LSB-Steganographie

Entwicklung einer Applikation zum Verstecken von Dateien in Bildern

Huwiler Samuel  
Maturaarbeit im Fach Informatik

Eingereicht an der Kantonsschule Reussbühl  
am

Betreuung Kantonschule Reussbühl  
Daniel Zurmühle

# Vorwort

Diese Arbeit setzt sich mit dem Thema der `Least Significant Bit`-Steganographie, auch bekannt als LSB-Steganographie, auseinander und stellt den Prozess der steganographischen Methode, in Form einer Python Command-Line-Interface Applikation zum Verstecken von Dateien innerhalb eines beliebigen PNG-Bildes, dar.

In dieser Arbeit wird die Frage beantworten, ob es möglich ist Informationen innerhalb eines PNG-Bildes zu verstecken ohne wahrnehmbaren Effekt auf die Qualität des Originalbildes.

Im theoretischen Teil beschäftigt sich die Arbeit mit den Grundlagen der Steganographie. Es werden die Teile eines steganographischen Systems aufgezeigt, die LSB-Substitution erklärt und grundlegende Begriffe der Kryptographie erklärt.

Das erworbene Wissen aus dieser Theorie wird in einem Programm implementiert. Das entwickelte Programm wird anhand von verschiedenen Testszenarien mathematisch ausgewertet und die Resultate mit der Realität abgeglichen.

Im Schlussteil wird die Fragestellung, ob es möglich ist, Informationen innerhalb eines PNG-Bildes zu verstecken ohne wahrnehmbaren Effekt auf die Qualität des Originalbildes, beantwortet und mögliche Verbesserungen und Erweiterungen des Programmes aufgezeigt.

# Inhaltsverzeichnis

[Vorwort 2](#_Toc178933542)

[Inhaltsverzeichnis 3](#_Toc178933543)

[1 Einleitung 5](#_Toc178933544)

[1.1 Motivation 5](#_Toc178933545)

[1.2 Zielsetzung 5](#_Toc178933546)

[2 Theoretische Grundlagen 7](#_Toc178933547)

[2.1 Begriffsklärung Steganographie 7](#_Toc178933548)

[2.2 Steganographische Systeme 7](#_Toc178933549)

[2.3 LSB-Substitution 8](#_Toc178933550)

[2.4 Verlustfreie Bildformate 8](#_Toc178933551)

[2.5 Tags 9](#_Toc178933552)

[2.6 AES-Verschlüsselung 9](#_Toc178933553)

[2.7 Hashing 10](#_Toc178933554)

[2.8 Kompilierte und interpretierte Sprachen 10](#_Toc178933555)

[3 Praktische Umsetzung 11](#_Toc178933556)

[3.1 Bit-Format 11](#_Toc178933557)

[3.2 Programm Ablauf 12](#_Toc178933558)

[3.3 Limitationen des Vorgang 14](#_Toc178933559)

[3.4 Programmierentscheidungen 15](#_Toc178933560)

[4 Auswertung 17](#_Toc178933561)

[4.1 MSE und PSNR 17](#_Toc178933562)

[4.2 Messungen 18](#_Toc178933563)

[4.3 Analyse 19](#_Toc178933564)

[5 Diskussion 22](#_Toc178933565)

[5.1 Resultate 22](#_Toc178933566)

[5.2 Fehleranalyse 23](#_Toc178933567)

[5.3 Stärken des Programms 23](#_Toc178933568)

[5.4 Schwächen des Programms 24](#_Toc178933569)

[5.5 Mögliche Verbesserungen und Erweiterungen 24](#_Toc178933570)

[6 Fazit 27](#_Toc178933571)

[Literaturverzeichnis 28](#_Toc178933572)

[Abbildungsverzeichnis 30](#_Toc178933573)

[Tabellenverzeichnis 30](#_Toc178933574)

[Danksagung 31](#_Toc178933575)

[Anhang 32](#_Toc178933576)

[Deklaration 33](#_Toc178933577)

# Einleitung

In der heutigen Zeit der Digitalisierung ist das Vermitteln von sensiblen digitalen Daten zwischen zwei Computer oft schwierig umzusetzen. Es gibt immer wieder neue Sicherheitslücken und menschliche Fehler sind nie auszuschliessen. Eine Nachricht per Internet zu verschicken heisst, dass die Nachricht öffentlich sichtbar ist. Es ist deshalb wichtig sicherzustellen, dass nur der richtige Empfänger die Nachricht bekommt und niemand sonst Zugriff auf die Informationen hat. Um dieses Sicherheitsproblem zu lösen, gibt es zwei Möglichkeiten: Man kann die Daten physisch übertragen, ohne dass dabei die Daten ins Web müssen, oder man verschlüsselt die Informationen so, dass nur der Empfänger diese öffnen kann.

Die Steganographie beschäftig sich damit, geheime oder sensible Informationen so zu übertragen, dass man sie nicht als wichtig wahrnimmt. Es geht darum, die Informationen so zu versteckt, dass eine Drittperson nicht weiss, dass Informationen übertragen wurden. Die Informationen werden also nicht verschlüsselt oder gesichert, sondern getarnt.

## Motivation

Ich habe ein grosses Interesse für Informatik und setze mich in meiner Freizeit gerne mit Computern auseinander. Ich habe bereits einige kleine Vorkenntnisse im Bereich der Kryptographie und ich interessiere mich für die Theorie der Datensicherheit. Ein Teilbereich der Datensicherheit, der mir bis dahin noch unbekannt war, ist die Steganographie. Deshalb entschied ich mich, mich in meiner Maturaarbeit damit auseinander zu setzten. Da ein weiteres Hobby von mir das Fotografieren ist, habe ich mich für die Bild-Steganographie, im spezifischen die LSB-Steganographie, entschieden. Diese Form der Steganographie ist heutzutage die meist verwendete und somit am relevantesten. Die LSB-Steganographie ist der Grundbaustein für viele der modernen Bild-Steganographie Methoden.

Da ich neben der Theorie auch meine Kenntnisse im Softwaredesign verbessern möchte, habe ich mich entschieden, ein benutzerorientiertes Programm zu schreiben, in welchem die Theorie der LSB-Steganographie angewendet wird.

## Zielsetzung

Ich schreibe ein Programm, welches die LSB-Steganographie einsetzt, um eine Datei innerhalb eines Bildes zu verstecken.

Der Bildmodifizierungsvorgang sollte so verlaufen, dass kein visueller Unterschied zwischen der original und der modifizierten Bild-Datei zu erkennen ist. Ein aussenstehender menschlicher Betrachter sollte also nicht erkennen können, dass das Bild verändert wurde. Zusätzlich sollten die verstecken Nachrichten ausschliesslich mit meinem Programm wieder lesbar sein.

Die Steganographie hat sich in den letzten Jahren stetig weiterentwickelt und beinhaltet viel mehr als nur die einfache LSB Steganographie. In meiner Arbeit beschränke ich mich jedoch nur auf diesem Teilbereich. Das Verhindern von Computerdetektion wird weggelassen.

Für die Demonstration meiner Implementierung werde ich ausschliesslich Bilder verwenden, die ich kreiert habe.

# Theoretische Grundlagen

## Begriffsklärung Steganographie

Die moderne Definition von Steganographie lautet: Informationen oder Dateien in digitalen Bilder, Videos oder Audiodateien zu verstecken und wieder hervorzurufen[1].  
Dieser Prozess kann in fünf Komponenten aufgeteilt werden: die Cover-Datei, die Stego-Datei, die Nachricht, der Schlüssel und die Embedding Domain. Diese bilden zusammen ein steganographisches System (vgl. 2.2).

Die Steganographie ist nicht limitiert auf digitale Dateien. Die Steganographie an sich gibt es schon seit mehreren Jahrhunderten. So wurden oft in Kriegszeiten Nachrichten innerhalb von Briefen versteckt, um den feindlichen Spionen den Anschein zu geben, dass keine wichtigen Informationen zu finden sind. Oft wurden Texte geschrieben, bei denen z.B. nur alle dritten Wörter zu lesen waren oder es wurden Schablonen genutzt, um nur bestimmte Wörter sichtbar zu machen. Der Teil, der nicht abgedeckt wurde, beinhaltet dann die geheime Nachricht (vgl. Abbildung 1). Auch das Schreiben mit Zitronensaft auf Papier, so dass die Nachricht nur mit Wärme wieder sichtbar wird, fällt unter das Gebiet der Steganographie [2].

Heutzutage wird die Steganographie hauptsächlich für digitale Wassermarken in Bildern verwendet. Dabei werden Nachrichten innerhalb eines Bildes versteckt, die Aussagen darüber machen, wer das Recht auf dieses Bild besitzt, wann es gemacht wurde etc. Falls jemand nun das Bild verwendet, kann überprüft werden, wer das Urheberrecht besitzt und somit bewiesen werden, ob es sich um ein Plagiat handelt [3].

Ein Bild, das Text, Brief, Handschrift, Papier enthält.

Automatisch generierte BeschreibungEin Bild, das Kunst enthält.

Automatisch generierte BeschreibungEin Bild, das Text, Brief, Handschrift, Papier enthält.

Automatisch generierte Beschreibung

Abbildung : Verstecke Nachricht in einem Brief, mit Hilfe einer Schablone [4]

## Steganographische Systeme

Ein Steganographisches System besteht aus fünf Teilen [1].

* **Nachricht:** Die zu versteckende Nachricht. Dies kann ein Text oder eine ganze Datei sein.
* **Cover-Datei:** Die Datei in welcher die Informationen übertragen werden. Sie wird oft so gewählt, dass sie für einen äusseren Betrachter als unwichtig wirkt.
* **Stego-Datei:** Die fertige Datei, in der die Nachricht gespeichert wurde. Sie unterscheidet sich von aussen nicht von der Cover-Datei.
* **Schlüssel:** Der Schlüssel oder eine andere Information, mit der man die Nachricht aus der Stego-Datei herauslesen kann. Dies kann zum Beispiel die Anzahl an geänderten LSB-Bits sein.
* **Embedding Domain:** Die bestimmte Eigenschaft der Cover-Datei, die ausgenützt wird, um die Nachricht zu verbergen. Zum Beispiel die letzten Bits der Pixelwerte.

## LSB-Substitution

Bei der LSB-Steganographie werden die Informationen mittels LSB-Substitution in der Cover- Datei versteckt.

In der LSB-Substitution werden die `Least Significant Bits` (LSB) eines Byte ersetzt durch die zu versteckende Information. LSB sind die letzten Bits eines Byte (vgl. Abbildung 2). Um die Information wieder zu lesen, werden die LSB ausgelesen und wieder zu Bytes zusammengefügt. Die Farbe der Pixels ändert sich, jedoch ist die Änderungen so klein, dass die Farbunterschiede nicht wahrnehmbar sind (vgl. Abbildung 3) [1].

Ein Bild, das Text, Screenshot, Zahl, Reihe enthält.

Automatisch generierte Beschreibung

Abbildung : Hervorhebung der LSB bei 4 Bytes (in Anlehnung an [1])

Ein Bild, das Screenshot, Text, Diagramm, Reihe enthält.

Automatisch generierte Beschreibung

Abbildung : LSB-Substitution bei einem RGB Pixel (in Anlehnung an [1])

## Verlustfreie Bildformate

Farben werden digital immer zusammengesetzt aus Rot, Grün und Blau. Wenn man rotes, grünes und blaues Licht addiert, erhält man Weiss. Deshalb bestehen die Pixel eines Bild immer aus diesen drei RGB-Werten, die die Farbe des Pixel vorgeben.

Es gibt zusätzlich zwei Ansätze, wie man ein Bild digital speichern kann. Entweder werden alle Rot, Grün und Blau Werte jedes Pixels nacheinander gespeichert oder man komprimiert, mit Hilfe von Algorithmen, das Bild so, dass man minimal an Qualität verliert, dafür aber die Grösse der Datei stark verkleinert [5].

Die meisten Bilder müssen nicht in voller Grösse gespeichert werden, deshalb sind Formate wie JPEG der Standard.

Für meine Applikation benötige ich verlustfreie Bildformate, denn sonst gehen die minimalen Veränderungen an den einzelnen Pixel beim Speichern verloren. Mein Verfahren basiert darauf, dass an den jeweiligen Pixeln minimale Änderungen vorgenommen werden. Diese sind von Auge nicht sichtbar, werden deshalb auch von Formaten wie JPEG weggelassen.

Aus diesem Grund benutze ich PNG-Bilder. Dieses Format speichert die Pixel ohne Verlust. Zusätzlich kann es gut bearbeitet werden. PNG ist von Grund auf für Programmier-Benutzer Freundlichkeit aufgebaut und kann deshalb gut modifiziert werden [6].

PNG unterstützt auch Transparenz, dies wird erreicht, indem für jedes Pixel vier Werte gespeichert werden. Zusätzlich zu Rot, Grün und Blau kommt noch Alpha dazu. Dieser Wert gibt an, wie transparent das Pixel ist. Ein Alpha-Wert von 255 bedeutet, das Pixel wird nicht angezeigt, ein Wert von 0 bedeutet, dass das Pixel vollständig sichtbar ist.

Um Platz zu sparen, kann eine PNG-Datei auch diesen Alpha Chanel weglassen und nur RGB speichern.

## Tags

Um verschiedene Werte, mit nicht bekannter Länge, zu unterscheiden, werden Tags zwischen den Werten eingefügt. So werden die Informationen voneinander getrennt. Wenn alle Werte eine bekannte Länge haben, dann müssen sie nicht voneinander getrennt werden [7].

Der Vorteil von Tags ist, dass sie die Grösse der Nachricht verkleinern können, weil sie nicht überflüssige, leere Informationen übertragen. Ein anderer Ansatz, um Daten voneinander zu trennen, wäre das Allokieren von einem vorgegebenen Bereich. Damit kann man sich den zusätzlichen Speicher, der sonst von Tags verwendet wird, sparen. Dies bedeutet jedoch, dass man bereits im Voraus wissen muss, wie gross die Nachricht ist.

## AES-Verschlüsselung

Für den Fall, dass eine Drittperson die geheime Nachricht innerhalb der Cover-Datei entdeckt und extrahiert hat, kann die Nachricht zusätzlich mit dem AES Cypher verschlüsselt werden. Der `Advanced Encryption Standard` oder AES ist ein Standard für Verschlüsselung welcher drei verschiedene `Symmetric Key Encryption Cypher` definiert [8].

`Symmetric Key Encryption` ist eine Form der Verschlüsselung, bei der ein Schlüssel gebraucht wird, um eine Nachricht zu verschlüsseln und zu entschlüsseln [9]. Dies dient dazu, dass die Nachricht völlig unzugänglich ist, auch wenn die geheime Nachricht entdeckt wurde.

## Hashing

Eine Hash-Funktion ist eine Einwegfunktion, die eine beliebig lange Nachricht auf eine feste Länge kürzt, wobei eine kleine Änderung an der Nachricht zu einem komplett anderen Resultat führt.

Eine Hash-Funktion funktioniert nur in eine Richtung. Das bedeutet, dass es nicht möglich ist mit dem Resultat der Funktion die Originalnachricht zu generieren. Zwei identische Nachrichten erzeugen immer den gleichen Hash. Sie können deshalb benutzt werden, um die Validität einer Nachricht zu überprüfen, weil es höchst unwahrscheinlich ist, dass zwei verschiedene Nachrichten die gleichen Hash-Werte haben [10].

## Kompilierte und interpretierte Sprachen

Bevor man ein Programm auf einem Computer laufen lassen kann, muss die Programmiersprache, in der das Programm geschrieben wurde, umgewandelt werden in Maschinen Code. Programmiersprachen abstrahieren die Funktionsweisen des Computers, um das Entwickeln von Programmen zu erleichtern. Je näher eine Sprache an den einzelnen CPU-Befehlen ist, desto mehr muss man sich als Programmierer Sorgen über Systemspeichernutzung, Daten- und Variablenverwaltung und Prozessoptimierungen machen. Dafür hat man mehr Kontrolle über die Hardware selbst. Diese Sprachen werden oft direkt von einem `Compiler` in Maschinencode umgewandelt. Dieser Maschinencode kann dann von einem Benutzer ausgeführt werden.

Sprachen, die mehr Abstraktionen einführen, machen es für den Programmierer einfacher, seine Ideen mit Code zu beschreiben. Speicher- und Variablenverwaltung werden automatisiert. Solche Sprachen sind oft interpretiert. Anstelle von einer einmaligen Übersetzung in Maschinencode, wird bei interpretierten Sprachen das Programm Schritt für Schritt in Computerbefehle umgesetzt. Dieser Interpretierschritt hat den Nachteil, dass das Programm jedes Mal neu interpretiert werden muss, was zu einer längeren Laufzeit führt. Dafür ermöglicht es dem Programmierer, sich auf die grosse Idee zu fokussieren, weil viele Teile der Speicher- und Variablenverwaltung vom `Interpreter` übernommen werden.

# Praktische Umsetzung

## Bit-Format

Ein wichtiger Schritt in der Entwicklung meines Programmes ist das Erstellen eines Bit-Formates.

Dieses gibt vor, wie die verschiedenen notwendigen Informationen über die geheime Nachricht innerhalb des Bildes angeordnet werden. Es definiert, was in die LSB-Bits geschrieben und gelesen werden soll, um sicherzustellen, dass das Programm so funktioniert wie ich es mir vorgestellt habe.

Mein Programm soll folgendes leisten (vgl. Anhang C):

* Ein Benutzer soll zeit- und plattformunabhängig die gleiche Datei aus einem Bild entschlüsseln können, ohne Vorwissen über die Art der Verschlüsselung oder die verschlüsselte Datei selbst.
* Es soll, zusätzlich zum Inhalt der Datei, auch der Dateiname verschlüsselt werden.
* Der Leser sollte überprüfen können, ob die Datei verändert wurde, etwa durch ungewollte Komprimierung oder Bildbearbeitung.
* Das Programm soll in der Lage sein, mit verschiedenen Stärken an Bildmanipulation zu funktionieren. Also mit einer variablen Anzahl an modifizierten LSB-Bits.

Um diese Aufgaben zu erfüllen, habe ich diese Anforderungen an mein Programm gesetzt:

* Alle notwendigen Informationen zum Verschlüsselungsprozess müssen innerhalb der Bilddatei gespeichert werden. Somit kann garantiert werden, dass jede Person, die das Programm benutzt, die gleiche Datei entschlüsselt.
* Um die erhaltene Datei verifizieren zu können, muss ein SHA-256 Hash eingebettet werden. Dieser funktioniert wie ein digitaler Fingerabdruck, mit dem die empfangene Datei dann verglichen werden kann.
* Die Anzahl der geänderten Bits muss gespeichert werden. Diese Information muss als erstes gespeichert werden, weil sie angibt, wie die Datei zu entschlüsseln ist.
* Alle diese Informationen müssen immer auf die gleiche Art gespeichert werden, da sie für den Start des Entschlüsselungsprozesses benötigt werden und nicht im Voraus bekannt sind.

Um all dem gerecht zu werden, bin ich zu diesem Bit-Format gekommen:

Das Allererste, was beim Verschlüsseln mitgeschrieben wird, ist die Anzahl der Bits, die geändert werden. Diese Information wird separat im ersten Rot-Wert des ersten Pixels gespeichert. Danach werden alle Informationen in den kommenden LSB-Bits gespeichert.

Das nächste Element ist der SHA-256 Hash. Dieser hat immer eine konstante Länge von 256 Bit und kann daher einfach ausgelesen werden.

Die nächsten Elemente sind schwieriger, da sie keine feste Länge haben. Der Empfänger weiss nicht, wie gross die verschlüsselte Datei ist oder wie lang der Dateinamen ist.  
Ich habe mich deshalb für ein System entschieden, bei dem der Dateiname, der Dateiinhalt und der Bildrest jeweils durch ein Tag getrennt wird. In meinem Programm ist dies die Zeichenfolge «[STEG]». Dieser Tag wird als 48 ASCII decodierte Bits mitgeschrieben. Beim Auslesen wird alles nach dem Hash als Dateiname betrachtet, bis man auf die Bitfolge für «[STEG]» stösst. Ab dann ist alles nur noch Dateiinhalt. Wenn man wieder auf den Tag stösst, weiss man, dass die Datei fertig ist (vgl. Abbildung 4). Dies dient auch als zusätzliche Kontrolle. Wenn mehr als zwei oder nur ein Tag vorhanden ist, dann ist das Bild nicht kompatibel mit meinem Programm. In diesem Fall wird dem Nutzer eine Meldung angezeigt. Das kann passieren, wenn ein beliebiges Bild per Zufall die richtige Reihenfolge an Bits beinhaltet oder weil die Stego-Datei zugeschnitten, resp. einer der Tags abgeschnitten wurde.

Ein Bild, das Text, Screenshot, Reihe, Diagramm enthält.

Automatisch generierte Beschreibung

Abbildung : Visuelle Darstellung des Bit-Format

## Programm Ablauf

Mein Programm besteht aus zwei Komponenten. Dem `Embeding` und dem `Extracting`, jeweils zum Schreiben und Lesen der Nachricht aus dem Bild.

Der Embedingvorgang sieht folgendermassen aus:

* Das Programm liest die notwendigen Nutzereingaben aus der Konsole heraus und speichert diese intern ab.
* Die Benutzerargumente werden geprüft. Es wird kontrolliert, ob das angegebene Cover-Image wirklich eine Bild-Datei ist, ob der Nutzer alle notwendigen Argumente angegeben hat und ob diese den Erwartungen entsprechen. Es kann zum Beispiel sein, dass bei einem Zahlenfeld ein Buchstabe eingegeben wurde oder umgekehrt. Falls ein Fehler auftritt, wird das Programm abgebrochen und der Nutzer wird über den Fehler informiert.
* Danach werden die Cover-Datei gelesen und das Bild wird in RGB- und Alpha Arrays unterteilt. Der erste Wert des RGB-Array wird separat gespeichert. Dieser wird benötigt, um die Anzahl an modifizierten LSB-Bits zu speichern.
* Das Bit-Format wird aus den gegebenen Daten erstellt (vgl. 3.1).
  + Die Datei wird komprimiert und verschlüsselt. Wenn kein Passwort angegeben wird, dann wird ein festgelegtes Passwort als Schlüssel für den `AES Cypher’ gebraucht.
  + Die Datei wird gehasht.
  + Hash, Dateiname, Trenn-Tag und Dateiinhalt werden zusammengesetzt.
* Die erstellten Bits werden in den RGB-Werten gespeichert. Die Bits werden in Abschnitte mit Länge n aufgeteilt und dann jeweils in die n letzten Bits jedes RGB-Wertes gespeichert.
* Der modifizierte RGB-Array wird mit dem Alpha Array zusammengefügt.
* Das Bild mit den neuen Pixel-Werten wird gespeichert.

Das Extractingvorgang läuft praktisch gleich ab wie das Schreiben, nur in umgekehrter Reihenfolge.

* Das Programm liest die notwendigen Nutzereingaben aus der Konsole heraus und speichert diese intern ab.
* Die Argumente werden überprüft.
* Das Stego-Bild wird getrennt in RGB- und Alpha Arrays.
* Es werden die letzten drei Bits des ersten Wertes des RGB-Array gelesen und abgespeichert. Dieser Wert gibt an, wie die Informationen gespeichert wurden.
* Es werden jeweils die letzten n Bits jedes RGB-Wertes gelesen und abgespeichert.
* Danach werden die ersten 256 Bits als Hash gespeichert.
* Die nächsten Bits werden getrennt nach dem Tag.
* Der erste Teil der Nachricht ist der Dateiname, der zweite ist der Dateiinhalt und der Rest kann ignoriert werden.
* Der Dateiinhalt wird entschlüsselt und dekomprimiert.
* Danach wird der Dateiinhalt, mit der gleichen Hash-Funktion wie beim Schreiben, gehasht und die Werte werden verglichen. Wenn die Werte identisch sind, dann ist die Datei korrekt übertragen worden, wenn nicht, ist das Stego-Bild entweder manipuliert worden oder es war gar nie eine Nachricht gespeichert worden.
* Die Datei wird auf der Festplatte des Nutzers gespeichert.

Ein Bild, das Text, Diagramm, Plan, Screenshot enthält.

Automatisch generierte Beschreibung

Abbildung : Programmablauf beim Embeding

Ein Bild, das Text, Diagramm, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

Abbildung : Programmablauf beim Extracting

## Limitationen des Vorgang

In ein Cover-Bild können nur Nachrichten versteckt werden die kleiner sind als das Dreifache der Anzahl Pixel multipliziert mit der Anzahl an modifizierten LSB . Die Maximale Grösse der geheimen Nachricht in Bits wird wie folgt definiert, wobei in dieser Formel die Grössen des Hash, des Dateinamens und der Tags vernachlässigt werden:

Da die geheimen Nachrichten oft kleiner sind als die maximale Grösse, stellt sich die Frage, wie die Nachricht im Cover-Bild aufzuteilen ist. Ich habe mich bei meinem Programm dafür entschieden, die gesamte Nachricht am Anfang des Bildes zu verstecken. Das bedeutet, die Nachricht fängt im Bild oben rechts an und hört auf, wenn die gesamte Nachricht versteckt ist. Der Rest des Bildes bleibt unverändert. Abbildung 7 zeigt, welcher Bereich des Bildes geändert wird und wie sich diese Fläche verändert, wenn die Nachricht grösser wird.

Ein Bild, das Muster, Screenshot, Rechteck, Text enthält.

Automatisch generierte Beschreibung

Abbildung : Manipulierter Bereich eines Cover-Bild bei grösser werdenden Nachrichten.

## Programmierentscheidungen

Als Programmiersprache für meine Arbeit habe ich Python gewählt, weil ich bereits Erfahrung mit der Sprache habe und sie sich gut für schnelles Programmieren und Entwickeln eignet. Im Gegensatz zu kompilierten Sprachen wie C++ oder Rust ist Python interpretiert und somit langsamer in der Ausführung. Ich habe mich aber trotzdem für Python entschieden, weil ich, durch meine Erfahrung in der Sprache und das Dynamische Design, einfacher meine Ideen umsetzten kann und diese anschliessend auch besser erweitern und ändern kann. Das Ziel meiner Arbeit ist nicht ein robustes, schnelles und sicheres Produkt für den allgemeinen Verbrauch herzustellen, sondern die Theorie der Steganographie umzusetzen. Somit bietet mir Python mehr Möglichkeiten diese Theorie anzuwenden als eine statisch-typisierte kompilierte Sprache.

Für das Lesen, Schreiben und Manipulieren von Bilddateien habe ich das Python-Modul Pillow benutzt. Dieses erlaubt es mir, die Cover-Bilddateien auszulesen, zu modifizieren und anschliessend das Stego-Bild zu speichern. Es erlaubt mir wiederum auch, die Stego-Bilddatei zu lesen und somit die geheime Nachricht zu sehen [11].

Pillow ist eine moderne Erweiterung von PIL, ein weiteres älteres Bildbearbeitungsmodul. Mittlerweile ist Pillow zum Standard für Python Bildmanipulation geworden und es wird von vielen anderen Modulen unterstützt.

Für das leichtere Manipulieren und Speichern der Pixel-Werte benutze ich Numpy.

Numpy ist ein wissenschaftliches Python Modul für Manipulationen von grossen Zahlen, Matrizen und Arrays. Pillow integriert sich gut mit Numpy und ein Pillow-Image lässt sich direkt in einen Numpy-Array konvertieren. Umgekehrt kann man auch ein Image direkt aus einem Array generieren [12]. Dies bietet eine gute Abstraktion für das Arbeiten mit den Pixel-Werten, da das ganze Bild nun nur noch eine lange Liste von Zahlen ist, die sich leicht ändern lassen.

# Auswertung

Das Ziel meiner Arbeit ist es herauszufinden, ob es möglich ist eine Datei innerhalb eines Bildes zu verstecken, ohne äussere Veränderung des Erscheinungsbildes.

Da das Auswerten der Resultate ausschliessend mit menschlichem Auge kein objektives Bild über die Performance meines Programms gibt, habe ich die Bilder mittels PSNR ausgewertet. PSNR oder `Peak Signal to Noise Ratio` ist ein Mass dafür, wie sich zwei Wertepaare voneinander unterscheiden. PSNR wird oft in der Bildkompression genutzt, um zu quantifizieren, wie viel Information durch das Komprimieren verloren gegangen ist.

Ein tiefer PSNR-Wert deutet auf eine grosse Unterscheidung hin. Das menschliche Auge erkennt visuelle Unterschiede ab 40 – 50db. Für meine Auswertung habe ich deshalb 50db als untere Schwelle für akzeptable Resultate gesetzt. Alles über 50db ist demnach als genügend zu betrachten.

## MSE und PSNR

`Mean Squared Error`, auch MSE genannt, wird definiert als der Durchschnitt der Quadrate aller Abweichungen zwischen zwei 2D Matrizen. In meinem Fall sind das die Pixel Werte des Bildes. Diese Formel gilt aber nur für Monochromatische Bilder. Bei Farbbilder, wie bei meinem Programm verwendet, wird der MSE für jeden Farbkanal berechnet und anschliessend summiert. Dabei ist zu betrachten das dieser Wert nun grösser ist als in Wirklichkeit. Deshalb wird der Wert durch 3 und die Grösse des Bildes geteilt.

`Peak Signal to Noise Ratio` oder PSNR ist definiert als das Verhältnis zwischen dem grössten Wert und dem Wert des Störungsrauschen, das die Qualität des Original beeinflusst [13].

Die Formeln für MSE und PSNR lauten:

Dabei gilt:

x = Cover-bild  
y = Stego-Bild  
*m* = Anzahl an Reihen des Bild (Breite in Pixel)  
*n* = Anzahl an Spalten des Bild (Höhe in Pixel)  
*MAXI* = Grösster Wert des Originalbildes; für RGB-Bilder: 255  
i und j repräsentieren den Index eines Pixel

Für die Auswertung mit Python verwende ich Numpy. Damit kann ich die Bilder in einen 3D Array konvertieren, welcher die RGB-Triplets für jeden Pixel beinhaltet. Numpy hat eine Funktion `mean` welche direkt auf 3D Array anwendbar ist. Um den MSE zu berechnen, subtrahiere ich den Array des Cover-Bilds vom Array des Stego-Bilds und quadriere alle Werte. Als Resultat erhalte ich den `Squared Error`. Anschliessend berechne ich mit Numpy das `mean` dieses `Squared Error` und erhalte so den MSE. Für die Berechnung des PSNR wird die Formel verwendet. (vgl. Anhang B)

## Messungen

Ich habe für meine Messungen folgende Bilder und Testdateien verwendet:



Tabelle : Bilder für die Auswertung des Programms



Tabelle : Lorem-Ipsum Testdateien für die Auswertung des Programms

Abbildung 8 stellt die PSNR-Werte dar, die entstehen, wenn man in den verschiedenen Cover-Bilder (Tabelle 1) die Testdateien (Tabelle 2) oder die Cover-Bilder versteckt. Hier dargestellt anhand von PSNR-Wert in Relation zur Anzahl an modifizierten LSB.



Abbildung : PSNR abhängig von der Anzahl modifizierter LSB

## Analyse

Die Messungen führen zu folgender Erkenntnis:

Die Qualität des Stego-Bildes nimmt mit zunehmender Anzahl an modifizierter LSB ab. Dies ist in Abbildung 8 gut ersichtlich. Dies bestätigt die Annahme, dass die Änderungen am Cover-Bild bei einem bis drei Bits nicht zu sehen sind. Auch bei vier Bits ist bei grösseren Cover-Bildern keine Änderung zu sehen. Es ist jedoch möglich bei vier geänderten LSB minimale Änderungen festzustellen, wenn man das Stego-Bild Pixel für Pixel vergleicht. Ab fünf Bits sind die Änderungen auch ohne Vergleich sichtbar und es bilden sich sichtbare Farbblöcke bei Stellen mit hohem Kontrast. Diese Farbfehler werden mit zunehmender Anzahl an überschriebenen LSB verstärkt. Bei acht LSB bleibt von der Originalfarbe nichts mehr übrig (vgl. Abbildung 9, Abbildung 10Abbildung 9 und Abbildung 11).

Nachrichten, die nur im letzten Bit versteckt wurde, führen zu einer minimalen Abweichung zum Cover-Bild. Diese Abweichungen werden zunehmend grösser mit der Anzahl an Bits. Auch erkennbar ist, dass die Kurve abflacht nach vier modifizierten LSB. Die Abnahme der Qualität scheint linear zu verlaufen bis vier LSB danach sinkt die Qualität minimal ab. Anzunehmen wäre jedoch, dass sich die Qualität kontinuierlich verschlechtert, je mehr LSB geändert werden.

Die Grösse der Nachricht beeinflusst die Qualität des Stego-Bild. Eine kleine Nachricht führt zu einem kleinen Verlust der Qualität im Vergleich zu einer grösseren Nachricht. Diese Qualitätsabnahme scheint konstant zu sein für jede Anzahl an LSB.

Somit kann zusammengefasst werden, dass die Grösse der Cover-Bild Datei keinen erheblichen Einfluss auf die Qualität des Stego-Bild hat. Die Qualität nimmt ausschliesslich mit der Anzahl LSB und der Grösse der Nachricht ab.

 

Abbildung : Farbfehler bei fünf LSB (Ausschnitt brille.png)

 

Abbildung : Farbfehler bei fünf LSB (Ausschnitt auto.png)



Abbildung : Wasser.png mit 8 modifizierten LSB

# Diskussion

## Resultate

Die Resultate der Auswertung bestätigen die Annahme, dass die Änderungen am Cover-Bild bei einem bis drei Bits nicht zu sehen sind. Auch bei vier Bits ist, bei grösseren Cover-Bildern, keine Änderung zu sehen. Es ist jedoch möglich minimale Änderungen festzustellen, wenn man das Stego-Bild Pixel für Pixel vergleicht. Ab fünf Bits sind die Änderungen auch ohne Vergleich sichtbar.

Die Korrelation zwischen Grösse der Nachricht und dem Verschlechtern des PSNR-Wertes ist darauf zurückzuführen, dass bei einer grösseren Nachricht ein grösserer Anteil des Bildes manipuliert werden muss. Dies führt wiederum zu einem grösseren PSNR-Wert.

Auch ersichtlich ist, dass sich die Qualität nach vier `Least Significant Bits` nicht zusätzlich verschlechtert. Zu erwarten wäre eine kontinuierliche Abnahme des PSNR-Wertes. Dies ist hier jedoch nicht der Fall. Dieses Phänomen entsteht dadurch, dass die Grösse der modifizierten Fläche mit der Anzahl an LSB abnimmt. Eine Substitution mit acht Bits nimmt immer auch acht Mal weniger Platz im Stego-Bild ein, als eine bei der nur das letzte Bit geändert wurde. Dies führt dazu, dass sich die gesamte Abänderung im Bild nach vier LSB nicht gross verschlechtert. Es ist jedoch gut von Auge ersichtlich, dass das Stego-Bild manipuliert wurde.

Abbildung 12 zeigt das Cover-Bild Wasser.png bei dem Lorem\_50k.txt innerhalb der letzten 3, 6 und 8 Bits versteckt wurde. Ab sechs LSB-Bits ist klar ersichtlich, dass das Bild manipuliert wurde, auch wenn der PSNR-Wert sich nur um ein Dezibel verändert zwischen 6 und 8 Bits (vgl. Tabelle 3).

Ein Bild, das Wasserfall, draußen, gefäßlose Landpflanze, Strom enthält.

Automatisch generierte BeschreibungEin Bild, das Wasserfall, Wasser, Pflanze, draußen enthält.

Automatisch generierte BeschreibungEin Bild, das Wasserfall, Wasser, Natur enthält.

Automatisch generierte Beschreibung

Abbildung : Lorem\_50k versteckt innerhalb von wasser.png 3,6 und 8 modifizierter LSB



Tabelle : PSNR-Wert und Anzahl an LSB bei lorem\_50k versteckt in wasser.png

## Fehleranalyse

Die Messung der Qualität anhand von PSNR führt zu einigen Abweichungen gegenüber der Realität.  
PSNR gibt die gesamte Abweichung zwischen zwei Bildern an. Dabei kann es dazu kommen, dass das Minimum von 40db nicht unterschritten wird und man trotzdem eine visuelle Veränderung wahrnehmen kann. Das menschliche Auge nimmt nicht nur die durchschnittliche Abweichung war. Es wäre deshalb von Vorteil, wenn man nur den PSNR-Wert vom manipulierten Bereich betrachten würde. Dies würde den Effekt der Verkleinerung der manipulierten Fläche, mit zunehmend grösseren Anzahl an modifizierten Bits eliminieren. Trotzdem sind keine visuellen Änderungen zu erkennen bei 1,2 oder 3 geänderten Bits, auch wenn der gemessene PSNR-Wert höher ist als zu erwarten.

## Stärken des Programms

Mein Programm hat alle von mir gestellten Erwartungen erfüllt und ist in der Lage alle notwendigen Operationen durchzuführen.

* Mein Programm setzt die LSB-Steganographie erfolgreich um und kann eine beliebige Datei innerhalb eines PNG speichern.
* Mein Programm ist in der Lage, eine versteckte Nachricht wieder zu extrahieren, wenn diese Nachricht zuvor von einem gleichen Programm versteck wurde.
* Mein Programm ist in der Lage zu überprüfen, ob ein Bild eine geheime Nachricht enthält, die von einem gleichen Programm versteck wurde oder ob sie keine enthält.
* Mein Programm kann Dateien in ein- bis acht `Least Significant Bits` verstecken.
* Mein Programm kann unabhängig vom Benutzer herausfinden, wie viele LSB verwendet wurden, um eine Nachricht zu verstecken.
* Die Wahl des Bitformat ermöglich, dass man allein mit dem Stego-Bild die vollständige geheime Datei erhalten kann.

## Schwächen des Programms

Mein Programm ist in keiner Hinsicht perfekt, es gibt überall Kleinigkeiten, die dazu führen, dass mein Programm nicht optimal läuft oder nicht zu 100% unerkannt sein kann. Dies ist mir im Vorhinein bereits bekannt gewesen und ich habe dies auch so in Kauf genommen.

* Die Platzierung der Daten innerhalb der Cover-Datei ist suboptimal für das Verstecken von Informationen. Dadurch, dass sich alle Daten am Start des Bild befinden, ist es einfacher möglich eine Änderung zu erkennen.
* Durch das Ändern der Farben im Cover-Image ist es möglich, mittels Vergleich festzustellen, dass das Bild modifiziert wurde. In dem man das Stego-Bild mit dem Cover-Bild vergleicht.
* Der Gebrauch von Tags im Bitformat führt dazu, dass beim Entschlüsseln nicht bekannt ist, wie gross die geheime Datei ist. Es ist deshalb notwendig, die LSB aller Pixel zu bestimmen, um an die Nachricht zu gelangen. Dies führt zu erheblichen Performance Einschränkungen bei Fällen, bei denen eine kleine Nachricht innerhalb eines grossen Cover-Bildes versteckt wurde, weil man die Mehrheit der Pixel gar nicht auslesen müsste, um die Nachricht zu erhalten.
* Durch das Benutzen von Python ist das Entschlüsseln und Verschlüsseln zeitintensiv und es kann, bei grösseren Bildern, 60 Sekunden oder länger dauern, bis der Vorgang abgeschlossen werden kann.
* Das Modifizieren der Farben des Cover-Bildes führt dazu, dass sich die Dateigrösse im Vergleich zum Original verändert. PNG-Bilder benutzen, um Platz zu sparen, verlustfreie Kompression. Diese ist nicht so effizient wie eine `Lossy-Kompression`, jedoch spart sie viel Platz im Vergleich zu unkomprimierten Bildern. Die meisten Bilder besitzen Regionen, in denen mehrere Pixel die gleiche Farbe haben. Zum Beispiel eine Hauswand oder eine Wiese in der Ferne. Solche Bildregionen können einfach zusammengefasst werden, ohne Informationen zu verlieren. Wenn sich jedoch jedes Pixel minimal von seinem Nachbar unterscheidet, dann können diese Regionen nicht mehr komprimiert werden und die Datei wird zwangsmässig grösser.

## Mögliche Verbesserungen und Erweiterungen

Folgende Bereiche meines Programms könnten verbessert werden:

* die Erkennbarkeit gegenüber statistischen Mittel
* die Laufzeit
* das Verhindern der Dateivergrösserung

Es ist in der LSB-Steganographie nie möglich, ein Bild so zu manipulieren, dass keine Veränderung statt stattfindet, weil man gezwungen ist, Informationen zum Cover-Bild hinzuzufügen oder das Cover-Bild zu ändern.

Wichtig ist jedoch, dass man im Normalfall als Aussenstehender nie Zugriff auf das Originalbild haben sollte. So kann ein Vorher-Nachher Vergleich verhindert werden. Denn jede kleine Änderung kann, mit Hilfe von Computer, erkennt werden. Zusätzlich ist es möglich durch Verfahren wie Histogramm Analyse [14] festzustellen, ob ein Bild modifiziert wurde, auch ohne direkten Vergleich.

Mit heutigen Methoden ist es trivial, solch einfache Änderungen durch LSB-Steganographie zu erkennen. Dennoch gibt es in der Literatur einige Ansätze wie man die LSB-Steganographie verbessern könnte. Es ist zum Beispiel möglich die Nachricht nur innerhalb von Bereichen mit hohen Kontrasten, wie zum Beispiel Ecken, zu speichern. Dies führ dazu, dass sich die Bildfehler nicht auf einem Haufen befinden [15]. Diese Bildfehler sind Abweichungen zwischen Cover- und Stego-Bild, die durch die LSB-Substitution verursacht werden.

Dadurch, dass mein Programm die Nachricht innerhalb der ersten Pixel versteckt (vgl. Abbildung 7), entsteht im Stego-Bild eine lokalisierte Anhäufung an kleinen Farbunterschieden. Diese Bildstörungen hören abrupt auf am Ende der Nachricht. Dieses lokalisierte Rauschen kann mittels statistischer Analyse einfach erkennt werden. Hier abgebildet ist das Stego-Bild einer Blume in der die Datei Muster.png in den letzten zwei `Least Significant Bits` versteckt wurde. Abbildung 13 ist das Resultat einer Subtraktion der Pixel Werte von dem Cover-Bild und dem Stego-Bild. Die Werte wurden dann anschliessend noch normalisiert auf 128 und verstärkt, um ein gut sichtbares Resultat zu erhalten. Änderungen der Farbwerte von +1 werden zu 255 und Änderungen von -1 zu 0.

 

Abbildung : Pixel Subtraktion des Stego-Bild einer Blume und dem original Cover-Bild

Um das Problem mit der Performance zu lösen, gibt es grundsätzlich zwei Hauptansätze:  
Es wäre möglich von Python auf eine kompilierte Sprache zu wechseln, was dazu führen würde, dass das Programm schneller laufen würde.

Auch wäre es möglich, beim Auslesen einer geheimen Nachricht, die Bits sequenziell auszulesen. Anstelle davon, alle `Least Significant Bits` zu bestimme, würde man Stück für Stück die Pixel Werte auslesen, bis man auf den letzten Tag stösst. Wenn man diesen erreicht hat, kann das Programm mit dem Lesen aufhören und anfangen, die Nachricht zu dekomprimieren und zu validieren. Dieser Ansatz würde meiner Meinung nach am meisten Laufzeitverbesserung bringen.

Für das Verändern der Dateigrösse gibt es keine einfachen Lösungen. Eine Möglichkeit wäre es, ein Bildformat zu wählen, welches keine verlustfreien Kompressionen einsetzt, zum Beispiel Bitmap. Eine andere Lösung wäre es, andere steganographischen Methoden zu verwenden, die nicht direkt die Farbe der Pixel beeinflusst. Diese Methoden übersteigen jedoch den Rahmen meiner Arbeit.

# Fazit

Die Steganographie ist die Wissenschaft der verborgenen Nachrichten. Sie beschäftigt sich damit, wie Informationen getarnt weitergegeben werden können. Eine Form der Steganographie ist die LSB-Steganografie. Auf diese habe ich mich bei meiner Arbeit fokussiert und mit Hilfe dieser Methode aufgezeigt, dass es möglich ist, Informationen innerhalb eines PNG-Bildes zu verstecken ohne wahrnehmbaren Effekt auf die Qualität des Originalbildes.

Mein erworbenes Wissen aus der Theorie der LSB-Steganographie habe ich in einem selbstentwickelten Computerprogramm angewendet. Es ist mir gelungen, ein Programm zu entwickeln, welches alle daran geknüpften Ziele erfüllt und in der Überprüfung die mathematischen Tests bestanden hat. In meinem Programm wird das Mitverstecken von zusätzlichen Informationen über die geheime Nachricht mittels meines Bit-Format (vgl. 3.1) vorgenommen. Dies vereinfacht die Programmnutzung erheblich gegenüber von anderen Steganographie-Tools.

Der Prozess des Programmierens war mit einigen Hürden verbunden. Da mein Programm auf Benutzerfreundlichkeit festgelegt ist, musste ich in jedem Schritt meines Programms überprüfen, ob die Eingaben der Nutzer korrekt sind. Ein Grossteil der Entwicklung war darauf fokussiert, ein robustes `Error-Handling` System zu schaffen, was mir schlussendlich auch gelungen ist.

Auch das Auswerten des Programms war nicht ohne Probleme. Weil ich am Anfang als Testbilder ausschliesslich hochauflösende Bilder und mittelgrosse Dateien verwendete, waren meine Resultate in den meisten Fällen besser als die Realität. Dies deshalb, weil nur ein minimaler Teil manipuliert wurde (vgl. 5.2). Das Problem konnte gelöst werden, in dem ich grössere geheimen Nachrichten und kleinere Cover-Bilder verwendete.

Meine Arbeit hat mir aber auch gezeigt, dass sich das Feld der Steganographie stark wegentwickelt hat von der einfachen LSB-Steganographie. Die Grundprinzipien der Steganographie, die ich in dieser Arbeit besprochen habe, sind jedoch auf viele der heutigen steganographischen Methoden universell anwendbar.

# Literaturverzeichnis

[1] S. A. Jebur, A. K. Nawar, L. E. Kadhim, und M. M. Jahefer, „Hiding Information in Digital Images Using LSB Steganography Technique“, *International Journal of Interactive Mobile Technologies (iJIM)*, Bd. 17, Nr. 07, Art. Nr. 07, Apr. 2023, doi: 10.3991/ijim.v17i07.38737.

[2] National Security Agency, *NCM: Introduction to Steganography*, (23. Februar 2022). Zugegriffen: 21. September 2024. [Online Video]. Verfügbar unter: https://www.youtube.com/watch?v=GPdIY6ObKJU

[3] University of Nottingham. *Secrets Hidden in Images (Steganography) - Computerphile*, (4. August 2015). Zugegriffen: 4. September 2024. [Online Video]. Verfügbar unter: https://www.youtube.com/watch?v=TWEXCYQKyDc

[4] „Mask Letter“, UM Clements Library. Zugegriffen: 2. Oktober 2024. [Online]. Verfügbar unter: https://clements.umich.edu/exhibit/spy-letters-of-the-american-revolution/secret-methods/mask-letter/

[5] G. K. Wallace, „The JPEG still picture compression standard“, *IEEE Trans. Consumer Electron.*, Bd. 38, Nr. 1, S. xviii–xxxiv, Feb. 1992, doi: 10.1109/30.125072.

[6] *Information technology – Computer graphics and image processing – Portable Network Graphics (PNG): Functional specification*. in International Standard ISO/IEC ISO 15948. Geneva: ISO/IEC, 2004.

[7] „IBM Integration Bus for z/OS 10.1“. Zugegriffen: 22. August 2024. [Online]. Verfügbar unter: https://www.ibm.com/docs/en/integration-bus/10.1?topic=msmc-message-sets-specifying-data-element-separation-methods-model-message

[8] National Institute of Standards and Technology (US), „Advanced Encryption Standard (AES)“, National Institute of Standards and Technology (U.S.), Washington, D.C., NIST FIPS 197-upd1, Mai 2023. doi: 10.6028/NIST.FIPS.197-upd1.

[9] „Symmetric Key Encryption - why, where and how it’s used in banking“. Zugegriffen: 18. September 2024. [Online]. Verfügbar unter: https://www.cryptomathic.com/news-events/blog/symmetric-key-encryption-why-where-and-how-its-used-in-banking

[10] Q. H. Dang, „Secure Hash Standard“, National Institute of Standards and Technology, NIST FIPS 180-4, Juli 2015. doi: 10.6028/NIST.FIPS.180-4.

[11] „Python Pillow“. Zugegriffen: 29. August 2024. [Online]. Verfügbar unter: https://python-pillow.org/

[12] „Pillow 10.4.0 documentation“, Pillow (PIL Fork). Zugegriffen: 29. August 2024. [Online]. Verfügbar unter: https://pillow.readthedocs.io/en/stable/index.html

[13] „Peak signal-to-noise ratio“, *Wikipedia*. 2. Juli 2024. Zugegriffen: 28. August 2024. [Online]. Verfügbar unter: https://en.wikipedia.org/w/index.php?title=Peak\_signal-to-noise\_ratio&oldid=1232208089

[14] K.-H. Jung, „Comparative Histogram Analysis of LSB-based Image Steganography“, *WSEAS Transactions on Systems and Control*, Bd. 13, S. 103–112, Jan. 2018.

[15] S. N. M. Al‐Faydi, S. K. Ahmed, und H. N. Y. Al‐Talb, „Improved LSB image steganography with high imperceptibility based on cover‐stego matching“, *IET Image Processing*, Bd. 17, Nr. 7, S. 2072–2082, Mai 2023, doi: 10.1049/ipr2.12773.

# Abbildungsverzeichnis

[Abbildung 1: Verstecke Nachricht in einem Brief, mit Hilfe einer Schablone [4] 7](#_Toc178934097)

[Abbildung 2: Hervorhebung der LSB bei 4 Bytes (in Anlehnung an [1]) 8](#_Toc178934098)

[Abbildung 3: LSB-Substitution bei einem RGB Pixel (in Anlehnung an [1]) 8](#_Toc178934099)

[Abbildung 4: Visuelle Darstellung des Bit-Format 12](#_Toc178934100)

[Abbildung 5: Programmablauf beim Embeding 14](#_Toc178934101)

[Abbildung 6: Programmablauf beim Extracting 14](#_Toc178934102)

[Abbildung 7: Manipulierter Bereich eines Cover-Bild bei grösser werdenden Nachrichten. 15](#_Toc178934103)

[Abbildung 8: PSNR abhängig von der Anzahl modifizierter LSB 19](#_Toc178934104)

[Abbildung 9: Farbfehler bei fünf LSB (Ausschnitt brille.png) 20](#_Toc178934105)

[Abbildung 10: Farbfehler bei fünf LSB (Ausschnitt auto.png) 20](#_Toc178934106)

[Abbildung 11: Wasser.png mit 8 modifizierten LSB 21](#_Toc178934107)

[Abbildung 12: Lorem\_50k versteckt innerhalb von wasser.png 3,6 und 8 modifizierter LSB 22](#_Toc178934108)

[Abbildung 13: Pixel Subtraktion des Stego-Bild einer Blume und dem original Cover-Bild 25](#_Toc178934109)

# Tabellenverzeichnis

[Tabelle 1: Bilder für die Auswertung des Programms 18](#_Toc178933591)

[Tabelle 2: Lorem-Ipsum Testdateien für die Auswertung des Programms 18](#_Toc178933592)

[Tabelle 3: PSNR-Wert und Anzahl an LSB bei lorem\_50k versteckt in wasser.png 23](#_Toc178933593)

# Danksagung

Ich bedanke mich bei allen Personen, die mir beim Erschaffen dieser Maturaarbeit geholfen haben.

Als erstes möchte ich mich bei Herr Daniel Zurmühle bedanken. Er hat meine Maturaarbeit betreut und mir bei der Überarbeitung der Arbeit geholfen. Um die hilfreichen Feedbacks zur Ideeneingrenzung, um die konstruktiven Rückmeldungen bei der Überarbeitung und um seine Betreuung war ich sehr froh.

Ein weiterer Dank geht an Herr Christoph Schaufelberger, der sich während des Schreibprozesses Zeit genommen hat, mit mir Teile der Arbeit zu überarbeiten. Er gab mir Feedbacks zu meinem Schreibstil und zeigte mir Verbesserungsmöglichkeiten auf.

Auch möchte ich mich bei Herr Timmy Liniger bedanken, der mir am Anfang meiner Arbeit geholfen hat, mich für ein Thema zu entscheiden.

Zum Schluss danke ich auch meinen Eltern, für das Gegenlesen meiner Arbeit und die konstante Unterstützung im Verlauf der ganzen Maturaarbeit.

# Anhang

1. Programm  
   Mein Programm, welches ich im Ramen dieser Arbeit geschrieben habe, ist öffentlich zugänglich auf GitHub: <https://github.com/Samhuw8a/Steganography>.
2. Auswertungsdaten  
   Die Resultate der Auswertung so wie auch alle Scripts, Abbildungen, Bilder sind zugänglich auf Github: <https://github.com/Samhuw8a/Auswertung-Steganography>
3. Anforderungsanalyse  
   TODO

# Deklaration

a) «Ich erkläre hiermit,

* dass ich die vorliegende Arbeit selbständig und unter Benutzung der angegebenen Quellen verfasst habe,
* dass ich eine allfällige Nutzung von Künstlicher Intelligenz (z.B. ChatGPT) in der Arbeit ausgewiesen habe,
* dass ich auf eine eventuelle Mithilfe Dritter in der Arbeit ausdrücklich hinweise,
* dass ich vorgängig die Schulleitung und die betreuenden Lehrpersonen informiere, wenn ich diese Maturaarbeit bzw. Teile oder Zusammenfassungen davon veröffentlichen werde und/oder Kopien dieser Arbeit zur weiteren Verbreitung an Dritte aushändigen werde,
* dass mir die möglichen Folgen eines Plagiats bekannt sind (vgl. Weisung Unredlichkeiten, SRL 506).»

b) Einverständnis zur Plagiatsprüfung

Ich nehme zur Kenntnis, dass meine Arbeit zur Überprüfung der korrekten und vollständigen Angabe der Quellen mit Hilfe einer Software (Plagiatserkennungstool) geprüft werden kann. Zu meinem eigenen Schutz wird die Software auch dazu verwendet, später eingereichte Arbeiten mit meiner Arbeit elektronisch zu vergleichen und damit Abschriften und eine Verletzung meines Urheberrechts zu verhindern. Falls Verdacht besteht, dass mein Urheberrecht verletzt wurde, erkläre ich mich damit einverstanden, dass die Schulleitung meine Arbeit zu Prüfzwecken herausgibt.

Unterschrift für a) und b):

Malters, 4. Oktober 2024

Samuel Huwiler