SMK IT-Sicherheit - StegoDetect: Steganographie und verdeckte Kommunikation

Bernhard Birnbaum

Zusammenfassung—Im Zentrum dieses Projekts steht die Fragestellung, inwiefern steganographische Inhalte in JPEG-Bildern detektiert und attributiert werden können. Weiterhin soll daraus geschlossen werden, wie sich Variationen der Einbettungsparameter auf die Attributierung auswirken.

Index Terms—IT-Security, Steganography, JPEG Format, Open Source

1 Motivation

1.1 Was ist Steganographie?

Die Steganographie bezeichnet Verfahren zum Verstecken von Informationen, um Nachrichten verdeckt übermitteln zu können (Folie 2 in [1]). Dabei stehen Vertraulichkeit und Geheimhaltung der Datenübertragung im Vordergrund. Für eine solche Kommunikation wird zwingend ein Trägermedium benötigt; diese Art von Daten werden als „Cover-Daten“ bezeichnet (Folie 6 in [2]). Ein Dritter sollte so im Idealfall beim Betrachten des Cover-Datums nichts von der Manipulation mitbekommen, sodass die Kommunikation über den vom Trägermedium gebildeten Kanal unbemerkt erfolgen kann (Folie 15 in [2]).
Allerdings bieten steganographische Verfahren auch Missbrauchsmöglichkeiten, wodurch sie von Angreifern ausgenutzt werden können.

1.2 Risiko und Sicherheitsaspekte

Das Risiko hängt von der Bedrohung und der Schwachstelle ab (Folie 24 in [3]). Das Eintrittspotential ist dabei als hoch bis sehr hoch einzustufen [4], da Steganographie bereits aktiv in Exploits ausgenutzt wird [5]. Das Schadenspotential ist abhängig von den steganographisch eingebetteten Daten. Im Worst Case könnte ein Angreifer mit einer per Steganographie verdeckt übertragenen Schadware Kontrolle über das System erlangen, was ein sehr hohes Schadenspotential impliziert [4].
Das entstehende Risiko ist demzufolge sehr hoch [4]. Es werden die Basisangriffe des Erzeugens/Fälschens bzw. Veränderns von Daten genutzt, was wiederum die IT-Sicherheitsaspekte der Integrität, der Vertraulichkeit sowie der Authentizität verletzt (Folie 20-21 in [3]).

1.3 Gegenmaßnahmen

Zur Prävention solcher Angriffe ist neben der Detektion der steganographischen Einbettung auch deren Attributierung wichtig, um Auskunft über Art und Weise der Einbettung zu erhalten und geeignete Gegenmaßnahmen entwickeln zu können.

2 Stand der Technik

Es gibt drei Möglichkeiten, Steganographie zu praktizieren (Folie 24 in [1]):

1) **Steganographie per Cover-Auswahl**: der Sender wählt ein Cover aus einer großen Menge aus, sodass die Nachricht kommuniziert wird
2) **Steganographie per Cover-Synthese**: der Sender erstellt ein Cover, welches die Nachricht kommuniziert
3) **Steganographie per Cover-Modifikation**: der Sender modifiziert ein existierendes Cover, um die Nachricht einzubetten

Diese Untersuchung beschäftigt sich mit der Attributierung von Steganographie per Cover-Modifikation in JPEG-Bildern. Dabei wurde in der Aufgabenstellung (Folie 37-40 in [6]) bereits die Auswahl an zu betrachtenden Steganographie-Tools vorgegeben als die Werkzeuge aus dem Open-Source-Projekt „*Steganography Toolkit*“ [7].

2.1 Detektion

Aufgrund der Definition von Steganographie haben die Entwickler von Steganographie-Tools, d.h. von Programmen, die Einbettungen in Cover-Daten vornehmen, ein Interesse daran, dass die Einbettungsmechanismen nicht analysiert werden, um die Detektion und dementsprechend ebenfalls die Attributierung zu verhindern. Deswegen können bei dieser Untersuchung lediglich ältere, bereits bekannte Stego-Verfahren betrachtet werden.
Genau für diesen Zweck bietet das *Stego-Toolkit* [7] eine Sammlung von mehreren bekannten Stego-Tools und (Stego-)Analyse-Tools. Zu einigen der in dieser Kollektion beinhalteten Tools gibt es bereits Detektionsansätze:

2.1.1 Detektion von jphide, jsteg und outguess-0.13

Auf Basis bereits vorangegangener Forschung können steganographische Manipulationen in JPEG-Bildern durch eine statistische Analyse erkannt werden, insofern die verwendeten Verfahren die Einbettung in den LSBs der DCT (discrete cosine transform) vornehmen (Abs. 4 „Statistical Analysis“ in [8]).

Im Falle von *jphide* (Fig. 6 in [8]) und *jsteg* (Fig. 5 in [8]) findet die Einbettung von Beginn des Bildes an sukzessive statt.
Das Tool *outguess-0.13* hingegen wählt die zu ändernden Koeffizienten pseudo-zufällig aus, wodurch sich keine klare Signatur ergibt (Fig. 7 in [8]).

2.1.2 Detektion von outguess-0.2, f5 und steghide

Im Vergleich zu *outguess-0.13* ist *outguess-0.2* (*outguess*) nicht durch statistische Analyse detektierbar, da in dieser Version entsprechende Bildeigenschaften erhalten werden, sodass nicht mehr auf eine Einbettung geschlossen werden kann (Abs. 5.3 „Outguess“ in [8]).
Ähnlich verhält es sich mit *f5*, welches ebenfalls Maßnahmen zur Erschwerung einer statistischen Analyse ergreift [9].
Für *steghide* gibt es Möglichkeiten, die DCT-Koeffizienten zu extrahieren und dadurch an die Einbettungsinhalte zu kommen, dies umfasst allerdings keine generelle Detektion von Einbettungen [9].

3 KONZEPT

3.1 Vorüberlegungen

3.1.1 Einordnung in forensische Methodik

Die Konzeptionierung der Untersuchung ist am forensischen Daten- und Vorgehensmodell [10] angelehnt; die angewandte Methodik orientiert sich an DPE („Data processing and evaluation“ (Folie 61 in [3])). Dabei wird die Untersuchung durch Methoden zur Datenverarbeitung, -dokumentation und -visualisierung vorangetrieben.
Für diese Arbeit sind zwei der im IT-Forensik-Leitfaden beschriebenen Datenarten (Folie 60 in [3] bzw. [11]) von Interesse, die ausschließlich den Massenspeicher DS_T (Folie 61 in [3]) betreffen:

1) DT_3 **Details über Daten**: Metadaten zu den Nutzdaten
2) DT_8 **Anwenderdaten**: vom Nutzer bearbeitete oder konsumierte Inhalte - Medien-Daten, z.B. Bilder

3.1.2 Coverbild-Testset

Für eine statistisch signifikante Untersuchung müssen die Testdaten einen ausreichenden Umfang sowie Diversität aufweisen. Bilder in öffentlichen Steganographie-Testdatenbanken wie „kaggle/alaska2“ [12] oder „bows2“ [13] weisen oft ähnliche bis identische Metadaten auf (z.B. Auflösung und Farbkanäle), wodurch einige Attribute (wie z.B. die verwendete Aufnahmekamera) nicht existieren und dementsprechend nicht betrachtet werden können oder die Attributierung sich an falschen Merkmalen orientiert. Deshalb wurde das Coverbild-Testset [14] aus mehrere Bildquellen zusammengestellt (siehe Tabelle 1).
Die Umsetzung des Testsets sowie die konkrete Bildauswahl werden später in Abschnitt 4.1.1 beschrieben.

Tabelle 1
Coverbild-Testset-Zusammensetzung

Quelle	Anzahl	Charakteristik
alaska2	640	Farbbilder, 512x512
bows2	192	Schwarz-Weiß-Bilder, 512x512
privat	187	Smartphone-Kamera-Bilder, WhatsApp-komprimierte Bilder
privat	5	hochauflösende Bilder

3.1.3 DCT-Änderung durch JPEG-Kompression

Da sowohl Stego-Verfahren als auch normale JPEG-Kompressionen die DCT-Koeffizienten des Cover-Bildes abändern (Folie 35 in [1]), müssen beide Einflüsse voneinander differenziert werden. Um diesen Effekt untersuchen zu können, sollte das komplette Testset neu komprimiert und anschließend eine Differenzbildbetrachtung zwischen den JPEG-Neukompressionen und den Originalbildern angefertigt werden. So wird der Einfluss einer normalen JPEG-Kompression auf das Differenzbild sichtbar.

3.2 Werkzeugauswahl

3.2.1 Stego-Toolkit

Als Ausgangspunkt für die Untersuchung sollte das Steganographie-Toolkit „*Stego-Toolkit*“ [7] verwendet werden, was eine Sammlung von bekannten Steganographie-Tools (Stego-Tools) und (Stego-)Analyse-Werkzeugen (Steganalysis-Tools) bereitstellt. Einige Programme sind dabei allerdings nur unvollständig vorhanden oder nicht (mehr) lauffähig, sodass die Werkzeugauswahl abgeändert werden musste.
Trotz dass das *Stego-Toolkit* Werkzeuge für verschiedene Dateiformate bietet (jpg, png, bmp, mp3, wav), wurde sich in dieser Arbeit auf Bilder, insbesondere auf das JPEG-Format, beschränkt.

3.2.2 Steganographie-Tools

Die anfängliche Werkzeugauswahl bestand aus einer Teilmenge an Stego-Tools des *Stego-Toolkits*, die mit JPEG-Dateien arbeiten können: *jphide*, *jsteg*, *outguess-0.13*, *outguess*, *steghide*, *f5*.
Allerdings ließ sich auch nach mehreren Versuchen keine Einbettung oder Extraktion mit *jphide* bzw. *jpseek* durchführen, vermutlich aufgrund veralteter Software (siehe Abs. 4.5.3). Die beiden Programme wurden daraufhin für die Untersuchung abgeschrieben.
Daraus ergibt sich die finale im Testprotokoll verwendete Stego-Tool-Auswahl (Tabelle 3).

Tabelle 3
Steganographie-Tools mit Einbettungsvariationenanzahl

Tool	Schlüssel-variationen	Daten-variationen	Einbettungen
jsteg	1	5	**5**
outguess	3	5	**15**
outguess-0.13	3	5	**15**
steghide	2	5	**10**
f5 (≤ 1024)	3	4	(**12**)

3.2.3 (Stego-)Analyse-Tools

Im *Stego-Toolkit* wird zwischen „general screening tools“ und „detecting tools“ unterschieden [7]. Für die eigentliche Attributierung ist diese Einteilung allerdings nicht relevant, weshalb für die Untersuchung diese Tools als „(Stego-)-Analyse-Tools“ zusammengefasst werden.
Wie auch bei der Stego-Tool-Auswahl wurden zunächst alle Tools des *Stego-Toolkits* ausgewählt, die auf JPEG-Bilder anwendbar sind: *file*, *exiftool*, *binwalk*, *strings*, *foremost*, *imagemagick* (identify), *stegoveritas*, *stegdetect*, *stegbreak*.
Das Programm *stegbreak* war in der Form so wie es im *Stego-Toolkit* vorhanden ist, zunächst nicht ausführbar. Wie im Abschnitt 4.5.4 beschrieben liefert das Tool aber nun in ca. 25% der Fälle ein sinnvolles Ergebnis, wodurch es jetzt für die Attributierung als ergänzendes Werkzeug genutzt werden kann.
Das Tool *file* liefert keine ergänzenden Informationen (lediglich Dateiformat und JFIF-Version bei JPEG-Dateien) zu denen von *exiftool* und *binwalk*, ist somit redundant und wird deshalb nicht weiter betrachtet.
imagemagick besteht aus mehreren Teilmodulen [15]. Im Stego-Toolkit ist lediglich das Modul *identify* zum Auslesen von Metainformationen aus Bildern vorhanden. Zusätzlich wird allerdings das Modul *compare* zur Differenzbilderstellung benötigt [16], weshalb *imagemagick* neu und vollständig in der Umgebung installiert werden musste.
Alle final verwendeten Analyse-Tools sind abschließend in Tabelle 2 gelistet.

Tabelle 2
(Stego-)Analyse-Tools mit potentiellen Attributen und Ausgabeformat

Tool	potentielle Attributierungsmerkmale	Ausgabeformat
statistische Merkmale		
sha1sum	Validierung erfolgreicher Stego-Einbettungen	Text
exiftool	Dateigröße, Aufnahme-Kamera, MIME-Type, JFIF-Version, Encoding, Anzahl Farbkomponenten, Auflösung, Megapixel	Text
binwalk	Dateityp, JFIF-Version	Text
strings	Datei-Header	Text
foremost	Bildintegrität durch Datenextraktion, Dateigröße	Dateistruktur
inhaltsbasierte Merkmale		
identify	Dateiformat, Farbminima, Farbmaxima, Farb-Mittelwert, Farb-Standardabweichng, Kurtosis, Skewness, Entropie	Text
compare	Differenzbilderstellung	Dateistruktur
stegoveritas	RGB-Zerlegungen, Kanten, Gaußscher Blur, Invertierung, Minima, Maxima, Mittelwert, scharfgezeichnet, weichgezeichnet	Dateistruktur
Stego-Detektionstools		
stegdetect	Detektion von jsteg, jphide, outguess-0.13 (Abs. 6.1. in [8])	Text
stegbreak	Detektion von jsteg, jphide, outguess-0.13 [7]	Text

3.3 Testprotokoll

3.3.1 Einbettungsvariationen

Um den Einfluss von Einbettungsparametern auf die Attributierung von Steganographie untersuchen zu können, werden die in Tabelle 4 dargestellten Variationen von Einbettungsschlüssel und Einbettungsdaten betrachtet:

Tabelle 4
Testprotokoll-Variationen

Einbettungs-schlüssel (12-14 in [17])	**Einbettungsdaten** [18]	**nicht unterstützte Tools**
-	67 Bytes ASCII	steghide
-	1.53 KB ASCII	steghide
-	17.5 KB ASCII	steghide
-	16 KB ASCII, min. Entropie	steghide
-	16.8 KB Binärdaten	steghide, f5
4 Bytes	67 Bytes ASCII	jsteg
4 Bytes	1.53 KB ASCII	jsteg
4 Bytes	17.5 KB ASCII	jsteg
4 Bytes	16 KB ASCII, min. Entropie	jsteg
4 Bytes	16.8 KB Binärdaten	jsteg, f5
50 Bytes	67 Bytes ASCII	jsteg
50 Bytes	1.53 KB ASCII	jsteg
50 Bytes	17.5 KB ASCII	jsteg
50 Bytes	16 KB ASCII, min. Entropie	jsteg
50 Bytes	16.8 KB Binärdaten	jsteg, f5

Die ASCII-Daten sind dabei zum Testen der Einbettungskapazität der einzelnen Tools in der Länge variiert. Zusätzlich werden ASCII-Daten mit minimaler Entropie verwendet, um eine mögliche Datenkompression vor der Einbettung zu erkennen. Als Gegentest werden zusätzlich Binärdaten verwendet, um ASCII-verschiedene Daten ebenfalls betrachten zu können.
Wie bereits Tabelle 4 aus der Spalte „nicht unterstützte Tools“ zu entnehmen ist, ist die Möglichkeit der Einbettungsparametervariation bei jedem Steganographie-Tool verschieden:

- **steghide**: forciert einen Einbettungsschlüssel
- **jsteg**: unterstützt keine Einbettungsschlüssel
- **f5**: unterstützt keine Binärdateneinbettung, da die Einbettungsdaten als Kommandozeilenargument übergeben werden müssen
- **outguess/outguess-0.13**: lediglich *outguess* bzw. *outguess-0.13* unterstützen theoretisch alle im Testprotokoll geforderten Parametervariationen

Die Anzahl der jeweils unterstützten Schlüssel- und Datenvariationen sowie die daraus resultierende Gesamtanzahl an Einbettungen pro Tool sind in Tabelle 3 dargestellt.
Konkret bedeutet das 57 Stego-Einbettungen pro Cover, bzw. 45 Einbettungen bei Bildern, die in mindestens einer Dimension 1024 Pixel überschreiten. Dies hat technische Gründe, die später in Abschnitt 4.2.2 erläutert werden.

3.3.2 Ablauf der Untersuchung

Ziel der Untersuchung ist die Datenakquise, um basierend darauf Hinweise auf mögliche Attribute zur Bestimmung einer steganographischen Einbettung auszuarbeiten. Der hier beschriebene Ablauf ist außerdem im Ablaufdiagramm A.1 dargestellt.

1) **Qualitätssicherungsmaßnahmen**: Bevor mit dem eigentlichen Testprotokoll begonnen werden kann, muss die Umgebung geprüft werden, um größere Fehler während der Untersuchung zu vermeiden.
   Dieser Schritt umfasst die Prüfung der virtuellen Umgebung (Docker), der angegebenen Kommandozeilen-Parameter, der verfügbaren Tools sowie der Cover- und Einbettungsdaten.
2) **Einbettungsphase**: In der Einbettungsphase werden die Stego-Bilder erzeugt. Jedes Stego-Tool führt dabei die in Tabelle 3 berechnete Anzahl an Einbettungen bzw. Extraktionen aus. Die Extraktion der zuvor eingebetteten Daten ist ein wichtiger Beitrag zur Qualitätssicherung, da so festgestellt werden kann, ob die Einbettung erfolgreich war und ob die Daten wiederhergestellt werden konnten.
   Im Falle von *jsteg*, *outguess* und *outguess-0.13* wird anschließend das Stego-Analyse-Tool *stegbreak* ausgeführt, da es von sich behauptet, die zuvor genannten Stego-Tools erkennen zu können [7]. Bei *f5* ist die Einbettung zusätzlich von der Bildauflösung abhängig (siehe Abs. 4.2.2).
3) **Stego-Analyse**:
   a) **Screening-Phase**: In der Screening-Phase werden alle (Stego-)Analyse-Tools aufgerufen und deren erzeugte Ausgabe zwischengespeichert. In dieser Phase werden die Daten erzeugt, die später für die Analyse benötigt werden.
      Abgesehen von *stegoveritas* und *foremost*, die eine Dateistruktur erzeugen, liefern alle Tools eine Text-Ausgabe (siehe Abs. 4.2.3.1).
   b) **Parsing-Phase**: Aus den zuvor generierten Daten werden nun die Attribute, wie sie in Tabelle 2 gelistet sind, mit verschiedenen Parsing-Mechanismen ausgelesen und mit Hilfe einer Tabelle im CSV-Format aufbereitet (siehe Abs. 4.2.3.2).
   c) Wiederholung für nächste Einbettung ab Schritt **a)**
4) **Evaluation**: Die Evaluation eines Covers erfolgt nach dem Untersuchen aller Stego-Einbettungen der verschiedenen Tools. Dabei sollen die einzelnen Tools gezielt - u.a. durch Berechnen von Durchschnittswerten und Differenzen - verglichen werden. Diese Daten werden ebenfalls als CSV-Tabelle gespeichert.
5) Wiederholung für nächstes Cover ab Schritt **2)**
6) **Detailanalyse**: Nach der Datenakquise müssen die Daten detailliert betrachtet und ausgewertet bzw. visualisiert werden, um Erkenntnisse über Attributierungsmöglichkeiten zu erlangen.

4 IMPLEMENTIERUNG

4.1 Aufsetzen der Umgebung

4.1.1 Coverbild-Testset

Die Bilder in der BOWS2-Datenbank [13] liegen im PGM-Format (Portable GrayMap) vor. Deshalb mussten die Dateien konvertiert werden. Dies erfolgte mit dem *imagemagick*-Modul „convert“ wie folgt:

```
for img in *.pgm; do convert $img $(basename $img .pgm).jpg; done
```

Die Dateien für das finale Testset wurden anschließend aus den Cover-Quellen per Zufall (inhaltsunabhängig) ausgewählt. Die Zufallsauswahl wurde dabei mit folgendem Befehl umgesetzt:

```
find -type f -name '*.jpg' | sort -R
```

Die finale Zusammenstellung der hier verwendeten Testdaten [14] wurde vom Script [19] durchgeführt.

4.1.2 Vorbereiten der Docker-Umgebung

Um das Aufsetzen der Umgebung zu vereinfachen, sind die in folgenden Punkten 1 bis 4 beschriebenen Schritte und Befehle bereits in einem Shell-Script „stego-docker.sh“ [20] implementiert, welches sich auf die Verwaltung der Docker-Umgebung konzentriert.

1) **Docker-Installation**: `./stego-docker.sh -s`
   Da das *Stego-Toolkit* [7] als Docker-Image bereitgestellt wird, muss Docker installiert werden. Auf Ubuntu-basierten Linux-Distributionen kann Docker mit `apt` installiert werden:
   `sudo apt install docker.io -y`
   Danach wird Docker als Service im System registriert:
   `sudo systemctl enable --now docker`
   Abschließend wird der Nutzer der neuen Gruppe hinzugefügt: `sudo usermod -aG docker $USER`
   Jetzt ist Docker vollständig konfiguriert.
2) **Stego-Toolkit-Setup**: `./stego-docker.sh -p`
   Das *Stego-Toolkit* wird nun mit folgendem Befehl heruntergeladen:
   `docker pull dominicbreuker/stego-toolkit`
3) **Starten des Containers**: `./stego-docker.sh -r`
   Der Docker-Container kann jetzt gestartet werden, um anschließend die für die Untersuchung benötigten Daten kopieren zu können:
   `docker run -it --rm -v $(pwd)/data:/data dominicbreuker/stego-toolkit /bin/bash`
   Dieser Befehl öffnet eine Shell im Container.
4) **Kopieren von Daten**: Für die in diesem Projekt durchgeführte Untersuchung müssen mehrere verschiedene Daten in den Container kopiert werden. Dies umfasst neben dem Coverbild-Testset [14] einige Utility-Scripte [21], [22], das Attributierungs-Script [17], Tests [23], die Einbettungsdaten [18], sowieso die reparierte Version des Programms *stegbreak*, inklusive benötigter Konfigurationsdaten.
   Docker bietet für diesen Zweck folgenden Befehl:
   `docker cp <src> container:<dest>`
   Um auch hier das Setup so einfach wie möglich zu gestalten, können mit Script [24] alle zuvor genannten Daten mit einem Aufruf importiert werden:
   `./stego-docker-importDefaults.sh`

4.2 Umsetzung der Datenakquise/des Testprotokolls

Der Ablauf des im Diagramm A.1 konzeptionierten Testprotokolls ist als Shell-Script „stego-attrib.sh“ [17] implementiert. Dabei wurde sich an die in Punkt 3.3.2 beschriebenen Phasen gehalten:

4.2.1 Qualitätssicherungsmaßnahmen

Für eine Ausführung der Untersuchung ohne größere Fehler während der Laufzeit werden vor Beginn folgende Qualitätssicherungsaspekte geprüft:

1) **Docker-Umgebung**: Da das Script innerhalb des Docker-Containers ausgeführt werden soll, wo der `docker`-Befehl selbst nicht vorhanden sein sollte, kann dadurch geprüft werden, ob das Script auch wirklich in der virtuellen Umgebung läuft. Sollte der `docker`-Befehl verfügbar sein, läuft das Script sehr wahrscheinlich in der falschen Umgebung.
2) **Parameter-Prüfungen**: Einige Parameter erfordern die Übergabe von Ganzzahl-Werten. Dabei wird das korrekte Format mit einem regulären Ausdruck überprüft. Insofern Dateipfade übergeben werden, wird abhängig von der Semantik die Existenz des Pfades geprüft.
3) **Tool-Installation**: Wie bereits in Abschnitt 3.2 beschrieben, musste *imagemagick* nachinstalliert und *stegbreak* neu kompiliert werden (siehe Abs. 4.5.4). Vor der Ausführung wird deshalb kontrolliert, dass die Programme im Container vorhanden sind und die Konfigurationsdatei für *stegbreak* an der korrekten Stelle im System liegt.
4) **Daten**: Als letztes werden die vorhandenen JPEG-Dateien im angegebenen Testset (insofern es existiert) gezählt und mit der Anzahl der zu prüfenden Cover-Daten abgeglichen; außerdem wird das Vorhandensein der Einbettungdaten sichergestellt.

4.2.2 Einbettungsphase

Die Einbettungen werden wie in Tabelle 3 und 4 gezeigt durchgeführt. Nachdem eine temporäre CSV-Datei zum Mitschreiben der durchgeführten Einbettungen angelegt wurde, werden die Tools in folgender Reihenfolge ausgeführt:
jsteg, *outguess*, *outguess-0.13*, *steghide*, *f5*
Dabei ist wichtig zu erwähnen, dass *f5* nur ausgeführt wird, wenn die Auflösung des Cover-Bildes in beiden Dimensionen kleiner oder gleich 1024 Pixel ist, da bei größeren Bildern die Laufzeit des Tools massiv steigt (siehe Abs. 4.5.6).
Zusätzlich werden in dieser Phase auch Datenextraktionen der zuvor erstellten Einbettungen versucht. Im Anschluss werden die Prüfsummen der Einbettungsdaten in die CSV-Datei geschrieben, wodurch später festgestellt werden kann, ob die Einbettung erfolgreich war.
In diesem Zuge wird ebenfalls eine Extraktion über *stegbreak* für Einbettungen der Tools *jsteg* und *outguess-0.13* versucht. Da ab dem nächsten Schritt in der Implementierung nicht weiter zwischen den verschiedenen Stego-Tools differenziert wird, wird *stegbreak* bereits in der Einbettungsphase aufgerufen.

4.2.3 Stego-Analyse

Auf Basis der in der Einbettungsphase erstellten temporären CSV-Datei werden nun alle dort eingetragenen Einbettungen von den (Stego-)Analyse-Tools betrachtet. Dieser Vorgang lässt sich pro Einbettung in eine „Screening-Phase“ (ausführen der Tools und speichern deren Outputs) und eine „Parsing-Phase“ (auslesen der generierten Daten und als CSV aufbereiten) gliedern. Dabei werden nur Stego-Einbettungen betrachtet, die erfolgreich waren, d.h. wo die Stego-Datei nicht leer ist und die Daten erfolgreich extrahiert werden konnten.

4.2.3.1 Screening-Phase

Folgende (Stego-)Analyse-Tools werden in angegebener Reihenfolge ausgeführt:
file, *exiftool*, *binwalk*, *strings*, *foremost*, *imagemagick*, *stegoveritas*, *stegdetect*
foremost und *stegoveritas* erstellen jeweils eine Dateistruktur, um die Daten auszugeben, die Text-Ausgaben aller anderen Werkzeuge werden jeweils in eine Datei umgeleitet. Ähnlich wie beim Stego-Tool *f5* wurde die Ausführung von *stegoveritas* an eine Bedingung geknüpft, sodass die Bildzerlegungen nur für Bilder kleiner oder gleich 1024 Pixel pro Bilddimension angefertigt werden. Da für die von *stegoveritas* erzeugten Zerlegungen jeweils noch Differenzbilder erstellt werden, ist die Laufzeit sonst zu hoch (siehe Abs. 4.5.6).

4.2.3.2 Parsing-Phase

In diesem Schritt werden die in Tabelle 2 gelisteten potentiellen Attributierungsmerkmale für die spätere Auswertung aus den zuvor zwischengespeicherten Daten ausgelesen. Dafür wurden bei der Implementierung verschiedene Linux-Tools in Kombination je nach Bedarf verwendet:
find, *cut*, *grep*, *tr*, *xargs*, *head*, *tail*, *sed* und *awk*
Abschließend werden die Daten pro Cover-Bild als CSV-Tabelle gespeichert, wobei in jeder Zeile die Daten einer Einbettung stehen (`/*-csv/_*.csv` in [25]).

4.2.4 Evaluationsphase

In der Evaluationsphase wird eine Zusammenfassung des zuvor untersuchten Covers ermittelt. Dafür werden die untersuchten Einbettungen nach Stego-Tools gruppiert und Durchschnittswerte einiger Attribute gebildet, um bei der folgenden Auswertung relevante Attribute einfacher erkennen zu können.
Die Ergebnisse werden auch hier als Zeile in einer CSV-Datei gespeichert, sodass pro Cover-Bild eine Zeile geschrieben wird (`/*[^ -]*.csv` in [25]).

4.3 Umsetzung der Auswertung

Für die Auswertung der Untersuchung stehen alle CSV-Dateien in [25] zur Verfügung. In einem ersten Schritt wurden die Daten manuell gesichtet, um Auffälligkeiten zu identifizieren und interessante Merkmale von uninteressanten (Merkmale, die keine Veränderungen aufweisen, z.B. MIME-Type, Bildauflösung, Megapixel, ...; siehe Abs. 5.2.5) zu trennen. Anschließend wurden für einige ausgewählte Attribute Diagramme erstellt, um Auffälligkeiten sichtbar zu machen.

4.3.1 Manuelle Detailanalyse

Insgesamt wurden 6 Bilder genauer untersucht, die jeweils aus verschiedenen Bildklassen des Testsets stammen [26]. Dafür wurden die vom Script [17] generierten Daten in Excel geladen und anschließend relevante Attribute identifiziert und irrelevante ausgeblendet. Dabei sind mehrere annotierte Tabellen entstanden (`/detailedAnalysis.xlsx` in [27]).
Die gewonnenen Erkenntnisse für die Attributierung werden später in Abschnitt 5.2 beschrieben.

4.3.2 Erstellung der Diagramme

Um mögliche Korrelationen zwischen Einbettungsvariationen und Daten herzustellen, ist es hilfreich, diese zu visualisieren. Zur Verrechnung der pro untersuchtem Cover generierten CSV-Tabellen (`/*-csv/_*.csv` in [25]) wurden die Scripte [21] und [22] geschrieben.
In [21] werden insgesamt 6 Attribute in jeweils ein Diagramm zusammengerechnet: Differenzbild, Entropie, Dateigröße, RGB-Kanal-Differenzbilder sowie *stegdetect* und *stegbreak*-Ergebnisse. Script [22] prüft den Einfluss einer JPEG-Neukompression auf Farbkanal-Differenzbilder, um das in Abschnitt 3.1.3 beschriebene Phänomen zu evaluieren.
Die Scripte wurden jeweils für die verschiedenen Bildklassen aus Tabelle 1 ausgeführt, sodass die Ausgabe 20 CSV-Dateien sind (`/*-*.csv` in [25]). Darauf aufbauend konnten anschließend die Diagramme (Anhang C) mit Excel erstellt werden (`/valuesAndDiagrams.xlsx` in [27]).

4.4 Umsetzung der Attributierung

Als Grundlagen für die Umsetzung der Attributierung wurden die Ergebnisse aus Abschnitt 5.2 verwendet. Die Umsetzung erfolgte Dabei im Script „stego-attrib.sh“ (Funktion `jpg_examination` in [17]). Das Script kann eine Stego-Datei direkt überprüfen oder zusätzlich (insofern vorhanden) einen Vergleich mit dem Original-Bild vornehmen. Wenn das Original vorliegt ist die Attributierung eindeutiger möglich, da Bildmerkmale vergleichend berücksichtigt werden können.

Tabelle 5
auf Auswertung basierende, implementierte Attributierungsmerkmale

Merkmal	Attributierungs-Tool	Stego-Tool
direkte Attribute		
JFIF-Version	*exiftool*, *binwalk*	*jsteg*
Dateiformat	*binwalk*	*jsteg*
Header	*strings*	*jsteg*, *outguess*, *outguess-0.13*, *f5*
Datei-Integrität	*foremost*	*jsteg*
Detektion	*stegdetect*, *stegbreak*	*jsteg*, *outguess-0.13*
vergleichende Attribute		
Dateigröße	*exiftool*	*steghide*
Differenzbild	*imagemagick*	*steghide*
Farbkanal-Differenzbild	*stegoveritas*, *imagemagick*	*steghide*

Ähnlich wie bei der Testprotokoll-Umsetzung werden hier zunächst in der „Screening-Phase“ die (Stego-)Analyse-Tools ausgeführt (siehe Abs. 4.2.3.1) und anschließend die Zwischenergebnisse wie in der „Parsing-Phase“ ausgelesen (siehe Abs. 4.2.3.2).
Durch mehrere Abfragen werden zusätzlich die in Abschnitt 5.2 identifizierten und in Tabelle 5 gelisteten Merkmale überprüft. Die dadurch erzielten Ergebnisse werden in Abschnitt 5.4 erläutert.

4.5 Probleme bei der Umsetzung

4.5.1 Referenzen

Mehrere der bei der Aufgabenstellung angegebenen Referenzen wiesen beim näheren Betrachten Fehler auf.
Zum einen waren der Link auf die Stego-Bild-Datenbank „BOSS“ veraltet, was in einem 404-Fehler resultierte. Deshalb wurde als Alternative die kaggle/alaska2-Datenbank [12] verwendet.
Außerdem, wie bereits in Abschnitt 4.1.1 beschrieben, passte das Dateiformat der BOWS2-Datenbank [13] aus den Referenzen nicht zu den Werkzeugen des *Stego-Toolkits*, welches als Vorgabe verwendet werden sollte. Die Bilder wurden deshalb in das JPEG-Format umgewandelt.

4.5.2 Projekt-Umfang und Team-Organisation

Da diese Projektarbeit für 3-6 Personen ausgelegt ist, das ursprüngliche Team jedoch nur aus 2 Personen bestand, musste der eigentliche Umfang eingeschränkt werden. Als dann nach dem DR1-Meilenstein klar wurde, dass das Projekt aufgrund von einer zu einseitigen Kommunikation und Initiative nur noch von einer Person bearbeitet werden wird, mussten zusätzlich noch einmal die Aufgabenstellung in Bezug auf den Umfang und die Tiefe abgeändert werden.
Aus diesem Grund konnten viele Attribute von Interesse vor allem im späteren Teil der Arbeit nicht im ausreichenden Detail bearbeitet werden und müssen auf weiterführende Arbeiten geschoben werden (siehe Abs. 6.2.1).

4.5.3 jphide/jpseek

Zunächst bestand das Problem, dass *jphide* (und *jpseek*) keine automatisierte Einbettungsschlüsselübergabe im Programmaufruf zuließen. Daraufhin wurde diese Funkionalität nachträglich implementiert.
Nun tritt allerdings unter allen Umständen ein Segmentation Fault-Fehler (Shell-Fehlercode 139) auf, wenn eine Stego-Einbettung mit diesem Tool durchgeführt werden soll. Aufgrund dass dieser Fehler allerdings bereits vor den im Quelltext getätigten Änderungen auftritt, wurde *jphide* daraufhin für diese Untersuchung abgeschrieben, da vermutlich eine in diesem Tool verwendete Bibliothek mit modernen Prozessor-Architekturen nicht kompatibel ist, weil die im *Stego-Toolkit* verwendete *jphide*-Version [28] von 1999 ist.

4.5.4 stegbreak

Auch bei *stegbreak* gab es ähnlich wie bei *jphide* bei jeder Ausführung einen Segmentation Fault Fehler. Das Problem tritt konkret bei Verwendung von Ubuntu-Derivaten auf [29]. Durch eine Neukompilierung aus einer alternativen Quelle [30] mit einer zusätzlichen, im *Stego-Toolkit* fehlenden Konfigurationsdatei [31] konnte der Fehler so weit behoben werden, wie in Abbildung 1 zu sehen ist.

4.5.5 Speicherplatz

Für eine Untersuchung mit Detailanalyse wäre es von Vorteil gewesen, die bei der Datenakquise erzeugten Daten alle zu speichern und zu nutzen. Pro untersuchtem Bild liegt der minimale Speicherverbrauch allerdings bei 250 MB, was für das Testset der Größe 1024 mindestens 250 GB Speicherbedarf bedeuten würde. Deshalb wurden im Vorfeld einige wenige Attribute ausgewählt, die beim Parsing in CSV-Tabellen überschrieben werden, und die restlichen Daten anschließend zu löschen, um den Speicherbedarf so zu minimieren.

4.5.6 Ausführungszeiten

Da das Stego-Tool *f5* sowie das (Stego-)Analyse-Tool *stegoveritas* bei Bildern höherer Auflösungen eine Laufzeit im zweistelligen Minutenbereich pro Einbettung aufweisen, wurde die Ausführung dieser Tools abhängig von der Bildauflösung implementiert. Es hat sich ergeben, dass das Überspringen der Ausführung dieser Tools für Bilder, die in mindestens einer Bilddimension 1024 Pixel überschreiten, zu einer signifikanten Verbesserung der Performance beiträgt, sodass eine Cover-Bild-Analyse nun durchschnittlich 3-5 Minuten beansprucht, im Worst-Case bis zu 9 Minuten.
Die Ausführung der Untersuchung wurde über mehrere Tage hinweg mit verschiedenen Computern mit verschieden großen Teilmengen des Testsets durchgeführt. Im Nachhinein hätte eine von Anfang an auf Parallelisierung ausgerichtete Implementierung zu einer deutlich besseren Analyse-Performance geführt.

4.5.7 Automatisierte Inhaltsbetrachtung

Durch Betrachtung von Cover-Stego-Differenzbildern können Context-Sensitive-Einbettungen erkannt werden. Diesen Schritt zu automatisieren ist mit den gegebenen Werkzeugen allerdings kaum im nötigen Umfang möglich. Deshalb und aufgrund von zeitlicher Beschränkung wurde in dieser Arbeit nur sehr oberflächlich auf eine inhaltsbasierte Betrachtung eingegangen (siehe Abs. 6.2.5).

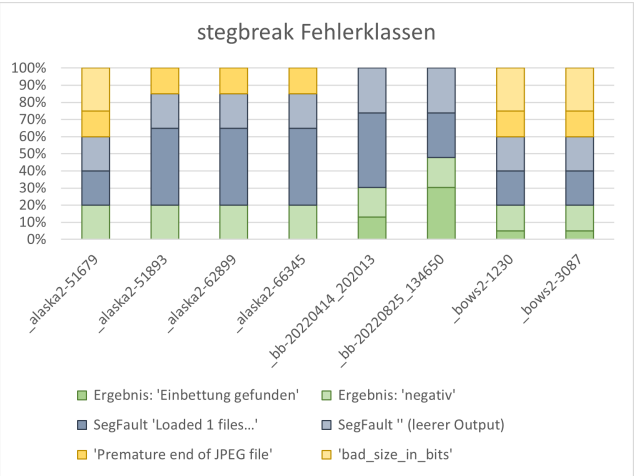


Abbildung 1. *stegbreak* Fehlerklassen der neukompilierten Version

5 EVALUIERUNG

5.1 Einbettungsgrenzen

Bei der Auswertung der erfolgreichen Einbettungen durch Abgleichen der mit *sha1sum* ermittelten Prüfsummen hat sich ergeben, dass nicht alle Einbettungsvariationen wie sie im Testprotokoll gefordert sind, tatsächlich bei allen Cover-Bildern möglich sind.

- **alaska2**- & **bows2**-Bilder:
  outguess und *outguess-0.13* konnten nur 2 von 5 Datenlängen erfolgreich einbetten, da die Einbettungskapazität der Bilder für diese Tools überschritten wurde.
  jsteg-Einbettungen führen bei 3 von 5 Datenlängen zu beschädigten Bilddateien, da die Einbettungskapazität der Bilder für dieses Tool überschritten wurde.
- **Smartphone**-Bilder:
  Da die Bilder in dieser Teilmenge eine höhere Auflösung haben, ist auch die Einbettungskapazität der Cover-Daten größer. Es konnten für alle geforderten Variationen Einbettungen durchgeführt werden.
- **Hochauflösende** Bilder: Wie auch bei den Smartphone-Bildern ist die Größe der Bilder dieser Teilmenge groß genug, um alle Testdaten erfolgreich einzubetten. *f5*-Einbettungen wurden wie in Abschnitt 4.5.6 begründet nicht betrachtet.

Es lässt sich festhalten, dass vor allem die Bilder aus den Stego-Datenbanken eine (zu) kleine Einbettungskapazität für einige Tools (durch die geringe Auflösung von 512x512) bieten, um komplexere Inhalte wie kompilierte Anwendungen im Bild zu verstecken. Die Kapazität reicht für einige hundert Worte aus, sodass trotzdem kurze Nachrichten oder Befehle eingebettet werden können.

5.2 Detailanalyse der Attributierungsmerkmale

Beim Auswerten potentieller Attributierungsmerkmale muss überprüft werden, ob Werte-Änderungen tatsächlich mit einer Stego-Einbettung in Verbindung stehen und kein Artefakt einer Bildkompression sind, da viele Stego-Tools beim Erstellen einer Einbettung das Bild neu komprimieren. Um den Einfluss einer JPEG-Kompression bzw. Stego-Einbettung auf ein Merkmal zu differenzieren, wurden die Attribute zusätzlich bei einer neu komprimierten Version des Covers ohne Stego-Einbettung betrachtet, die ebenfalls in den folgenden Diagrammen zu sehen sind. Das heißt, dass die Attribut-Werte der Stego-Bilder nicht nur mit dem Original, sondern auch mit einer JPEG-Kompression des betrachteten Covers verglichen werden müssen.

5.2.1 Entropie

Die Entropie des Bildes wurde als näher zu betrachtendes Attribut ausgewählt, da sie ein bekanntes Merkmal ist, welches sich durch Stego-Einbettungen in der Regel ändert. Zu erwarten ist, dass das Bildrauschen durch zunehmende Einbettungsdatenlängen ansteigt. Dieses Merkmal wurde dabei mit dem *imagemagick*-Modul *identify* ausgelesen. In den folgenden ausgewerteten Diagrammen wird die Entropie der verschiedenen Bildklassen in Abhängigkeit des verwendeten Stego-Tools, der Einbettungsdatenlänge sowie des Einbettungsschlüssels visualisiert.

Das Diagramm C.4 für die alaska2-Entropie hat viele Parallelen zum Diagramm C.5 für bows2-Bilder. Zunächst ist zu sehen, dass bei *jsteg*-Einbettungen die Entropie mit steigender Einbettungsdatenlänge sinkt, wobei sich Binärdaten als auch ASCII-Daten mit minimaler Entropie gleich verhalten. Ähnlich verhält es sich mit *outguess* und *outguess-0.13* bei Smartphone-Bildern und Bildern mit höheren Auflösungen wie in C.6 und C.7 erkennbar ist, wobei alle langen Einbettungsdaten eine gleichhohe Entropie aufweisen, sodass eine Daten-Kompression vor der Einbettung ausgeschlossen werden kann. Bei *steghide* in C.4 und C.6 ist außerdem zu erkennen, dass die Entropie bei langen Einbettungsschlüssellängen und allen Datenvariationen bis auf Binärdaten geringer ist als bei kurzem Einbettungsschlüssel. *steghide* und *f5* weisen in allen Bildklassen eine ähnlich geringe Abweichung von der Original- bzw. JPEG-Kompression-Entropie auf.
Abschließend kann festgehalten werden, dass eine **Abweichung der Entropie vom Original mit mehr als 1% auf eine Stego-Manipulation hinweist**, da bei einer JPEG-Neukompressionen die Abweichung nicht größer als 0.5% ist. Die Entropie steigt oder sinkt mit zunehmender Einbettungsdatenlänge, abhängig von Stego-Tool und Bildklasse. Signifikante, zur Attributierung nutzbare **Einflüsse des Einbettungsschlüssels konnten nur bei *steghide* bei alaska2- und Smartphone-Bildern** beobachtet werden.

5.2.2 Dateigröße

Die Dateigröße ist ein Merkmal, wo anzunehmen ist, dass durch zusätzliche eingebettete Daten die Dateigröße ansteigt; da allerdings die meisten Werkzeuge das Bild neu komprimieren, schwanken die Werte relativ stark. In den folgenden Diagrammen wird die Dateigröße ebenfalls in Abhängigkeit der Testprotokoll-Variationen betrachtet.
Die Dateigröße steigt zwar mit zunehmender Einbettungsdatenlänge bei *outguess* und *outguess-0.13* an bzw. sinkt bei *f5* (C.8 und C.9), allerdings sind die Veränderungen in einem Bereich, der praktisch nicht von einer JPEG-Kompression zu unterscheiden ist. Bis auf bei ***steghide*, wo die Dateigröße von Stego-Bildern sehr nah am Original ist, wird von allen Tools eine JPEG-Kompression durchgeführt**, wobei sich die Dateigröße mindestens halbiert. Zusätzlich ist zu erkennen, dass der relative Einfluss der Einbettungsdatenlänge auf die Stego-Dateigröße zumindest bei Smartphone-Bildern geringer ist als bei anderen Bildklassen (C.10). Bei den hochauflösenden Bildern ist hingegen eine relativ starke Abnahme der Dateigröße durch Kompression im Vergleich zu den anderen Bildklassen erkennbar, die Stego-Dateigrößen haben maximal 25% der Original-Bildgröße (C.11).
Dadurch, dass die Dateigrößen entweder **auf dem Niveau des Originalbildes oder dem einer JPEG-Kompression ist, bietet sie keine Anhaltspunkte als Attributierungsmerkmal**. Auch *f5* in Diagramm C.10 bildet keine Ausnahme, da hier aufgrund der im Testprotokoll (siehe Abs. 3.3.1) festgelegten Einschränkungen nur eine kleine Teilmenge der Bildklasse betrachtet werden konnte, was die geringen Stego-Dateigrößen im Vergleich zu den anderen Tools erklärt. Der Einfluss des Einbettungsschlüssels, der ausschließlich bei *steghide* in C.8 zu beobachten ist, kann

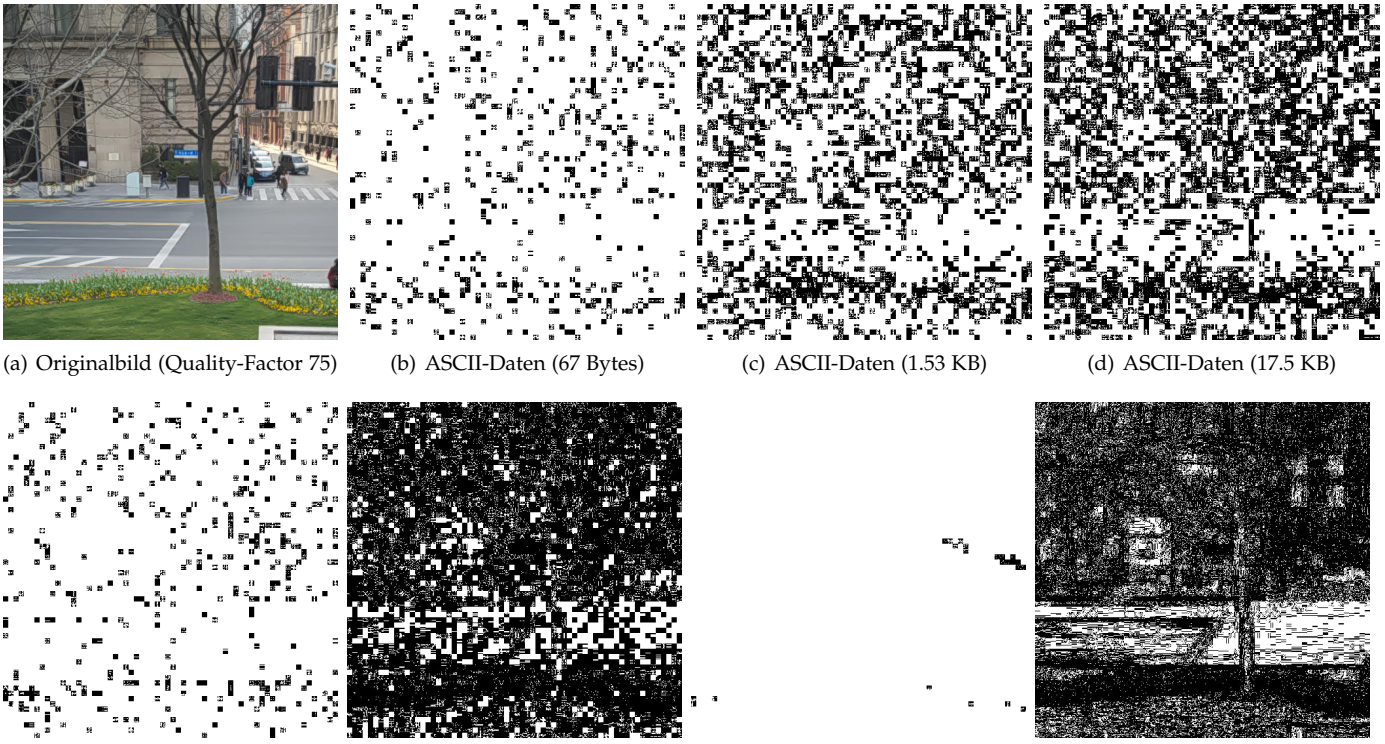
kaum zur Attributierung beitragen, da er sich auf diese Bildklasse beschränkt.

5.2.3 Differenzbild

Mit Hilfe des Differenzbildes zwischen einem Stego-Bild und dem Original können context-sensitive Einbettungen identifiziert werden. Alle Differenzbilder wurden mit dem *imagemagick*-Modul *compare* erstellt.
In Abbildung 2 wird exemplarisch der Einfluss von den verschiedenen Datenlängen auf eine *steghide*-Einbettung gezeigt. Zu erkennen ist, dass durch Zunahme der Datenlänge mehr Bildbereiche geändert werden (2b-d), wobei *steghide* Bildbereiche mit vielen Kanten bevorzugt. Interessant ist außerdem, dass das Differenzbild 2e ein ähnliches Bild wie 2b zeigt, obwohl sich die Einbettungsdatenlängen um den Faktor 240 unterscheiden. Ebenso kann man mehr Änderungen in Abbildung 2f als in 2d sehen, obwohl sich die absoluten Datenlängen diametral verhalten, was auf eine Datenkomprimierung vor der Einbettung hindeutet, wobei sich Binärdaten aufgrund einer hohen Entropie weniger effizient komprimieren lassen als ASCII-Texte. Die durch eine JPEG-Komprimierung hinterlassenen Artefakte im Differenzbild sind in Abbildung 2g-h für verschiedene Qualitätsfaktoren dargestellt. Änderungen werden im Differenzbild mit schwarz (0) dargestellt, unangetastete Bereiche sind weiß (1).
Da das automatisierte Identifizieren solcher Einbettungsstrategien auch mit den im *Stego-Toolkit* bereitgestellten Werkzeugen nicht ohne größeren Aufwand möglich ist, wurde sich bei der Auswertung dieses Attributs auf das Verhältnis zwischen schwarz (0) und weiß (1) im Differenzbild beschränkt.
Die meisten Tools (*jsteg*, *outguess*, *outguess-0.13* und *f5*) haben einen ziemlich starken Einfluss von ca. 90% auf das Differenzbild, vor allem bei Farbbildern (C.12 und C.14), was sehr wahrscheinlich zum Großteil an der angewandten Kompression liegt. Bei bows2-Bildern (C.13) hingegen geht der Schwarz-Anteil selten über 80%, wodurch Stego-Einbettungen kaum von einer JPEG-Kompression zu unterscheiden sind. Im Falle von *steghide* ist in C.12 zu sehen, dass sich alle alaska2-Bilder ähnlich wie in Abbildung 2 dargestellt verhalten, wobei auch hier die Datenkompression vor der Einbettung am Beispiel von ASCII mit minimaler Entropie zu erkennen ist; auch bei diesem Attribut ist **die Nähe von *steghide*-Einbettungen zum Original-Bild auffällig**. Für alle Stego-Tools über alle Bildklassen und Tools hinweg (mit Ausnahme von *outguess* bei C.15) kann man sagen, dass mit zunehmender Einbettungsdatenlänge die sichtbarkeit im Differenzbild zunimmt. Ein **Einfluss der Schlüssellänge konnte bei keiner Einbettungsvariation** bzw. Bildklasse erkannt werden.

5.2.4 Farbkanal-Differenzbild

Um Stego-Tools in Bezug auf Priorisierung von spezifischen Farbkanälen bei der Einbettung zu überprüfen, wurden zusätzlich die Differenzbilder der einzelnen RGB-Farbkanäle betrachtet. Dazu wurde in der Darstellung der Einfluss des Einbettungsschlüssels gegen die Farbkanäle ausgetauscht, da ab diesem Punkt bereits abzusehen ist, das **der Einbettungsschlüssel nahezu keinen Einfluss auf**



(a) Originalbild (Quality-Factor 75) (b) ASCII-Daten (67 Bytes) (c) ASCII-Daten (1.53 KB) (d) ASCII-Daten (17.5 KB)

(e) ASCII-Daten (16 KB), minimale Entropie (f) Binärdaten (16.8 KB) (g) JPEG-Neukompression im Differenzbild (Quality-Factor 75) (h) JPEG-Neukompression im Differenzbild (Quality-Factor 80)

Abbildung 2. Einfluss verschiedener Einbettungsdaten bei *steghide* und JPEG-Kompression auf das Differenzbild; am Beispiel von Bild 00978 aus Alaska2-Datenbank

die Einbettungen hat. Dieser Punkt behandelt lediglich die RGB-Planes der *stegoveritas*-Zerlegungen ausführlich, weitere Detailbetrachtungen mussten in spätere Arbeiten verlagert werden (siehe Abs. 6.2.1).
Auch bei den Differenzbildern der Farbkanäle weisen die Bildklassen alaska2 und bows2 einige Ähnlichkeiten auf (C.16 und C.17). Bei *jsteg*, *outguess*, *outguess-0.13* und *f5* ist der Einfluss der Einbettungsdatenlänge auf das Differenzbild durch einen ansteigenden Schwarz-Anteil erkennbar, obwohl die Werte hierbei nicht stark von einer einfachen JPEG-Kompression abweichen. Auffällig bei allen Variationen, die eine Kompression einschließen ist, dass der rote und der blaue Farbkanal mehr verändert werden als der grüne (C.16). Bei Smartphone-Bildern hingegen konnte dieser Effekt außer bei *f5* nicht beobachtet werden (C.18). Da die bows2-Bilder schwarz-weiß sind, müssen auch bei einer Kompression alle Farbkanäle gleichmäßig geändert werden, um die Charakteristik zu erhalten (C.17).
Interessant ist, dass bei Smartphone-Bildern die Stego-Tools *jsteg*, *outguess* sowie *outguess-0.13* ähnliche Änderungswerte aufweisen wie *steghide*, obwohl dieses Programm als einziges keine JPEG-Kompression durchführt; eventuell verhalten sich einige Stego-Tools bei höheren Bildauflösungen anders, dies müsste mit einer Codeanalyse der Tools bestätigt werden (siehe Abs. 6.2.2). Aufgrund der in Abschnitt 3.3.1 beschriebenen Einschränkungen wurden im Diagramm C.18 nicht alle Smartphone-Bilder betrachtet. Deshalb gibt es auch für hochauflösende Bilder für dieses Merkmal keine Daten.

5.2.5 (Kurz-)Auswertung sonstiger Merkmale

- **Dateityp**: Bilder, die von *jsteg* manipuliert worden sind, können über *binwalk* nicht korrekt attributiert werden, d.h. der Dateityp kann nicht ausgelesen werden.
- **Aufnahme-Kamera**: Die in den Metadaten eines Bildes eingebetteten Informationen, wie z.B. die Aufnahme-Kamera, gehen bei allen Stego-Tool-Einbettungen verloren.
- **MIME-Type***: Der von *exiftool* ausgelesene MIME-Type bleibt bei allen Einbettungen erhalten.
- **JFIF-Version**: Bilder, die von *jsteg* manipuliert worden sind, können über *binwalk* als auch *exiftool* nicht korrekt attributiert werden, d.h. die JFIF-Version kann nicht ausgelesen werden.
- **Encoding**: Alle von Stego-Tools manipulierten Bildern werden mit „Baseline-DCT“ gespeichert, auch wenn sie vorher z.B. mit „Progressive DCT“ encodiert waren.
- **Farbkomponenten-Anzahl***: Die Anzahl an Farbkomponenten bleibt bei allen Einbettungsvariationen wie beim Original gleich.
- **Auflösung***: Die Auflösung bleibt bei allen Einbettungsvariationen wie beim Original gleich.
- **Megapixel***: Die Anzahl an Farbkomponenten bleibt bei allen Einbettungsvariationen wie beim Original gleich.

(* Merkmale ohne Mehrwert für Attributierung)

5.2.6 Detektionstools

Wie bereits in Abschnitt 4.5.4 beschrieben, ist die Ausführung von *stegbreak* problematisch. Trotzdem wurde eine Betrachtung der erfolgreichen Detektionen der verschiedenen Bildklassen durchgeführt. In Diagramm B.2 sind die Ergebnisse beider Detektionstools (*stegdetect* und *stegdetect*) für Stego-Tool-Einbettungen von *jsteg* und *outguess-0.13* dargestellt. Nur von diesen zwei Stego-Tools wird behauptet, dass diese erfolgreich detektiert werden könnten [7].
stegdetect konnte dabei keine *jsteg*-Einbettung erfassen, aber es konnte von *outguess-0.13* erstellte Stego-Bilder besser detektieren als *stegbreak*, insbesondere alaska2- und Smartphone-Bilder. Die Detektionsrate liegt dabei zwischen 1 und 22%, abhängig von Bildklasse und Einbettungsdatenlänge.
stegbreak kann (bis auf in hochauflösenden Bildern) *jsteg*-Einbettungen mit einer Chance von 2 bis 9% erfolgreich detektieren. *outguess-0.13*-Einbettungen kurzer Datenlänge sowie von Binärdaten konnten hingegen nicht erkannt werden. Für sonstige *outguess-0.13* Detektionen liegt die Erfolgsrate bei 5 bis 20%.
Kurze Einbettungsdatenlängen sind dabei in allen Variationen nur in sehr seltenen Fällen detektierbar.

5.3 Bewertung der Testfälle

5.3.1 Stego-Detektion

Bei dieser Untersuchung wurden in Abschnitt 5.2.5 zwei Merkmale identifiziert, die bei allen untersuchten Stego-Bildern aller Stego-Tools den gleichen Wert haben. Wenn also beide Merkmale in einem Bild nachweisbar sind, ist es möglich, dass eine Stego-Einbettung vorliegt.
Dabei handelt es sich konkret um die für das Bild verwendete Aufnahme-Kamera, sowie die verwendete Encodierung. Eine eventuell verwendete Aufnahme-Kamera ist aus Stego-Bildern nicht auslesbar, da bei der Speicherung Bildmetadaten verworfen werden (vermutlich als Sicherheitsmaßnahme, um beispielsweise auch Geo-Daten aus den Bildern zu entfernen, um so den Ursprung zu verschleiern). Die Encodierung aller Stego-Bilder ist außerdem immer „Baseline-DCT“. Auch die Smartphone-Bilder, die im Original als „Progressive-DCT“ vorliegen, wurden nach der Einbettung mit „Baseline-DCT“ gespeichert.
Da diese Merkmale allerdings lediglich Indizien für eine Stego-Einbettung sind und auch nicht auf ein spezielles Stego-Tool hinweisen, wurden diese Merkmale allerdings nicht implementiert.

5.3.1.1 Triviale Detektion

Einige Stego-Tools führen keine Überprüfung der Einbettungskapazität vor dem Einbetten durch, weshalb einige der erzeugten Stego-Bilder beschädigt sind. Wie beispielsweise in Abbildung 1 zu sehen ist, bricht *stegbreak* bei *jsteg*-Einbettungen, wo die Kapazität überschritten wurde, mit der Ausgabe „`Premature end of JPEG file`“ ab. Die Beschädigung ist auch visuell in Abbildung 3 im unteren Bildteil klar erkennbar. Bei *outguess* und *outguess-0.13* gibt es dieses Problem auch, allerdings ist in diesem Fall die Ausgabe-Datei komplett leer.

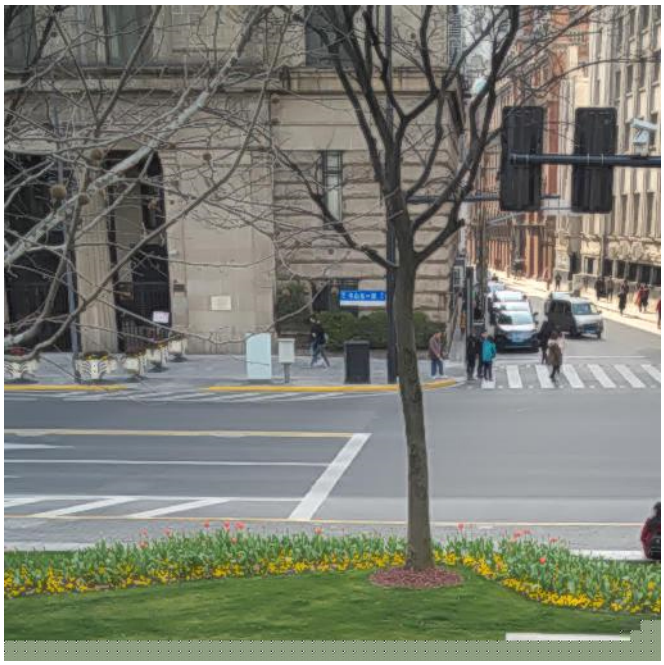


Abbildung 3. *jsteg*-Einbettung von Daten mit min. Entropie

5.3.2 Wie: Stego-Tool Attributierung

Zur Attributierung des verwendeten Stego-Tools konnten mehrere Merkmale identifiziert und implementiert werden. Diese sind in Tabelle 5 dargestellt und wurden in „direkte“ und „vergleichende“ Attribute eingeteilt. Mit „direkten“ Attributen kann ohne Betrachtung des Original-Bildes erkannt werden, welches Tool vermutlich eine Manipulation vorgenommen hat. Dabei ist das Betrachten des Bild-Headers mit dem Tool *strings* besonders wichtig, da *jsteg*, *outguess*/*outguess-0.13* und *f5* durch die bei der JPEG-Kompression verwendeten Bibliotheken Tool-spezifische Signaturen hinterlassen. „Vergleichende“ Attribute können lediglich beim Vorhandensein des Original-Bildes herangezogen werden. Vor allem lässt sich so *steghide* gut attributieren, da solche Stego-Einbettungen durch ihre Nähe zum Original auffallen, wie auch bereits in der Detailanalyse in Abschnitt 5.2 erkennbar ist.

5.3.2.1 Kompressionen bei Stego-Einbettungen

Die Stego-Tools *jsteg*, *outguess*, *outguess-0.13* und *f5* führen keine Datenkompression der Daten vor der Stego-Einbettung durch, allerdings wurde das erzeugte Stego-Bild mit einer neuen JPEG-Kompression gespeichert. Diametral dazu verhält sich *steghide*, welches vor der Einbettung die Daten komprimiert, allerdings beim Speichern des Stego-Bildes keine Neu-Kompression vornimmt.

5.3.3 Was: Attributierung des Inhalts

Die Attributierung der Einbettungsdaten ist nach dem Erkenntnisstand dieser Untersuchung am ehesten mit dem Attribut Entropie möglich, insofern vergleichend das Originalbild vorliegt. Damit können wie in Abschnitt 5.2.1 beschrieben mit steigender Entropie-Differenz auf längere Einbettungsdaten geschlossen werden: Insofern die Differenz größer als 1% ist, sollte von einer Stego-Einbettung ausgegangen werden, da eine Neukompression in der Regel die Entropie nicht um mehr als 0.5% ändert. Umso größer die Entropie-Differenz wird, umso mehr Daten wurden eingebettet (Ausnahme bei *f5*, dort sinkt die Entropie mit steigender Einbettungsdatenlänge).
Zur Attributierung des verwendeten Einbettungsschlüssels konnten in dieser Arbeit keine zur Attributierung nutzbaren Anhaltspunkte erkannt werden, die über verschiedene Bildklassen/Stego-Tools hinweg konsistent sind. Die einzigen Auffälligkeiten wurden dabei in den Diagrammen C.4, C.6 und C.8 beim Stego-Tool *steghide* ausgemacht, wobei zu erkennen ist, dass die Entropie bzw. Dateigröße bei kurzen Einbettungsschlüsseln höher ist; die Entropie allerdings nur bei ASCII-Daten. Da diese Effekte allerdings schwer nachzuvollziehen und minimal sind, sowie lediglich bei zwei untersuchten Merkmalen, zwei Bildklassen und einem Tool auftraten, sollte zunächst eine Codeanalyse von *steghide* durchgeführt werden, um diese Ergebnisse zu verifizieren und weitere Erkenntnisse zu gewinnen (siehe Abs. 6.2.2).

5.3.4 Wo: Ort der Einbettung

Wie in Abschnitt 2.1.1 und 2.1.2 erwähnt wurden für *jsteg*, *outguess-0.13* und *outguess* bereits Erkenntnisse zum Ort der Einbettung in [8] untersucht. In Abbildung 2 ist erkennbar, dass *steghide* vermehrt in Bildbereichen mit vielen Kanten einbettet. Dies ist dadurch zu erklären, dass Stego-Einbettungen das Bildrauschen in der veränderten Region verstärken, und dies bei großen, einfarbigen Bereichen früher auffällig ist als in Bereichen mit vielen verschiedenen Strukturen. Für genauere Aussagen zum Ort der Einbettungen können die in Abschnitt 6.2.5 beschriebenen Möglichkeiten betrachtet werden.

5.4 Genauigkeit der implementierten Attributierung

Zum Testen der implementierten Tool-Attributierung wurde der Testumfang aus Zeitgründen auf ein Minimum reduziert: Da die Einflüsse einer Schlüsselvariation auf die Detektion einen verschwindend geringen Einfluss aufweist, wurde für die durchgeführten Tests jeweils nur ein kurzer 4-Byte-Schlüssel bzw. kein Schlüssel (Tool-abhängig) verwendet. Für die Tests wurde außerdem nur die kurze 67-Byte-Einbettungsdatenlänge gewählt, da diese durch den geringen Einfluss auf das Bild tendenziell am schwierigsten zu erkennen ist. Mit diesen Einschränkungen ließ sich der Testumfang auf 5 Stego-Einbettungen plus 2 nicht-manipulierte Bilder (Originalbild und einfache JPEG-Kompression) eingrenzen. Außerdem wurde das komplette Coverbild-Testset insgesamt zweimal durchlaufen, einmal als Einzelattributierung der Einbettungen, und einmal vergleichend mit dem Original-Bild. Da das Original-Bild in der Praxis in den seltensten Fällen verfügbar ist, ist die Einzelattributierung der Stego-Einbettungen (vor allem in den Smartphone-Bildern) mit Blick auf die Realität der interessanteste Testfall. Dabei konnten die in Diagramm B.3 dargestellten Ergebnisse erzielt werden.
Zunächst ist zu erkennen, dass *outguess-0.13* vom Detektor in keinem Fall erkannt wurde. Aus diesem Grund sind für *outguess-0.13* keine direkten Ergebnisse zu sehen; zusätzlich wurden dafür *outguess-0.13*-Einbettungen, die als *outguess-0.2* detektiert wurden, mit „outguess-0.13 als 0.20“

(98-100% erfolgreich) im Diagramm visualisiert, woraus man schließen kann, dass *outguess-0.13* nicht im nötigen Umfang von *outgues-0.20* differenziert werden konnte, da vermutlich zu wenige Merkmale an der Attributierung beteiligt sind.
Außerdem ist zu sehen, dass *steghide* ausschließlich mit vorliegendem Original-Bild detektiert werden konnte. Dies ist damit zu erklären, dass die Attributierung von *steghide* stark auf vergleichenden Merkmalen, wie dem Differenzbild beruht.
Original-Bilder sowie neukomprimierte Bilder ohne Stego-Einbettungen werden vergleichsweise häufig fälschlicherweise als Stego-Cover detektiert, so dass diese lediglich mit einer Erfolgsrate von 60-97% korrekt als solche attributiert werden.

5.4.1 Vergleich mit stegdetect und stegbreak

stegdetect und *stegbreak* weisen, wie in Abschnitt 5.2.6 beschrieben, relativ niedrige Detektionsraten auf. Trotzdem sind beide Tools in der direkten Erkennung von *outguess-0.13*-Einbettungen besser als der auf den in Tabelle 5 gelisteteten Merkmalen basierende Detektor. In allen anderen Testfällen allerdings ist der hier beschriebene Ansatz (zumindest in Bezug auf das konkret genutzte Coverbild-Testset) deutlich genauer in der Bestimmung des verwendeten Stego-Tools. Dabei konnten Detektionsraten von bis zu 100% erreicht werden, wohingegen *stegdetect* im Optimalfall maximal 22% aufwies.

5.5 Als Angreifer Attributierung umgehen

Steganographische Einbettungen sind einfacher zu detektieren, wenn (nahezu) die volle Einbettungskapazität des Covers genutzt wird, wodurch Merkmale wie die Entropie auf die Einbettung hinweisen könnten. So gesehen sind Einbettungen in hochauflösenden Bildern schwerer zu erfassen, da sich inhaltsbasierte Merkmale relativ wenig verändern, d.h. umso größer das Bild, desto weniger fällt die Stego-Manipulation ins Gewicht.
Wichtig beim Entwickeln eines Stego-Tools ist ebenfalls, dass statistische Merkmale, wie z.B. der Dateiheader authentisch bleiben und Metadaten wie Aufnahme-Kamera und -ort vorhanden sind. Aus diesen Gründen sollte ein Angreifer keine Cover-Bilder aus bekannten Stego-Datenbanken verwenden.
Trivial sollte sein, dass die erzeugte Stego-Datei nicht beschädigt ist; wie allerdings in Abschnitt 5.1 am Beispiel von *outguess-0.13/outguess* sowie *jsteg* zu sehen ist, wurde im Vorfeld die Einbettungskapazität nicht überprüft bzw. wurden Daten auch in die Header-Informationen eingebettet, wodurch die Bilddatei beschädigt wird („`Premature end of JPEG file`“).

6 Zusammenfassung und Ausblick

6.1 Zusammenfassung der Ergebnisse

6.1.1 Attributierungsmerkmale

Die untersuchten Bildmerkmale können in verschiedene Attributierungsebenen eingeteilt werden. Dabei gibt es zunächst Merkmale, die allgemein auf Steganographie hinweisen (Detektionsmerkmale), dazu zählt z.B. die Entropie. Um anschließend Aussagen über Einbettungsparameter treffen zu können, müssen zunächst das Stego-Tool und Details zur Einbettungsstrategie bestimmt werden. Dabei hat jedes Tool eine bestimmte Signatur aus zusammenwirkenden Eigenschaften (z.B. Dateiheader und Metadaten), anhand derer es identifiziert werden kann (Tool-Attributierung, siehe Tabelle 5). Abhängig vom verwendeten Stego-Tool, können Aussagen über Einbettungsdaten und -schlüssel getroffen werden. Dazu ist unter anderem relevant, ob und wie Stego-Programme Einbettungsdaten komprimieren und ob eine JPEG-Kompression ausgeführt wird. Auswirkungen des Einbettungsschlüssels auf die Stego-Einbettung konnten bis auf eine Ausnahme nicht erkannt werden. Insofern das Original-Cover vergleichend zum potentiellen Stego-Bild vorliegt, können weitere Merkmale herangezogen werden, wie z.B. die Dateigröße oder Differenzbild-Betrachtungen.

6.1.2 Einfluss der Einbettungsparametervariation

Die Variation der Einbettungsdaten hat einen Einfluss auf die mögliche Attributierung. Besonders kurze Stego-Einbettungen sind gegenüber längeren Einbettungen bzw. Einbettungen mit hohen Entropien (wie Binärdaten) schwerer zu erkennen, da sie einen geringeren Einfluss auf inhaltsbasierte Bildmerkmale haben. Verallgemeinert bedeutet das, desto größer das Verhältnis zwischen Stego-Kapazität (definiert durch Bildauflösung) des Covers und der Einbettungsdatenlänge, umso unauffälliger verhält sich das Covermedium gegenüber einer Attributierung. Da die in dieser Arbeit getesteten Einbettungsdaten bei allen Bildklassen dieselben sind, lassen sich kleinere Bilder wie alaska2- und bows2-Bilder demzufolge insgesamt einfacher detektieren und attributieren als Bilder mit höheren Auflösungen. Allerdings sind einige relevante Attributierungsmerkmale davon nicht betroffen, wie beispielsweise der Datei-Header der Stego-Einbettungen, wodurch auch hier eine solide Attributierung zumindest des verwendeten Stego-Tools möglich ist. Da die Schlüssellänge einen verschwindend geringen Einfluss auf verschiedenste untersuchte Attribute aufweist, kann der Einbettungsschlüssel über die hier betrachteten Merkmale allgemein nicht attributiert werden.

6.2 Ausblick für zukünftige Arbeiten

6.2.1 Erweiterte Detailanalyse

Die Analyse kann in mehreren Aspekten ausgeweitet werden. Beispielsweise können zu den in dieser Arbeit verwendeten Bildklassen weitere hinzugezogen werden, oder Bildklassen nach Bildinhalt zusammengestellt werden. Da die im *Stego-Toolkit* [7] vorhanden Programme noch viele weitere Informationen liefern, die zum Attributieren verwendet werden könnten, können in einer weiterführenden Untersuchung weitere Attribute herausgearbeitet werden. Beispielsweise liefert *stegoveritas* mehrere Bildzerlegungen, von den

jeweils nur die RGB-Planes in Abschnitt 5.2.4 betrachtet wurden, wobei zusätzlich unter anderem Kanten, Minima, Median, Maxima sowie weich- und scharfgezeichnete Bearbeitungen zur Verfügung stehen. Außerdem sollte bei der Auswertung der beispielsweise in den Diagrammen im Anhang C dargestellten Daten zusätzlich die Standardabweichung bzw. Varianz mit einbezogen werden, um die Verteilung innerhalb der Bildklassen besser erfassen zu können. Da ein Einfluss des Einbettungsschlüssels nicht in ausreichendem Umfang nachgewiesen werden konnte, muss die Betrachtung der Kombination aus Einbettungsdaten- und -schlüsselvariation ebenfalls auf weiterführende Arbeiten ausgelagert werden.

6.2.2 Codeanalyse der Stego-Tools

Um die Einbettungsstrategien der Tools vollständig nachvollziehen zu können, muss eine Codeanalyse der Stego-Tools erfolgen. Dadurch können durch Verständnis der Funktionsweise der Programme gezielt sinnvolle Attributierungsmerkmale hergeleitet werden. Zusätzlich ist es so möglich, die maximal mögliche Einbettungsdatenlänge zu bestimmen sowie den kompletten Einbettungsschlüsselraum zu identifizieren.

6.2.3 Intermedienvergleich

In dieser Arbeit wurden lediglich JPEG-Bilder betrachtet. Interessant ist aber auch ein Intermedienvergleich mit z.B. einer Repräsentation eines JPEG-Covers als PNG-Bild. Da steganographische Verfahren nicht auf Bild-Medien beschränkt sind, ist die Attributierung von beispielsweise PNG-, BMP- oder auch MP3-, WAV- und MP4-Dateien ebenso sinnvoll. Ausgangspunkt könnten hierbei die weiteren Werkzeuge aus dem *Stego-Toolkit* [7] sein.

6.2.4 Parallele Implementierung

Um größere (Sub-)Coverbild-Testsets von Medien betrachten zu können, kann eine effizientere, parallele Implementierung realisiert werden. Dadurch wird eine bessere statistische Signifikanz erreicht, da bei dieser Untersuchung zumindest die Bildklasse „höherauflösende Bilder“ nicht ausreichend repräsentiert ist.

6.2.5 Verbesserte inhaltsbasierte Untersuchung

Um eine context-sensitive Attributierung effizient zu automatisieren, wären für die Stego-Analyse auch Methoden aus dem AI-Bereich wie beispielsweise Neuronale Netze möglich zu verwenden. Damit kann der Bildinhalt als Merkmal in die Untersuchung mit einbezogen werden, was mit der Werkzeugauswahl dieses Projekts nicht möglich ist. Bei einer zusätzlich verbesserten inhaltsbasierten Analyse kann das Bild in z.B. Quadranten zerlegt werden, um mit einer folgenden Differenzbildanalyse der 4 Teilbilder auf die angewandte Einbettungsstrategie schließen zu können.

6.2.6 Stego-Analysepipeline

Wie in Abschnitt 6.1.1 bereits erwähnt, lassen sich Attributierungsmerkmale in mehrere Stufen einteilen. Daraus kann in weiteren Arbeiten eine Analysepipeline, mit beispielsweise folgenden Schritten implementiert werden:
Detektion → Tool-Attributierung → Datenattributierung

LITERATUR

[1] K. Lamshöft and J. Dittmann, “Grundlagenvorlesung zu steganographie (auszug steganographie von multimedia and security) in kw43,” Oct. 2022.

[2] K. Lamshöft, “Grundlagenvorlesung zu steganographie in kw43,” Oct. 2022.

[3] J. Dittmann, “Grundlagenvorlesung smkits/mmdap in kw42,” Oct. 2022.

[4] BSI, “Risikostufen für schwachstellen,” Aug. 2022. [Online]. Available: https://www.bsi.bund.de/dok/9202084

[5] A. Jeyasekar, D. Bisht, and A. Dua, “Analysis of exploit delivery technique using steganography,” 2016. [Online]. Available: https://dx.doi.org/10.17485/ijst/2016/v9i39/102075

[6] J. Dittmann, “Modul-, themenvorstellung und -vergabe in kw41,” Oct. 2022.

[7] “Stego-toolkit,” Mar. 2020. [Online]. Available: https://github.com/DominicBreuker/stego-toolkit

[8] N. Provos and P. Honeyman, “Detecting steganographic content on the internet,” Aug. 2001. [Online]. Available: http://www.citi.umich.edu/u/provos/papers/detecting.pdf

[9] “Extracting data embedded with jsteg,” Feb. 2004. [Online]. Available: http://www.guillermito2.net/stegano/jsteg/index.html

[10] S. Kiltz, “Data-centric examination approach (dcea) for a qualitative determination of error, loss and uncertainty in digital and digitised forensics,” Sep. 2020. [Online]. Available: http://dx.doi.org/10.25673/34647

[11] BSI, “Leitfaden it-forensik, p. 83,” Mar. 2011. [Online]. Available: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Cyber-Sicherheit/Themen/Leitfaden_IT-Forensik.pdf?__blob=publicationFile&v=1

[12] “Alaska2 image steganalysis,” Jul. 2020. [Online]. Available: https://www.kaggle.com/competitions/alaska2-image-steganalysis/data

[13] “Bows2 web page,” Apr. 2008. [Online]. Available: http://bows2.ec-lille.fr/

[14] B. Birnbaum, “/coverdata (coverbild-testset) in smkits5-stegodetect-main.tar.gz,” Oct. 2022.

[15] “Imagemagick,” Dec. 2022. [Online]. Available: https://imagemagick.org/

[16] “‘diff’ an image using imagemagick,” Feb. 2011. [Online]. Available: https://stackoverflow.com/questions/5132749/diff-an-image-using-imagemagick

[17] B. Birnbaum, “/stego-attrib.sh (umsetzung testprotokoll und attributierung) in smkits5-stegodetect-main.tar.gz,” Jan. 2023.

[18] ——, “/embeddingdata (einbettungsdaten) in smkits5-stegodetect-main.tar.gz,” Nov. 2022.

[19] ——, “/stego-utils-buildtestset.sh (zusammenkopieren des coverbild-testsets) in smkits5-stegodetect-main.tar.gz,” Jan. 2023.

[20] ——, “/stego-docker.sh (docker-umgebungsverwaltung) in smkits5-stegodetect-main.tar.gz,” Jan. 2023.

[21] ——, “/stego-utils-generatediagrams.sh (vorbereitende verrechnung ausgew. attribute) in smkits5-stegodetect-main.tar.gz,” Jan. 2023.

[22] ——, “/stego-utils-recompress.sh (neukomprimierung des coverbild-testsets) in smkits5-stegodetect-main.tar.gz,” Jan. 2023.

[23] ——, “/stego-attrib-test.sh (testen der tool-attributierung) in smkits5-stegodetect-main.tar.gz,” Jan. 2023.

[24] ——, “/stego-docker-importdefaults.sh (docker-standarddatensetup) in smkits5-stegodetect-main.tar.gz,” Jan. 2023.

[25] ——, “/docs/acquireddata (datengrundlage für auswertung) in smkits5-stegodetect-main.tar.gz,” Dec. 2022.

[26] ——, “smkits5-stegodetect-full-analysis-data-6x.tar.gz,” Dec. 2022.

[27] ——, “/docs/annotatedanalysis (manuell annotierte auswertung) in smkits5-stegodetect-main.tar.gz,” Dec. 2022.

[28] A. Latham, “jphide/jpseek,” Aug. 1999. [Online]. Available: http://linux01.gwdg.de/~alatham/stego.html

[29] “Stegdetect und stegbreak (seite 2),” Apr. 2008. [Online]. Available: https://www.linux-community.de/ausgaben/linuxuser/2008/04/stegdetect-und-stegbreak/2/

[30] “stegdetect/stegbreak repository,” Oct. 2018. [Online]. Available: https://github.com/abeluck/stegdetect

[31] “stegbreak ’rules.ini’,” Jun. 2012. [Online]. Available: https://github.com/poizan42/stegdetect/blob/master/rules.ini

ANHANG A

A.1 Testprotokoll als Ablaufdiagramm

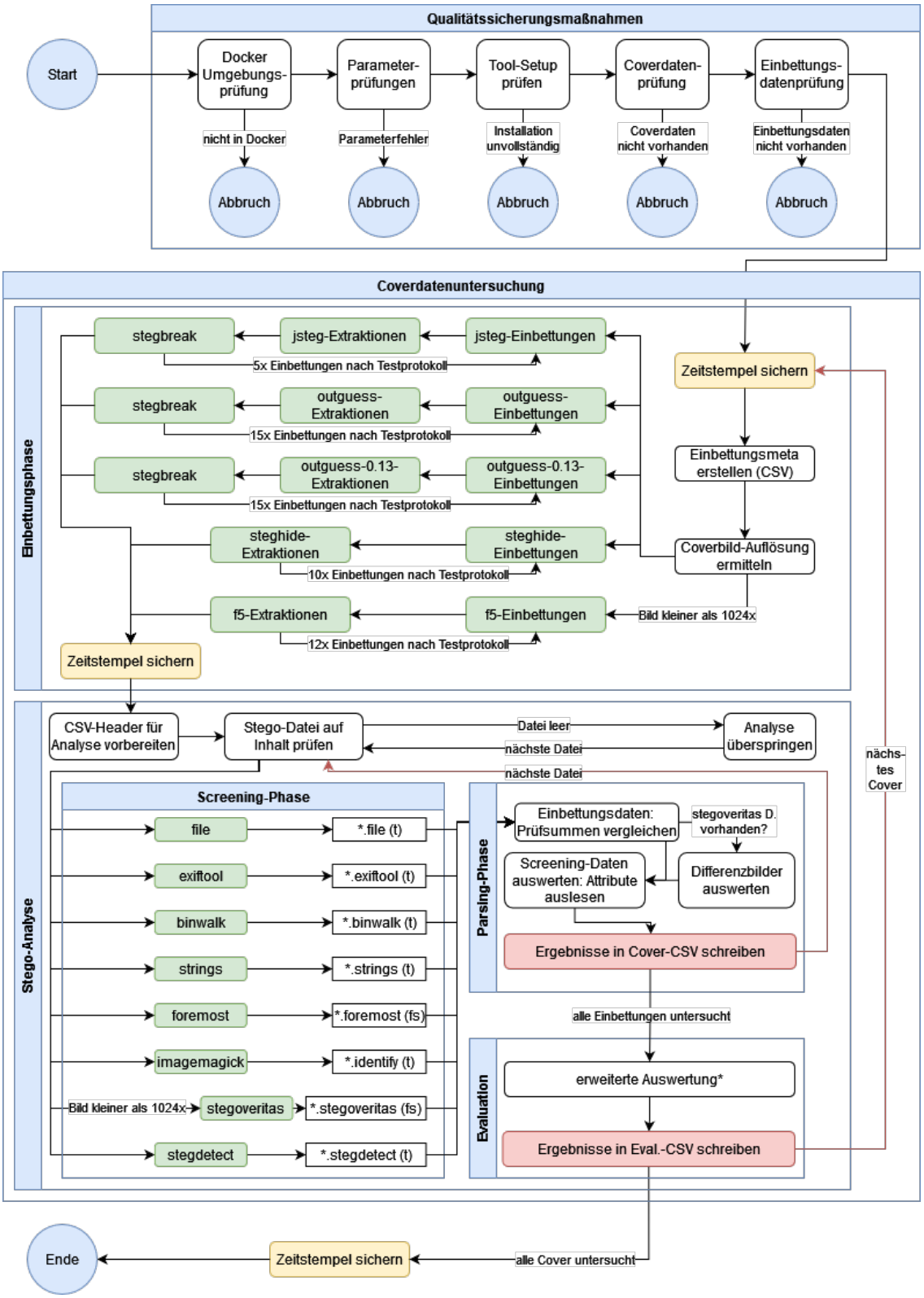


Abbildung A.1. Ablaufdiagramm zur Umsetzung des Testprotokolls

ANHANG B

B.1 Detektoren im Vergleich

B.1.1 Detektionen von stegdetect und stegbreak

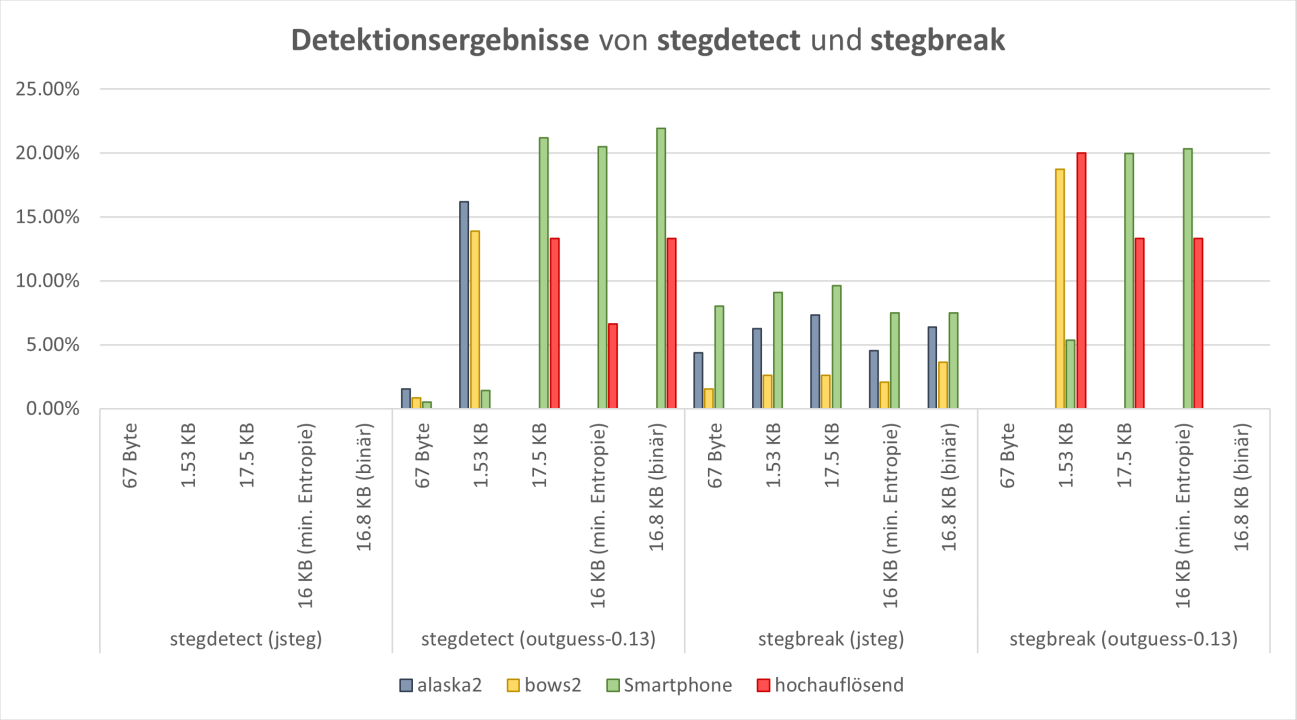


Abbildung B.2. erfolgreiche Detektionen von stegdetect und stegbreak

B.1.2 Detektionen der hier vorgestellten Tool-Attributierung

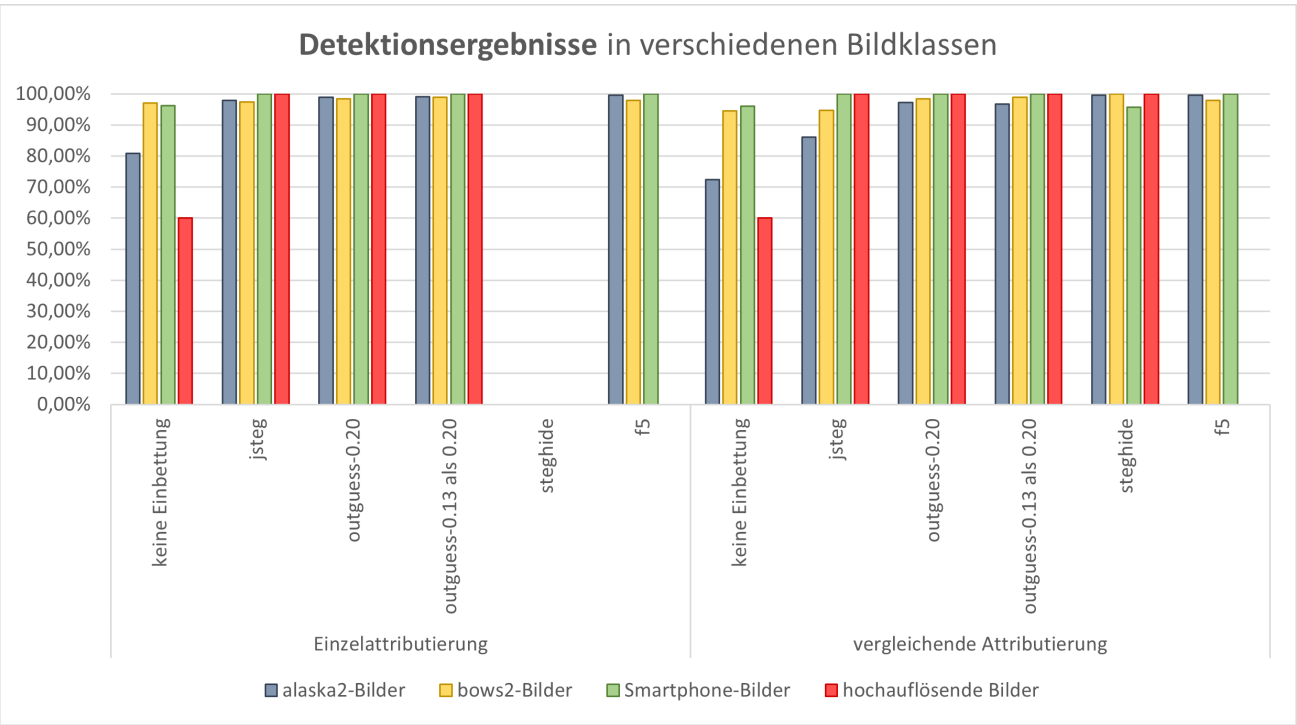


Abbildung B.3. erfolgreiche Detektionen aller Bildklassen und Stego-Tools, jeweils mit Original-Vergleich; kein/kurzer Schlüssel und kurze Einbettungsdaten

ANHANG C

C.1 Vergleich der Entropie

C.1.1 kaggle/alaska2-Bilder

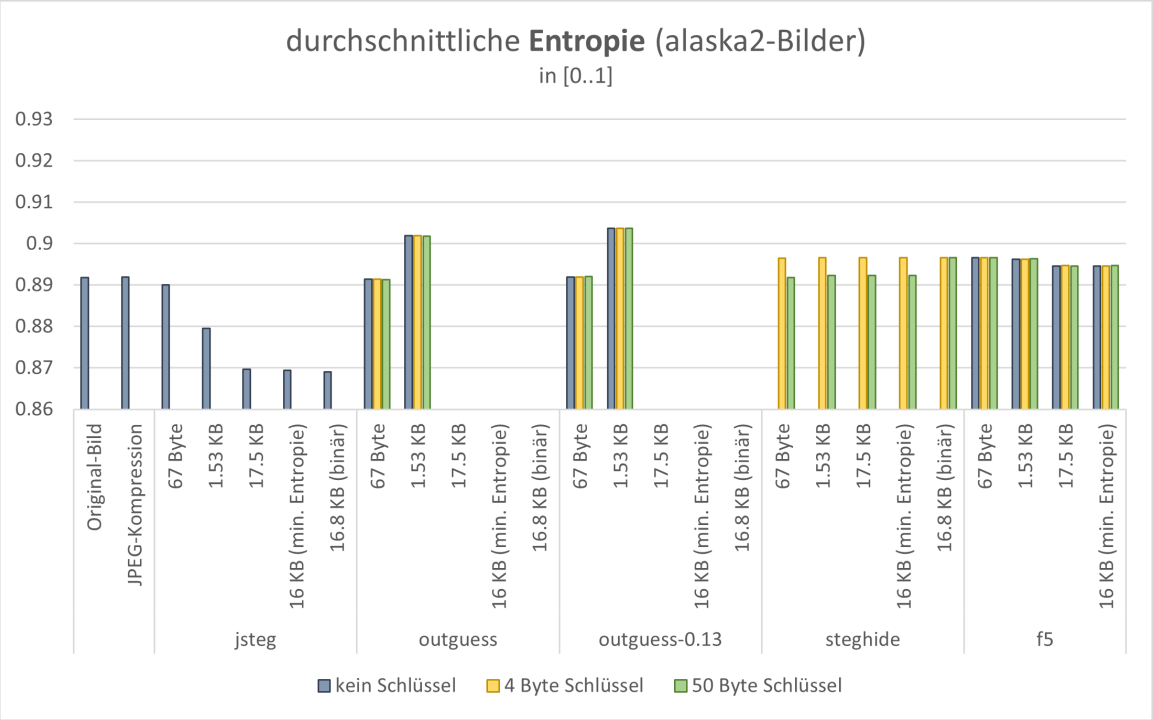


Abbildung C.4. Entropie in 640 betrachteten alaska2-Bildern

C.1.2 bows2-Bilder

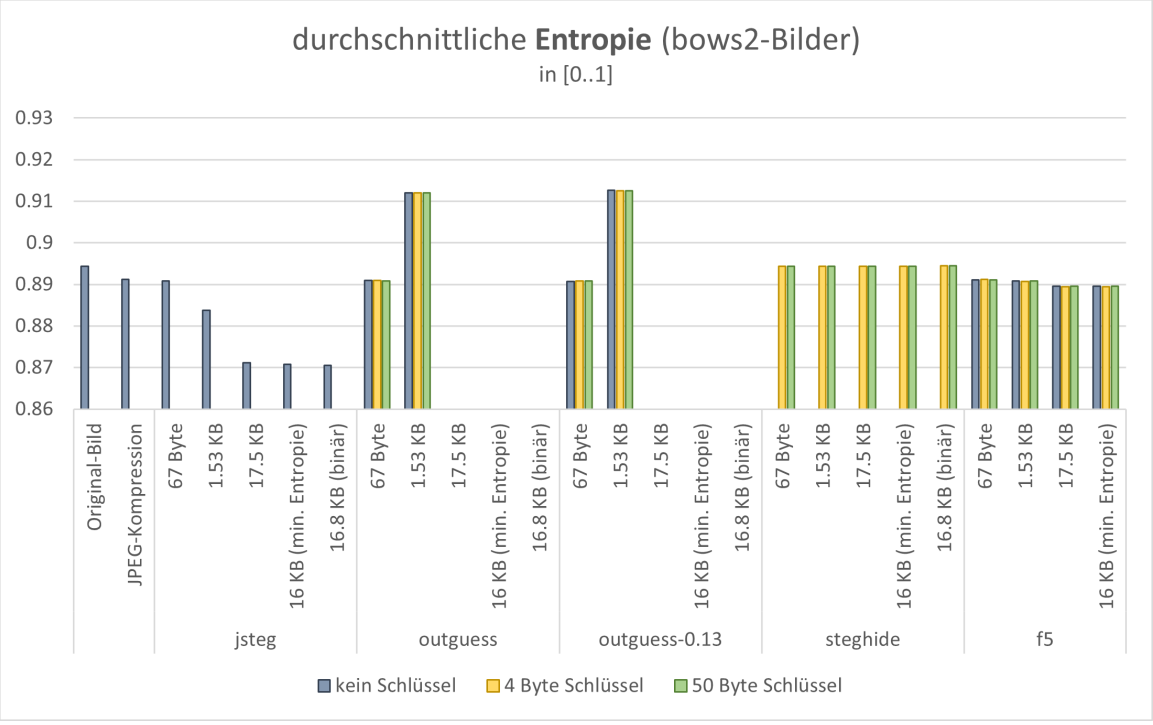


Abbildung C.5. Entropie in 192 betrachteten bows2-Bildern

C.1.3 Smartphone-Kamera-Bilder

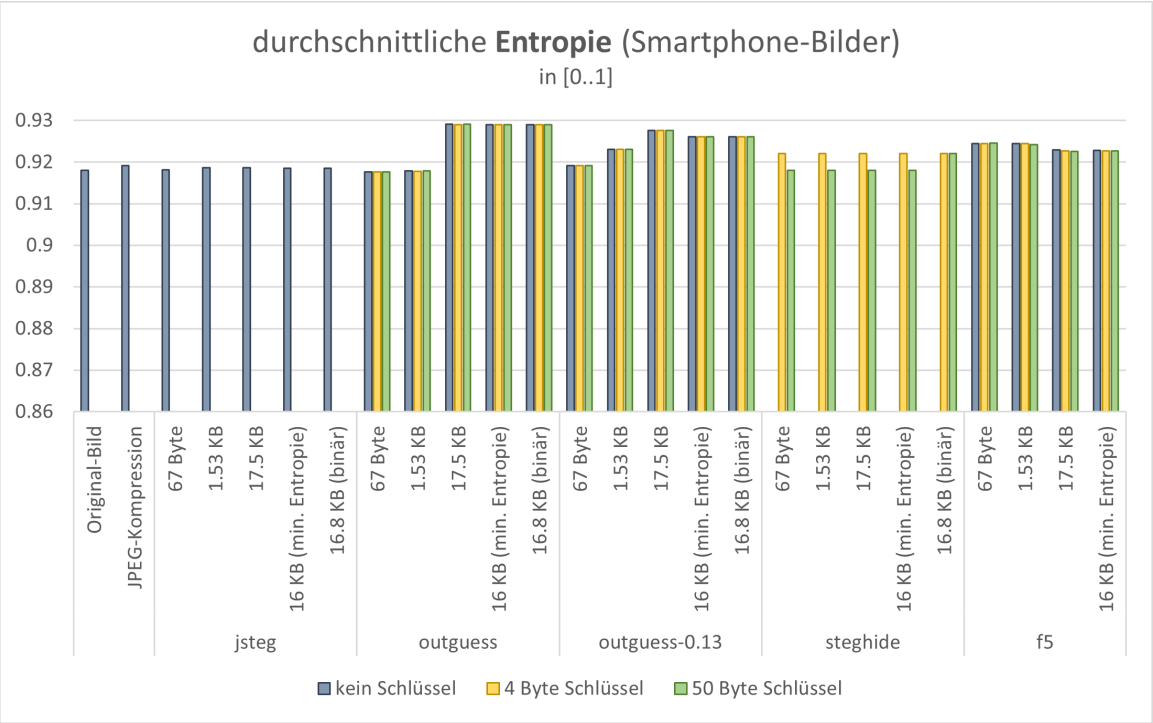


Abbildung C.6. Entropie in 187 betrachteten Smartphone-Kamera-Bildern

C.1.4 Hochauflösende Bilder

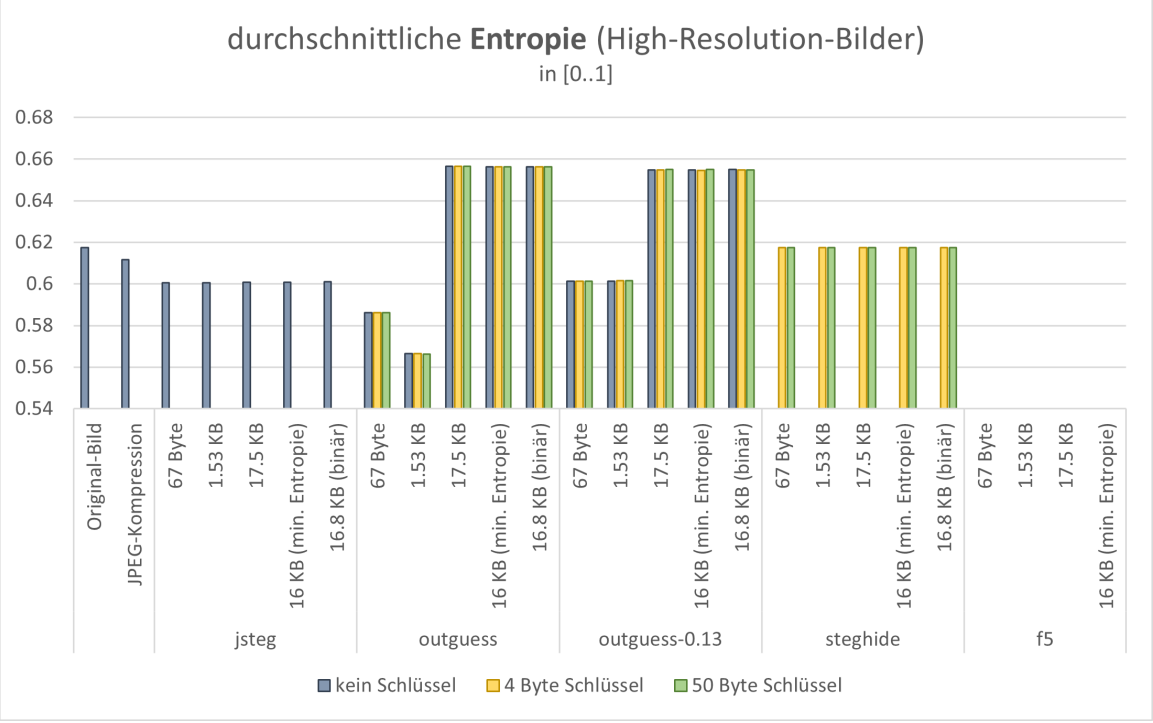


Abbildung C.7. Entropie in 5 betrachteten High-Resolution-Bildern

C.2 Vergleich der Dateigröße

C.2.1 kaggle/alaska2-Bilder

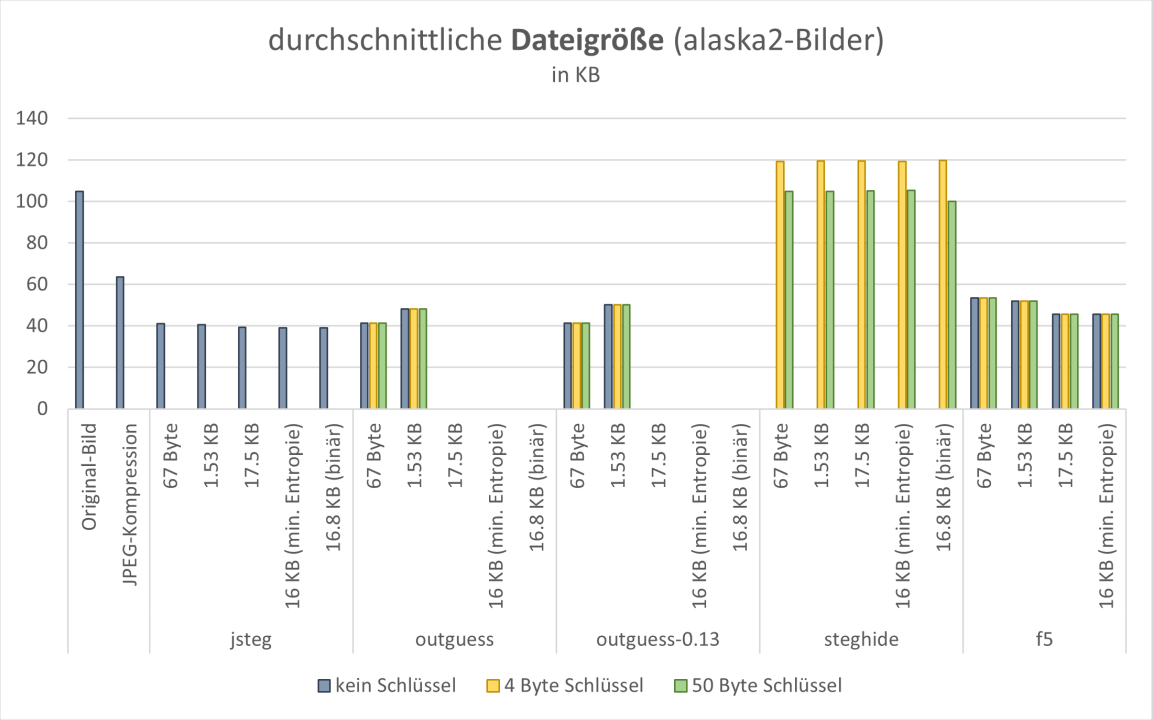


Abbildung C.8. Dateigröße in 640 betrachteten alaska2-Bildern

C.2.2 bows2-Bilder

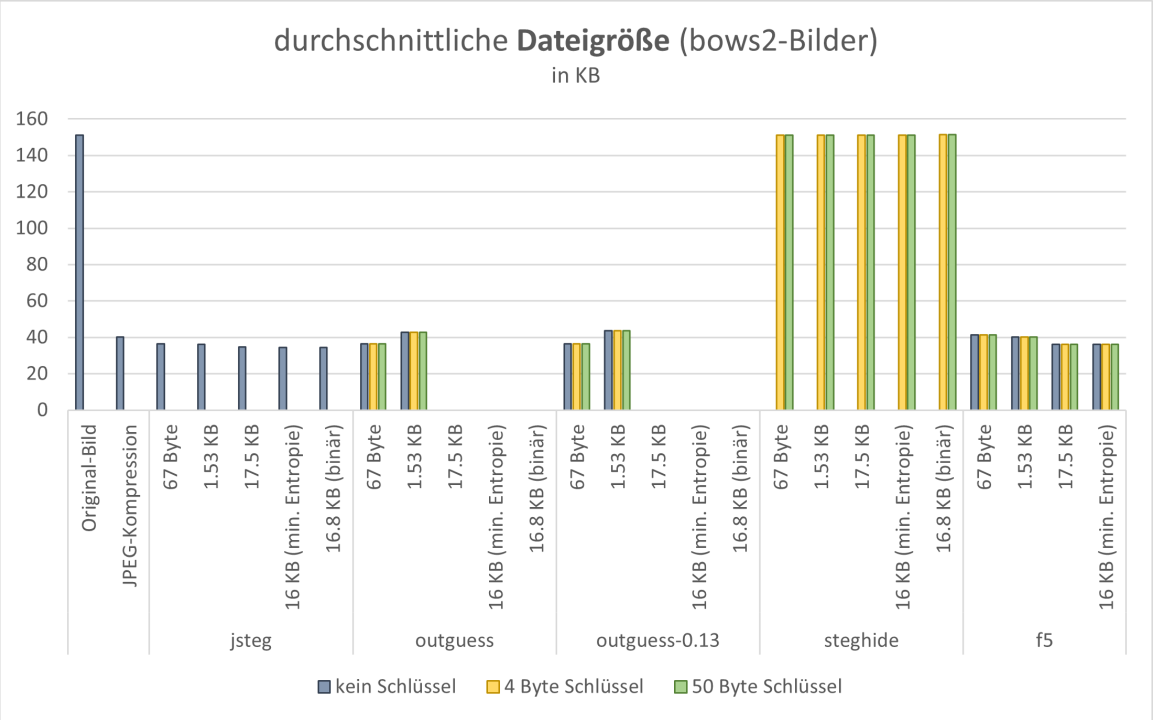


Abbildung C.9. Dateigröße in 192 betrachteten bows2-Bildern

C.2.3 Smartphone-Kamera-Bilder

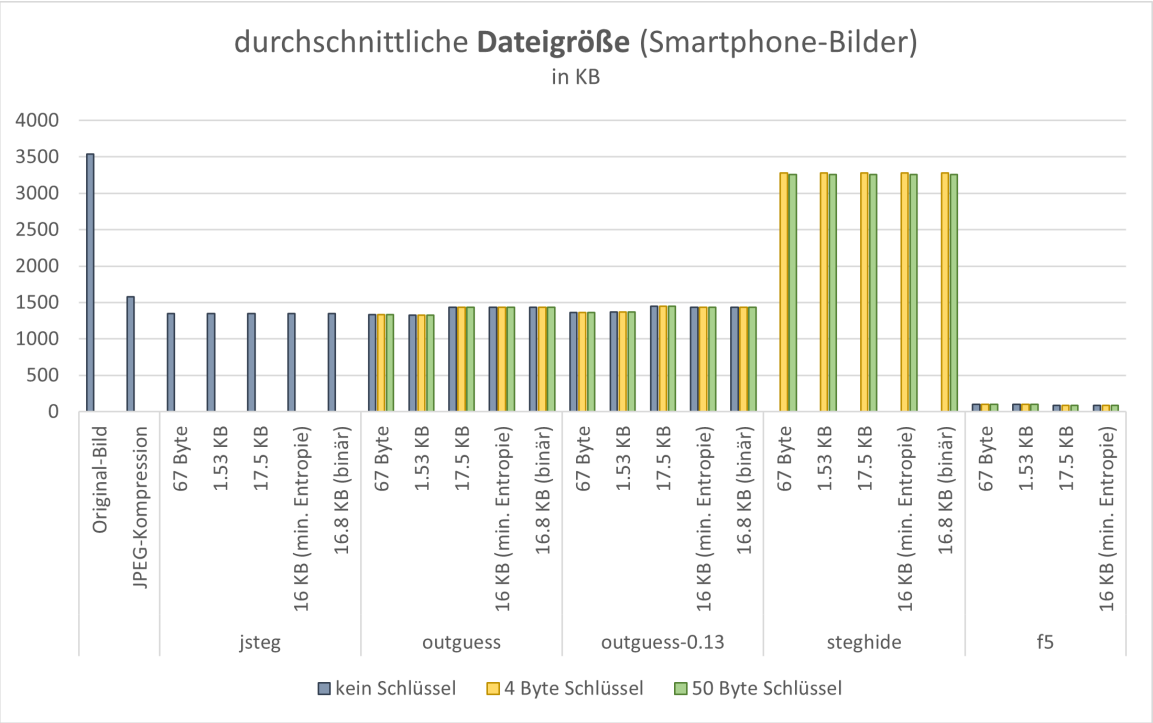


Abbildung C.10. Dateigröße in 187 betrachteten Smartphone-Kamera-Bildern

C.2.4 Hochauflösende Bilder

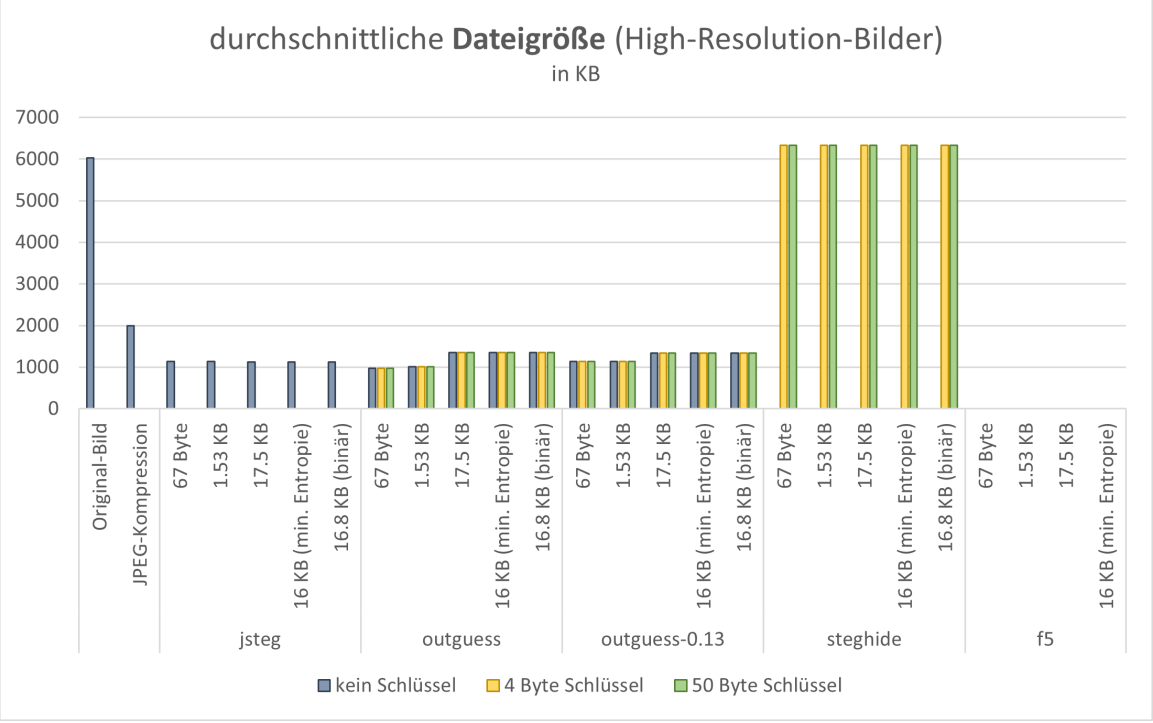


Abbildung C.11. Dateigröße in 5 betrachteten High-Resolution-Bildern

C.3 Vergleich der Differenzbilder

C.3.1 kaggle/alaska2-Bilder

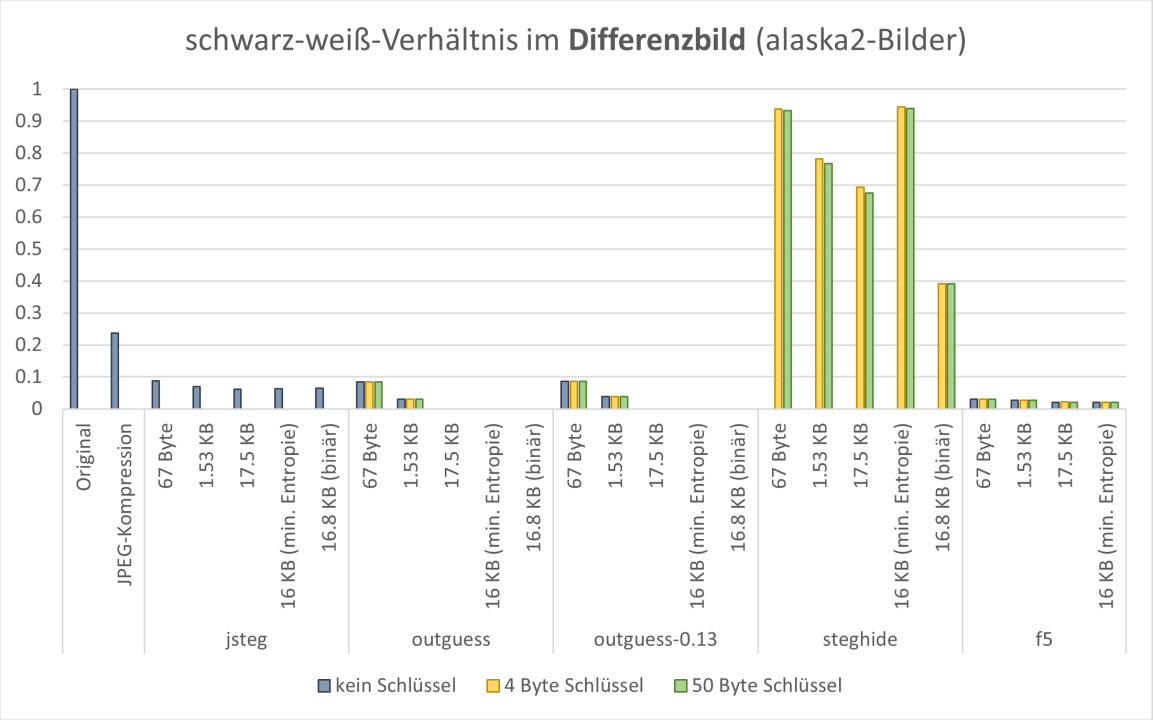


Abbildung C.12. Differenzbilder von 640 betrachteten alaska2-Bildern

C.3.2 bows2-Bilder

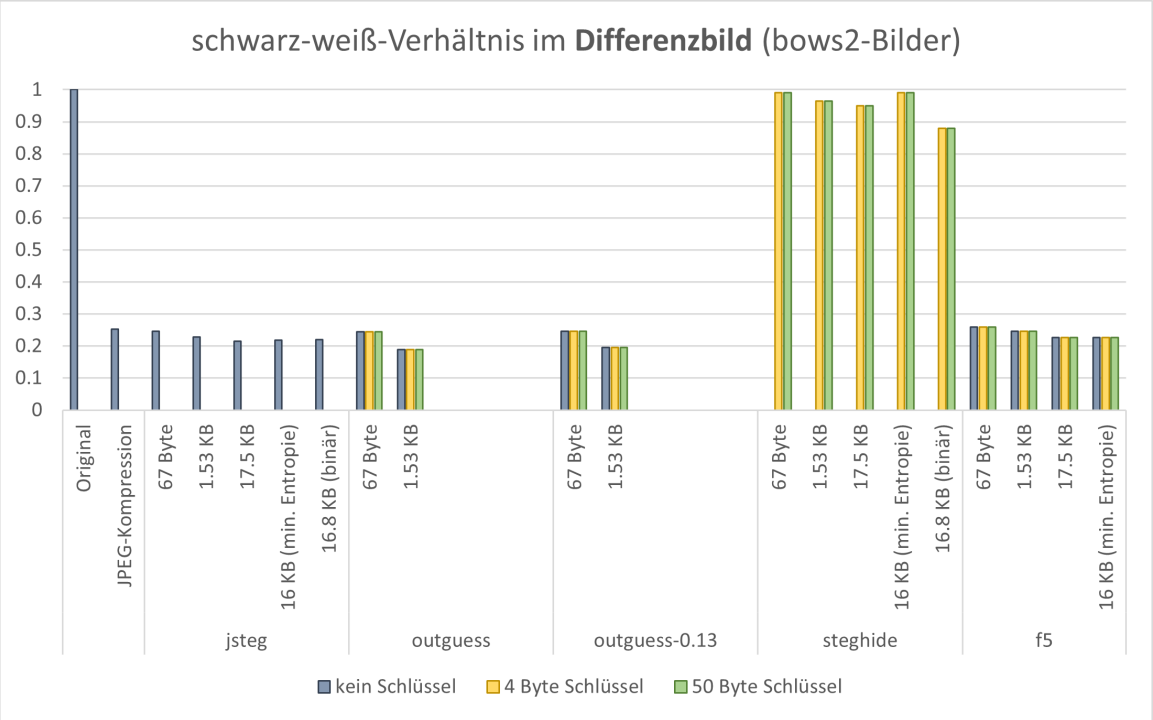


Abbildung C.13. Differenzbilder von 192 betrachteten bows2-Bildern

C.3.3 Smartphone-Kamera-Bilder

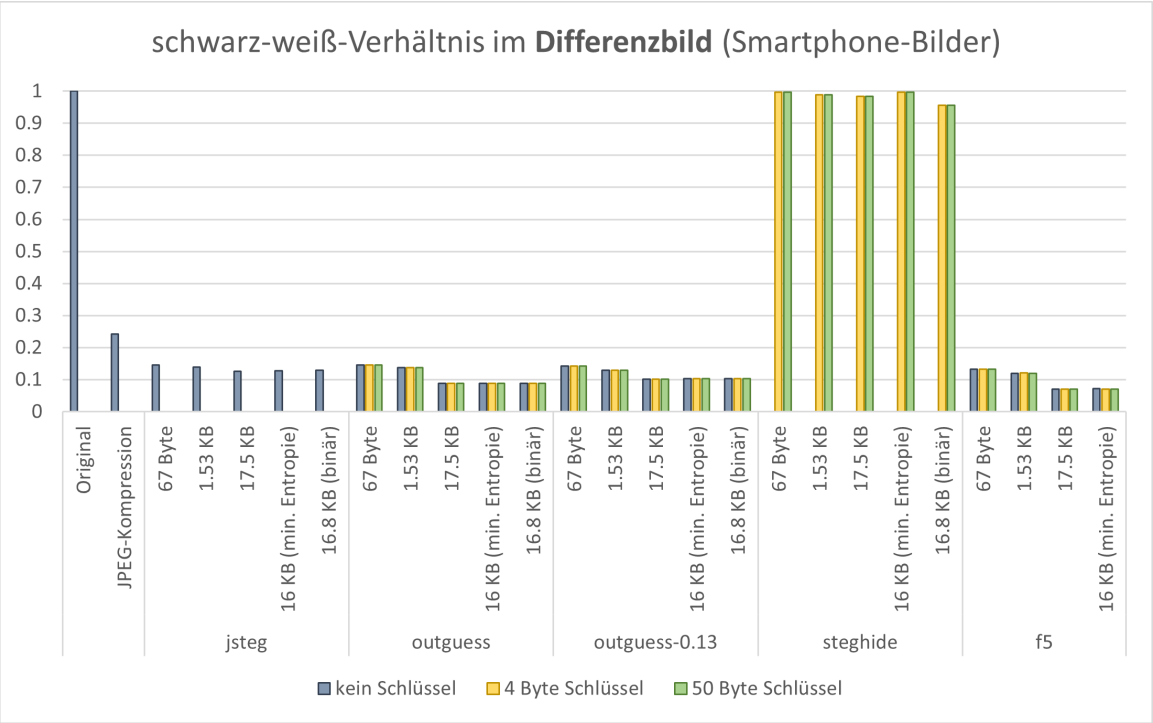


Abbildung C.14. Differenzbilder von 187 betrachteten Smartphone-Kamera-Bildern

C.3.4 Hochauflösende Bilder

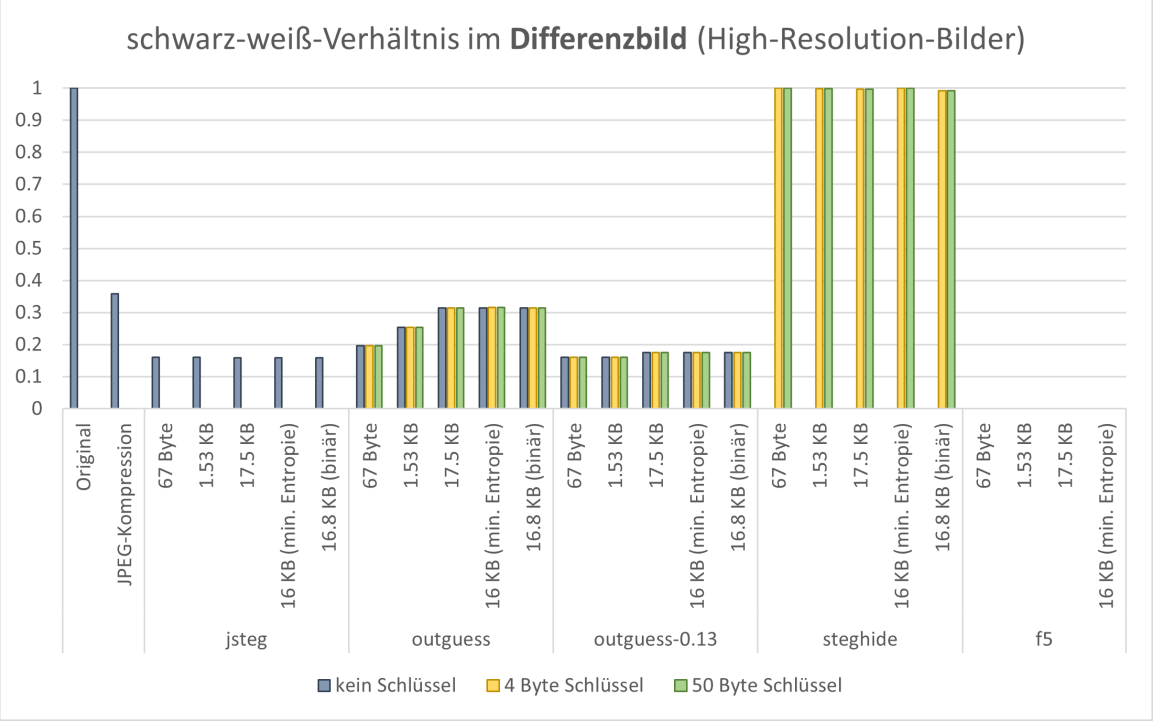


Abbildung C.15. Differenzbilder von 5 betrachteten High-Resolution-Bildern

C.4 Vergleich der Farbkanal-Differenzbilder

C.4.1 kaggle/alaska2-Bilder

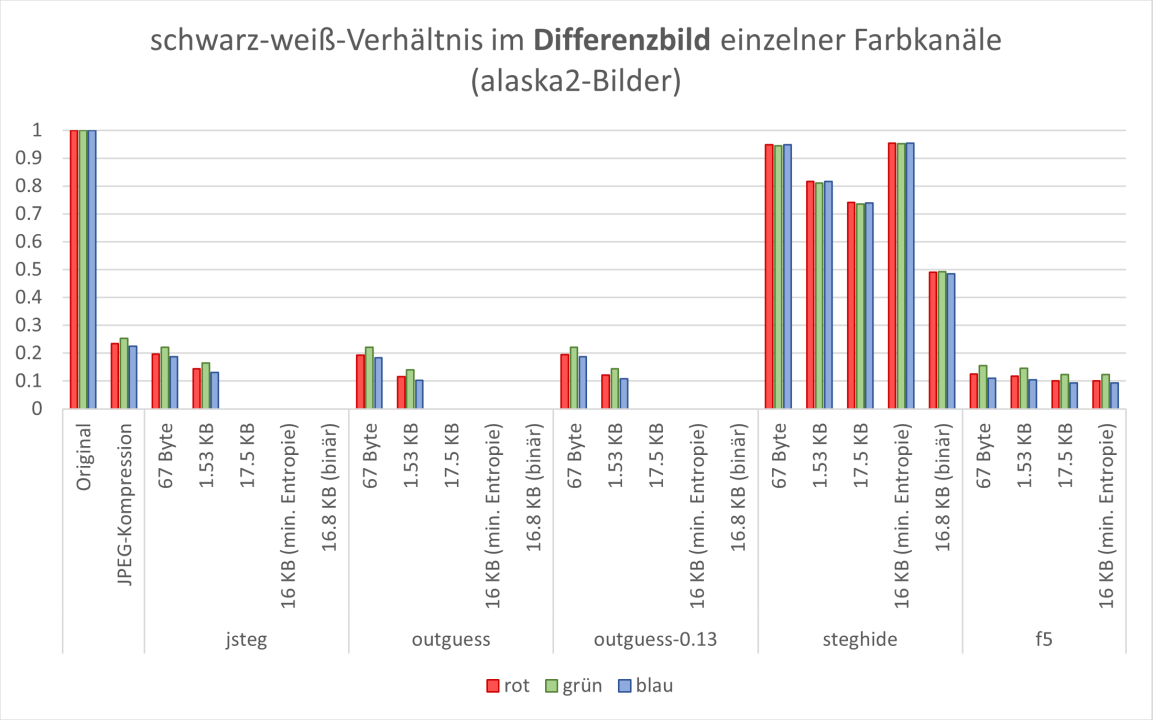


Abbildung C.16. Farbkanal-Differenzbilder von 640 betrachteten alaska2-Bildern

C.4.2 bows2-Bilder

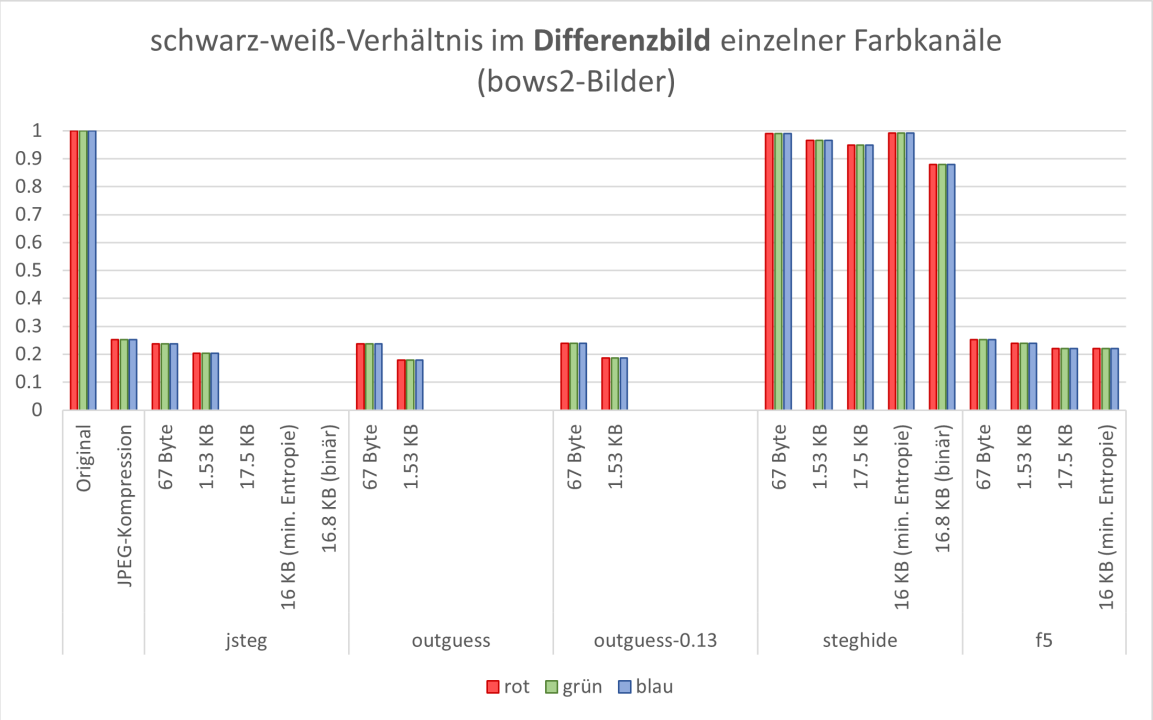


Abbildung C.17. Farbkanal-Differenzbilder von 192 betrachteten bows2-Bildern

C.4.3 Smartphone-Kamera-Bilder

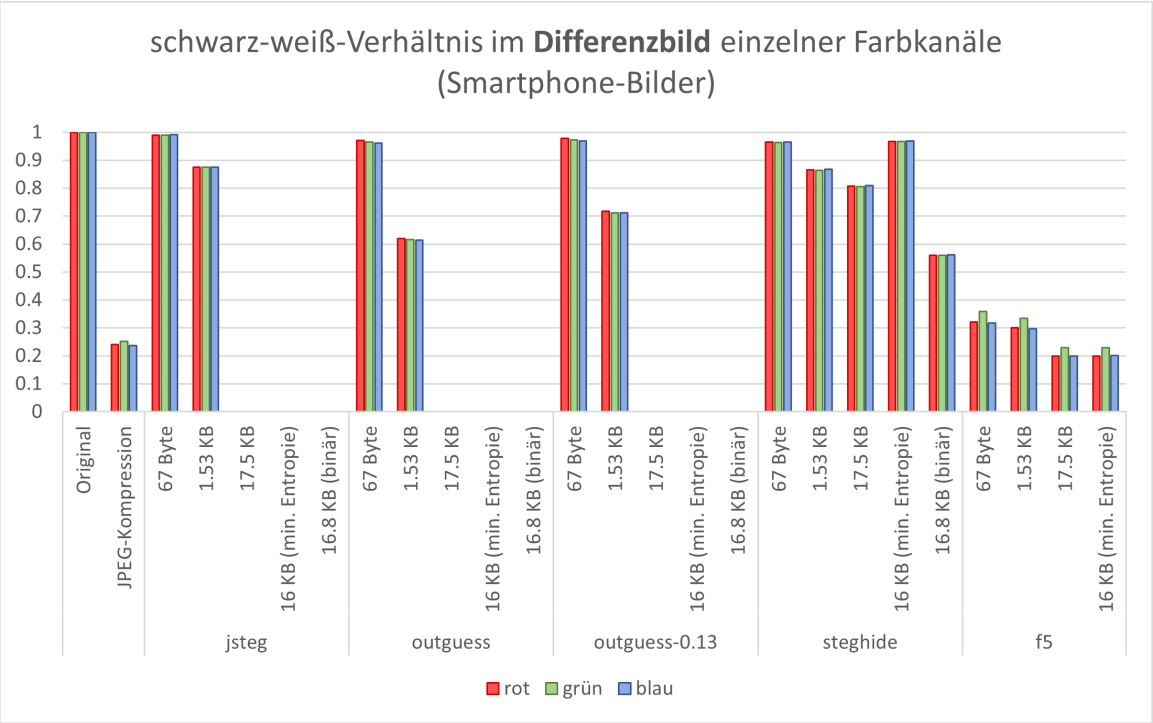


Abbildung C.18. Farbkanal-Differenzbilder von 187 betrachteten Smartphone-Kamera-Bildern