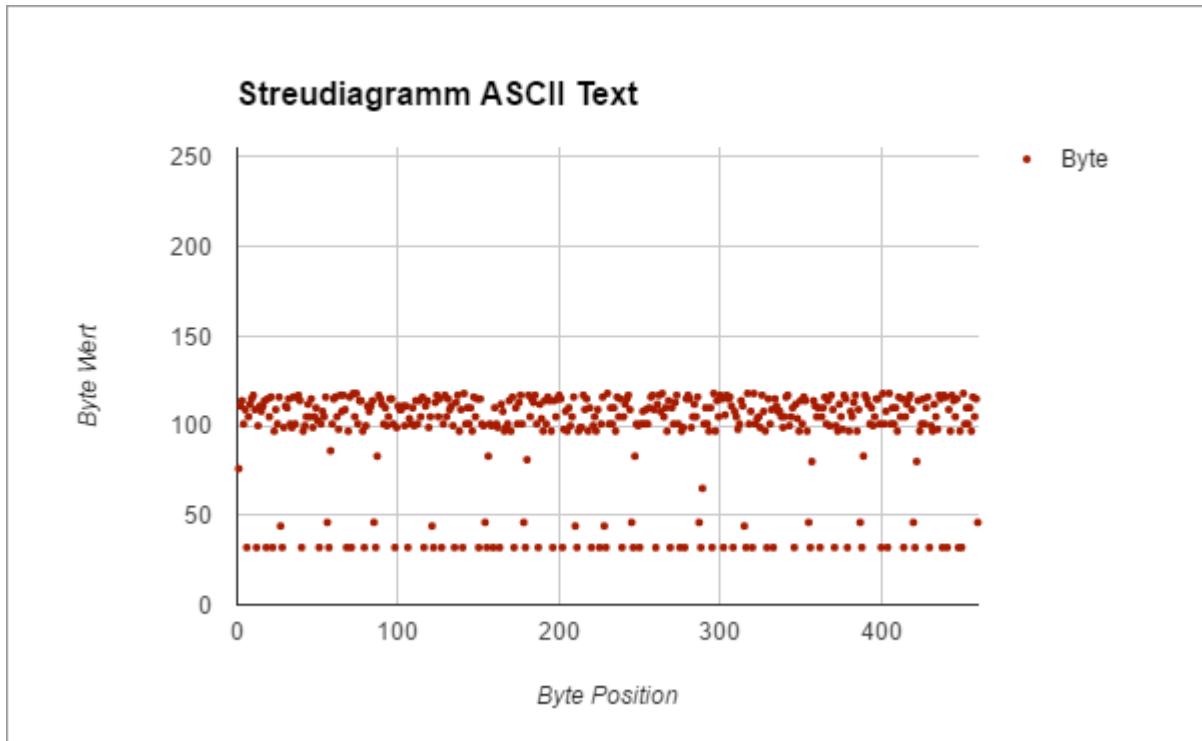


Figure 1.5: Hier deutlich zu erkennen die verwendeten ASCII Zeichen.

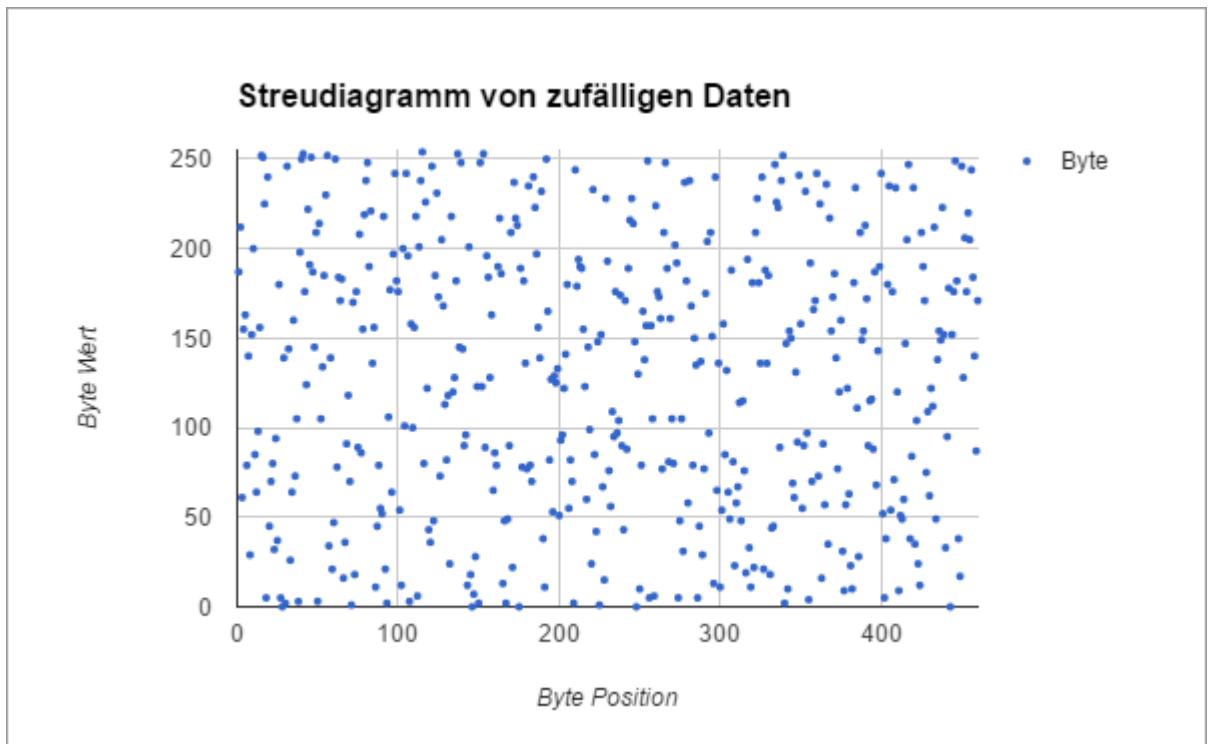


Figure 1.6: In diesen zufälligen Daten kann kein Muster erkannt werden.

8

Pseudozufallsgenerator mit bitweiser Verknüpfung

Bei Verfahren, welche auf Pseudozufallsgeneratoren bauen, ist die größte Stärke auch gleich die größte Schwäche: "Sie sind vorhersehbar". Sie schützen zwar das Steganogramm vor unwissenden Dritten, wie so oft aber nicht, wenn jemand gezielt danach sucht. Ihre sehr beschränkte Anzahl an Seeds und die Reproduzierbarkeit führen dazu, dass ein Angreifer sehr einfach auf die gängigsten Generatoren testen kann.

Das hier zur Anwendung kommende Prinzip "Security through obscurity" war bereits zu Beginn des 20. Jahrhunderts bekannt und auch als nicht sicher eingestuft. Das bekannteste Beispiel für Security through obscurity ist, einen nicht standardisierten Port für eine Webanwendung zu verwenden. Dies schützt zwar vor automatisierten Bots, ein gezielter Angriff hebelt diesen "Sicherheitsmechanismus" jedoch innerhalb von Minuten aus. Genauso ist es mit schwachen Whitening-Verfahren welche auf dieses Prinzip setzen.

⁸TODO Höhere Auflösung für Whitening Diagramme <https://docs.google.com/spreadsheets/d/1xDMWNJa9r2ggm8Hh9ZFuKrGrx7Gw>

In dem nachfolgenden Beispiel wird als Whitening der Datenstrom mit Hilfe der XOR-Operation mit einem Pseudozufallszahlengeneratordatenstrom verknüpft. Zum Einsatz kommt hier ein Kongruenzgenerator, wie er auch in der Standard Java Klasse `java.lang.Random` verwendet wird.

[L-Kongruenzgenerator]

	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
Inhalt	L	o	r	e	m	
Datenstrom	76	111	114	101	109	32
Random (Seed=0)	187	212	61	155	163	79
Datenstrom XOR Random	247	187	79	254	206	111

Figure 1.7: Wenn der Inhalt wiederhergestellt werden soll, muss nur von unten nach oben alles wieder rückgängig gemacht werden.

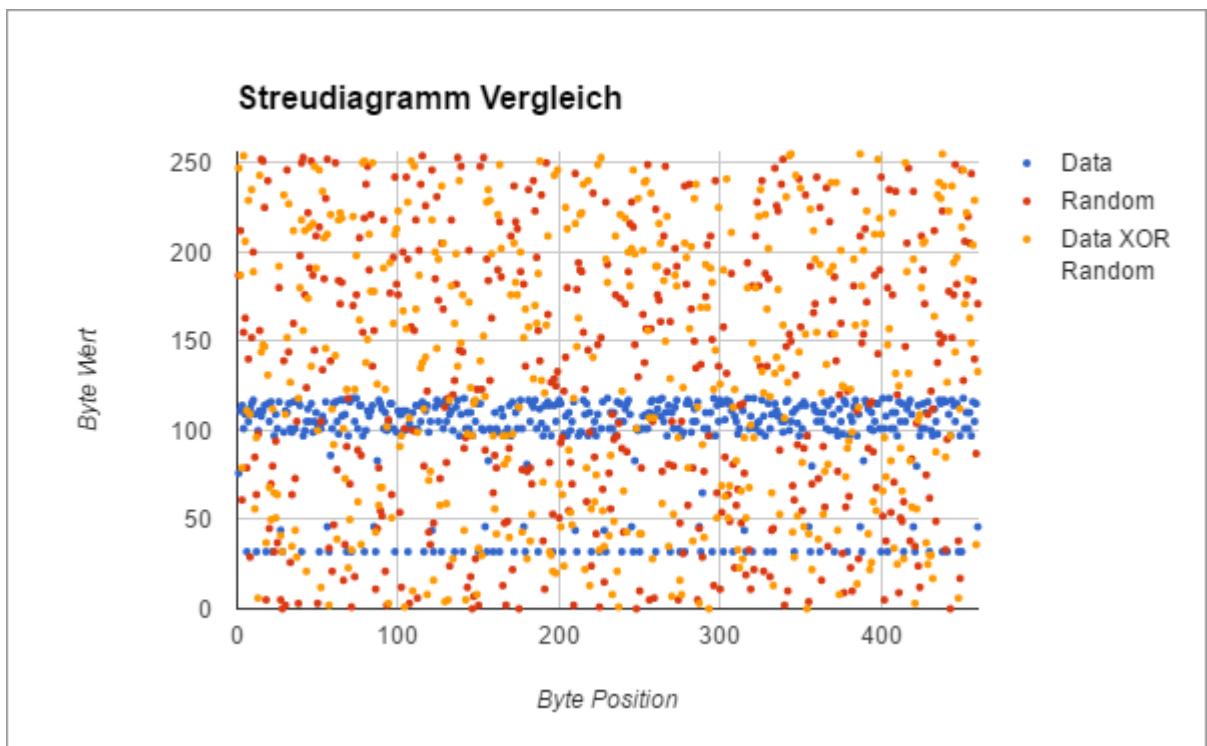


Figure 1.8: In diesem Vergleich ist gut zu sehen, dass nach Anwendung von diesem Verfahren der Datenstrom nicht mehr von zufälligen Werten unterscheidbar ist.

AES-256 Verschlüsselung mit Key

Es kann auch die Eigenschaft von Verschlüsselung genutzt werden, Daten derart zu verarbeiten, dass sie nicht mehr von zufälligen Werten unterscheidbar ist. In dem nachfolgenden Beispiel wird die zurzeit als sicher geltende AES-256 Verschlüsselung verwendet.

Folgende Java Funktion wurde für die Verschlüsselung des Plaintext verwendet:

```
1 public static byte[] cryptit(int ciphermode, char[] password, byte[]
2 salt, byte[] iv, byte[] data) throws Exception {
3     /* Derive the key, given password and salt. */
4     SecretKeyFactory factory = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256");
5     KeySpec spec = new PBEKeySpec(password, salt, 65536, 128);
6     SecretKey tmp = factory.generateSecret(spec);
7     SecretKey secret = new SecretKeySpec(tmp.getEncoded(), "AES");
8
9     /* Encrypt the message. */
10    Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
11    cipher.init(ciphermode, secret, new IvParameterSpec(iv));
12    AlgorithmParameters params = cipher.getParameters();
13    return cipher.doFinal(data);
14 }
```

Listing 1.1: AESVerfahren.java

Wenn man die daraus resultierenden Daten auswertet, lässt sich wieder der gewünschte Whitening-Effekt beobachten.

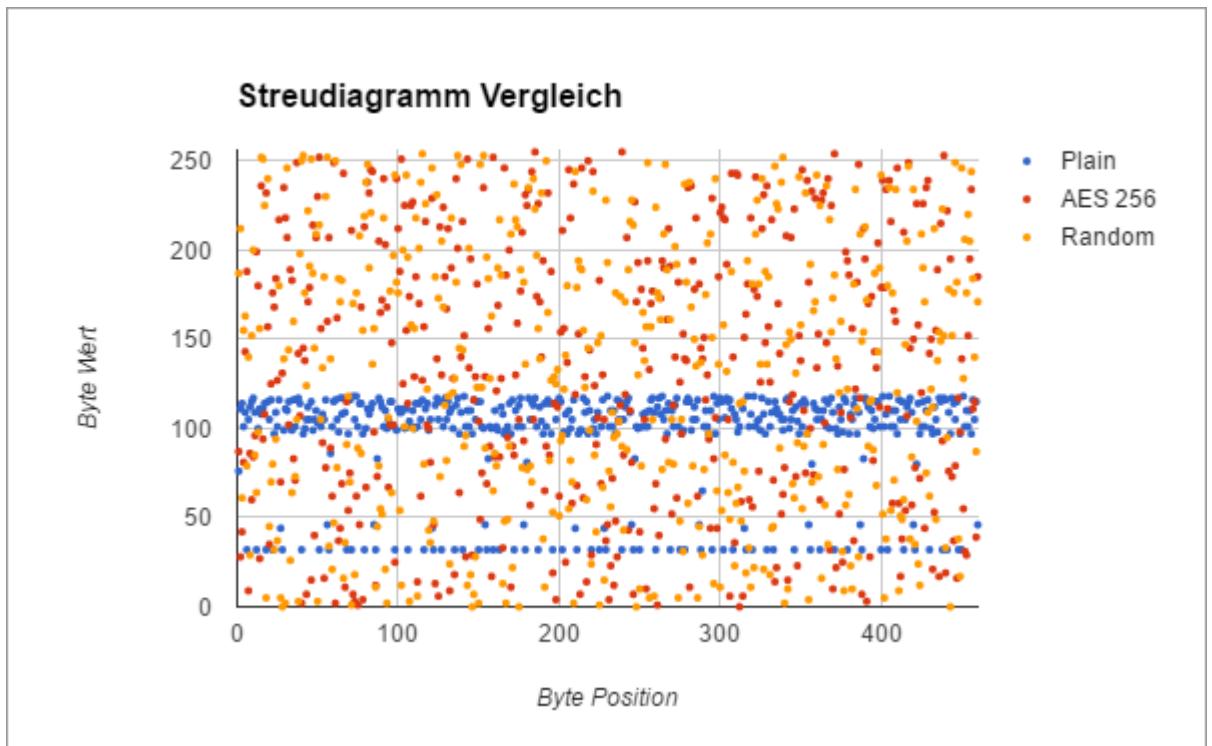


Figure 1.9: Vergleich zwischen AES verschlüsselten Daten und zufälligen Werten.

[L-AES-Encryption-Standard]

1.2.2 Fehlerkorrekturverfahren

Moderne Verfahren sind sehr anfällig auf verlustbehaftete Komprimierung. Dadurch, dass sie meistens darauf basieren, einzelne Bits zu verändern, passiert es auch sehr schnell, dass die erneute Manipulation des Trägermaterials zum Verlust der versteckten Daten führt. In dem folgenden Abschnitt werden Techniken vorgestellt, um diesen Informationsverlust zu verhindern oder wenigstens zu entdecken.

Redundante Datenspeicherung

Eine Möglichkeit ist es, die Daten mehrfach abzuspeichern. Das bietet Schutz vor Manipulation von einzelnen Teilen des Bildes. Zum Beispiel:

- Das Wegschneiden von Rändern

- Das Entfernen von Gesichtern
- Wasserzeichen
- Bild Über- und Unterschriften

Keine Verbesserung der Robustheit findet jedoch bei der Komprimierung des Trägermaterials statt, denn hier werden alle Teile geringfügig verändert, was bereits ausreicht, um einen kompletten Informationsverlust zu verursachen.

Wenn das Trägermaterial vergrößert/verkleinert wird, hilft redundante Speicherung meist nicht. Bei Skalierung von Bildern werden immer Filtering Methoden eingesetzt. Diese kombinieren oft mehrere Pixel zu einem einzelnen und wenden dabei zahlreiche Operationen an. Dadurch gehen fast immer sämtliche steganographisch versteckten Informationen verloren.

Paritätsbit

Die Parität einer Zahl beschreibt, ob eine Zahl durch zwei teilbar ist. Das Paritätsbit wird zur Überprüfung auf Übertragungsfehler verwendet, indem es nach einer festgelegten Anzahl an übertragenen Bits mitgesendet wird. Es werden die gesetzten Bits des zu überprüfenden Bereich gezählt. Ist dieser Wert nun eine gerade/ungerade Zahl, wird das Paritätsbit entsprechend gesetzt.

Wenn im Laufe der Übertragung ein einzelnes Bit umgedreht wird, kann dies durch das Paritätsbit entdeckt werden. Der Fehler selbst kann dadurch aber nicht ausgebessert werden, da nicht bekannt ist, wo er in der Bitfolge aufgetreten ist. Außerdem kann nur eine ungerade Anzahl an Fehlern festgestellt werden.

Zyklische Redundanzprüfung (z.B. CRC-32)

Die zyklische Redundanzprüfung basiert auf Polynomdivision. Es wird ein CRC-Polynom gewählt, dessen Koeffizienten entweder 1 oder 0 sind. So entspricht etwa die Bitfolge 110101 dem Polynom $x^5 + x^4 + x^2 + 1$.

Die zu prüfende Bitfolge wird nun mit einem Padding aus n Nullen versehen, wobei n dem Grad des Polynoms entspricht. Das Ergebnis wird durch das CRC-Polynom dividiert, der Rest der Division an die ursprüngliche Bitfolge ohne Padding angehängt und das Ganze dann übertragen. Dabei muss beachtet werden, dass bei der Division ausschließlich der XOR-Operator verwendet wird.

Table 1.2: Kodierungstabelle der Payload

Zeichen	DEC	BIN	Zeichen	DEC	BIN	Zeichen	DEC	BIN	Zeichen
A	0	000000	I	8	001000	Q	16	010000	Y
B	1	000001	J	9	001001	R	17	010001	Z
C	2	000010	K	10	001010	S	18	010010	Leerzeichen
D	3	000011	L	11	001011	T	19	010011	.
E	4	000100	M	12	001100	U	20	010100	,
F	5	000101	N	13	001101	V	21	010101	?
G	6	000110	O	14	001110	W	22	010110	!
H	7	000111	P	15	001111	X	23	010111	"

Auf der Empfängerseite wird die gesamte übertragene Bitfolge erneut durch das CRC-Polynom dividiert. Beträgt der Rest Null, dann ist entweder kein Fehler aufgetreten oder ein sehr unwahrscheinlicher. Mit diesem Verfahren können nicht nur Fehler entdeckt, sondern im besten Fall sogar ausgebessert werden.

[L-Fehlererkennung]

Aus diesem Grund wird in den Beispielen die Kodierung aus 1.2 für die Payload verwendet.

1.2.3 Least Significant Bit Verfahren

Das Least Significant Bit (LSB) Verfahren nützt die Tatsache aus, dass der Farbraum einer modernen Bilddatei sehr groß ist⁹. Dadurch ist es für das menschliche Auge schwierig, sehr ähnliche Farbtöne zu unterscheiden. Computerprogramme können nun gezielt einzelne Farben manipulieren, um Informationen in den Pixel des Bildes zu kodieren.

Bei einem 24-Bit RGB Bild besteht jeder Farbkanal aus 8 Bit. Um die Nachricht zu kodieren, wird das LSB von einem oder mehreren der Farbkanäle des Pixel auf den jeweiligen Wert aus der kodierten Nachricht gesetzt. Da das LSB nur einen wertmäßigen Unterschied von $+/-1$ ausmacht wenn es verändert wird, fällt diese Änderung kaum auf. Wenn wir bei einem Pixel einen der Farbkanäle bearbeiten, verändern wir den Farbwert des Kanals um $\frac{1}{256} = 0.39\%$. Der Farbwert des Pixel ändert sich sogar nur um $\frac{1}{16777216} = 0.00000596\%$. Es ist so gut wie unmöglich, mit dem freien Auge hier noch einen Unterschied zu erkennen.

Ein große Rolle spielt, wieviel Trägermaterial zur Verfügung steht und wie viele Daten

⁹Bei den meisten Formaten 24-Bit, wodurch 16,777,216 Farben dargestellt werden können

darin eingebettet werden. Je nachdem kann man die einzelnen Bits weiter verteilen oder muss sogar mehrere Bits auf ein Pixel legen. Es ist auch durchaus sinnvoll diverse Prüfsummen in die eingebetteten Daten einzubauen um sicher zu stellen das die extrahierten Daten auch keine Fehler enthalten. Wenn genug Platz zur Verfügung steht ist es auch möglich die Daten redundant zu speichern.

¹⁰

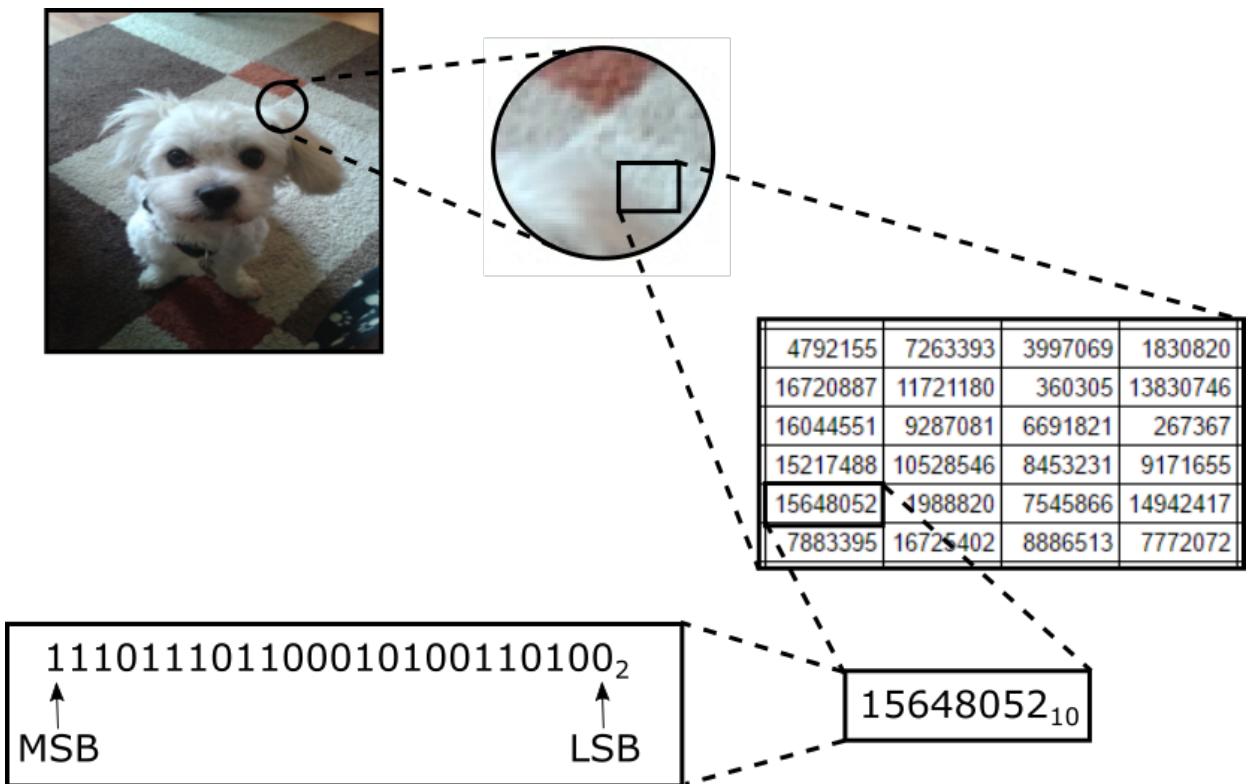


Figure 1.10: LSB Verfahren Erklärung

In dem Beispiel aus 1.11 wurde eine Datendichte von einem Zeichen pro Pixel gewählt. Bei einem Bild mit 2 Megapixel lassen sich bis zu 2 Millionen Zeichen abspeichern. Das Buch "Schöne neue Welt" von Aldous Huxley hat rund 430.000 Zeichen, geht sich also bereits vier Mal in einem doch recht kleinen Bild aus. Das setzt aber voraus, dass die in 1.2 angeführte Kodierungstabelle verwendet wird. Diese unterstützt aber weder Kleinbuchstaben noch Zeilenumbrüche, Umlaute oder Zahlen. Das Buch wäre durchaus noch lesbar, aber eben doch recht umständlich. In der ASCII Kodierung hat das Buch eine Größe von 449.414 Bytes, also 3.595.312 Bits. Bei einer Datendichte von 6 Bit / Pixel geht sich das Buch dennoch in einem 2 Megapixel großen Bild aus.

¹⁰ TODO Tabelle mit Vergleich wie viele Bit Farbe / wie viele Bit Daten und wie viel Prozent das ausmacht + Bildbeispiele für den Farbunterschied den das ausmacht

Pixel Nummer	1	2	3	4	5
Nachricht	T	R	E	F	F
Dec	19	17	4	4	5
Binär	10011	10001	00100	00100	00101
Roter Farbwert	RRRR RR10	RRRR RR10	RRRR RR00	RRRR RR00	RRRR RR00
Grüner Farbwert	GGGG GG01	GGGG GG00	GGGG GG10	GGGG GG10	GGGG GG10
Blauer Farbwert	BBBB BBB1	BBBB BBB1	BBBB BBB0	BBBB BBB0	BBBB BBB1

Figure 1.11: Beispiel für Kodierung von einem Zeichen pro Pixel.

In dem in 1.12 gezeigten Beispiel wurde das Buch "Schöne neue Welt" in ein Bild der Erde integriert. Verwendet wurde das in 1.2 gezeigte Java-Programm. Die kurze Länge des Programms zeigt, dass dieses Verfahren sehr einfach zu implementieren ist. Man kann durchaus, egal wo man sich gerade befindet, mit Hilfe eines Computers ein Programm schreiben und geheime Nachrichten versenden. Durch den zusätzlichen Einsatz von Whitening kann also von überall aus eine steganografisch sichere Datenübermittlung stattfinden.

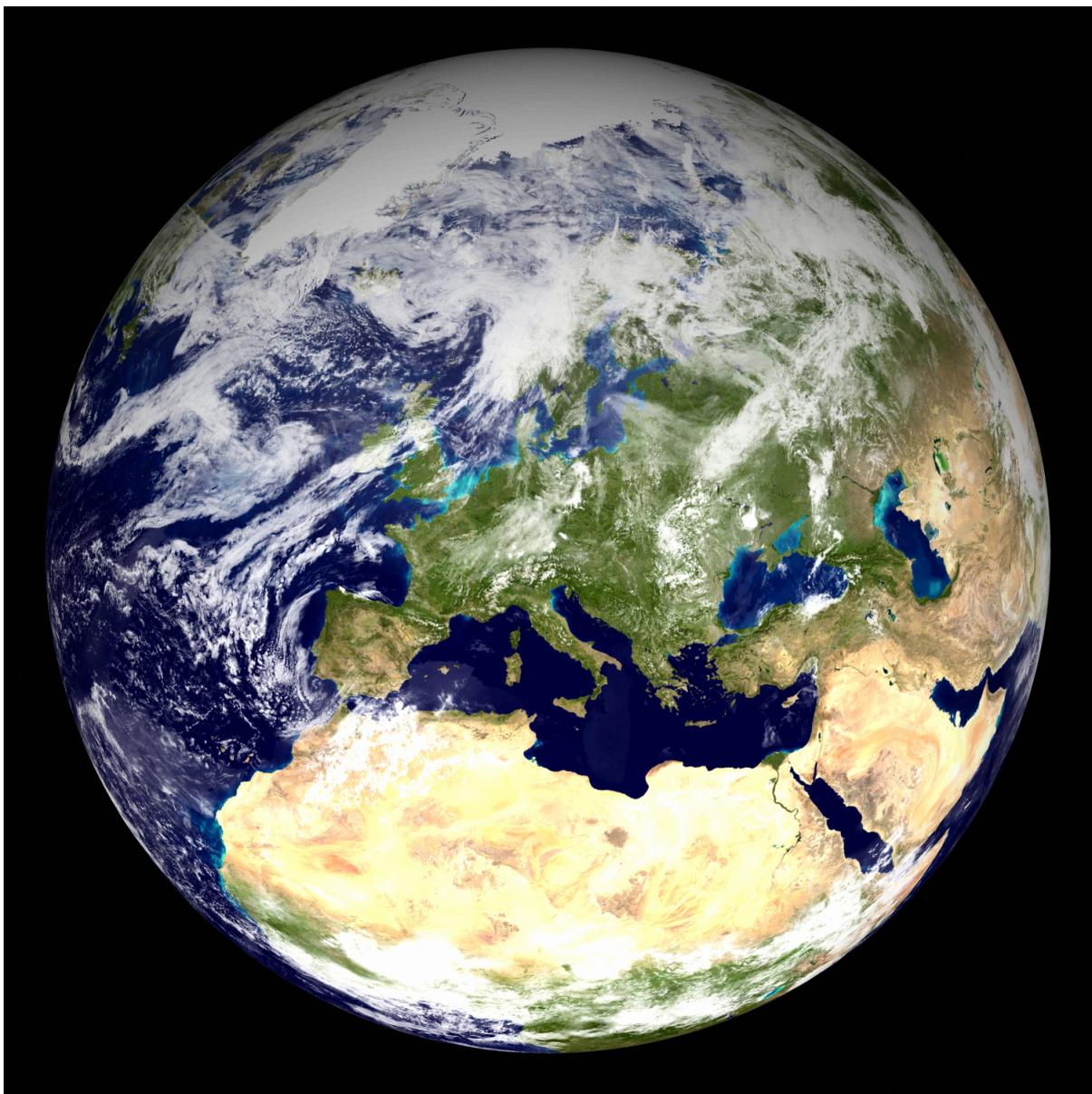


Figure 1.12: In diesem Bild wurde ein ganzes Buch versteckt, und es ist trotzdem nicht zu erkennen.

```
1 import java.awt.image.BufferedImage;
2 import java.io.File;
3 import java.io.IOException;
4 import java.nio.file.Files;
5 import static java.nio.file.StandardOpenOption.CREATE;
6 import static java.nio.file.StandardOpenOption.WRITE;
7 import java.util.Random;
8 import javax.imageio.ImageIO;
9
10 /**
11 *
```

```
12 * @author Simon Lehner-Dittenberger
13 */
14 public class LSBVerfahren {
15
16     public static final int SEED = 1234;
17
18     public static int bitpair(byte data[], int offset) {
19         if (offset / 8 >= data.length) {
20             return 0;
21         }
22         return (data[offset / 8] >> (offset % 8)) & 0x3;
23     }
24
25     public static void encode(BufferedImage image, byte[] data) {
26         Random random = new Random(SEED);
27         int bitOffset = -2;
28         for (int x = 0; x < image.getWidth(); x++) {
29             for (int y = 0; y < image.getHeight(); y++) {
30                 int pixel = image.getRGB(x, y);
31                 pixel = (pixel & (~0x030303)); //overwrite the two LSB bits
32                     with 0
33                 pixel |= (bitpair(data, bitOffset += 2) ^ random.nextInt(4)
34                         ) << 16;
35                 pixel |= (bitpair(data, bitOffset += 2) ^ random.nextInt(4)
36                         ) << 8;
37                 pixel |= (bitpair(data, bitOffset += 2) ^ random.nextInt(4)
38                         ) << 0;
39                 image.setRGB(x, y, pixel);
40             }
41         }
42     }
43
44     private static byte[] decode(BufferedImage image) {
45         Random random = new Random(SEED);
46         byte[] result = new byte[1 + (image.getWidth() * image.getHeight()
47             * 6) / 8];
48         int bitOffset = -2;
49         for (int x = 0; x < image.getWidth(); x++) {
50             for (int y = 0; y < image.getHeight(); y++) {
51                 int pixel = image.getRGB(x, y);
52                 bitOffset += 2;
53                 result[bitOffset / 8] |= (((pixel >> 16) ^ random.nextInt
54                     (4)) & 0x3) << ((bitOffset) % 8);
55                 bitOffset += 2;
56                 result[bitOffset / 8] |= (((pixel >> 8) ^ random.nextInt(4)
57                     ) & 0x3) << ((bitOffset) % 8);
58                 bitOffset += 2;
59                 result[bitOffset / 8] |= (((pixel >> 0) ^ random.nextInt(4)
60                     ) & 0x3) << ((bitOffset) % 8);
61             }
62         }
63     }
64 }
```

```

55     return result;
56 }
57
58 public static void extractDataFromImage(String args[]) throws
59     IOException {
60     BufferedImage image = ImageIO.read(new File("schÄne-neue-earth.png
61         ")); //load the original image
62     byte data[] = decode(image); //extract payload from image
63     Files.write(new File("schÄne-neue-welt-extract.pdf").toPath(),
64         data, WRITE, CREATE); //write extracted payload to hdd to test if
65         it worked
66 }
67
68 public static void encodeDataIntoImage(String args[]) throws
69     IOException {
70     BufferedImage image = ImageIO.read(new File("earth.png")); //load
71         the original image
72     byte data[] = Files.readAllBytes(new File("schÄne-neue-welt.pdf").
73         toPath()); //load the original payload file
74     encode(image, data); //add payload to image
75     data = decode(image); //extract payload from image
76     ImageIO.write(image, "png", new File("schÄne-neue-earth.png")); ////
77         write image with payload to hdd
78     Files.write(new File("schÄne-neue-welt-extract.pdf").toPath(),
79         data, WRITE, CREATE); //write extracted payload to hdd to test if
80         it worked
81 }
82
83 public static void main(String[] args) throws IOException {
84     encodeDataIntoImage(args);
85     extractDataFromImage(args);
86 }
87 }
```

Listing 1.2: LSBVerfahren.java

Wenn die einzubettenden Daten nicht den gesamten verfügbaren "Speicherplatz" innerhalb des Steganogramms verwenden, sollte unbedingt ein Padding aus zufälligen Daten hinzugefügt werden. Sonst können sich deutliche Unterschiede in dem Aussehen des Bildes ergeben, wenn es große, einfarbige Flächen besitzt. Diesen Effekt kann man in 1.14 deutlich erkennen.

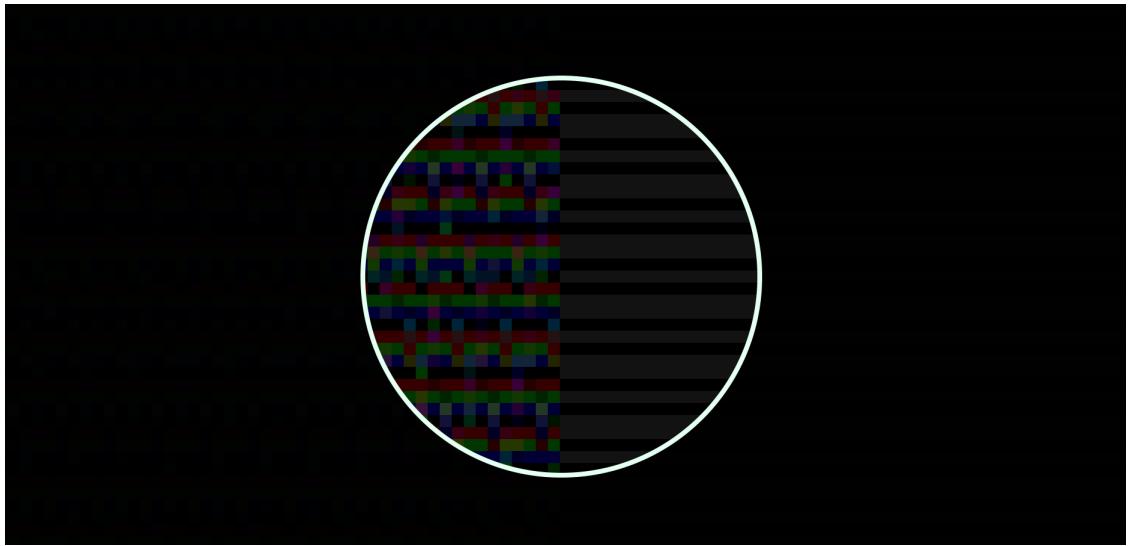


Figure 1.13: Auf der linken Seite befinden sich eingebettete Daten, auf der rechten Seite nicht. Innerhalb des Kreises wurde der Effekt verstkt dargestellt.

Die resultierende Datei ist deutlich grer als das Original. Das liegt daran, dass durch das Einbetten der Daten groe einfarbige Flchen minimal verndert wurden, wodurch sie sich nicht mehr so gut komprimieren lassen wie zuvor.

Varianten des Least Significant Bit Verfahren

Das LSB Verfahren gibt es in zahlreichen Varianten und Abstufungen. Deswegen ist es durchaus eine Herausforderung, automatische Erkennungsverfahren fr diese Steganogramme zu entwickeln. Des weiteren kommt es hufig vor, dass die eingebetteten Daten verschlselt wurden, was eine automatisch Erkennung beinahe unmglich macht.

Selbstverndlich kann das LSB Verfahren nicht nur auf verlustfreie Bildformate wie PNG Dateien angewendet werden. Es gibt auch Implementierungen, welche Musikdateien oder verlustbehaftete Bildformate wie JPEG als Trgerdatei verwenden.

Der bekannteste Algorithmus dafr ist JSteg. Bei jedem verlustbehafteten Format gibt es irgendwann einen Teil, wo Daten, auch wenn bereits komprimiert, verlustfrei gespeichert werden. Das ntzt JSteg aus und wendet dort das LSB Verfahren an. Bei dem JPEG Format geschieht dies wie in ?? gezeigt, nach dem verlustbehafteten Teil des JPEG Algorithmus. Es wird dabei darauf geachtet, dass nur Bits verndert werden, welche eine mglichst geringe Auswirkung auf das originale Bild haben.

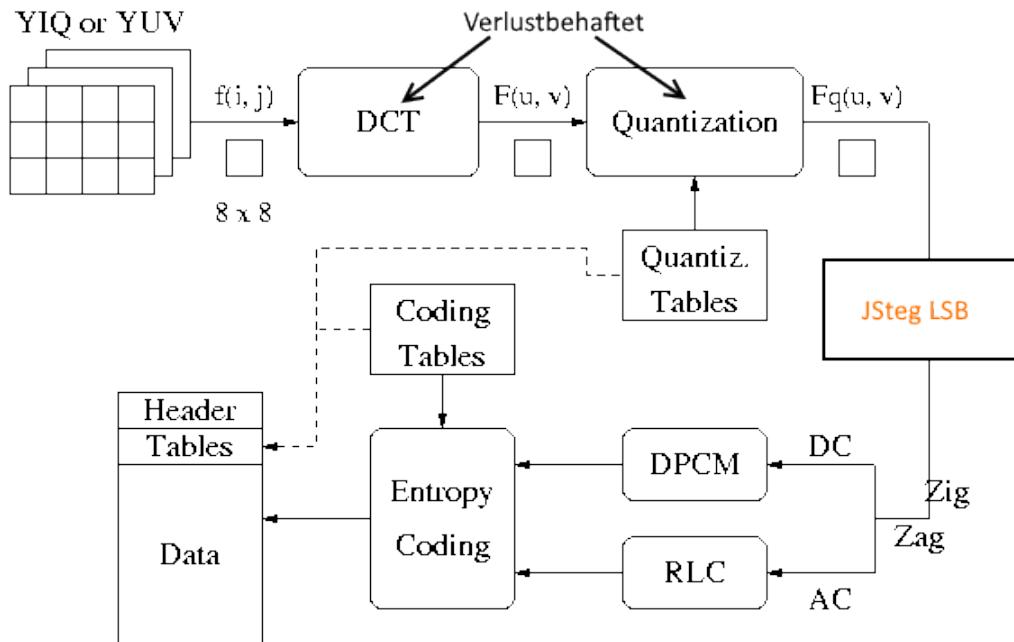


Figure 1.14: Die Daten werden im verlustfreien Teil des JPEG Algorithmus hinzugefügt.

[L-Jpeg-Algorithm] [L-JSteg]

1.2.4 Barcode Verfahren

Das Barcodeverfahren ist ein selbsterfundenes Verfahren, welches einen Barcode in einem Bild versteckt. Dies geschieht, indem das Bild zuerst in ein 8-Bit Graustufenbild konvertiert und anschließend das Histogramm des Bildes generiert wird.

Ein Histogramm ist eine grafische Abbildung der Anzahl einzelner Farbwerte. In das 1.15 wurde das Bild zuerst in ein 8-Bit Graustufenbild umgewandelt, die entstehenden 256 verschiedenen Werte gezählt und grafisch dargestellt.

Wenn man nun das Bild gezielt so bearbeitet, dass bestimmte Farbwerte nicht mehr vorkommen, dann ergeben sich dadurch auch im Histogramm entsprechende Spalten. Mit dieser Hilfe kann nun jedes beliebige eindimensionale Barcodeformat eingebettet werden.

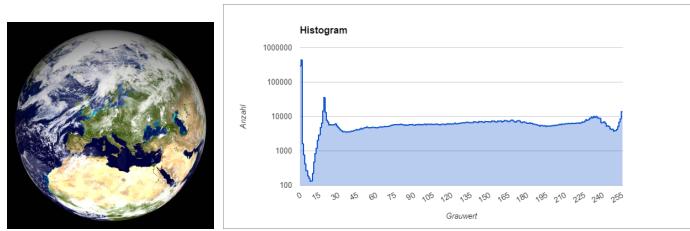


Figure 1.15

Damit gängige Barcodescanner jedoch den Barcode wahrnehmen können, muss das Histogramm grafisch gewisse Anforderungen erfüllen. Die meisten vorhandenen Histogramm-Generatoren stellen das Resultat in einer schönen ansprechenden Form dar, z.B. wird der Graph geglättet und in Farbe hinterlegt. Barcodescanner brauchen aber schwarze Striche, welche deutlich voneinander unterscheidbar sind. Deshalb muss ein spezieller Barcodegenerator verwendet werden.

Ein Beispiel für einen solchen Generator findet sich in 1.3. Dieses Programm generiert dann für Barcodescanner verwendbare Histogramme, wie in 1.16 gezeigt wird. In diesem Beispiel wurde das gängige Barcodeformat CODE_128 verwendet und der Text "Simon Lehner-D." damit kodiert.

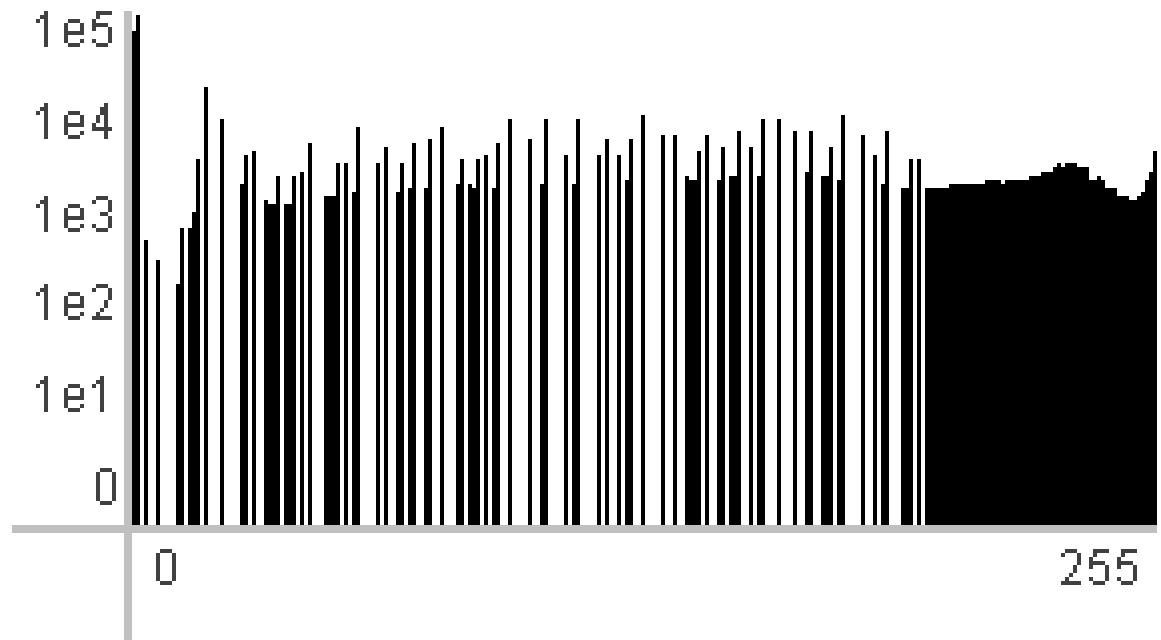


Figure 1.16: fig:L: Barcode Histogram Example

¹¹¹¹TODO Einfach nur den link zum git repo reintern?

```
1 import java.awt.Color;
2 import java.awt.Graphics2D;
3 import java.awt.image.BufferedImage;
4 import java.io.File;
5 import java.io.IOException;
6 import java.util.function.Consumer;
7 import javax.imageio.ImageIO;
8
9
10 /**
11  * 
12  * @author Simon
13  */
14 public class HistogramGenerator {
15
16     public static int r(int c) {
17         return (c >> 16) & 0xFF;
18     }
19
20     public static int g(int c) {
21         return (c >> 8) & 0xFF;
22     }
23
24     public static int b(int c) {
25         return (c) & 0xFF;
26     }
27
28     public static int avg(int c) {
29         return (r(c) + g(c) + b(c)) / 3;
30     }
31
32     public static String str(int c) {
33         return String.format("[%d,%d,%d]", r(c), g(c), b(c));
34     }
35
36     public static void forEach(BufferedImage img, Consumer<Integer>
37         consumer) {
38         for (int y = 0; y < img.getHeight(); y++) {
39             for (int x = 0; x < img.getWidth(); x++) {
40                 consumer.accept(img.getRGB(x, y));
41             }
42         }
43     }
44
45     public static void main(String[] args) throws IOException {
46         if (args.length != 2) {
47             System.out.println("java HistogramGenerator [inputfile] [
48                 outputfile]");
49             return;
50         }
51         File in = new File(args[0]);
52
53         BufferedImage img = ImageIO.read(in);
54
55         HistogramGenerator.forEach(img, c -> {
56             System.out.println(str(c));
57         });
58
59         File out = new File(args[1]);
60         ImageIO.write(img, "png", out);
61     }
62 }
```

```
50     File out = new File(args[1]);
51     if (out.isDirectory() || in.isDirectory()) {
52         System.out.println("Bitte zwei gültige Dateinamen angeben");
53         return;
54     }
55     if (!in.exists()) {
56         System.out.println("Datei " + in + " konnte nicht gefunden
57             werden");
58         return;
59     }
60     if (out.exists()) {
61         System.out.println("Datei " + out + " existiert bereits");
62         return;
63     }
64     BufferedImage source = ImageIO.read(in);
65     int[] histogram = new int[256];
66     int[] stat = new int[]{Integer.MIN_VALUE};
67     forEach(source, (color) -> {
68         int grey = avg(color);
69         histogram[grey]++;
70         stat[0] = Math.max(histogram[grey], stat[0]);
71     });
72     double mx = Math.max(0, Math.log10(stat[0]));
73     BufferedImage result = zeichneHistogram(histogram, 256, mx);
74     BufferedImage output = zeichneBeschriftung(result, mx);
75     ImageIO.write(output, "png", out);
76 }
77
78 public static BufferedImage zeichneHistogram(int histo[], int height,
79 double mx) {
80     double scale = 256.0 / mx;
81     BufferedImage img = new BufferedImage(histo.length, height,
82         BufferedImage.TYPE_INT_RGB);
83     for (int i = 0; i < histo.length; i++) {
84         //Verwende den 10er Logarithmus um große Werte darstellen zu
85         //können.
86         int to = (int) (scale * Math.log10(histo[i]));
87         //Begrenze werte auf den Bereich des Bild
88         to = Math.max(0, Math.min(height - 1, to));
89         //Färbe eine Spalte ein
90         for (int j = 0; j < height; j++) {
91             img.setRGB(i, j, ((j >= to) ? Color.white : Color.black) .
92                 getRGB());
93         }
94     }
95     return img;
96 }
97
98 public static BufferedImage zeichneBeschriftung(BufferedImage source,
99     double mx) {
100    double scale = source.getHeight() / mx;
```

```

95     BufferedImage result = new BufferedImage(30 + source.getWidth(), 30
96         + source.getHeight(), BufferedImage.TYPE_INT_RGB);
97     Graphics2D ctx = result.createGraphics();
98     //Zeichne die Achsen und das Histogram
99     Graphics2D ctx1 = (Graphics2D) ctx.create();
100    ctx1.scale(1, -1);
101    ctx1.translate(0, -result.getHeight());
102    ctx1.setColor(Color.white);
103    ctx1.fillRect(0, 0, result.getWidth(), result.getHeight());
104    ctx1.drawImage(source, 30, 30, null);
105    ctx1.setColor(Color.lightGray);
106    ctx1.drawRect(28, 0, 1, result.getHeight());
107    ctx1.drawRect(0, 29, result.getWidth(), 1);
108    ctx1.dispose();
109    //Zeichne die Beschriftung
110    Graphics2D ctx2 = (Graphics2D) ctx.create();
111    ctx2.setColor(Color.darkGray);
112    ctx2.drawString("0", 35, result.getHeight() - 15);
113    ctx2.drawString("0", 20, result.getHeight() - 35);
114    for (int i = 1; i <= Math.ceil(mx); i++) {
115        String txt = String.format("1e%d", i);
116        ctx2.drawString(txt, 5, (int) (result.getHeight() - 35 - i *
117            scale));
118    }
119    ctx2.drawString("255", result.getWidth() - 25, result.getHeight() -
120        15);
121    ctx2.dispose();
122    return result;
123}

```

Listing 1.3: HistogramViewer.java

1.2.5 Stereoskopische Verfahren

Die Stereoskopie beschreibt Methoden, um auf Bildern den Eindruck zu machen, dass diese dreidimensional sind, obwohl physikalisch gar keine Tiefe vorhanden ist. Sie macht sich dabei zur Hilfe, dass das Gehirn eigentlich nur zwei 2D-Aufnahmen der Umgebung bekommt und diese dann erst als 3D-Gebilde interpretiert.

Um im Gehirn den Eindruck von räumlicher Tiefe zu erzeugen, müssen beide Augen dieselbe Szene aus zwei verschiedenen Blickwinkeln zu sehen bekommen. Dies funktioniert normalerweise dadurch, dass beide Augen in einem Abstand von ungefähr 15cm voneinander entfernt sind. Wenn nun aber beide Blickwinkel mit einer Kamera aufgenommen und nebeneinander gelegt wurden, muss man dem Gehirn etwas nachhelfen. Mit der richtigen Blicktechnik, z.B. durch Schielen, können die beiden Teilbilder

wieder übereinandergelegt werden und der gewünschte Tiefeneffekt tritt auf.



Figure 1.17: Beispiel fr ein Stereogramm

¹²

Es gibt verschiedene Techniken, um Stereogramme zu erstellen, die sich vor allem durch die Wahl des Trägermaterials unterscheiden.

- Classic stereogram

Das klassische Stereogramm aus 2 Bildern, welche mit der richtigen Blicktechnik oder einem Stereoskop dreidimensional gesehen werden können.

- Single image stereogram (SIS)

Einzelbildstereogramme, auch oft Autostereogramm genannt, verwenden, wie der Name schon sagt, nur ein einzelnes Trägerbild, um die Formen zu beinhalten. Dazu benötigen sie ein sich wiederholendes Muster, in welches mit Hilfe eines Programms und einer Depthmap die Formen hinzugefügt werden.¹³

Grundsätzlich kann jede der hier genannten Techniken auch Autostereogramme erzeugen.

- Random dot stereogram (RDS)

Bei diesem werden Bilder verwendet, die aus zufälligen Punkten bestehen, um dreidimensionale Formen darzustellen. Dadurch ist es oft auf den ersten Blick nicht gleich ersichtlich, dass es sich hier um ein Stereogramm handelt.

- Text stereogram

Als Trägermaterial wird ausschließlich Text verwendet. Ein Beispiel dafür ist unter 1.18 zu sehen.

¹²TODO Quelle raus suchen oder eigenes Bild aufnehmen

¹³TODO was ist eine Depthmap wir leben in der Vergangenheit

- Map textured stereogram

Diese Technik ist sehr ähnlich wie RDS, nur dass hier Texturen - wie sie etwa in Videospielen vorkommen - verwendet werden. Diese haben die Eigenschaft, beliebig oft wiederholt werden zu können, ohne dass sich Kanten im Bild ergeben. Dadurch lässt sich sehr gut ein Stereogramm aus ihnen erzeugen.

Die bekannte Buchserie "Magic Eye" beinhaltet vor allem solche Stereogramme, weil die Blicktechnik hier einfacher anzuwenden ist als z.B. bei Textstereogrammen.

- Wallpaper stereogram / object array stereogram

Bei einem Objekt Array Stereogramm wird ein und dasselbe Bild mehrere Male wiederholt. Dabei werden aber die Abstände von einzelnen Elementen des Bild unterschiedlich gewählt um den gewünschten 3D Effekt zu erzielen.

```

IIIIIIIIIIIIIIII   IIIIIIIIIIIIIII   Marty Hewes
H ( ) \|/ H   H ( ) \|/ H   Tellabs Inc.
H( ) -O- H   H ( ) -O- H   System and Applications Test
H )/|\ H   H )/|\ H   Lisle, IL
H=====^=====H   H=====^=====H   nitram@tellabs.com
H- |----@----H   H----|---@---H   uunet!tellab1.TELLABS.COM!nitram
H /|\ @|\|/ @ H   H /|\@|\|/ @ H
H \||/ \|/ H   H \||/ \|/ H   W (708) 955-3038
III^IIIIIIII^III   III^IIIIIIII^III   H (708) 665-6671
Wide eyed stereo   Wide eyed stereo

```

Figure 1.18: Beispiel für ein Text Stereogramm

14



Figure 1.19: Beispiel für ein Objekt-Array Stereogramm

¹⁴TODO Quelle: <https://web.archive.org/web/20080517013244/archive.museophile.org/3d/ascii-3d.html>

¹⁵

Aus diesen Techniken ergeben sich nun einige praktische Anwendungsmöglichkeiten, welche im Folgenden genauer bearbeitet werden. Des weiteren werden Hilfsmittel gezeigt, mit deren Hilfe man Stereogramme automatisch entdecken und entschlüsselt darstellen kann.

Erzeugen von Stereogrammen mithilfe einer Depth Map

Zuerst benötigt man sowohl ein gutes Muster als Grundlage, als auch eine sogenannte Depth Map. Depth-Maps sind im Grunde einfach Schwarz-Weiß Bilder bei denen der Weißwert jedes Pixels die Tiefe/Höhe der jeweiligen Position angibt. Beide Teile gibt es zahlreich im Internet. Bei Bedarf können Depth Maps auch mit einer geeigneten 3D-Grafiksuite wie etwa Blender erzeugt werden. Aber auch einfache Bildbearbeitungsprogramme sind gut dazu geeignet, vor allem Text als Depth Map abzuspeichern.

Dabei gilt der Grundsatz: Je einfacher das dargestellte Objekt in der Depth Map, desto einfacher ist es auch, das Stereogramm ohne technische Hilfsmittel dreidimensional sehen zu können.

Um das Stereogramm nun zu erzeugen, wird ein neues, leeres Bild erzeugt mit derselben Größe wie die Depth Map. Auf diesem Bild wird nun das gewählte Muster auf dem linken Rand hineinkopiert, sodass es von oben bis unten einen Streifen mit dem Muster bildet. Die Größe dieses Grundstreifens wird von nun an N genannt, und gibt an, um wie viel die Augen das Bild verschieben müssen, um den dreidimensionalen Effekt sehen zu können.

Die restlichen noch leeren Teile des Bildes werden nun erzeugt, indem immer das Pixel auf derselben Höhe um N nach links verschoben kopiert wird. Um den gewünschten Tiefeneffekt zu bekommen, wird noch jeweils ein Offset hinzugefügt, das dem Wert der aktuellen Position auf der Depth Map entspricht.

$$E(x, y) = \begin{cases} M(x, y), & \text{if } x < N \\ E(x - N + (s \cdot D(x, y))), & \text{else} \end{cases}$$

- $E(x, y)$ das resultierende Stereogramm
- $M(x, y)$ das Muster welches als Grundlage verwendet werden soll.

¹⁵TODO Quelle: <http://www.pakin.org/scott/stereograms/>

- $D(x, y)$ die Depth Map [0.0; 1.0]
- s ein Faktor der Angibt wie groß der Tiefeneffekt sein soll [0.0; N]

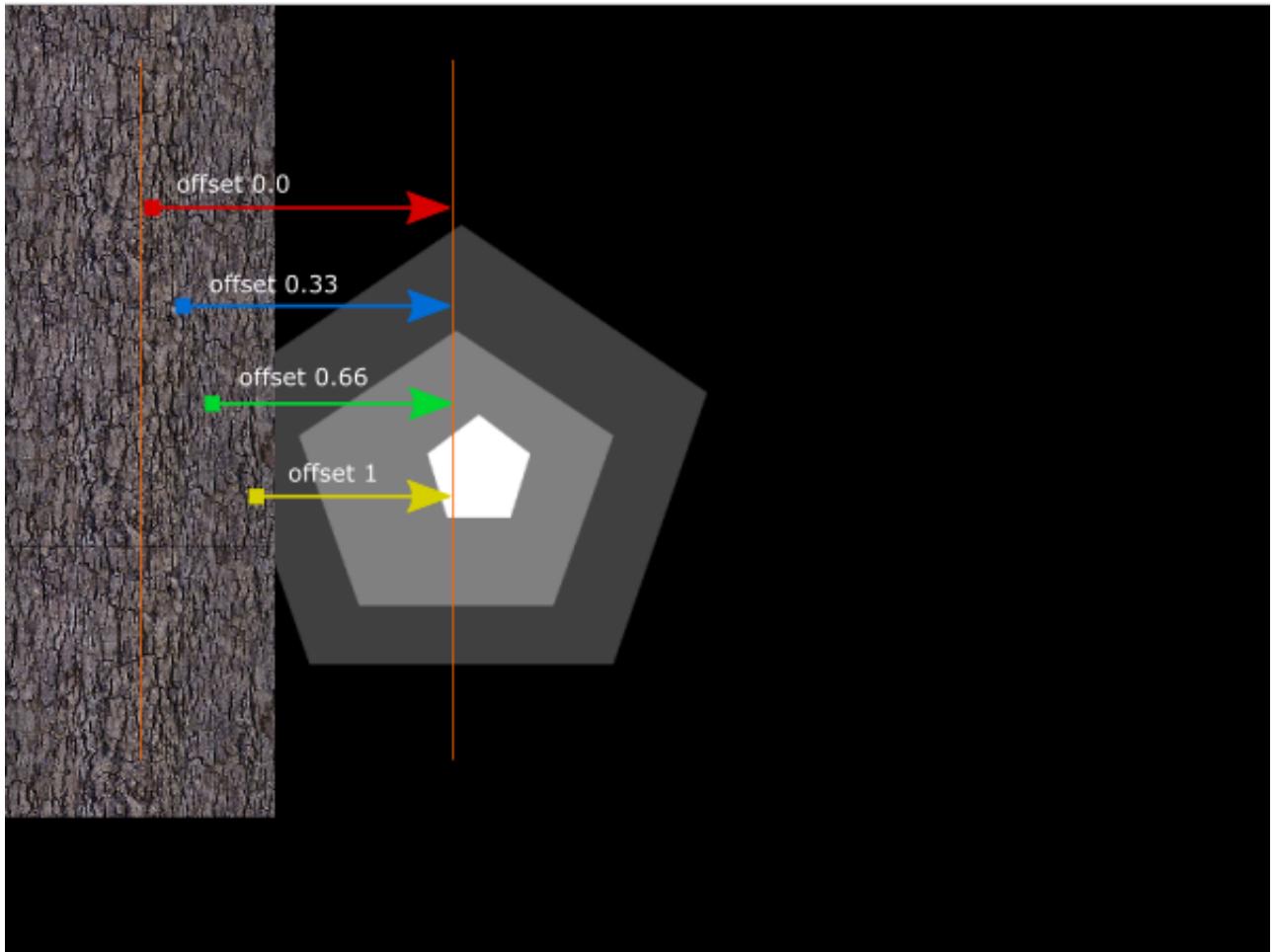


Figure 1.20: Die Pixel werden je nach Depth Map vom linken Strich + Offset zu dem rechten Kopiert.

Wiederholt man diesen Vorgang über das gesamte Bild ergibt sich dann folgendes Stereogramm.

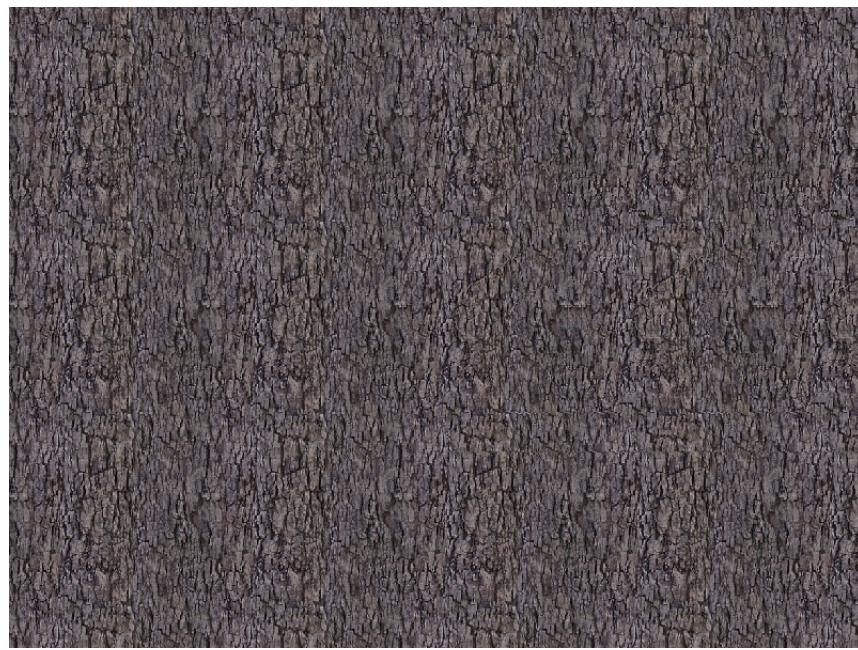


Figure 1.21: Kaum zu erkennen sind die versteckten Formen.

Verwendet wurde dafür dieses Java-Programm:

```
1 import java.awt.image.BufferedImage;
2 import java.io.File;
3 import javax.imageio.ImageIO;
4
5 /**
6 * 
7 * @author Simon
8 */
9
10 public class StereogramGenerator {
11
12     public static void main(String[] args) throws Exception {
13         double scale = 20;
14         BufferedImage source = ImageIO.read(new File("texture2.jpg"));
15         BufferedImage depthmap = ImageIO.read(new File("depthmap.png"));
16         BufferedImage result = new BufferedImage(depthmap.getWidth(),
17             depthmap.getHeight(), BufferedImage.TYPE_INT_RGB);
18         //Für jede Zeile
19         for (int y = 0; y < result.getHeight(); y++) {
20             //Kopiere das Muster das erste mal auf der linken Seite
21             for(int x = 0; x < source.getWidth(); x++) {
22                 result.setRGB(x, y, source.getRGB(x, y % source.getHeight()
23                     ));
24             }
25             //Kopiere alle weiteren Teile von links verschoben um das
26             //Offset aus der Depth Map
```

```

24     for (int x = source.getWidth(); x < result.getWidth(); x++) {
25         result.setRGB(x, y, result.getRGB(x - source.getWidth() +
26             (int)(offset(depthmap, x, y) * scale), y));
27     }
28     ImageIO.write(result, "png", new File("what.png"));
29 }
30
31 private static double offset(BufferedImage depthmap, int x, int y) {
32     return (depthmap.getRGB(x, y) & 0xFF) / 255.0;
33 }
34
35 }
```

Listing 1.4: StereogrammGenerator.java

Steganographie mithilfe von Stereogrammen

Ein Autostereogramm ist eigentlich schon ein Steganogramm, da ja der eigentliche Inhalt auf den ersten Blick nicht ersichtlich ist. Wenn man ein Schwarz/Weiß Bild mit Text drauf als Depth Map verwendet, ergibt das ein Stereogramm welches einen geheimen Text enthält.

Grafisches Hilfsmittel zum Entschlüsseln von Stereogrammen

Wenn man nun keine Motivation oder nicht genügend Zeit hat, die richtige Blicktechnik zu erlernen, kann man auf technische Hilfsmittel zurückgreifen.

Meistens reicht es aus, das Bild in einem geeigneten Bildbearbeitungsprogramm zu öffnen und dort richtig zu bearbeiten: Es muss das gesamte Bild kopiert und wieder eingefügt werden. Das eingefügte Bild wird dann auf derselben Höhe solange nach links und rechts verschoben, bis man den gewünschten Effekt erzielt. Dabei ist zu beachten, dass das Programm die Composite-Einstellungen "Differenz" unterstützt und diese auch verwendet wird.

Das Wort Composite kommt aus dem Englischen und bedeutet so viel wie Zusammensetzung oder Gemisch. Im Sinne von Bildbearbeitung ist damit gemeint, auf welche Art und Weise die Pixel miteinander verbunden werden sollen. Für Stereogramme müssen die Pixel voneinander subtrahiert werden, es wird also die Differenz der Pixel benötigt. Daher auch die entsprechende Einstellung.

Wenn der Bedarf da ist, des öfteren Stereogramme zu entschlüsseln, dann ist es oft

hilfreich, ein kleines Programm dafür zu verwenden. Es gibt zum Beispiel eine Implementierung eines solchen Programms unter <http://magiceye.ecksdee.co.uk/>. Diese verwendet jedoch Low-Level Javascript Array Operationen, wodurch die Anwendung bei sehr großen Dateien zu langsam und damit unbrauchbar ist.

Moderne Browser unterstützen für die Rendering Kontexte in Javascript die oben erwähnten Composite-Operationen. Daher kann eine eigene kleine Funktion als Solver in nicht mehr als 5 Lines of Code geschrieben werden. Diese verwendet Hardwarebeschleunigung und ist somit selbst bei riesigen Dateien noch ausreichend schnell. Diese Funktion muss nur noch in eine kleine Oberfläche verpackt werden und ist schon einsatzbereit.

```

1  /*
2   * This functions renders img two times onto the provided context.
3   * One time shifted by 'offset' with the composite op 'difference'
4   * @param {CanvasRenderingContext2D} ctx
5   * @param {Image} img
6   * @param {} offset
7   */
8   var render = function (ctx, img, offset) {
9     ctx.drawImage(img, 0, 0, img.width, img.height);
10    ctx.globalCompositeOperation = 'difference';
11    ctx.drawImage(img, offset, 0, img.width, img.height);
12 };

```

Listing 1.5: Einfacher Stereogramm Solver

16

17

Automatische Erkennung von Stereogrammen

Es gibt noch kein Standardverfahren zur Erkennung, ob es sich bei einem Bild um ein Stereogramm handelt. Deswegen wurde im Rahmen des Projekts Geocaching-Tools und für diese Diplomarbeit ein eigenes Verfahren entwickelt.

Wie bereits bei den grafischen Hilfsmitteln zum Entschlüsseln von Stereogrammen gezeigt, kann man den versteckten Inhalt eines Stereogramm sichtbar machen, indem man eine Kopie von dem Bild verschiebt und von dem Original subtrahiert. Die Herausforderung und das Ziel des Verfahrens ist es also, das richtige Offset, falls vorhanden,

¹⁶TODO Javascript wird nicht unterstuetzt

¹⁷TODO Stereogramme mit verlaufenden 3D Bildern sind etwas komplizierter zum schoen darstellen, muss ich nur implementieren, hab da schon ne idee

zu erkennen.

Zuerst werden alle möglichen Verschiebungen generiert. Zwei identische Farben voneinander subtrahiert ergeben die Farbe Schwarz. Es wird also nach einem Offset gesucht bei dem ein möglichst großer Teil des Bildes schwarz wird. Damit ist auch schon der größte Teil der Arbeit getan. Aus den prozentualen Schwarzanteilen vom Gesamtbild über die verschiedenen Verschiebungen ergibt sich für jedes Bild eine Datenserie. Diese muss nur noch analysiert werden.

In 1.22 sind fünf Bilder mit dieser Technik analysiert worden. Dabei kann man gut erkennen, dass sich ein Spike in der Datenreihe ergibt, wenn das Bild ein Stereogramm ist. Das größte Problem bei der Sache sind Bilder, welche im Vorhinein bereits größtenteils aus schwarz bestehen. Auch Signaturen von Bildern welche nur sehr wenige Farben beinhalten, sind oft den Signaturen von Stereogrammen sehr ähnlich. Ein Beispiel dafür sind Bilder mit viel weißem Text auf schwarzem Hintergrund.

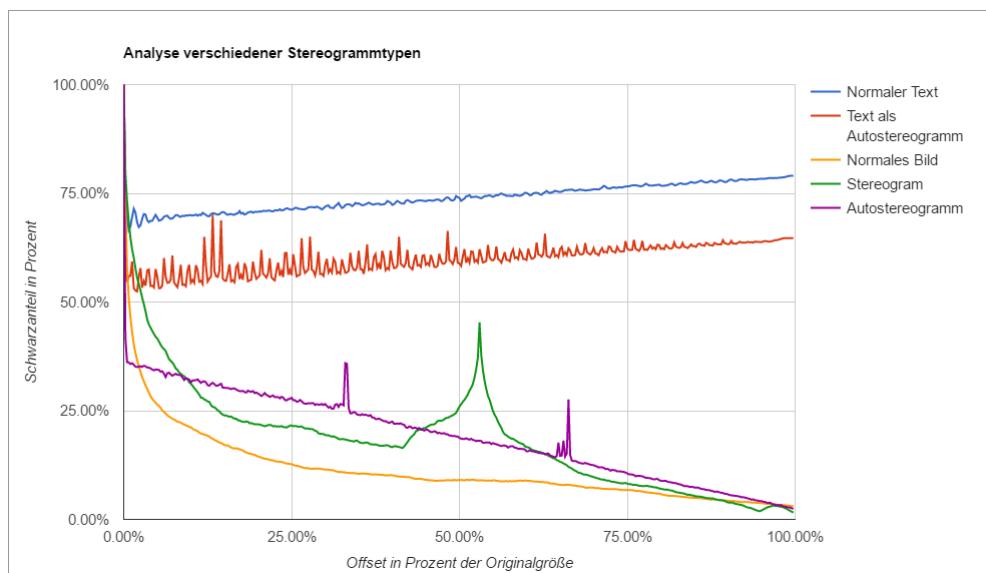


Figure 1.22: Bei dem Stereogramm kann ein Spike in der Datenreihe erkannt werden

Stereogramme haben mit dieser Analyse immer einen Spike in der Datenreihe, während normale Bilder einen eindeutigen Abwärtstrend haben. Lediglich Texte sind schwer zu unterscheiden und einzuordnen. Eine Möglichkeit wäre nun eine künstliche Intelligenz darauf zu trainieren, diese Datenreihen zu analysieren. Die hier generierten Daten eignen sich sehr gut für ein solches Verfahren, weil sie aufgrund der prozentuellen Angaben eine fixe Größe besitzen. Dafür wird jedoch sehr viel Rechenleistung benötigt, wodurch hier eine effizientere Methode gewählt wurde.

Gesucht wird aus der Menge D die größte Steigung zwischen zwei Punkten, $D_j -$

$D_i, i > j$ soll also maximiert werden. Dazu wurde folgender sehr einfacher und schneller Algorithmus gewählt:

```

1  public static double getBiggestDiff(double[] data) {
2      double mn = data[0];
3      double biggest = Double.MIN_VALUE;
4      for (int i = 0; i < data.length; i++) {
5          biggest = Math.max(biggest, data[i] - mn);
6          mn = Math.min(mn, data[i]);
7      }
8      return biggest;
9  }
10 }
```

Listing 1.6: Steigung

Hier ergibt sich das Problem, dass - wie in 1.22 zu sehen ist - Texte ein recht hohes Ergebnis bei diesem Test erzielen, da sie einen steigenden Trend besitzen. Dies ergibt sich aus dem hohen Schwarzanteil des Originalbildes. Dadurch, dass bei der Berechnung auch der sich nicht mit dem verschobenen Bild überschneidende Teil miteinbezogen wird, führt das zu diesen sehr hohen Werten.

Wenn bei dem Algorithmus nur der Teil zur Berechnung herangezogen wird, von dem das verschobene Bild subtrahiert wurde, verändern sich die Datenreihen deutlich: Die Spikes bei Stereogrammen vergrößern sich um ein Vielfaches und der generelle Trend wird flacher.

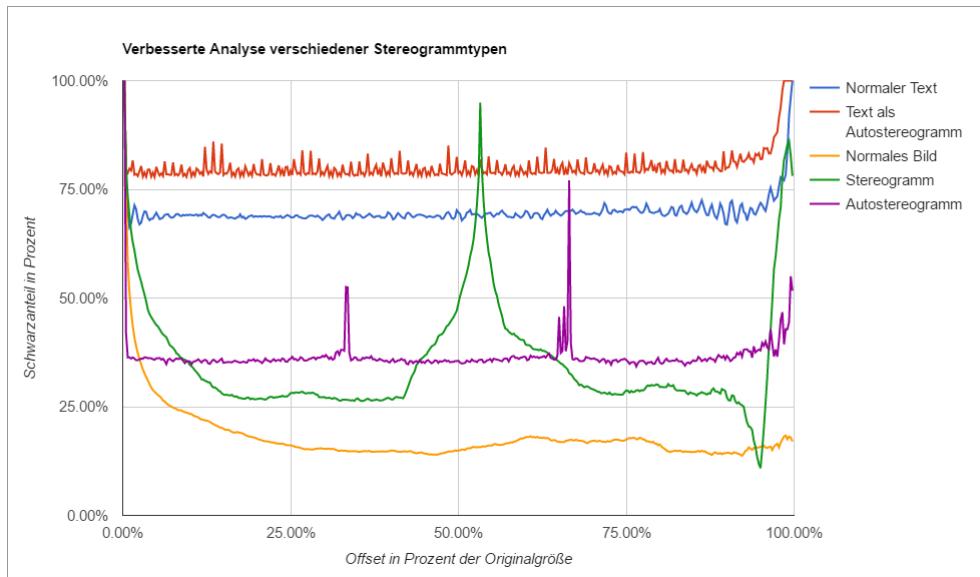


Figure 1.23: Die verbesserte Variante erzielt deutlichere Ergebnisse

Es ist aber auch zu beobachten, dass im letzten Viertel die Werte sehr stark zu

schwanken anfangen. Das liegt daran, dass der beobachtete Bereich immer kleiner wird und dadurch sehr wenige kleine Überschneidungen einen sehr großen Effekt auf das Ergebnis haben. Um noch bessere Ergebnisse zu erzielen kann also das letzte Viertel der Datenreihe vernachlässigt werden, da es bei einem Offset von mehr als 75% sehr unwahrscheinlich ist, dass es sich hier noch um ein Stereogramm handelt.

In der folgenden Grafik werden die drei Versionen miteinander verglichen.

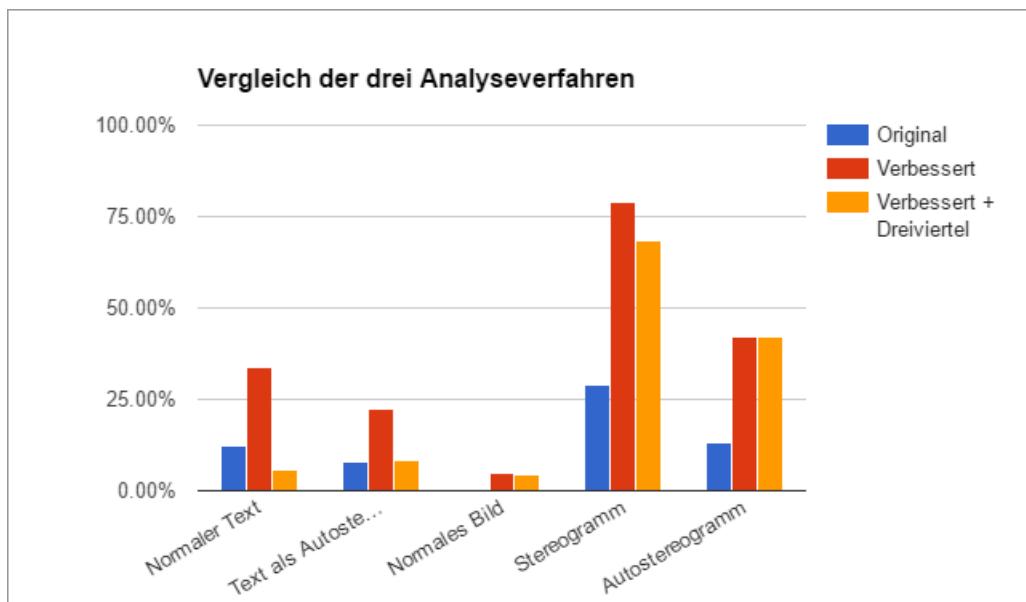


Figure 1.24: Ein Wert von mehr als 25% heißt, dass es sich Wahrscheinlich um ein Stereogramm handelt.

Wie man sehen kann, erzielt das eben vorgestellte Verfahren bei vier von fünf Testfällen ein korrektes Ergebnis. Um eine annähernd 100% korrekte Aussage treffen zu können, werden kompliziertere Verfahren benötigt, wie die oben vorgeschlagene Verwendung von künstlicher Intelligenz.

Bildfehler erkennen

Jeder kennt diese "Suche den Fehler"-Bilder aus Tageszeitungen. Diese bestehen aus zwei beinahe identischen Bildern. Bei einem der beiden wurden jedoch einzelne Details entfernt. Das ist ähnlich wie bei herkömmlichen Stereogrammen, bei denen die einzelnen Teile zwar nicht entfernt, sondern verschoben sind.

Dadurch, dass diese beiden Anwendungen so ähnlich sind, kann man sehr gut Hilfsmittel aus der Stereographie anwenden, um die Bildfehlerspiele zu lösen. Wenn man

die Blicktechniken von Stereogrammen auf diese Spiele anwendet, überlappen die beiden Bilder derart, dass auf derselben Stelle der entfernte Teil sichtbar und unsichtbar ist. Deshalb weiß das Gehirn nicht, welches der beiden Augen recht hat, weil das linke Auge zum Beispiel einen Schornstein sieht und das rechte nicht. Diese Teile des Bildes schauen dann aus, als würden sie flimmern.

Wenn ein automatisches Hilfsmittel auf ein derartiges Bildpaar angewendet wird, ergibt sich bei dem richtigen Offset ein schwarzes Bild, auf dem nur die Fehler als Farbhaufen zu sehen sind. Im Sinne der Steganographie kann zum Beispiel der Sekundanteil einer Koordinate als Position auf einem quadratischen Bild mit je 1000px Seitenlänge markiert werden, indem dort der Bildfehler platziert wird.

¹⁸

1.2.6 Odd-Pixel Verfahren

Das Wort "Odd" kommt aus dem Englischen und bedeutet seltsam oder merkwürdig. Es handelt sich also um merkwürdige Pixel. Damit ist zum Beispiel gemeint, wenn in einem fast komplett dunklen Bild irgendwo ein einzelnes Pixel Weiß, Grasgrün oder Rubinrot ist. So ein seltsames Pixel fällt einem sofort auf: Das passt da einfach nicht hinein.

Man kann sich das zu Nutze machen, indem man dem Odd-Pixel eine ganz bestimmte Position oder eine spezielle Farbe gibt. Ein Beispiel: Ein grünes Odd-Pixel bedeutet ja und ein rotes nein, oder die Position gibt die Koordinaten für einen Treffpunkt an.

Wenn man ein solches Odd-Pixel verstecken will, muss man aber beachten, dass die Wahrnehmungsschwelle eines Menschen nicht unterschritten wird. Sonst führt das unweigerlich dazu, dass niemand das Steganogramm entziffern kann. Jedoch ist es kaum beeinflussbar für wen das Pixel sichtbar ist, wessen Wahrnehmungsschwelle also ausreicht.

Eine große Herausforderung stellt hier vor allem die automatische Erkennung eines solchen Odd-Pixel dar. Weiß man, wonach man sucht, macht es die ganze Sache schon einfacher. Wird zum Beispiel ein komplett weißes Pixel gesucht, also mit den allen Werten der RGB-Skala auf 255, so muss das Programm nur eben dieses Pixel aufspüren. Es ist also das beste, ein Programm zur Verfügung zu stellen, welches diese Aufgabe zwar nicht von alleine lösen kann, jedoch einen Menschen bei der Arbeit unterstützt.

¹⁸TODO Finde ein Beispiel für ein solches Suchbildrätsel und löse es mit GC-Tools

Um das einfache Arbeiten zu ermöglichen, sind folgende Funktionen nützlich:

- Eine Zoomfunktion
- Markieren von Stellen mit einem Stift/Radierer Werkzeug
- Filtern nach bestimmten Farben. Ähnlich dem Füllwerkzeug bei Bildbearbeitungsprogrammen, nur als einstellbarer Filter welcher Bereiche verdunkelt, die nicht dem Kriterium entsprechen.
- Automatisches Markieren von gefundenen Stellen. Einzelne Pixel sind sehr schwer zu sehen, selbst wenn diese eingefärbt wurden. Eine Art bunte Umrandung ist dabei sehr hilfreich.
- Gruppieren von benachbarten Pixeln. Diese Pixelgruppen dann in einer auswählbaren Liste darstellen um Position, Farbe und weitere Eigenschaften abrufen zu können.

Mithilfe dieser Funktionen sollte es nicht mehr allzu schwer sein, Odd-Pixel zu finden.

Erwähnenswert ist hier eine Technik, mit deren Hilfe einzelne Pixel sowie Pixelgruppen umrandet werden können. Es handelt sich um einen Filter und zwei Pinsel Operationen welche nacheinander auf die Auswahlmaske angewendet werden.

Ein Pinsel wird auf das ganze Bild angewendet, indem für jeden Punkt auf dem Bild überprüft wird, ob eine bestimmte Bedingung eintrifft, und wenn ja wird ein Kreis in der angegebenen Größe mit einer bestimmten Farbe dort hin gezeichnet.

Ein Filter besteht aus einem sogenannten Kernel, das ist ein zweidimensionales Zahlenarray. Wie in 1.25 dargestellt, wird jeder Wert aus dem Kernel auf den Input mit einer festgelegten Operation verknüpft und die Ergebnisse summiert. Wenn nicht genauer definiert, ist die verwendete Operation die Multiplikation. Zahlreiche bekannte Filter und Effekte aus Bildbearbeitungsprogrammen können auf diese Weise implementiert werden.

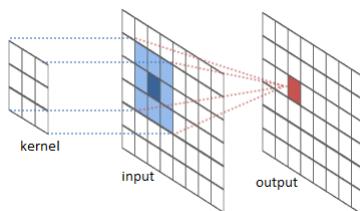


Figure 1.25: Kernel wird auf den Input angewendet um den Output zu erzeugen.

Einige Anwendungsbeispiele für Kernel sind:

- Laplacian Kantenerkennungs Kernel

$$K = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \quad K = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad K_H = \begin{bmatrix} 0 & 0 & 0 \\ -1 & 2 & -1 \\ 0 & 0 & 0 \end{bmatrix} \quad K_V = \begin{bmatrix} 0 & -1 & 0 \\ 0 & 2 & 0 \\ 0 & -1 & 0 \end{bmatrix}$$

- Gaußscher Weichzeichner Kernel

$$G = \frac{1}{16} \cdot \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

- Sobel Operator Kernel

$$S_H = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \quad S_V = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

- Emboss Kernel

$$E = \begin{bmatrix} -2 & -1 & 0 \\ -1 & 1 & 1 \\ 0 & 1 & 2 \end{bmatrix}$$

Java stellt in seiner Standardbibliothek die beiden Klassen `ConvolveOp` und `Kernel` zur Verfügung. Diese benötigen nur ein Double Array mit den Kernel-Werten und können dann auf Objekte der Klasse `BufferedImage` angewendet werden.

Um nun Details in einem Bild zu markieren, wird zuerst eine Pinselfunktion angewendet, welche jeden Punkt, der die Bedingung erfüllt, mit einem großen Pinsel übermalt. Auf das Resultat wird dann ein Kantenerkennungsfilter angewendet. Dann wird erneut ein Pinsel verwendet, jedoch ein kleinerer. Die Größe des ersten Pinsels gibt an, in welchem Abstand die Komponenten umrandet werden sollen. Die Größe des zweiten Pinsel gibt an wie dick der Rand sein soll.

[L-Convolution] In 1.26 sind die einzelnen Schritte des Verfahrens dargestellt.



Figure 1.26: Selbst kleine Details sind groß Umrandet und gut Sichtbar

[L-Convolution-Kernels]

```
2 import java.awt.Color;
3 import java.awt.Graphics2D;
4 import java.awt.image.BufferedImage;
5 import java.awt.image.ConvolveOp;
6 import java.awt.image.Kernel;
7 import java.io.File;
8 import java.io.IOException;
9 import javax.imageio.ImageIO;
10
11 /*
12  * To change this license header, choose License Headers in Project
13  * Properties.
14  * To change this template file, choose Tools | Templates
15  * and open the template in the editor.
16 */
17 /**
18  *
19  * @author Simon
20 */
21 public class ConvolveOpExample {
22
23     public static BufferedImage convert(BufferedImage input, int type) {
24         BufferedImage copy = new BufferedImage(input.getWidth(), input.
25             getHeight(), type);
26         Graphics2D g2d = copy.createGraphics();
27         g2d.drawImage(input, 0, 0, null);
28         g2d.dispose();
29         return copy;
30     }
31
32     public static void apply3x3(float[] kernel, BufferedImage input,
33         BufferedImage dest) throws IOException {
34         ConvolveOp op = new ConvolveOp(new Kernel(3, 3, kernel));
35         BufferedImage out1 = op.filter(input, dest);
36     }
37
38     public static void pinsel(int radius, Color color, BufferedImage source
39         , BufferedImage dest) {
40         Graphics2D ctx = dest.createGraphics();
41         ctx.setColor(color);
42         for (int i = 0; i < dest.getWidth(); i++) {
43             for (int j = 0; j < dest.getHeight(); j++) {
44                 if ((source.getRGB(i, j) & 0xFF) > 127) {
45                     ctx.fillOval(i - radius, j - radius, 2 * radius - 1, 2
46                         * radius - 1);
47                 }
48             }
49         }
50     }
51
52     public static void main(String[] args) throws Exception {
```

```
48     float[] laplacian8 = {1f, 1f, 1f, 1f, -8f, 1f, 1f, 1f, 1f};  
49  
50     BufferedImage mask = convert(ImageIO.read(new File("mask.png")),  
51         BufferedImage.TYPE_INT_RGB);  
52     BufferedImage mask1 = new BufferedImage(mask.getWidth(), mask.  
53         getHeight(), mask.getType());  
54     BufferedImage mask2 = new BufferedImage(mask.getWidth(), mask.  
55         getHeight(), mask.getType());  
56     BufferedImage mask3 = new BufferedImage(mask.getWidth(), mask.  
57         getHeight(), mask.getType());  
58     BufferedImage result = convert(mask, mask.getType());  
59  
60     pinsel(10, Color.white, mask, mask1);  
61     apply3x3(laplacian8, mask1, mask2);  
62     pinsel(5, Color.red, mask2, mask3);  
63     pinsel(5, Color.red, mask2, result);  
64  
65     ImageIO.write(mask1, "png", new File("mask1-pinsel10.png"));  
66     ImageIO.write(mask2, "png", new File("mask2-edges.png"));  
67     ImageIO.write(mask3, "png", new File("mask3-pinsel5.png"));  
68     ImageIO.write(result, "png", new File("mask4-result.png"));  
69 }  
70 }
```

sources/ConvolveOpCircleItExample.java

List of Figures

1.1	Buchstaben-Wort-Substitutionstabelle von Buch I der Polygraphia von Johannes Trithemius, Quelle: http://daten.digitale-sammlungen.de/lsb00026190/image_71	2
1.2	Hier wurde mit den Grashalmen links von der Brücke, auf der kleinen Mauer und entlang des Wasser Morsecode hinzugefügt	9
1.3	Wenn das grüne Band zum dekodieren verwendet wird ergibt sich die Nachricht "Renseignements arrivent", auf Deutsch: "Informationen angekommen"	10
1.4	In diesem Beispiel wurde Morsecode in Form der Größe der Vögel, der Zaunstangen und der Blumen kodiert. Dekodiert ergeben sich die Namen der drei Automarken VW, Buick und Volvo	11
1.5	Hier deutlich zu erkennen die verwendeten ASCII Zeichen.	13
1.6	In diesen zufälligen Daten kann kein Muster erkannt werden.	14
1.7	Wenn der Inhalt wiederhergestellt werden soll, muss nur von unten nach oben alles wieder rückgängig gemacht werden.	15
1.8	In diesem Vergleich ist gut zu sehen, dass nach Anwendung von diesem Verfahren der Datenstrom nicht mehr von zufälligen Werten unterscheidbar ist.	15
1.9	Vergleich zwischen AES verschlüsselten Daten und zufälligen Werten.	17
1.10	LSB Verfahren Erklärung	20
1.11	Beispiel für Kodierung von einem Zeichen pro Pixel.	21

1.12	In diesem Bild wurde ein ganzes Buch versteckt, und es ist trotzdem nicht zu erkennen.	22
1.13	Auf der linken Seite befinden sich eingebettete Daten, auf der rechten Seite nicht. Innerhalb des Kreises wurde der Effekt verstrkkt dargestellt.	25
1.14	Die Daten werden im verlustfreien Teil des JPEG Algorithmus hinzugefgt.	26
1.15	27
1.16	fig:L: Barcode Histogram Example	27
1.17	Beispiel fr ein Stereogramm	31
1.18	Beispiel fr ein Text Stereogramm	32
1.19	Beispiel fr ein Objekt-Array Stereogramm	32
1.20	Die Pixel werden je nach Depth Map vom linken Strich + Offset zu dem rechten Kopiert.	34
1.21	Kaum zu erkennen sind die versteckten Formen.	35
1.22	Bei dem Stereogramm kann ein Spike in der Datenreihe erkannt werden	38
1.23	Die verbesserte Variante erzielt deutlichere Ergebnisse	39
1.24	Ein Wert von mehr als 25% heit, dass es sich Wahrscheinlich um ein Stereogramm handelt.	40
1.25	Kernel wird auf den Input angewendet um den Output zu erzeugen.	42
1.26	Selbst kleine Details sind gro Umrandet und gut Sichtbar	43

List of Tables

1.1	Vergleich zwischen Steganographie und Kryptographie, Quelle: [L: Stego VS Crypto]	3
1.2	Kodierungstabelle der Payload	19

Listings

1.1	AESVerfahren.java	16
1.2	LSBVerfahren.java	22
1.3	HistogramViewer.java	28
1.4	StereogrammGenerator.java	35
1.5	Einfacher Stereogramm Solver	37
1.6	Steigung	39

Bibliography

[L: StegoGeschichte] <https://igw.tuwien.ac.at/designlehren/steganographie.pdf>

Eine kurze Geschichte der Steganographie

Peter Purgathofer

12.11.2016

[L: Stego VS Crypto] <http://digilib.happy-security.de/files/Steganographie.pdf>

Kryptographie und Informationstheorie: Steganographie

Prof. Dr. Richard Eier, Institut für Computertechnik TU Wien

Michaela Schuster

20.11.2016

[L: StegoVersteck] <http://www.tecchannel.de/a/vertrauliche-daten-perfekt-ver>

2024281

Vertrauliche Daten perfekt versteckt, Artikel vom 30.11.2009

13.12.2016

[L: USA-Crypto Export] https://en.wikipedia.org/wiki/Export_of_cryptography_from_the_United_States

[L-Confidentiality without Encryption] <http://people.csail.mit.edu/rivest/chaffing-980701.txt>

Chaffing and Winnowing: Confidentiality without Encryption

Ronald L. Rivest

MIT Lab for Computer Science

March 18, 1998 (rev. July 1, 1998)

[L-Convolution] <http://colah.github.io/posts/2014-07-Understanding-Convolutional-Networks/>

Understanding Convolutions

Christopher Olah

30.03.2017

[L-Convolution-Kernels] <http://matlabtricks.com/post-5/3x3-convolution-kernels-with-online-demo>

30.03.2017

[L-Stego] Jessica Fridrich: Steganography in Digital Media: Principles, Algorithms, and Applications

Cambridge University Press (12 Nov 2009)

ISBN: 978-052119019-0

30.03.2017

[L-Jpeg-Algorithm] <http://www-i6.informatik.rwth-aachen.de/web/Misc/Coding/365/li/material/notes/Chap4/Chap4.2/Chap4.2.html> 30.03.2017

[L-JSteg] <http://cise.ufl.edu/~makumar/proposalppt.pdf>
Steganography and Steganalysis of JPEG Images
Ph.D. Proposal
Mahendra Kumar
CISE Department 30.03.2017

[L-Fehlererkennung] <http://web.mit.edu/course/16/16.682/www/leca.pdf> 30.03.2017

[L-Peterson] W. W. Peterson: Cyclic Codes for Error Detection.
In: Proceedings of the IRE, Vol. 49, No. 1, 1961, S. 228-235

[L-AES-Encryption-Standard] <http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf> 30.03.2017

[L-Kongruenzgenerator] Donald E. Knuth: The Art of Computer Programming.
Volume 2 Seminumerical Algorithms. 3. Auflage. Addison-Wesley
1997, S. 10-26
ISBN 0-201-89684-2

[Kopka1] Helmut Kopka: *Latex Band 1, Einführung*
Addison-Wesley, 2000
ISBN: 3-8273-7038-8

[Demmig 1] Demmig, Thomas:
jetzt lerne ich Latex 2
Markt+Technik, 2004
ISBN 3-8272-6517-7

[Web 1] <http://www.meta-x.de/faq/LaTeX-Einfuehrung.html>
Latex-Einführung
28.September 2012

[JavaDoc05] http://docs.oracle.com/cd/E12839_01/core.1111/e10043/introjps.htm
Oracle Security Guide über das Java Sicherheits Model
13.11.2014