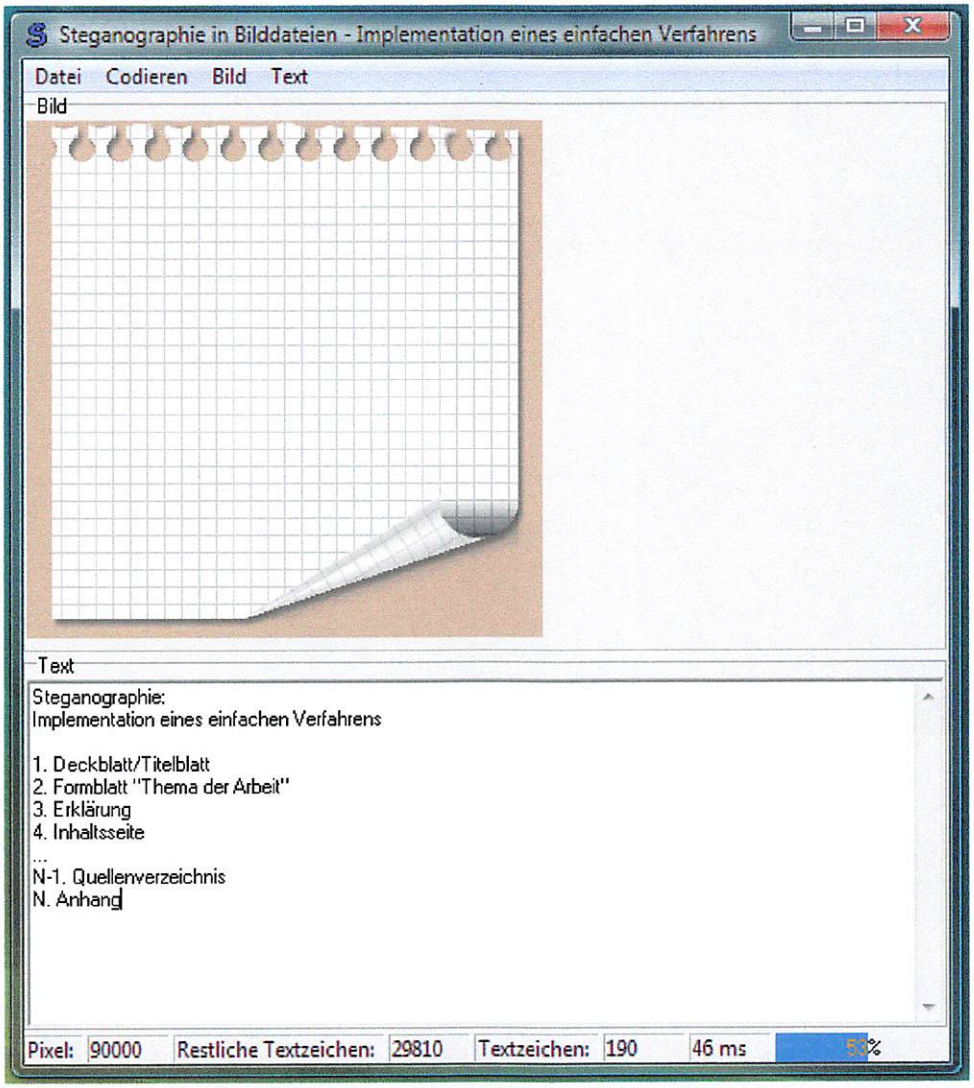
Steganographie -

Implementation eines einfachen Verfahrens



Facharbeit in Informatik

Geschrieben von

Marcel Carlé

Möhnesee 2008

Inhaltsverzeichnis

1. Steganographie

1.1 Was ist Steganographie?

Die Steganographie ist eine sehr alte Kunst, die bis ins antike Griechenland zurück zu verfolgen ist. Auch das Wort Steganographie kommt Ursprünglich aus dem Altgriechischen, und bedeutet übersetzt in etwa ‚geheimes Schreiben'. Daraus lässt sich die Zielsetzung der Steganographie ableiten, nämlich die Geheimhaltung von Information durch das Verbergen der Geheimhaltung oder auf Englisch „to embed confidential messages by inconspicuously changing the cover material"[1]. Sie wird häufig der Kryptographie zugeordnet, aber auch von einigen Wissenschaftlern als eigenständiger Wissenschaftsbereich angesehen. Beides ist möglich, da die Kryptographie eine ähnliche Zielsetzung hat, nämlich die Geheimhaltung von Informationen, jedoch nicht genau dieselbe.

Da sich die Kryptographie gut mit der Steganographie ergänzen lässt, werden sie häufig in Kombination genutzt, um so effektiver zu verhindern, dass nicht autorisierte Dritte die Informationen erhalten.

Man unterscheidet in der Steganographie die technische und die linguistische. Im Folgenden werde ich einige Techniken der technischen und der linguistischen Steganographie vorstellen:

1.2 Linguistische Steganographie

Die linguistische Steganographie beschäftigt sich zum einem mit dem so genannten Open Code und zum anderen mit den Semagrammen.

1.2.1 Semagramm

Bei einem Semagramm werden kleinere Details einer unverfänglich aussehenden Nachricht hinzugefügt, wie z.B. ein paar Tintenpatzer, oder ähnliches, in einem Text. Ein Beispiel für ein Semagramm in einem Bild wäre z.B. eines einer Landschaft, wo kurze und lange Grashalme zu sehen sind, welche dann als Morsecode aufzufassen sind. Die Semagramme werden die „sichtlich getarnten Geheimschriften" genannt, weil ein geübtes Auge solch kleine, unwichtig wirkende Details entdecken kann.[2]

---

[1] Barni, Mauro: Information Hiding. 7th International Workship. Springer 2006; S.189

[2] Eckert, Claudia: IT-Sicherheit. Konzepte – Verfahren –Protokolle. 4. Auflage. Oldenbourg April 2006; S.284

1.2.2 Open Code

Bei dem Open Code (oder auch Jargon Code) werden Informationen in einer unverfänglich aussehenden Nachricht dem Empfänger mitgeteilt, indem vorher eine Absprache gemacht wurde, was welche Bedeutung hat. Also z.B. ein bestimmtes Symbol in einem Text oder ähnlichem, welches für ‚SOS' steht. Auch möglich wäre, dass nur jeder x-te Buchstabe betrachtet wird. Ein Beispiel:

> *„Liebe Kolleginnen! Wir genießen nun endlich unsere Ferien auf der Insel vor Spanien. Wetter gut, Unterkunft auch, ebenso das Essen. Toll! Gruß, M. K."* [3]

Das sich in dieser Botschaft der Hilferuf ‚SOS' verbirgt, findet man heraus indem man die Buchstaben einschließlich Satzzeichen bis zum nächsten Leerzeichen zählt. Ist diese Zahl ungerade, ergibt sich eine 0, ansonsten eine 1:

„Liebe Kolleginnen! Wir genießen nun endlich unsere Ferien
0 1 0 1 0 0 1 1
auf der Insel vor Spanien. Wetter gut, Unterkunft
0 1 0 0 1 1 1 1
auch, ebenso das Essen. Toll! Gruß, M. K."
0 1 0 1 0 0 1 1

Für die ersten 8 Wörter ergibt sich also der Binärcode 01010011, welcher dem Buchstaben ‚S' entspricht. Für die nächsten 8 Wörter erhält man den Binärcode 01001111, welcher dem ‚O' entspricht und für die letzten 8 Wörter ergibt sich wieder der Binärcode 01010011, also wieder ein ‚S'. So erhält man den Hilferuf ‚SOS'.
Open Codes nennt man die „unersichtlich getarnten Geheimschriften". [4]

1.3 Technische Steganographie

Die technische Steganographie beschäftigt sich mit den „unsichtbaren Geheimschriften". Dabei wird die Information so gut versteckt, dass sie unsichtbar ist und nur mit z.B. bestimmten Hilfsmitteln zu lesen ist. Dazu gehört z.B. eines der ältesten technischen Verfahren der Steganographie, die Geheimtinte. Verwendet man z.B. Milch oder Zwiebelsaft als Tinte, so kann die Botschaft erst unter Einwirkung von UV-Licht oder durch Erwärmung sichtbar werden.

---

[3] Baur, Jochen; Gaertner Christoph; Mlinar Mari: Steganographie (online). Aufruf: 10.03.2008 21 Uhr http://www.it.hs-esslingen.de/~schmidt/vorlesungen/kryptologie/seminar/ws9798/html/stega/stega-1.html
[4] Eckert, Claudia: IT-Sicherheit. Konzepte – Verfahren –Protokolle S.284

Schon vor etwa 2500 Jahren (490-425 v. Chr.) gab es erste Ansätze der technischen Steganographie. So berichtet der griechische Dichter Herodot, dass einem Sklaven der Kopf rasiert und anschließend eine Nachricht auf seinem Kopf tätowiert wurde. Nachdem die Haare neugewachsen waren, wurde dieser dann zu seinem Ziel gesandt. Nach einer Kopfrasur konnte die Zielperson die Nachricht wieder frei lesen.
Eine weitere recht bekannte Technik der Steganographie ist der Mikropunkt (auch *Mikrodot* genannt), welcher im zweiten Weltkrieg zum Einsatz kam. Auf ihm kann man Information im Umfang einer DIN A4-Seite speichern und ihn dann z.B. als i-Punkt in einem Brief tarnen.[5]

1.4 Moderne Steganographie

Heutzutage wird die Steganographie häufig bei Dateien am Computer genutzt. Durch das Entstehen des Internets, wodurch via Computer kommuniziert werden kann, entstand auch ein ganz neues Betätigungsfeld für die technische Steganographie. Da auch das Internet nicht sicher beim Versenden von Informationen ist, wurden neue Techniken erstellt, um Information unsichtbar zu machen. Anfangs versandten Firmen untereinander ständig unwichtige Daten und verstecken die wichtigen in diesem Haufen vom Datenmüll. Da dies aber sehr auffällig ist und das Internet anfangs nicht gerade billig war, ist diese Methode nicht gerade Ideal.
Somit wurden neue Techniken entwickelt und man stellte fest, dass man Informationen in ganz normalen Daten, wie z.B. Bildern, speichern kann, ohne dass es jemanden auffällt. Auch in Audiodateien kann man heutzutage Texte, Bilder oder andere Informationen speichern.
Durch die Raubkopien entwickelten Firmen das *„digital watermarking"* um in ihren Bildern, Musikdateien, Programmen, etc. einen Verweis zu verstecken, der die Datei eindeutig identifiziert. So kann mit dem richtigen Programm untersucht werden, ob z.B. die Musikdateien aus dem Netz illegal runtergeladen wurden, oder legal gekauft wurden. Der Kopierschutz bei einem Spiel, wo vor dem starten des Spieles geguckt wird, ob die Original DVD im Laufwerk liegt, gehört ebenfalls zur Technik des Digitalen Wasserzeichens.
Diese Technik befindet sich zwar schon länger auf dem Markt, ist aber noch lange nicht ausgereift, da trotz Seriennummer und Kopierschutz fast jedes Programm illegal zu haben ist. Daher bedarf es noch weitere Forschungen in diesem Gebiet.

---

[5] Eckert, Claudia: IT-Sicherheit. Konzepte – Verfahren –Protokolle S.285

2. Die Bitmap Datei

Eine Bitmap Datei ist eine Bilddatei, die weit bekannt ist. Sie gehört zu den besten und weit Verbreitesten Grafikformaten, aber verbraucht sehr viel Speicherplatz.

2.1 Versionen

Von der Bitmap Datei gibt es mehrere Versionen, welche ich im Folgenden einmal kurz vorstellen möchte.

Die erste Version wurde von Microsoft für das Betriebssystem OS/2 entwickelt. Diese Version hat schon eine gewisse Ähnlichkeit zum heutigen Bitmap Format, hatte aber schlechte Kompressionsmethoden und das Farbsystem ist umgekehrt im Vergleich zum heutigen, nämlich BGR statt RGB.

Nachdem sich IBM und Microsoft für die Entwicklung des OS/2 zusammengeschlossen hatten, entwickelten sie die zweite Version des Bitmaps. Diese stellte verbesserte Kompressionsmethoden zur Verfügung, die allerdings auch nicht viel mehr Komprimierten. Weiterhin konnte man „das Bitmap-Format so erweitern, dass nicht nur Grafiken, sondern auch Icons und Zeiger damit abgelegt werden können".[6]

1991 trennte sich Microsoft von IBM um sich weiter der Windows-Weiterentwicklung zu widmen. Für die Benutzeroberfläche Windows 3.0 wurde eine neue Version, die dritte, des Bitmaps entwickelt, das Windows Bitmap. Diese Version verfügt über eine RLE8 und eine RLE4 Kompressionsmöglichkeit, welche aber selten benutzt werden und auch nur bei Bildern mit Flächenweise dem gleichen Farbwert. Aufgrund von der Beliebtheit von Microsofts Betriebssystemen wurde das Bitmap zu einem weit verbreiteten Grafikformat.

Die vierte Version entwickelte Microsoft für Windows 95 und NT4 und die fünfte für Windows 98 und 2000.

Im Folgenden werde ich auf den Aufbau der meist gebrauchten Version des Bitmaps, der Version 3, eingehen.

---

[6] Holtorf, Klaus: Das Handbuch der Grafikformate. 3. Neubearbeitete und erweiterte Auflage. Franzis-verlag Feldkirchen 1996; S. 52

2.2 Aufbau (Version 3)

Das Bitmap ist in verschiedene Strukturen unterteilt, die der Reihenfolge nach aus dem Bitmap ausgelesen werden müssen. Die erste Struktur bildet der Dateikopf, der Bitmapfileheader, in welchem Grundinformationen zum Bitmap gespeichert werden. Als nächstes folgt ein Informationsblock, welcher Bitmapinfo genannt wird. Diese Struktur hat weitere Unterstrukturen, nämlich die Bitmapinfoheader- und die RGBQuad-Struktur. Letzteres beinhaltet die eigentlichen Bilddaten.
Auf den Aufbau der Strukturen Bitmapfileheader, Bitmapinfoheader und RGBQuad werde ich im Kommenden kurz eingehen und die Funktionen einzelner Strukturen erklären.

2.2.1 Bitmapfileheader

Der Bitmapfileheader enthält Informationen über den Typ, die Größe und das Layout des Bitmaps.

> *„The Bitmapfileheader structure contains information about the type, size, and layout of a file that contains a DIB."* [7]

Als erstes wird in einer Bitmap-Datei die Dateierkennung gespeichert, also der Typ des Bildes. Dieser Wert muss immer ‚BM' sein, da es ansonsten keine Bitmap-Datei wäre. Danach folgt die Speichergröße des gesamten Bitmaps in Bytes. Die nächsten zwei Speicherplätze sind reserviert und müssen 0 sein. Im letzten Teil wird die Speichergröße vom Bitmapfileheader bis hin zu den Bilddaten gespeichert.[8]

2.2.2 Bitmapinfoheader

Im Bitmapinfoheader werden Informationen über die Abmessungen und dem Farbsystem des Bitmaps gespeichert.

> *„The Bitmapinfoheader structure contains information about the dimensions and color format of a DIB."* [9]

Den ersten Speicherplatz belegt die Größe des Bitmapinfoheaders. Im den nächsten Plätzen werden Breite und Höhe gespeichert. Danach folgen die Anzahl der Ebenen, welche momentan immer eine ist, die Anzahl der Bits pro Pixel (z.B. 24), die Komprimierung (ohne, RLE8 oder RLE4), die Größe des Bildes in Byte, die

---

[7] MSDN bzw. Microsoft, Bitmapfileheader (Online).
http://msdn2.microsoft.com/en-us/library/ms532321(VS.85).aspx (11.03.08)
[8] Born, Günter: Dateiformate – Die Referenz. 1Auflage. Galileo Press GmbH, Bonn 2001 S.611
[9] MSDN bzw. Microsoft, Bitmapinfoheader (Online).
http://msdn2.microsoft.com/en-us/library/ms532290(VS.85).aspx (11.03.08)

Horizontale und Vertikale Auflösung in Pixel pro Meter, die Anzahl der benutzen Farben und zum Schluss die Anzahl der wichtigen Farben.[10]

2.2.3 RGBQuad

In der RGBQuad Struktur werden einzelne Rot, Grün und Blau Werte gespeichert, welche zusammen eine Farbe ergeben. Diese Struktur ist für die Bilddaten verantwortlich. Nur in dieser Struktur werden die Farben eines Pixels gespeichert, welche hinterher für den Betrachter des Bildes sichtbar sind.

> *„The RGBQUAD structure describes a color consisting of relative intensities of red, green and blue."* [11]

Wie oben schon erwähnt, werden in dieser Struktur die Farbwerte gespeichert. Zuerst der Rot, dann der Grün und dann der Blau Wert. Zuletzt folgt noch ein reservierter Speicherplatz, der immer 0 sein muss.

3. Der Binärcode und die ASCII Zeichen

3.1 Was ist der Binärcode?

Der Binärcode besteht aus einer Reihe von Bits. Da ein Bit entweder den Wert 0 oder 1 hat, entsteht so z.B. folgender 8-Bit Binärcode: 01001011. Einen 8-Bit Binärcode nennt man auch Byte. Diese Bytes, bzw. Bits werden an Computern als Speichereinheiten bei Daten angegeben. Daraus lässt sich schon folgern, dass man durch eine Reihe von Bits, also einem Binärcode, Daten speichern kann. Es bestehen insgesamt 256 Möglichkeiten wie ein Byte aussehen kann.
Nimmt man 24 Bit, könnte man einen Bildpunkt in einem 24-Bit Bitmap darstellen. So steht der 24-Bit Binärcode ‚11111111 11111111 11111111' für die Farbe Weiß. Rechnet man ein 8-Bit Binärcode in den ASCII Code um, könnte man z.B. ein Satzzeichen darstellen. Zum Beispiel stellt der Code 01001011 den Buchstaben ‚K' dar. Dieses System nennt man das Binärsystem, oder auch Dualsystem oder Zweiersystem.

3.2 Was ist ASCII?

Die Abkürzung ASCII kommt aus dem Englischen und steht für **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange. ASCII ist eine Zeichenkodierung, die Standardmäßig aus 7-Bit Binärcodes besteht. Sie definiert 128 Zeichen, wovon 33

---

[10] Born, Günter: Dateiformate – Die Referenz. S.612
[11] MSDN bzw. Microsoft, Bitmapinfoheader (Online).
http://msdn2.microsoft.com/en-us/library/ms532316(VS.85).aspx (11.03.08)

nicht-druckbar und 95 druckbar sind. Die Zeichen umfassen das lateinische Alphabet in Groß- und Kleinschreibung, sowie die zehn arabischen Ziffern und einige Satz- und Steuerzeichen. Jedem Zeichen wird ein 7-Bit Code zugeordnet.[12]
Da einige Sprachen Sonderzeichen verwenden, wie z.B. die deutschen Umlaute, war eine Erweiterung auf 8-Bit Codes nötig. Die dadurch neuen 128 Zeichen wurden für jedes Land spezifisch festgelegt, sodass derselbe 8-Bit Code in zwei verschiedenen Ländern ein völlig unterschiedliches Zeichen sein kann. Insgesamt hat man durch die 8-Bit Codes 2^8 = 256 Zeichen. Im folgenden Schritt erläutere ich, wie die Umrechnung vom Binärcode zu ASCII und von ASCII zum Binärcode erfolgt. [13]

3.3 Umrechnung vom Binärcode zu ASCII

Hat man den Binärcode 00000000, so ist der ASCII Code dazu die 0, bei dem Binärcode 11111111 ist der zugehörige ASCII Code 256. Da bei jeder Ziffer nur zwei Möglichkeiten bestehen, entweder ist sie 0 oder 1, ist die maximale Anzahl verschiedener 8-Bit Binärcodes gleich 2^8 = 256. Daraus ergibt sich für die erste Ziffer, also die ‚**1**'1111111 eine Wertigkeit von 2^7, für die zweite 2^6 und so weiter. Folglich hat die letzte Ziffer eine Wertigkeit von 2^0 = 1. Wenn eine Ziffer eine 0 ist, so ist deren Wertigkeit auch 0. Rechnet man alle ergebenen Zahlen zusammen, erhält man den ASCII Code. [14]
Ein Beispiel:
Binärcode: 1 0 1 0 1 0 1 1
ASCII Code: $2^7 + 0 + 2^5 + 0 + 2^3 + 0 + 2^1 + 2^0 = 128+32+8+2+1 = 171$

3.4 Umrechnung von ASCII zum Binärcode

Die Umrechnung von ASCII zum Binärcode ist sehr schnell geschehen. Hat man einen ASCII Code, so schaut man ob er durch 2 zu teilen ist, also ob er ‚gerade' ist. Ist dies der Fall, so teilt man ihn durch 2 und fügt dem Ergebnis eine 0 hinzu. Ist er ungerade und lässt sich ohne Kommazahlen nicht durch 2 teilen, fügt man dem Ergebnis eine 1 hinzu und teilt den Code durch 2 und rundet das Ergebnis ab. Mit dem neuen Ergebnis macht man genau dasselbe und zwar solange bis das Ergebnis 0 ist.

---

[12] Stamm. Tobias: Zahlensysteme (Online). Erstellt 2003, letztes Update: 31. Januar 2008 http://mandalex.manderby.com/z/zahlensysteme.php (11.03.08 21 Uhr)
[13] Stamm, Tobias: ASCII (Online). http://mandalex.manderby.com/a/ascii.php (11.03.08 21 Uhr)
[14] Külgen, Holger: Dual-Dezi (Online). Erstellt 2003, letztes Update: 10. März 2008, Aufruf: 11.03.08 21 Uhr http://www.its05.de/computerwissen-computerhilfe/programmierung/zahlensysteme/dual-dezi_umrechnung.html

Hier ein Beispiel:

ASCII Code: 159	Binärcode:
1. Schritt: 159 / 2 = 79	1
2. Schritt: 79 / 2 = 39	1
3. Schritt: 39 / 2 = 19	1
4. Schritt: 19 / 2 = 9	1
5. Schritt: 9 / 2 = 4	1
6. Schritt: 4 / 2 = 2	0
7. Schritt: 2 / 2 = 1	0
8. Schritt: 1 / 2 = 0	1

Der ergebene Binärcode ist jedoch nicht 11111001, sondern 10011111. Folglich bestimmt man den Binärcode von Hinten nach Vorne, also bei der letzten Ziffer angefangen. Auch müssen keine 8-Bit langen Binärcodes das Ergebnis sein. Bestes Beispiel liefert der ASCII Code 0. Der Binärcode zur Dezimalzahl 0 ist 0. Zur Dezimalzahl 333 ist der Binärcode 9 Stellig, nämlich 101001101.[15]

3.5 Delphi Umrechnung

Im Folgenden erkläre ich meine Quellcodes zur Umrechnung von ASCII zu Binär und umgekehrt. Da die Quellcodes etwas länger sind, werde ich nur auf die wichtigen Abschnitte eingehen, die kompletten Quellcodes sind aber im Anhang zu finden.

3.5.1 Delphi - Umrechnung von ASCII zum Binärcode

```
while z>=1 do
begin
  b:=inttostr(z mod 2)+b;
  z:=z div 2;
end;
```

In dieser Schleife wird die Dezimalzahl (bzw. Der ASCII Code) solange durch 2 geteilt, bis sie kleiner als 1 ist. Weiterhin wird dem Ergebnis-String b eine 0 hinzugefügt, wenn die Dezimalzahl durch 2 Teilbar ist, ansonsten eine 1. Dies wird mit ‚z mod 2' festgestellt. Nachdem diese Schleife durchgelaufen ist, gibt die Funktion den Binärcode zurück.

---

[15] Külgen, Holger: Dezi-Dual (Online). Aufruf: 11.03.2008 21 Uhr
http://www.its05.de/computerwissen-computerhilfe/programmierung/zahlensysteme/dezi-dual_umrechnung.html

3.5.2 Delphi - Umrechnung vom Binärcode zu ASCII

```
count:=length(z);    //steht für die Hochzahl
for i:=1 to length(z) do
begin
  dec(count);        //die nächste Stelle hat eine niedirgere Hochzahl
  a:=a+strtoint(z[i])*round(intpower(2,count));
end;
```

Count steht für die Hochzahl der jeweiligen Stelle und wird am Anfang auf die Länge des Binärcodes gesetzt. Die Schleife läuft solange, bis die Schleifen-Variable i die Länge des Binärcodes erreicht hat. Solange wird zuerst die Hochzahl runter gezählt und dann die Wertigkeit der Stelle errechnet und dem Ergebnis a hinzugefügt. Am Ende übergibt die Funktion das Ergebnis.

4. Umsetzung der Steganographie in Bitmaps

Wie unter 2.2.3 schon erklärt, besteht ein Bitmaps aus vielen einzelnen Pixeln. Die Farbe dieser Pixel wird durch den Rot-, Grün- und Blau-Wert bestimmt. Wandelt man diese Werte in einen Binärcode um und ändert bei jedem Wert den letzten Bit, so wird für das menschliche Auge kein Unterschied in den Farben bestehen. Genau dies macht man sich zu nutzen um Informationen im sogenannten LSB zu speichern. Im Folgenden werde ich zuerst den LSB vorstellen und danach eine von vielen Möglichkeiten, wie man einen Buchstaben in ein Bitmap schreibt und wie man diesen wieder heraus liest.

4.1 Der LSB

Die Abkürzung LSB steht für *‚least significant bit'* und bedeutet Übersetzt in etwa *‚das niedrigstwertige Bit'*. Der LSB stellt also den Bit mit dem niedrigsten Stellenwert dar. Dieser ist immer das letzte Bit, da dessen Stellenwert $2^0 = 1$ ist. Bei einem 4-Bit Code wäre dies das also vierte Bit.

4.2 Schreiben von Buchstaben in Bitmaps

Um einen Buchstaben in einem Bitmap zu speichern wird erstmal der Binärcode dieses Buchstaben gebraucht. Diesen erhält man, indem man seinen ASCII Code bestimmt und dann nach 3.5.1 umrechnet. Dieser Binärcode besteht aus 8 Bit, man müsste also in einem Bitmap 8 Bit verändern, damit dieser Buchstabe in dem Bitmap gespeichert ist. Also nimmt man 3 benachbarte Pixel, teilt sie in ihre RGB Werte auf und wandelt sie in ihre Binärcodes um. So erhält man insgesamt 9 Binärcodes mit

jeweils einem LSB den man verändern kann, ohne dass das menschliche Auge eine Veränderung bemerkt. Man nimmt nun die erste Stelle des Binärcodes vom Text und speichert diese in die letzte des Binärcodes von Rot vom ersten Pixel. Dies macht man solange weiter, bis man am Ende des Binärcodes vom Buchstaben ist. Da dieser Binärcode aus 8-Bit besteht, verändert man z.B. den Blau-Wert vom letzten Pixel nicht. Hier ein kurzes Beispiel:

Buchstabe: A →ASCII Code: 65 → Binärcode: 01000001

Pixel: 1) R: 11100101 G: 00011100 B: 01100110
2) R: 11011010 G: 00011011 B: 00111111
3) R: 10111100 G: 00100000 B: 01100010

Speichert man nun den Buchstaben in diese drei Pixel, so ergibt sich:

Pixel: 1) R: 11100100 G: 00011101 B: 01100110
2) R: 11011010 G: 00011010 B: 00111110
3) R: 10111100 G: 00100001 B: 01100010

Die rot markierten Bits stellen die veränderten LSB's dar.

Man kann solange in einem Bild Buchstaben reinschreiben, wie drei noch nicht beschriebene Pixel vorhanden sind. Also beträgt die maximale Anzahl der Buchstaben ein Drittel der Pixelanzahl.

4.3 Lesen von Buchstaben aus Bitmaps

Um einen Buchstaben aus einem Bitmap raus zu lesen, muss man sich 3 benachbarte Pixel nehmen und diese in ihre RGB Werte auflösen. Diese Wandelt man dann in die zugehörigen Binärcodes um. Nun nimmt man vom ‚roten Binärcode' des ersten Pixels das LSB und schreibt es zum Ergebnis dazu. Danach nimmt man das LSB vom ‚grünen Binärcode' des ersten Pixels und schreibt dies dahinter. Dies macht man so weiter, jedoch lässt man das LSB vom ‚blauen Binärcode' des dritten Pixels aus, da dort nie etwas geändert wird, weil dieses Bit beim Schreiben immer übersprungen wird. Nachdem man den Binärcode ermittelt hat, wandelt man ihn nach 3.5.2 in den ASCII Code um und diesen danach in den zugehörigen Buchstaben. Hier ein kurzes Beispiel:

Pixel: 1) R: 00000010 G: 00000011 B: 11110110
2) R: 00001001 G: 00000100 B: 10101111
3) R: 11001101 G: 00101111 B: 00010010

Nun betrachtet man die blau markierten LSB's und schreibt sie nacheinander auf.

Binärcode: 01010111 → ASCII: 87 →Buchstabe: W

4.4 Umsetzung in Delphi

Im Folgenden werde ich auf einige Schritte der Codierung (dem Schreiben) und der Decodierung (dem Lesen) eingehen. Da der Gesamte Quellcode etwas lang ist, sind hier nur einige wichtige Schritte aufgelistet. Der Komplette Quellcode ist aber in der CD zu finden.

a)

```
r:=getrvalue(bmp.Canvas.Pixels[x,y]);
g:=getgvalue(bmp.Canvas.Pixels[x,y]);
b:=getbvalue(bmp.Canvas.Pixels[x,y]);
```

In diesem kleinen Quellcode-Ausschnitt werden die RGB Werte aus dem Pixel an der Position [X, Y] in die Variablen r, g und b gespeichert.

b)

```
bin:=Convert.ToBinaer(r);
bin[8]:=txtbin[binc];
r:=Convert.ToAscii(bin);
```

Hier wird der Rot-Wert zuerst in einen Binärcode umgewandelt. Danach wird dessen LSB (das 8.Bit) durch das Bit an der aktuellen Position im Binärcode des Buchstabens ersetzt. Danach wird der Binärcode wieder in ASCII umgewandelt und in der Variablen r gespeichert.

c)

```
bmp.Canvas.Pixels[x,y]:=RGB(r,g,b);
```

Die mit Schritt 2 neu ermittelten Farben werden in ihr Pixel geschrieben.

d)

```
s:=convert.ToBinaer(r1)[8]+convert.ToBinaer(g1)[8]+
   convert.ToBinaer(b1)[8]+convert.ToBinaer(r2)[8]+
   convert.ToBinaer(g2)[8]+convert.ToBinaer(b2)[8]+
   convert.ToBinaer(r3)[8]+convert.ToBinaer(g3)[8];
d:=d+chr(convert.ToAscii(s));
```

Nachdem die einzelnen Rot-, Grün- und Blau-Werte von drei Pixeln in die Variablen r1, r2, r3 und g1, g2, g3 und b1 und b2 gespeichert wurden, wird ihr Binärcode ermittelt und dem Ergebnis ihr LSB (8. Bit) hinzugefügt. Am Ende erhält man dann einen neuen Binärcode, welchen man in den zugehörigen ASCII Code umwandelt und dessen zugehörigen Buchstaben dem Ergebnis hinzufügt.

5. Aufspüren von Steganographie

Wenn man sich das Verfahren der Steganographie anschaut, erscheint es doch wirklich sehr gut zu sein und auch nicht so leicht auffindbar. Leider ist das Gegenteil der Fall, denn durch die Veränderung des LSB verändert man das Rauschen eines Bildes. Siehe dazu im Anhang auf Bild i). Dies ist das Rauschen des Bildes auf dem Deckblatt im umbeschriebenen Zustand. Danach habe ich irgendeinen Text aus dem Internet in das Bild geschrieben. Das Rauschen dieses Bildes ist das Bild ii) im Anhang.

Bei dem ersten Bild kann man deutlich sehen, wo im Originalbild gleichfarbige Flächen sind. Beim zweiten Bild kann man diese zwar auch sehen, aber es liegt ein starkes Rauschen über diesen Flächen.

Wenn man also eine Botschaft in ein Bild speichern will, sollte das Bild am besten keine, oder auch wenige kleine, Flächen derselben Farbe haben. Weiterhin sollte man das Bild nicht mit sich immer wiederholenden Sätzen, Wörtern oder Buchstaben vollschreiben. Warum zeigt das Bild iii) im Anhang, denn dort sieht man im Rauschen die Regelmäßigkeit der Sätze.[16]

Aber trotzdem kann man schon sagen, dass die Steganographie ein teils sicheres Verfahren ist, denn wer kommt auf die Idee jedes Bild, jede Audiodatei, etc. zu überprüfen ob sie ein ungewöhnliches Rauschen aufweist? Wahrscheinlich eher wenig Leute. Weiterhin ist die Nachricht mit der Rauschprüfung auch noch nicht aus dem Bild entnommen. Dies geht zwar bei meinem Verfahren die Zeichen ins Bitmap zu speichern relativ leicht und schnell, aber bei anderen Verfahren müssten z.B. Berechnungen des nächsten Pixels, oder ähnliches, gemacht werden.

Und wenn einem die Steganographie alleine trotzdem nicht genügend Sicherheit bietet, kann man den zu speichernden Text auch mit Hilfe von kryptografischen Verfahren verschlüsseln bevor man ihn ins Bild schreibt.

Insgesamt denke ich auch, dass die Steganographie in Dateien schon sehr sicher ist, da niemand auf die Idee kommt und 15.000 Dateien untersucht, solange man nicht gerade ein Schwerverbrecher ist.

---

[16] Westfeld, Andreas: Unsichtbare Botschaften. c't Magazin 9/2001. Heise Zeitschriftenverlag, Hannover 2001

6. Quellenverzeichnis

Barni, Mauro: Information Hiding. 7th International Workship. Springer 2006

Born, Günter: Dateiformate – Die Referenz. 1Auflage. Galileo Press GmbH, Bonn 2001

Eckert, Claudia: IT-Sicherheit. Konzepte – Verfahren –Protokolle. 4. Auflage. Oldenbourg April 2006

Holtorf, Klaus: Das Handbuch der Grafikformate. 3. Neubearbeitete und erweiterte Auflage. Franzis-verlag Feldkirchen 1996

Westfeld, Andreas: Unsichtbare Botschaften. c't Magazin 9/2001. Heise Zeitschriftenverlag, Hannover 2001

Internetquellen

Hochschule Esslingen:
http://www.it.hs-esslingen.de/ (10.03.08 21 Uhr)

ITS05:
http://www.its05.de/ (11.03.08 21 Uhr)

Mandalex:
http://mandalex.manderby.com/ (11.03.08 21 Uhr)

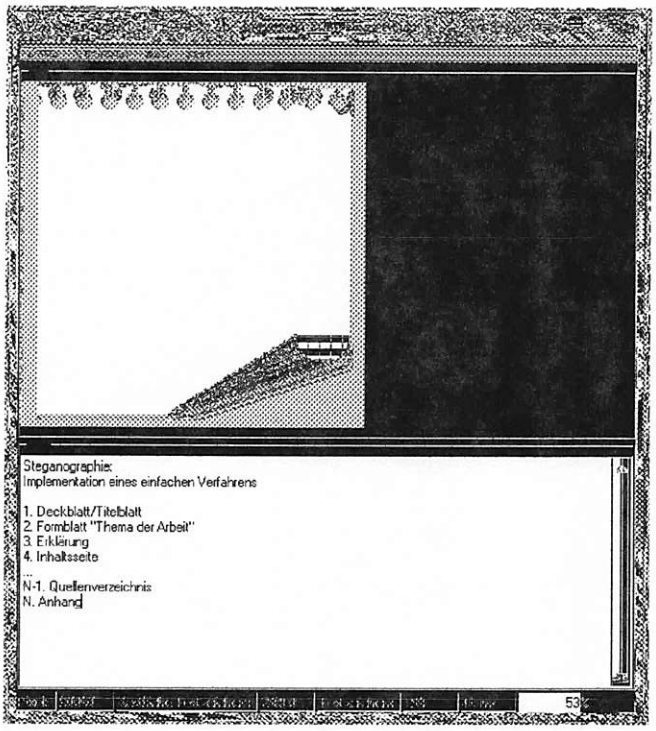
MSDN bzw. Microsoft:
http://msdn2.microsoft.com/ (11.03.08 21 Uhr)

Bildquelle

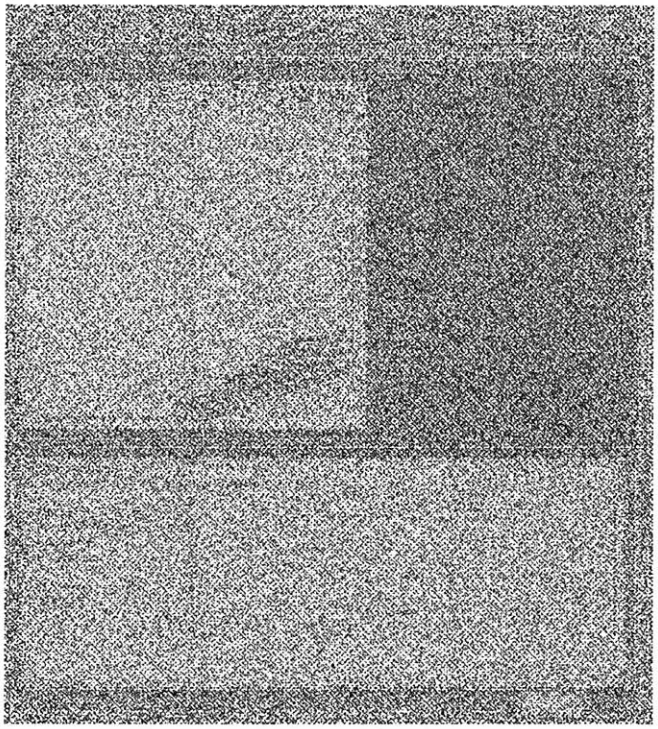
Photoshop-Lernen:
http://www.photoshop-lernen.de/Gratis%20Workshops/uwps07/karierter_zettel/PS_Karierter_Zettel14.gif (14.01.2008)
(Notizzettel aus Deckblatt)

1. Bilder

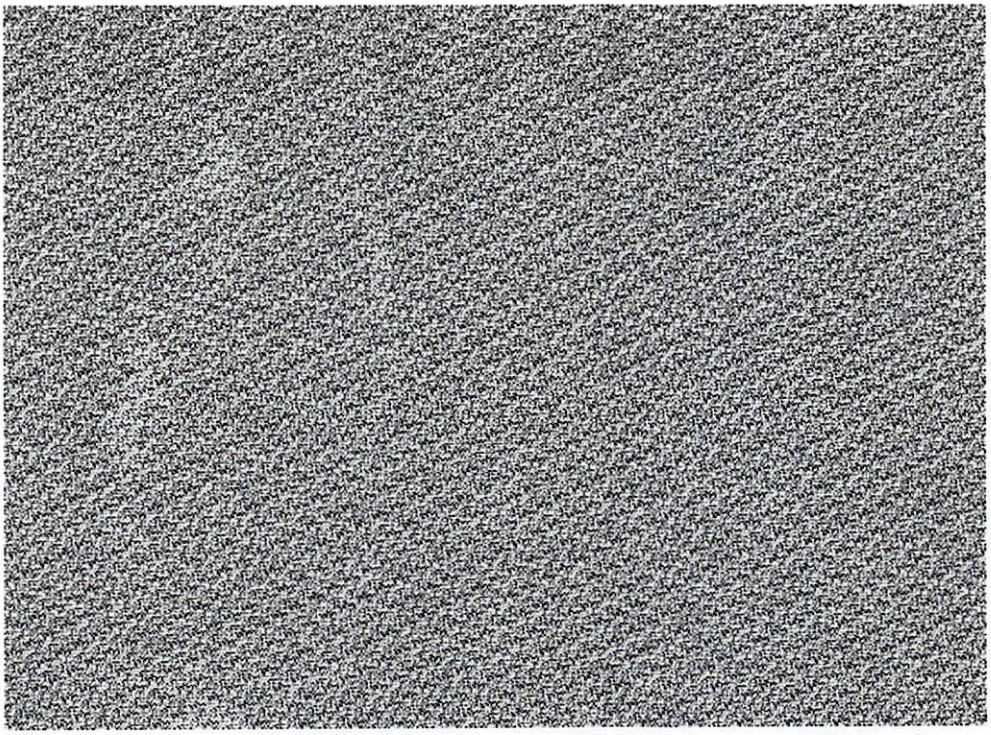
i)


Steganographie:
Implementation eines einfachen Verfahrens
1. Deckblatt/Titelblatt
2. Formblatt "Thema der Arbeit"
3. Erklärung
4. Inhaltsseite
...
N-1. Quellenverzeichnis
N. Anhang

ii)



iii)



2. Quelltexte

2.1 ASCII -> Binär

```
function KConvert.ToBinaer(Ascii:integer) : string;
//Der übergebene ASCII Code wird in seinen Binärcode umgerechnet
var z:integer;
    b:string;
begin
  b:='';
  z:=ascii;          //z bekommt den ASCII Code zugewiesen
  while z>=1 do      //Solange z Größer oder Gleich 1 ist
  begin
    //Dem Ergebnis wird eine Ziffer hinzugefügt. Ist z ungerade eine '1'
    //ist z gerade eine '0'
    b:=inttostr(z mod 2)+b;
    //z wird durch 2 geteilt und abgerundet (also: 5 Div 2 = 2)
    z:=z div 2;
  end;
  //Wenn der Binärcode weniger als 8 Stellen hat, wird er auf 8 Stellen erweitert
  while length(b)<8 do
    b:='0'+b;

  result:=b;
end;
```

2.2 Binär -> ASCII

```
function KConvert.ToAscii(Binaer:string) : integer;
//Der übergebene Binärcode wird in seinen ASCII Code umgerechnet
var count,i,a:integer;
    z:string;
begin
  z:=binaer;
  a:=0;
  count:=length(z);      //steht für die Hochzahl
  for i:=1 to length(z) do
  begin
    dec(count);          //die nächste Stelle hat eine niedirgere Hochzahl
    //Wenn an der Stelle i eine 0 steht wird dem Ergebnis 0 hinzugefügt.
    //Steht an der Stelle eine 1, so wird dem Ergebnis 2^count hinzugefügt.
    a:=a+strtoint(z[i])*round(intpower(2,count));
  end;
  result:=a;
end;
```