
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 27. März 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

Inhaltsverzeichnis

Vorwort	i
Diplandenvorstellung	ii
Danksagungen	iii
Zusammenfassung	iv
Abstract	v
Inhaltsverzeichnis	vi
1 Steganographie Übersicht	1
1.1 Grundlagen Steganographie	1
1.2 Abgrenzung zur Kryptographie	2
1.3 Einsatzgebiete	3
1.4 Steganographie als "Wicked Problem"	4
2 Klassische Verfahren der Steganographie	6
2.1 Spreu und Weizen Verfahren	6
2.1.1 Sender-Seite	7
2.1.2 Die Spreu	7
2.1.3 Empfänger-Seite	8

2.1.4	Sicherheitsaspekte	9
3	Moderne Steganographie	10
3.1	Allgemein	10
3.1.1	Whitening	10
	Pseudozufallsgenerator mit bitweiser Verknüpfung	12
	AES-256 Verschlüsselung mit Key	13
3.1.2	Fehlerkorrekturverfahren	15
	Redundante Datenspeicherung	15
	Paritätsbit	15
	Zyklische Redundanzprüfung (z.B. CRC-32)	16
3.2	Verstecken von Dateien	16
3.2.1	Alternativer Datenstream	16
	Erstellen von ADS unter Windows	17
3.2.2	Dateien Verketten	18
3.3	Modifikation von Bilddateien	19
3.3.1	Least Significant Bit Verfahren	20
	Varianten des Least Significant Bit Verfahren	26
3.3.2	Barcode Verfahren	26
Anhang		31
	Tabellenverzeichnis	33
	Verzeichnis der Listings	34
	Literaturverzeichnis	35

Kapitel 1

Steganographie Übersicht

1.1 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 ersetzt werden. Diese Wortfolge wird dann mit weiteren nicht in der Tabelle vorkommenden Wörtern ergänzt um vollständige, grammatisch korrekte Sätze bilden zu

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

können. Eine solche Tabelle findet man zum Beispiel in dem Buch 1 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	Cpfex	D	pifsumus
E	Dominus	E	magnus
F	Dominator	F	excellens
G	Consolator	G	maximus
H	Arbiter	H	optimus
I	Iudex	I	sapientissimus
K	Illuminator	K	inuifibilis
L	Illustrator	L	immortalis
M	Reclor	M	eternus
N	Rex	N	sempiternus
O	Imperator	O	gloriosus
P	Gubernator	P	fortissimus
Q	Factor	Q	sanctissimus
R	Fabricator	R	incomprehensibilis
S	Conferuator	S	omnipotens
T	Redemptor	T	pacificus
V	Auctor	V	mifericors
X	Princeps	X	mifcrordissimus
Y	Pafbor	Y	cunclipotens
Z	Moderator	Z	magnificus
W	Saluator	W	excellentissimus

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

1.2 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 aber bei genauerer Betrachtung zwei komplett unterschiedliche Verfahren. Wichtig ist

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

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 versteckte 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	krypte = 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

Tabelle 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 unterschieden werden. Wenn man dieses "Rauschen" dann mit Hilfe von Steganographie in ein unauffälliges Trägermedium einbettet, ist es selbst mit elektronischer 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.

1.3 Einsatzgebiete

Steganographie schützt Daten nicht vor Missbrauch, warum sollte man sie dann überhaupt verwenden, wenn Kryptographie viel sicherer ist? In der westlichen Welt ist Verschlüsselung durch das Internet so weit verbreitet, dass es als selbstverständlich er-

scheint, 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]

1.4 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.

Kapitel 2

Klassische Verfahren der Steganographie

Mit klassischen Steganographie Verfahren sind Techniken gemeint, welche größten Teils oder gänzlich ohne Computersysteme funktionieren und somit nicht zwingend auf das "digitale Zeitalter" angewiesen sind. In dem nachfolgenden Kapitel wird das "Spreu und Weizen" Verfahren vorgestellt. Dies soll mit einem praxisnahen Beispiel einen Überblick darüber verschaffen, was genau der Grundgedanke von Steganographie ist.

2.1 Spreu und Weizen Verfahren

Spreu und Weizen Verfahren¹ stellen eine Mischform zwischen Steganographie und Kryptographieverfahren 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 Verfahren ist es, die zu versteckenden Daten in einem Haufen nicht relevanter Information zu verstecken, wie die Nadel im Heuhaufen.

² 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

¹eng.: Chaffing and Winnowing

²TODO Nachfolgenden Absatz und Zitat ans Ende der Section schieben?

abstreiten und argumentieren, sie wollen nur die Authentizität ihrer Nachrichten sicherstellen. Dadurch 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

2.1.1 Sender-Seite

Der Sender muss seine Nachricht in Pakete unterteilen. Ihre Größe kann beliebig gewählt werden. Er muss die Pakete außerdem in irgendeiner Weise durchnummrieren, um sie auf der Empfängerseite wieder in der richtigen Reihenfolge zusammenzusetzen 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.

2.1.2 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 verwendeten Methode führt sehr leicht zu zufällig gültigen MAC's, ist also in einem realen Anwendungsfall nicht ausreichend. Hier könnten zum Beispiel Codes aus der HMAC³ Familie zur Anwendung kommen, wie der recht bekannte HMAC-SHA256.

Spreu Nachrichten könnten etwa so aufgebaut sein:

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.

2.1.3 Empfänger-Seite

Der Empfänger sortiert nun die Pakete nach der ID und überprüft deren MAC's. Für ihn ist nun gut ersichtlich, welche der 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.

³Hash-based Message Authentication Code

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
3	um 10 Uhr	7881
3	um 18 Uhr	2816
4	am Hauptbahnhof	8370
4	am Westbahnhof	9821
4	am Bahnhof Meidling	1155

⁴

2.1.4 Sicherheitsaspekte

In dem obigen Beispiel gibt es $3 * 2 * 2 * 3 = 36$ verschiedene Lösungswege eine Nachricht zusammen zu setzen. 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.

⁴TODO Statt einer langen unübersichtlichen Tabelle ein Diagramm mit den möglichen Lösungswegen welches man selber durchgehen kann.

Kapitel 3

Moderne Steganographie

Mit moderner Steganographie sind Verfahren gemeint, welche nur mit Hilfsmittel der elektronischen Datenverarbeitung funktionieren. Sie verlassen sich meist darauf das 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 digital Fotografie.

In dem nachfolgendem Kapitel werden einige dieser Verfahren erklärt und etwaige Fehler und Schwierigkeiten die damit verbunden sind aufgezeigt. Außerdem werden einige Implementierungen gezeigt, verglichen und auf die Verwendbarkeit im Alltag getestet.

3.1 Allgemein

In dem folgenden Abschnitt wird der Begriff Whitening erklärt und auf die Möglichkeit von Fehlerkorrekturmaßnahmen bei der Übertragung von Daten eingegangen.

3.1.1 Whitening

Als Whitening werden Prozesse und Techniken bezeichnet mit dem Ziel, 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 einzubettendem Medium nicht so stark auffallen wie zuvor.

Whitening Verfahren wandeln auf deterministische Weise Datenströme, welche in etwa wie 3.1 aussehen können, um damit sie dem Profil von 3.4 ähneln. Dies muss auf eine Art und Weise geschehen das 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 das sich die angewandten Operationen auch wieder rückgängig machen lassen. Es werden auch oft Methoden aus der Kryptologie verwendet, welche nicht nur herausragende Ergebnisse in Hinsicht auf Zufälligkeit erzielen, sondern zugleich auch die Daten mit Hilfe eines Schlüssel schützen.

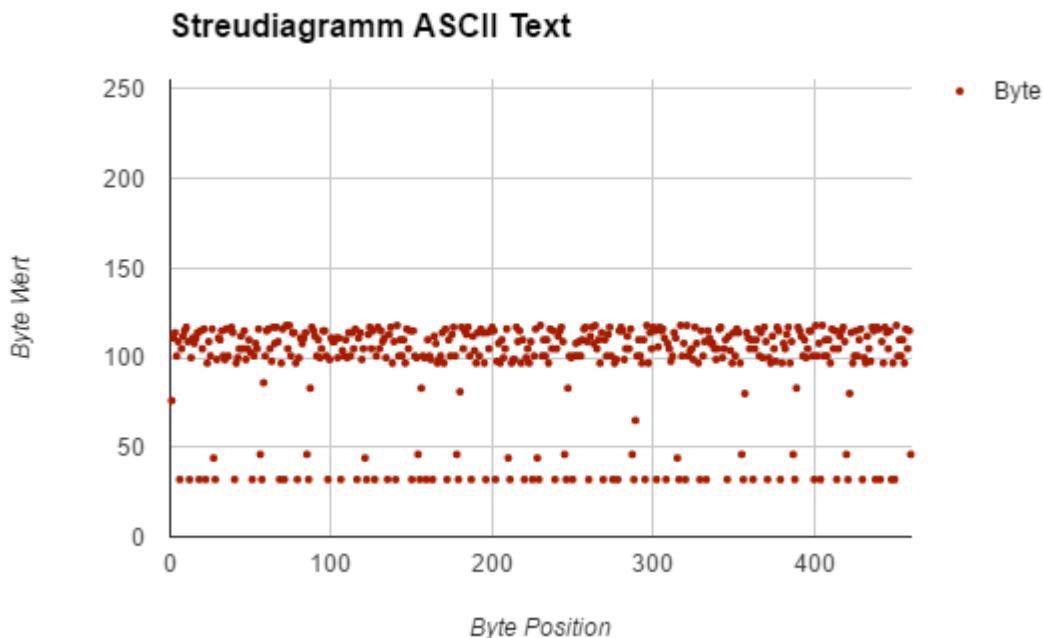


Abbildung 3.1: Hier deutlich zu erkennen die verwendeten ASCII Zeichen.

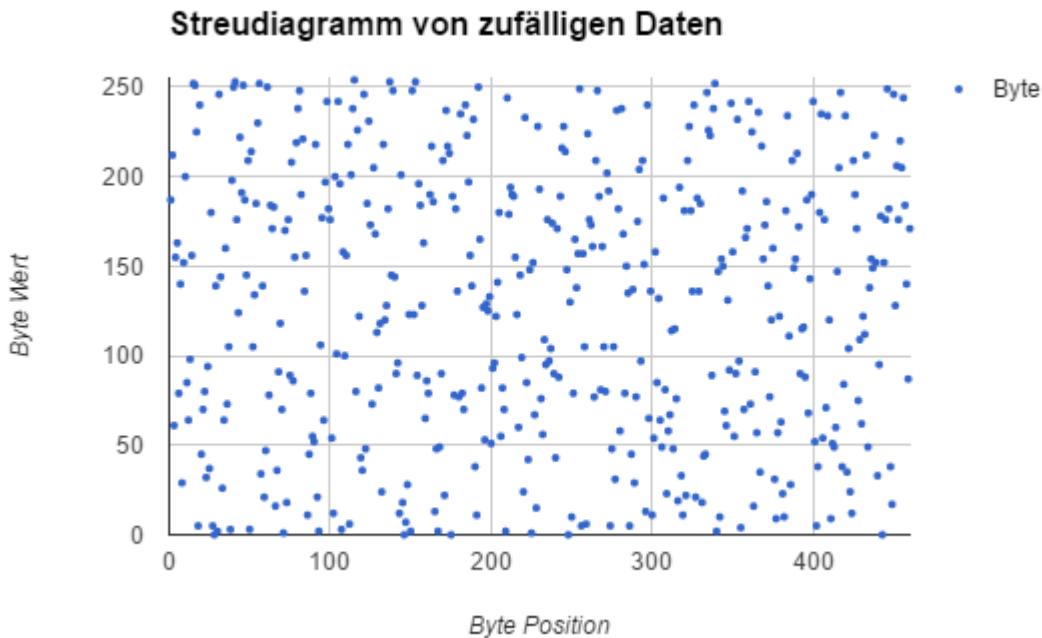


Abbildung 3.2: In diesen zufälligen Daten kann kein Muster erkannt werden.

1

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 das 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" innerhalb von Minuten aus. Genau so ist es mit schwachen Whitening Verfahren welche auf Obskurität setzen.

¹TODO 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 Pseudozufallszahlengeneratoren Datenstrom verknüpft. Zum Einsatz kommt hier ein Kongruenzgenerator wie er auch in der Standard Java Klasse `java.lang.Random` verwendet wird.

	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

Abbildung 3.3: Wenn der Inhalt wieder hergestellt werden soll muss nur von unten nach oben alles wieder Rückgängig gemacht werden.

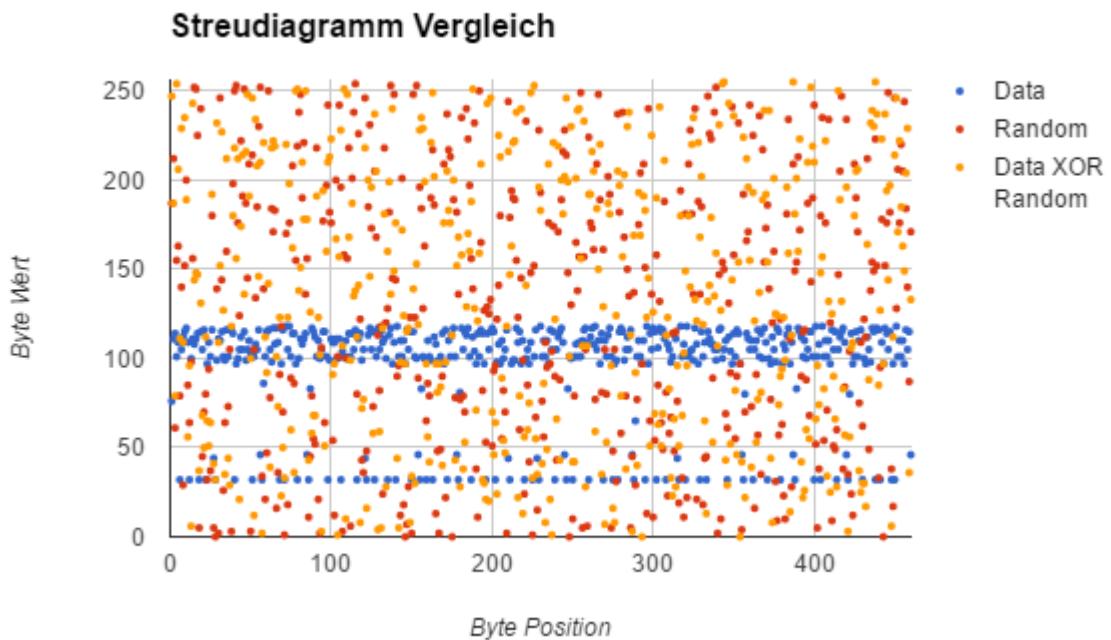


Abbildung 3.4: In diesem Vergleich ist gut zu sehen das nach Anwendung von diesem Verfahren der Datenstrom nicht mehr von zufälligen Werten unterscheidbar ist.

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 nach-

folgenden 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 3.1: AESVerfahren.java

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

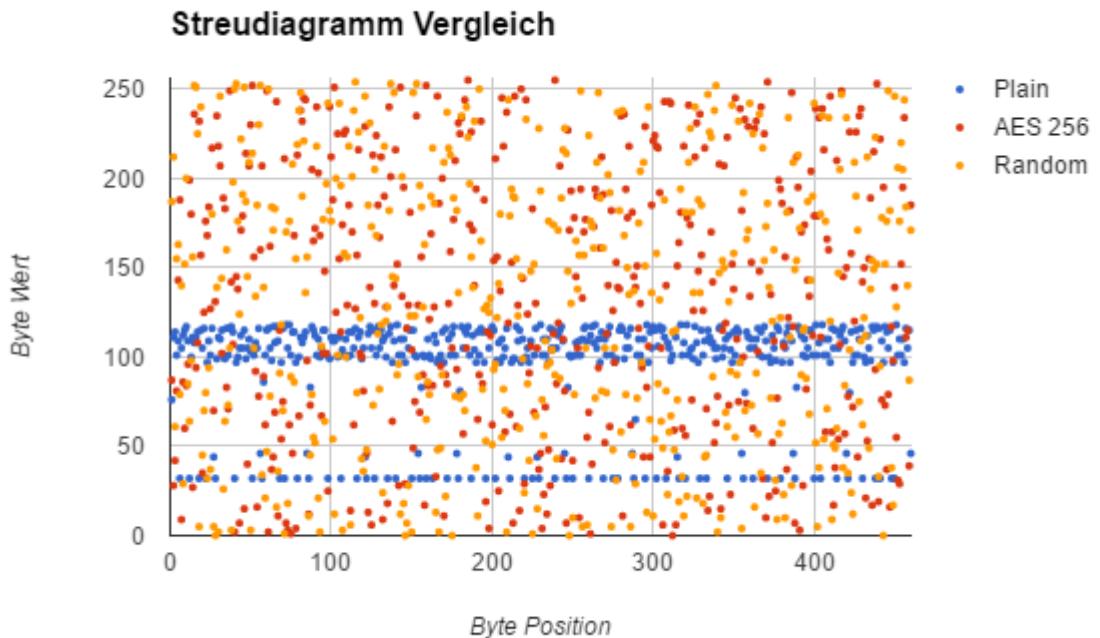


Abbildung 3.5: Vergleich zwischen AES verschlüsselten Daten und zufälligen Werten.

3.1.2 Fehlerkorrekturverfahren

Moderne Verfahren sind sehr anfällig auf verlustbehaftete Komprimierung. Dadurch das sie meistens darauf basieren einzelne Bits zu verändern, passiert es auch sehr schnell das 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. Des weiteren werden mehrere Verfahren nach Robustheit verglichen.

Redundante Datenspeicherung

Eine Möglichkeit ist es die Daten mehrfach abzuspeichern. Das bietet Schutz vor Manipulation von einzelnen Teilen des Bild. Zum Beispiel das Wegschneiden von Rändern, das Entfernen von Gesichtern, Wasserzeichen oder einfach Bild Über- und Unterschriften welche erst später zu dem Bild hinzugefügt werden.

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

Auch wenn das Trägermaterial vergrößert/verkleinert wird hilft redundante Speicherung meist nicht. Bei Skalierung von Bildern werden immer Filtering Methoden eingesetzt werden. Diese kombinieren oft mehrere Pixel zu einem einzelnen und wenden dabei zahlreiche Operationen an. Dadurch geht fast immer sämtliche steganographisch versteckte Information 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. Eine Weiterentwicklung dieses Verfahren stellt der unten vorgestellte Hamming-Code dar.

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. 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.

3.2 Verstecken von Dateien

² Die Dateisysteme der Betriebssysteme bieten zahlreiche Möglichkeiten um seine Daten für Dritte schwer auffindbar zu machen. In der folgenden Tabelle werden einige Verfahren verglichen und anschließend genauer erläutert. Es ist jedoch zu bemerken, dass es sich hier um sehr einfache Verfahren handelt die höchstens Schutz gegen Computeranfänger bietet und für jeden Forensiker keine Herausforderung darstellen.

Beschreibung	Anwendbarkeit	Human Attack	Computer Attack
Alternativer Datenstream	ext. Programm wird benötigt	mittel	einfach
Dateien Verketten	überall unterstützt	mittel	einfach
Versteckte Datei	überall unterstützt	einfach	einfach
Dateiendung verändern	nur auf Windows und OSX ³	einfach	sehr einfach

3.2.1 Alternativer Datenstream

Die alternativen Dateistreams wurden von Windows bei ihrem Dateisystem NTFS deshalb eingefügt, da sie auch das Dateisystem von Apple HFS unterstützen wollten. Dort werden sogenannte Resource Forks verwendet um weitere Informationen wie Icons für Dateien zu speichern. Diese Dateistreams sind mit dem normalen Datei Explorer von Windows nicht direkt sichtbar, können aber erahnt werden wenn man die in den Eigenschaften angezeigte Größe mit der tatsächlichen Größe vergleicht. Vor allem für

² TODO Ist das wirklich Steganographie?

große versteckte Dateien wie Bilder oder Videos können hier leicht große Unterschiede erkannt werden.

Erstellen von ADS unter Windows

Alternative Dateistreams können nicht einfach mit dem Windows Explorer erstellt werden. Die einfachste Möglichkeit unter Windows alternative Dateistreams zu erstellen ist über die Kommandozeile. Angesprochen werden die einzelnen Streams über den Dateinamen der sichtbaren Trägerdatei und einem mit Doppelpunkt getrennten Namen für den Stream. Über diesen Bezeichner kann nun jedes Programm auf den alternativen Stream zugreifen.

Als Trägerdatei kann jede beliebige Datei verwendet werden. Je größer die gewählte Trägerdatei ist, desto unwahrscheinlicher ist es das jemanden der zusätzliche Datenstream auffällt. In dem folgenden Beispiel wird die Datei "unscheinbare_textdatei.txt" über die Windows-Kommandozeile erstellt und mit einem unbedeutenden Text befüllt.

```
1 echo "Hallo Welt, hier ist nichts." > unscheinbare_textdatei.txt
```

Nun kann man mit dem gleichen Befehl und einem angepassten Dateinamen auf gleiche Weise alternative Datenstreams befüllen.

```
1 echo "Geheim" > unscheinbare_textdatei.txt:geheimer_stream.txt
```

Sowohl die Trägerdatei als auch der alternative Stream können mit jedem beliebigen Programm jederzeit bearbeitet werden. Zu beachten ist nur wieder das der Explorer und alle "Datei öffnen" Dialoge die geheimen Streams nicht anzeigt. Dazu muss man wieder auf die Kommandozeile zurückgreifen. Zum Beispiel kann man den Stream mit dem Programm Notepad von Windows öffnen.

```
1 notepad unscheinbare_textdatei.txt:geheimer_stream.txt
```

Wie man sehen kann können diese alternativen Streams von jedem Programm ohne viel Mehraufwand geöffnet werden. Sie stellen also keinen ausreichenden Schutz vor ungewollten Zugriff auf den Inhalt da.

Um sich eine Übersicht über alle Dateien und ihre Streams machen zu können gibt es zahlreiche fertige Tools, welche ohne viel Aufwand das gesamte Dateisystem nach solchen Strukturen durchsuchen. Bei einem konkreten Verdacht kann jedoch auch das

bei Windows mitgelieferte Programm *dir* verwendet werden. Man navigiert über die Kommandozeile in den jeweiligen Ordner und dort ruft man den *dir* Befehl mit dem Parameter */r* auf.

```
1 C:\stuff>dir /r
2 Datenträger in Laufwerk C: ist Windows
3 Volumeseriennummer: F8D1-CAC1
4
5 Verzeichnis von C:\stuff
6
7 16.02.2017 17:37    <DIR>      .
8 16.02.2017 17:37    <DIR>      ..
9 16.02.2017 17:44          28 unscheinbare_textdatei.txt
10                         11 unscheinbare_textdatei.txt:geheimer_stream
                           .txt:$DATA
```

3.2.2 Dateien Verketten

Viele Dateiformate besitzen eine interne Angabe über wie lang die Datei sein sollte. Dies kann ausgenutzt werden indem man eine Payload am Ende der Datei anfügt. Programme welche die Trägerdatei lesen wollen ignorieren den zusätzlich angehängten Teil. Die versteckte Datei kann schnell und einfach wieder aus der Trägerdatei extrahiert werden.

Zu beachten ist hier vor allem das man eine geeignete Trägerdatei wählt. Denn bei einer Datei welche sich oft in der Länge ändert kann es passieren das sein Versteckter angehänger Teil einfach überschrieben wird. Außerdem sollte darauf geachtet werden dass die Datei eine fix definierte Länge besitzt. Eine Beispiel für geeignete Dateiformate sind Bild- und Videoformate, sofern man sie nicht noch weiter bearbeiten will. Diese besitzen eine im Header der Datei angegebene Länge und haben auch eine relativ große Größe, wodurch kleinere angehängte Dateien im Speicherplatzverbrauch nicht so stark auffallen.

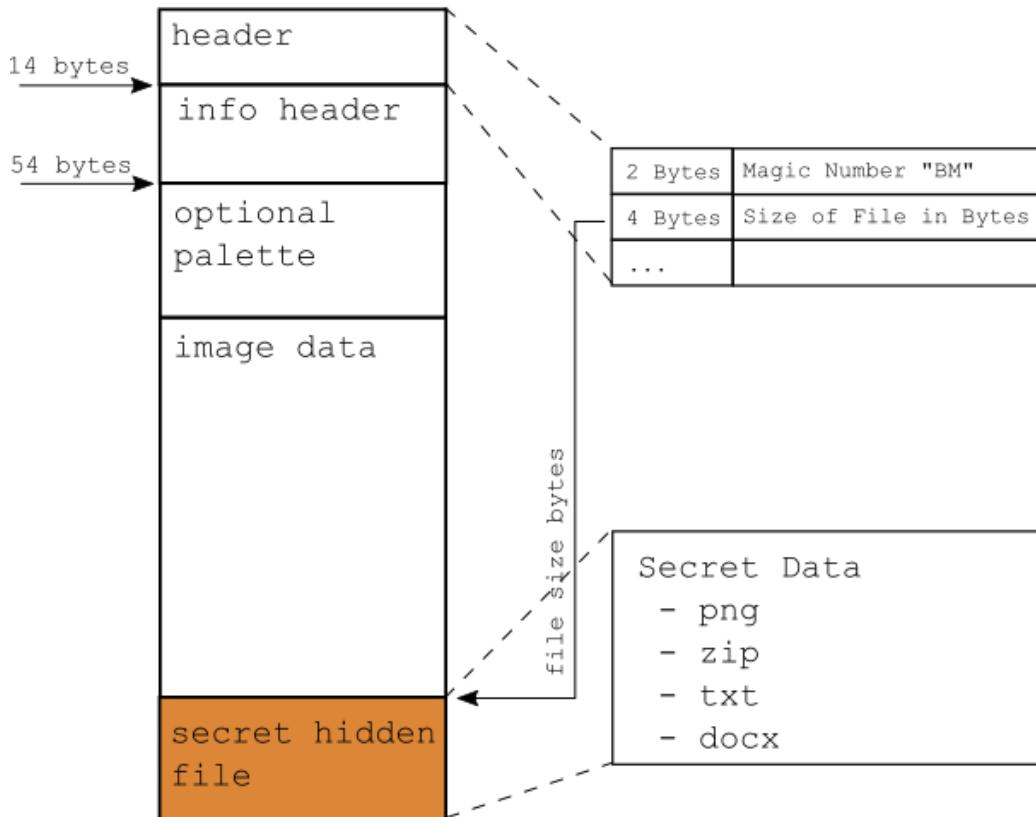


Abbildung 3.6: Beispiel für das Verstecken einer Datei am Ende einer Bitmap

3.3 Modifikation von Bilddateien

4

Bilder auf einem Computer sind nichts anderes als Matrizen welche Farbeinformationen für die einzelnen Pixel des Bildes beinhalten. Es gibt zahlreiche Verfahren in diesen 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.

Aus diesem Grund wird in den Beispielen die Kodierung aus 3.1 für die Payload verwendet.

⁴TODO Die ganze Sache gibt es auch in der klassischen wo es darum geht z.B. in der Länge der Gartenzaubretter Informationen zu verstecken

Tabelle 3.1: 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	"

3.3.1 Least Significant Bit Verfahren

Das Least Significant Bit (LSB) Verfahren nützt die Tatsache aus, dass der Farbraum einer modernen Bilddatei sehr groß ist⁵. Dadurch ist es für das menschliche Auge sehr schwierig, eng beieinander liegende 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 Kanal 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 groÙe Rolle spielt wie viel Trägermaterial zur Verfügung steht und wie viele Daten darin eingebettet werden. Je nachdem kann man die einzelnen Bits weiter verteilen oder muss sogar mehrere Bits auf ein Pixel. 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.

⁶

⁵Bei den meisten Formaten 24-Bit, wodurch 16,777,216 Farben dargestellt werden können

⁶TODO Tabelle mit Vergleich wie viele Bit Farbe / wie viele Bit Daten und wie viel Prozent das ausmacht + Bildbeispiele für den Farbunterschied den das ausmacht

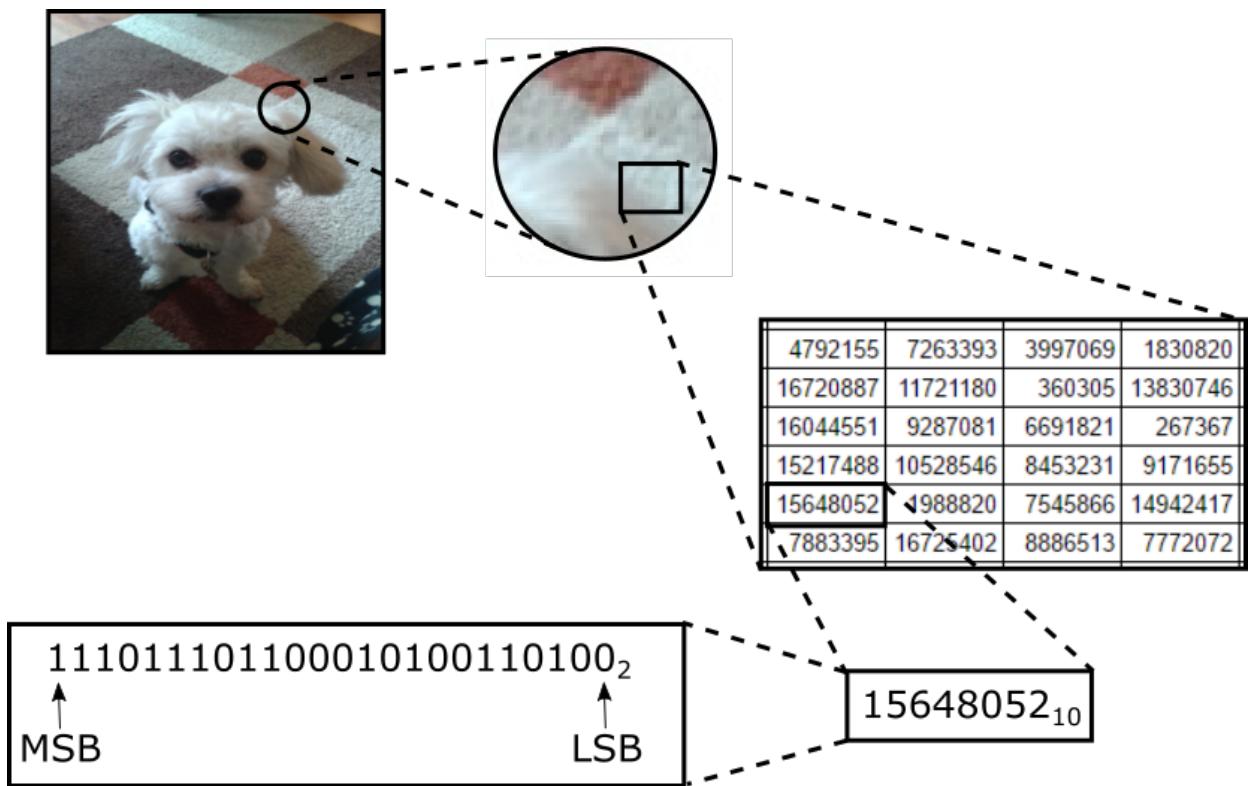


Abbildung 3.7: LSB Verfahren Erklärung

In dem Beispiel aus 3.8 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 das die in 3.1 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.

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

Abbildung 3.8: Beispiel für 1 Zeichen pro Pixel.

In dem in 3.9 gezeigtem Beispiel wurde das Buch "Schöne neue Welt" in ein Bild der Erde integriert. Verwendet wurde das in 3.2 gezeigte Java-Programm. Die kurze Länge des Programm zeigt das 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, ohne weitere Tools.

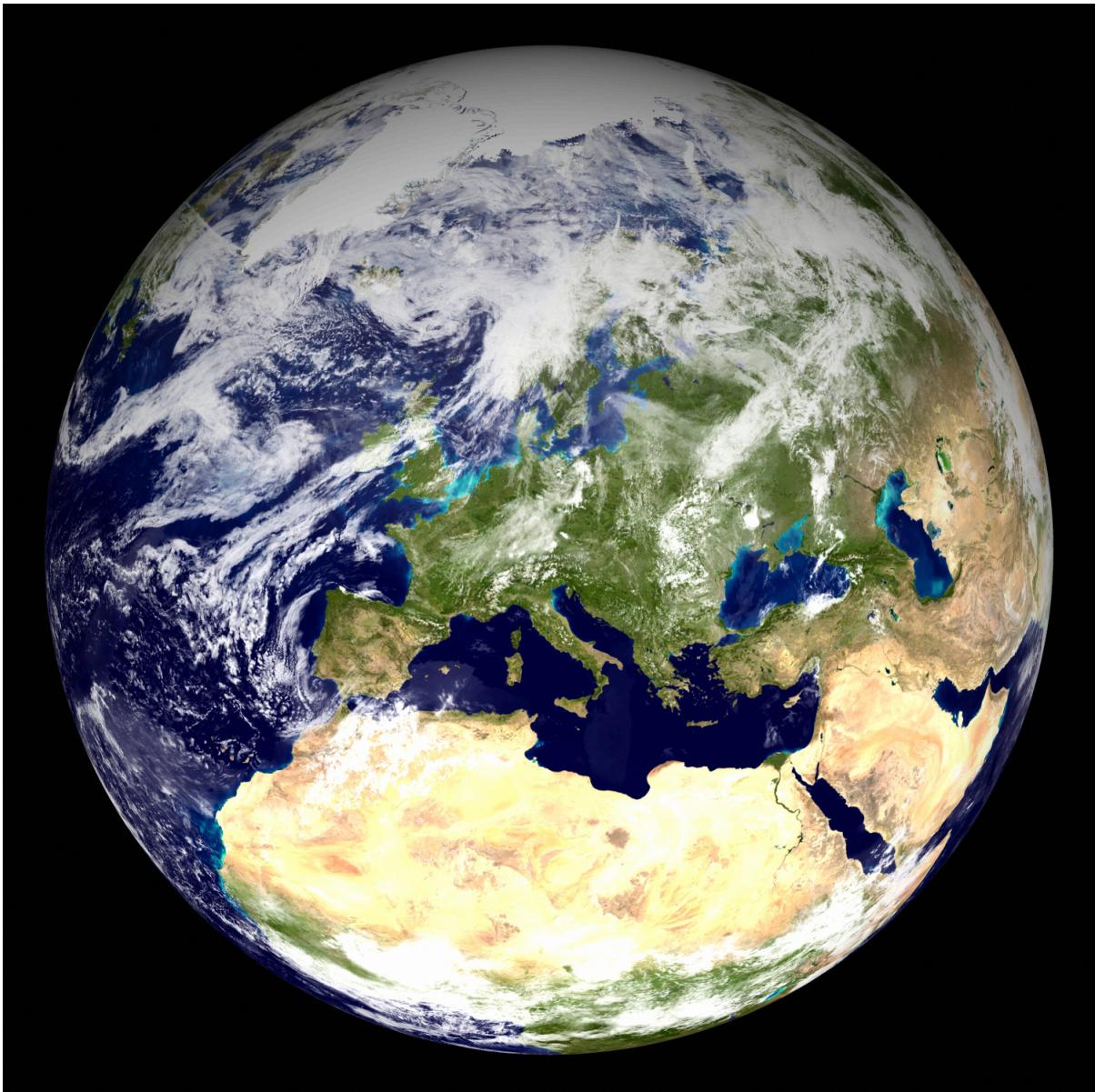


Abbildung 3.9: In diesem Bild wurde ein ganzes Buch versteckt, und es ist trotzdem nicht zu erkennen.

```
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;
```

```

48         result[bitOffset / 8] |= (((pixel >> 16) ^ random.nextInt
49             (4)) & 0x3) << ((bitOffset) % 8);
50         bitOffset += 2;
51         result[bitOffset / 8] |= (((pixel >> 8) ^ random.nextInt(4)
52             ) & 0x3) << ((bitOffset) % 8);
53         bitOffset += 2;
54         result[bitOffset / 8] |= (((pixel >> 0) ^ random.nextInt(4)
55             ) & 0x3) << ((bitOffset) % 8);
56     }
57 }
58
59     public static void extractDataFromImage(String args[]) throws
60         IOException {
61         BufferedImage image = ImageIO.read(new File("sch&ne-neue-earth.png
62             ")); //load the original image
63         byte data[] = decode(image); //extract payload from image
64         Files.write(new File("sch&ne-neue-welt-extract.pdf").toPath(),
65             data, WRITE, CREATE); //write extracted payload to hdd to test if
66             it worked
67     }
68
69     public static void encodeDataIntoImage(String args[]) throws
70         IOException {
71         BufferedImage image = ImageIO.read(new File("earth.png")); //load
72             the original image
73         byte data[] = Files.readAllBytes(new File("sch&ne-neue-welt.pdf").
74             toPath()); //load the original payload file
75         encode(image, data); //add payload to image
76         data = decode(image); //extract payload from image
77         ImageIO.write(image, "png", new File("sch&ne-neue-earth.png")); ////
78             write image with payload to hdd
79         Files.write(new File("sch&ne-neue-welt-extract.pdf").toPath(),
80             data, WRITE, CREATE); //write extracted payload to hdd to test if
81             it worked
82     }
83
84     public static void main(String[] args) throws IOException {
85         encodeDataIntoImage(args);
86         extractDataFromImage(args);
87     }
88 }
```

Listing 3.2: LSBVerfahren.java

Wenn die einzubettenden Daten nicht den gesamten Verfügbaren "Speicherplatz" innerhalb des Steganogramm verwenden, sollte unbedingt ein Padding aus zufälligen Daten hinzugefügt werden. Sonst können sich deutliche Unterschiede in dem Aussehen des Bild ergeben wenn es große, einfarbige Flächen besitzt, wie in 3.10 deutlich

zu erkennen ist.



Abbildung 3.10: Auf der linken Seite befinden sich eingebettete Daten, auf der rechten Seite nicht.

Die resultierende Datei ist deutlich größer als die Originale. Das liegt daran dass durch das einbetten der Daten große einfarbige Flächen minimal verändert wurden, wodurch sie sich nicht mehr so gut komprimieren lassen wie zuvor.

Varianten des Least Significant Bit Verfahren

3.3.2 Barcode Verfahren

Das Barcodeverfahren ist ein selbst erfundenes 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 3.11 wurde das Bild zuerst in ein 8-BGraustufenbild umgewandelt, die entstehenden 256 verschiedenen Werte gezählt und grafisch Dargestellt.

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

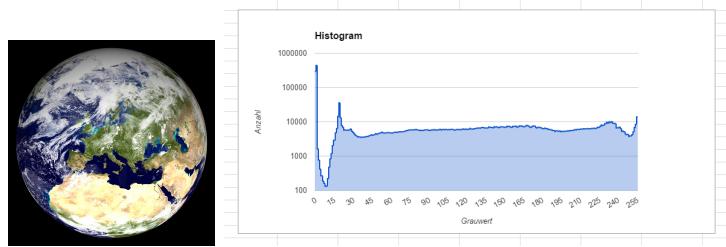


Abbildung 3.11

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

Ein Beispiel für einen solchen Generator findet sich in 3.3. Dieses Programm generiert dann für Barcodescanner verwendbare Histogramme, wie in 3.12 gezeigt wird. In diesem Beispiel wurde das gängige Barcodeformat CODE_128 verwendet und der Text "Simon Lehner-D." damit kodiert.

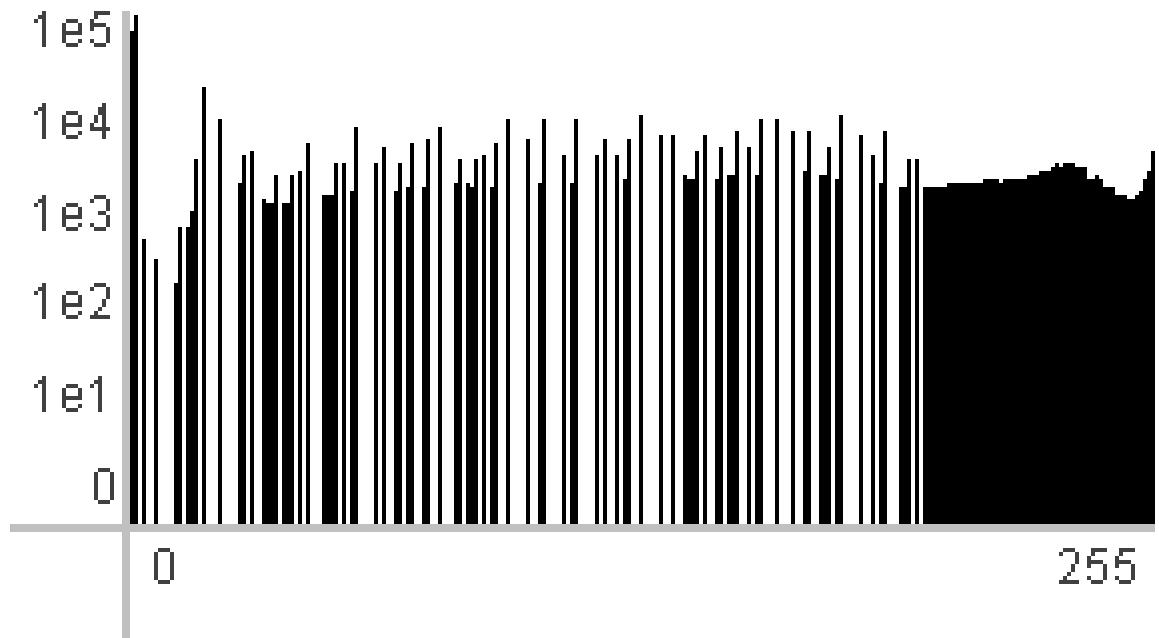


Abbildung 3.12: fig:L: Barcode Histogram Example

⁷⁷TODO 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 3.3: HistogramViewer.java

Abbildungsverzeichnis

1.1	Buchstaben-Wort-Substitutionstabelle von Buch I der Polygraphia von Johannes Trithemius, Quelle: http://daten.digitale-sammlungen.de/bsb00026190/image_71	2
3.1	Hier deutlich zu erkennen die verwendeten ASCII Zeichen.	11
3.2	In diesen zufälligen Daten kann kein Muster erkannt werden.	12
3.3	Wenn der Inhalt wieder hergestellt werden soll muss nur von unten nach oben alles wieder Rückgängig gemacht werden.	13
3.4	In diesem Vergleich ist gut zu sehen das nach Anwendung von diesem Verfahren der Datenstrom nicht mehr von zufälligen Werten unterscheidbar ist.	13
3.5	Vergleich zwischen AES verschlüsselten Daten und zufälligen Werten.	14
3.6	Beispiel für das Verstecken einer Datei am Ende einer Bitmap . . .	19
3.7	LSB Entwurf, muss ich noch verbessern.	21
3.8	Beispiel für 1 Zeichen pro Pixel.	22
3.9	In diesem Bild wurde ein ganzes Buch versteckt, und es ist trotzdem nicht zu erkennen.	23
3.10	Auf der linken Seite befinden sich eingebettete Daten, auf der rechten Seite nicht.	26
3.11	27

3.12 fig:L: Barcode Histogram Example 27

Tabellenverzeichnis

1.1 Vergleich zwischen Steganographie und Kryptographie, Quelle: [L: Stego VS Crypto]	3
3.1 Kodierungstabelle der Payload	20

Listings

3.1	AESVerfahren.java	14
3.2	LSBVerfahren.java	23
3.3	HistogramViewer.java	28

Literaturverzeichnis

[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 Schuster⁸

20.11.2016

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

2024281

Vertrauliche Daten perfekt versteckt, Artikel vom 30.11.2009

pte pte

13.12.2016

- [1] https://en.wikipedia.org/wiki/Export_of_cryptography_from_the_United_States

⁸TODO Ist Schuster der Autor oder Dr. Richard Eier? Außerdem - Ist das so richtig als Quelle drinnen?

[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