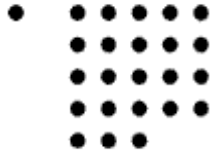


– Projektarbeit –

Im Fach Algorithmische Anwendungen

Gruppe „F_gelb_Ala0506“

Sebastian Skalec (11038991)
Stefan Kemper (11036985)

MD5 / JPEG–Integritätstest

Inhalt

1. Einführung

MD5 bezeichnet ein Verfahren, welches es ermöglicht, aus einem beliebigen Eingabedatenstrom einen 16 Byte bzw. 128 Bit langen Hashwert zu berechnen, auch „Message Digest“ genannt. Die Abkürzung „MD5“ bedeutet daher „Message Digest Algorithm 5“ (s. *Abb. 1*).

MD5 wurde als Ersatz für den als unsicher eingestuften Vorgänger MD4 ein Jahr nach der Veröffentlichung des letzten eingeführt. Er wurde von Prof. Ronald L. Rivest am MIT entwickelt.

Ziel des Algorithmus´ ist es, eine Eingabedatenmenge durch einen Hashwert zu repräsentieren, d.h. sozusagen einen „Fingerabdruck“ der Daten zu erzeugen, mit dem z.B. nach einer Datenübertragung überprüft werden kann, ob die Daten so angekommen sind, wie sie losgeschickt wurden.
Bei der Berechnung des Hashwertes muß also jedes Eingabebit mit möglichst gleicher Wirkung das Resultat beeinflussen.
Ein Beispiel verdeutlicht dies: unter Linux gibt es zum Berechnen von MD5-Hashwerten das Tool „`md5sum`“. Zwei geringfügig unterschiedliche Texte bewirken eine vollkommen unterschiedliche Ausgabe:

```
linux:~ # echo "Ein kleiner Text" | md5sum
9fdf5a6768d1e13946bf38467d420ead  -
```

Hingegen

```
linux:~ # echo "Ein kleiner Test" | md5sum
6679cdcb453072a6049a3b27f59b8f4d  -
```

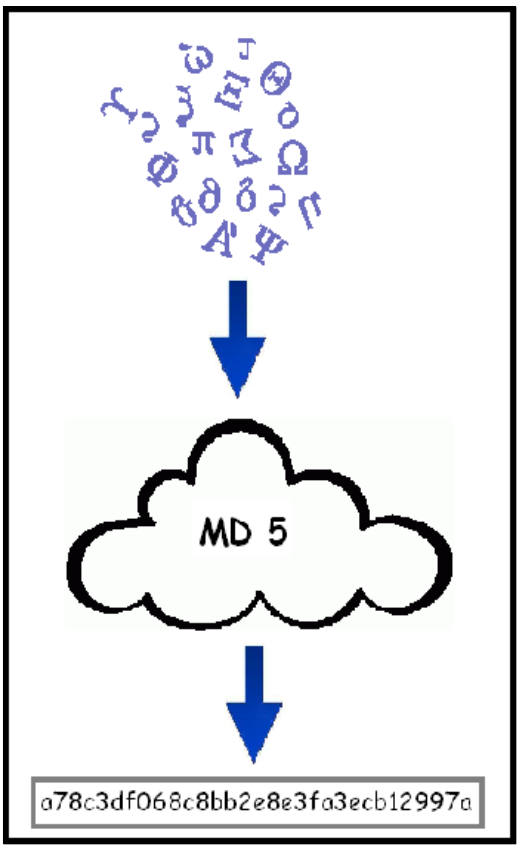


Abb. 1: Funktionsprinzip von MD5

MD5 in unserem Projekt

Im Rahmen dieses Projektes werden wir MD5 benutzen, um die Integrität, das heißt Unverändertheit, von JPEG-Bildern zu überprüfen. Als Eingabe dienen dabei der Datenbereich eines Bildes sowie ein geheimes Passwort. Der Hashwert wird im Kommentar-Bereich des Bildes untergebracht (ohne das Passwort könnte der Hashwert daher gefälscht werden).

Sinnvoll ist eine solche Anwendung z.B. bei Aufnahmen einer Überwachungskamera, bei denen es wichtig ist, sichergehen zu können, daß sie nicht verändert wurden.

Als primäre Quelle haben wir uns die Beschreibung von Ronald Rivest im RFC1321 (RFC = „Request for Comments“) zunutze gemacht. Dieses RFC stellt auch die Primärliteratur zu MD5 dar.

Ausserdem hilfreich waren die Artikel im deutschen und englischen „Wikipedia“, einer freien Internet-Enzyklopädie.

Quellenangaben

http://www.freesoft.org/CIE/RFC/1321/
http://de.wikipedia.org/wiki/Md5
http://en.wikipedia.org/wiki/Md5

2. Der Algorithmus

Die Beschreibung des Algorithmus´ ist in RFC1321 öffentlich verfügbar. Dieser Quelle entstammen auch die Informationen der folgenden Erörterungen.

Der Algorithmus lässt sich in fünf Teilschritte gliedern (s. RFC1321):

1. Padding-Bits anhängen
2. Länge der Eingabe anhängen
3. MD5 Buffer initialisieren
4. Eingabe in 512 Bit-Blöcken verarbeiten
5. Ausgabe

Schritt 1: Padding-Bits anhängen

Zur Verarbeitung wird die Nachricht in 512 Bit lange Blöcke aufgeteilt. Vorausschauend auf den nächsten Schritt wird ein Padding angefügt, so daß die Gesamtlänge modulo 512 Bit 448 Bit beträgt (s. *Abb. 2*). Somit bleibt ein Freiraum von 64 Bit. Das Padding wird auch ausgeführt, falls die Eingabe bereits die geforderte Länge besitzt.

Die Padding-Bits bestehen aus einer einzelnen '1' gefolgt von der benötigten Anzahl Nullen.

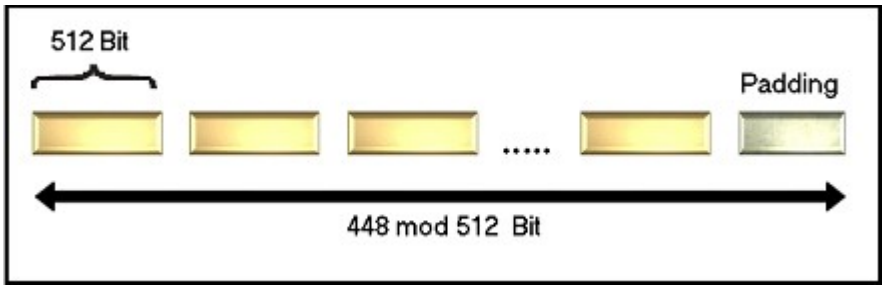


Abb. 2: Länge der Eingabe mit Padding

Schritt 2: Länge der Eingabe anhängen

An das Ende der Eingabedaten wird nun noch eine 64 Bit Integerzahl angehängt, welche die Länge der Eingabe (ohne Padding) ist, so daß die Gesamtlänge ein Vielfaches von 512 Bit ist (s. *Abb. 3*). Durch diese Maßnahme erhält die Nachricht einen individuelleren Charakter, was bei der Berechnung des Hashwertes vermutlich von Vorteil ist.

In dem unwahrscheinlichen Fall, daß die Länge der Eingabe länger als 2^{64} Bit (ca. 1,845 * 10^{19} Bit bzw. 16 Exa-Byte) ist, werden nur die unteren 64 Bit der Länge (modulo 2^{64}) angehängt.

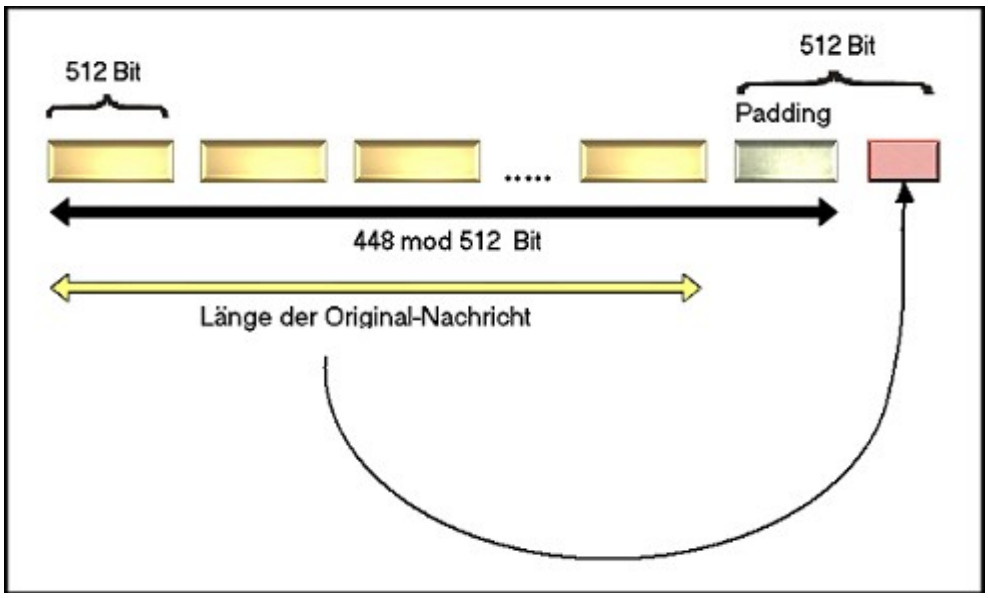


Abb. 3: Länge der Eingabe anhängen

Schritt 3: MD5 Buffer initialisieren

Als nächstes werden die vier Buffer, die am Ende des Algorithmus´ den Hashwert enthalten, mit vordefinierten Werten initialisiert. Die Buffer sind jeweils 32 Bit lang (Gesamtlänge dann 128 Bit) und ihr Inhalt in „Little Endian“ Reihenfolge interpretiert.
Die Länge der Register und deren Speicherungsart hängt damit zusammen, daß der Algorithmus konzipiert wurde, um auf 32-Bit Maschinen effizient zu arbeiten.

Die Vorbelegungen sind (Little Endian Notation, d.h. „lowest byte first“):

$A = (01\ 23\ 45\ 67)_h$
$B = (89\ ab\ cd\ ef)_h$
$C = (fe\ dc\ ba\ 98)_h$
$D = (76\ 54\ 32\ 10)_h$

Welche Überlegung zu diesen Werte geführt hat, konnten wir bis jetzt nicht feststellen. Es fällt nur auf das „A“ eine Primzahl ist und die restlichen Initialwerte zumindest relativ große Primzahlen als Faktor enthalten.

```
A = 1732584193
B = 4023233417 =  7 * 47 * 193 * 63361
C = 2562383102 =  2 * 29 * 44179019
D = 271733878 =  2 * 13 * 881 * 11863
```

Schritt 4: Eingabe in 512 Bit Blöcken verarbeiten

Die Eingabe wird in den im vorigen Schritt initialisierten „Registern“ verarbeitet. Darum werden die Eingabeblöcke (je 512 Bit) in 16 Worte (je 32 Bit) aufgeteilt. Die Verarbeitung geschieht in vier Runden für jeden der Eingabeblöcke.

Abb. 4: Verarbeitung der Eingabe

Für jede dieser Runden wird – in der gleichen Reihenfolge, wie aufgeführt – eine der folgenden binären Funktionen verwendet:

```
F(X,Y,Z) = XY v not(X) Z
G(X,Y,Z) = XZ v Y not(Z)
H(X,Y,Z) = X xor Y xor Z
I(X,Y,Z) = Y xor (X v not(Z))
```

In jeder Runde werden die Registerinhalte über eine für die jeweilige Runde spezielle binäre Funktion neu berechnet. Diese lautet für Runde (1):

```
[abcd k s i]  :

a = b + ((a + F(b,c,d) + X[k] + T[i]) <<< s)
```

In der zweiten Runde wird „**G**“, in der dritten „**H**“ und in der vierten „**I**“ anstelle von „**F**“eingesetzt (die Reihenfolge der Parameter **(b,c,d)** bleibt dabei gleich).

Das Feld **X[k]** bezeichnet das k-te Wort des gerade zu verarbeitenden Blocks. Das folgende Feld **T[i]** enthält vordefinierte Werte – und zwar ist

$$T[i] = \text{Int}(2^{32} \text{ mal } \text{abs}(\sin(i)))$$

In Worten: das i-te Element von **T** ist der Integerteil der Zahl, welche die 2^{32} – malige Berechnung des Absolutwertes des Sinus von „i“ ergibt (Sinus im Bogenmaß).

Das Ergebnis von **[abcd k s i]** wird dann dem Parameter, welcher Anstelle von „**a**“ übergeben wird, zugewiesen. Es folgt der aus RFC1321 entnommene Pseudocode des Verarbeitungsschrittes (Schritt 4) von MD5.

```
/* Process each 16-word block. */
  For i = 0 to N/16-1 do
    /* Copy block i into X. */
    For j = 0 to 15 do
      Set X[j] to M[i*16+j].
    end /* of loop on j */

    /* Save A as AA, B as BB, C as CC, and D as DD. */
    AA = A
    BB = B

    CC = C
    DD = D

    /* Round 1. */
    /* Let [abcd k s i] denote the operation
         a = b + ((a + F(b,c,d) + X[k] + T[i]) <<< s). */
    /* Do the following 16 operations. */
    [ABCD  0  7  1]  [DABC  1 12  2]  [CDAB  2 17  3]  [BCDA  3 22  4]
    [ABCD  4  7  5]  [DABC  5 12  6]  [CDAB  6 17  7]  [BCDA  7 22  8]
    [ABCD  8  7  9]  [DABC  9 12 10]  [CDAB 10 17 11]  [BCDA 11 22 12]
    [ABCD 12  7 13]  [DABC 13 12 14]  [CDAB 14 17 15]  [BCDA 15 22 16]

    /* Round 2. */
    /* Let [abcd k s i] denote the operation
         a = b + ((a + G(b,c,d) + X[k] + T[i]) <<< s). */
    /* Do the following 16 operations. */
    [ABCD  1  5 17]  [DABC  6  9 18]  [CDAB 11 14 19]  [BCDA  0 20 20]
    [ABCD  5  5 21]  [DABC 10  9 22]  [CDAB 15 14 23]  [BCDA  4 20 24]
    [ABCD  9  5 25]  [DABC 14  9 26]  [CDAB  3 14 27]  [BCDA  8 20 28]
    [ABCD 13  5 29]  [DABC  2  9 30]  [CDAB  7 14 31]  [BCDA 12 20 32]

    /* Round 3. */
    /* Let [abcd k s t] denote the operation
         a = b + ((a + H(b,c,d) + X[k] + T[i]) <<< s). */
    /* Do the following 16 operations. */
    [ABCD  5  4 33]  [DABC  8 11 34]  [CDAB 11 16 35]  [BCDA 14 23 36]
    [ABCD  1  4 37]  [DABC  4 11 38]  [CDAB  7 16 39]  [BCDA 10 23 40]
    [ABCD 13  4 41]  [DABC  0 11 42]  [CDAB  3 16 43]  [BCDA  6 23 44]
    [ABCD  9  4 45]  [DABC 12 11 46]  [CDAB 15 16 47]  [BCDA  2 23 48]

    /* Round 4. */
    /* Let [abcd k s t] denote the operation
         a = b + ((a + I(b,c,d) + X[k] + T[i]) <<< s). */
    /* Do the following 16 operations. */
    [ABCD  0  6 49]  [DABC  7 10 50]  [CDAB 14 15 51]  [BCDA  5 21 52]
    [ABCD 12  6 53]  [DABC  3 10 54]  [CDAB 10 15 55]  [BCDA  1 21 56]
    [ABCD  8  6 57]  [DABC 15 10 58]  [CDAB  6 15 59]  [BCDA 13 21 60]
    [ABCD  4  6 61]  [DABC 11 10 62]  [CDAB  2 15 63]  [BCDA  9 21 64]

    /* Then perform the following additions. (That is increment each
       of the four registers by the value it had before this block
       was started.) */
    A = A + AA
    B = B + BB
    C = C + CC
    D = D + DD

  end /* of loop on i */
```

Quelle: RFC1321

Im Pseudocode wird deutlich, wie die vorher beschriebenen binären Funktionen in den einzelnen Runden zur Anwendung kommen. Die große For-Schleife läuft für jeden der 512 Bit – Blöcke einmal durch (N ist daher die gesamte Anzahl der 32 Bit Worte der Eingabedaten).

Die dann folgende kleine For-Schleife kopiert die 16 Worte des jeweiligen Blocks in das Array „X“ („M“ steht für das Array der gesamten „Message“, d.h. der Eingabedaten, in Worten).

Das Einzige, was zusätzlich zu den Runden noch berechnet wird, ist die Summe (modulo 2^{32} Bit) von den Registerwerten vor - mit den Werten nach der Berechnung der Runden-Funktionen. Diese Summe beendet die Operationen, welche für einen Eingabeblock ausgeführt werden.

Schritt 5: Ausgabe

Nun folgt der dankbarste Schritt. Nach Beendigung aller Rundendurchläufe des Algorithmus muß nun nur noch der MD5-Hash ausgelesen werden (allerdings in umgekehrter Reihenfolge – von rechts (LSByte) nach links (MSByte). Mit der Rückgabe des Hashwertes in der üblichen hexadezimalen Darstellung ist der Algorithmus abgeschlossen.

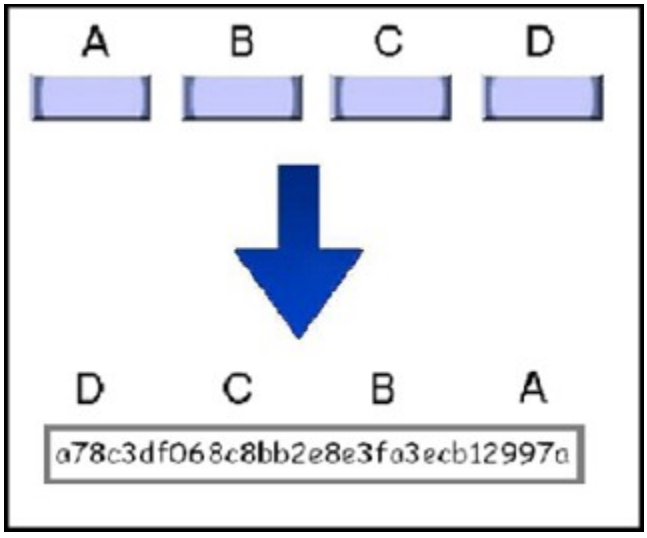


Abb. 5: Auslesen des MD5-Hashes

3. JPEG Integritätstest

Um die Integrität von Bildern mittels MD5 testen zu können, braucht man zusätzlich zu dem Bild einen aus den Bilddaten berechneten MD5-Hash (sozusagen Fingerabdruck des Bildes).

Dieser Hashwert soll im Kommentarbereich der Bilddatei untergebracht werden.

Da bei Bekanntsein des Verfahrens dieser Hashwert ohne weiteres gefälscht werden könnte, wird zusätzlich ein Passwort mit in den Hash gerechnet (das heißt, zusätzlich zu den Bilddaten als Eingabe für den Algorithmus bereitgestellt).

Abb. 6 zeigt ein Aktivitätsdiagramm für das Einfügen eines Hashwertes in ein JPEG-Bild. Die Kästen stellen die Aktivitäten des Benutzers / der Anwendung dar und sind kausal nacheinander angeordnet.

Der Benutzer wählt ein Bild, gibt sein Passwort ein, mit welchem dann – zusammen mit den Bilddaten – der MD5 Hashwert berechnet wird. Der Hashwert wird dann in das Bild eingefügt (wie, siehe nächste Seite). Dann erhält der Benutzer noch eine Erfolgsmeldung.

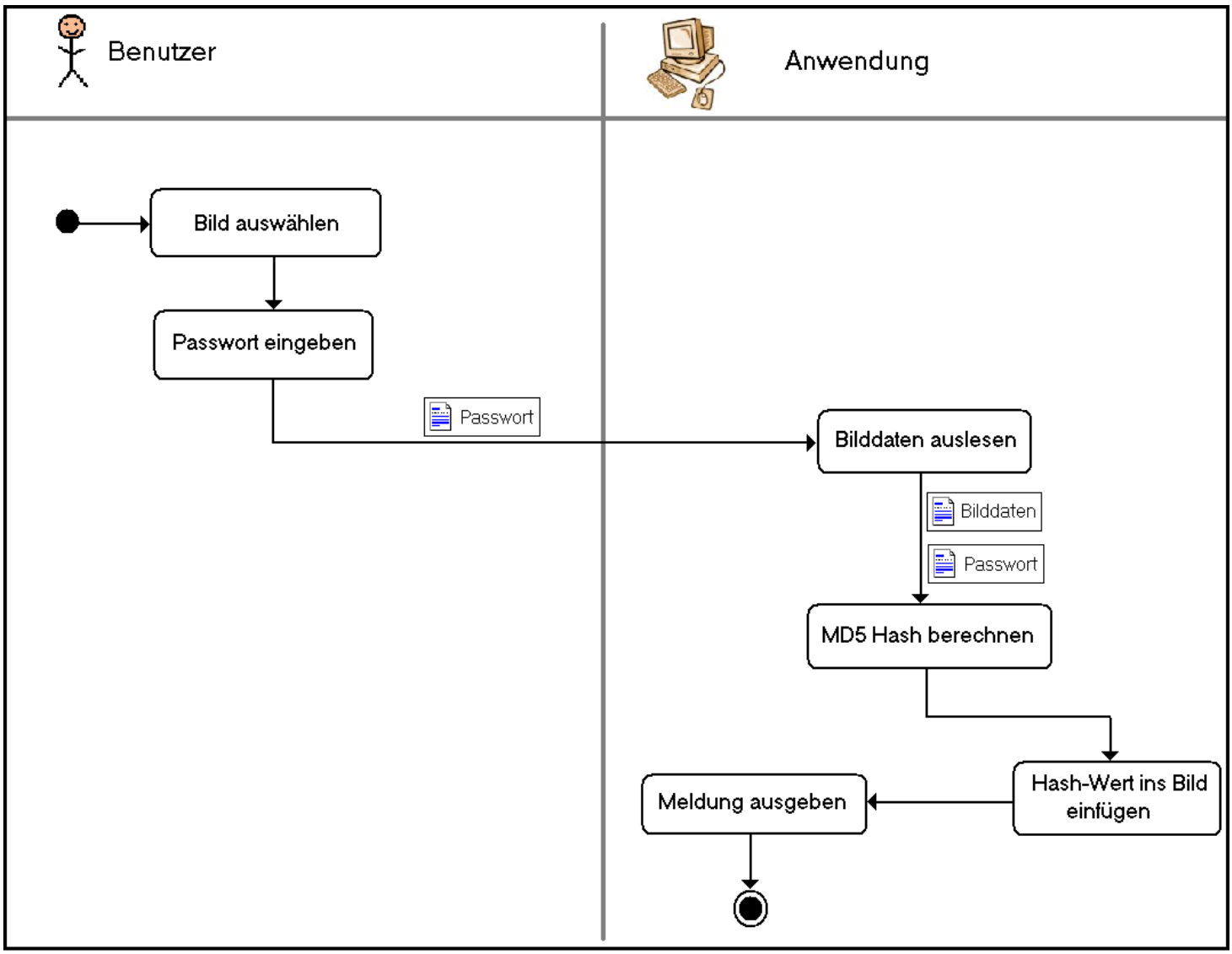


Abb. 6: Anwendungsfall: MD5-Hash berechnen und einfügen

Kurzer Exkurs: JFIF

JFIF steht für „*JPEG File Interchange Format*“ und bezeichnet eine Konvention für ein Dateiformat, welches die JPEG-Bilddaten enthält. Es beinhaltet ausserdem Mechanismen zur Synchronisation bei Übertragungsfehlern (daher „*Interchange Format*“) und ist in mehrere Bereiche gegliedert. Markiert sind diese Bereche im TLV (Tag, Length, Value) Stil. Jeder Tag, mit Ausnahme des SOI (Start Of Image) und EOI (End Of Image), besteht aus einem 2 Byte langen Bezeichner und einer ebensolangen Längenangabe (vorzeichenloses Integer). Die Längenangabe schließt die 2 dazu notwendigen Bytes ebenfalls ein.

Hinzu existieren noch weitere Tags zur Gliederung der Bilddaten. Diese Tags hatten für unsere Anwendung keine Bedeutung.

FF xx	Bezeichnung
FF D8	Start Of Image (SOI)
FF C0	Image Format Information (width, height etc.)
FF E0	JFIF tag
FF C4	Define Huffman Table (DHT)
FF DB	Define Quantisation Table (DQT)
FF E1	EXIF Daten
FF EE	Oft für Copyright Einträge
FF En	*n*=2..F allg. Zeiger
FF FE	Kommentare
FF DA	Start of Scan (SOS)
FF D9	End of Image (EOI)

Abb. 7: JFIF Tags (http://de.wikipedia.org/wiki/JFIF)

Wichtig war nur der Kommentar-Bereich „$fffe_h$“, den wir zur Unterbringung unseres Integritäts-Hashwertes benutzen wollten.

Falls das Bild bereits einen Kommentar enthält, muß der Benutzer mit einer entsprechenden Fehlermeldung darauf aufmerksam gemacht werden und falls kein Kommentar vorhanden ist, muß das Bild entsprechend verändert werden.

Abb. 8 zeigt das Funktionsprinzip des Anwendungsfalls „Hashwert überprüfen“ in einem weiteren Aktivitätsdiagramm. Der Benutzer wählt ein Bild aus und die Anwendung liest dessen Hashwert (bzw. Signatur) aus. Aus eingegebenem Passwort und den Bilddaten wird der Hashwert berechnet und mit dem ausgelesenen verglichen. Das Programm zeigt dem Benutzer eine entsprechende Meldung (Übereinstimmung / Unterschiedlichkeit).

Der Hashwert selbst wird aus der Datenmenge, aus der er berechnet wird (und dann auch enthalten ist), natürlich ausgeschlossen, weil sonst die Neuberechnung nicht mehr funktionieren würde. Zur Berechnung des Hashes werden die Bilddaten, welche in der Bilddatei auf den Kommentar folgen, benutzt.

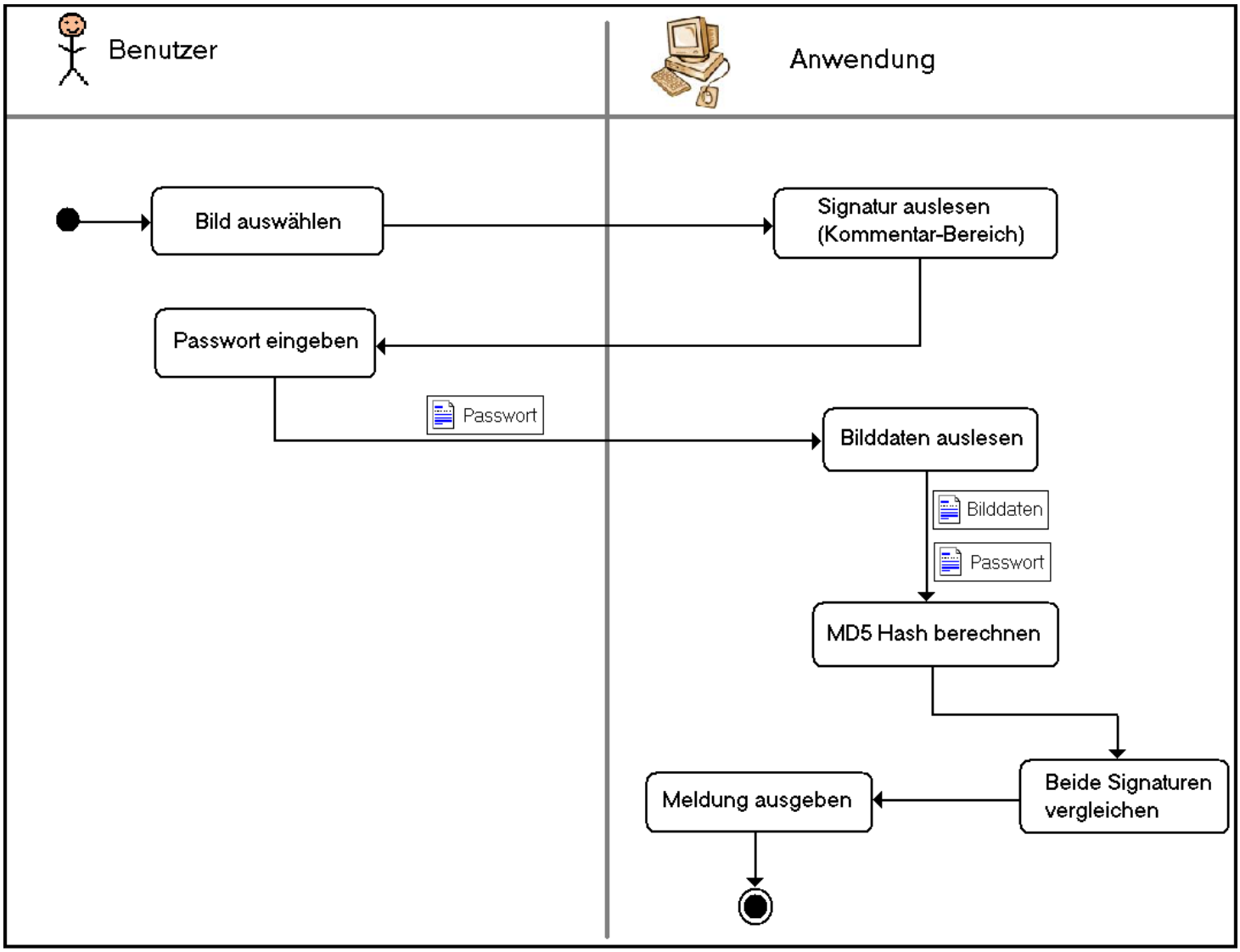


Abb. 8: Anwendungsfall: MD5-Hash prüfen

Nachstehender Screenshot (Abb. 9) zeigt unsere Java-Anwendung, welche die beschriebene Funktionalität realisiert.

Das geladene Bild enthält erst noch keinen keinen Hashwert.

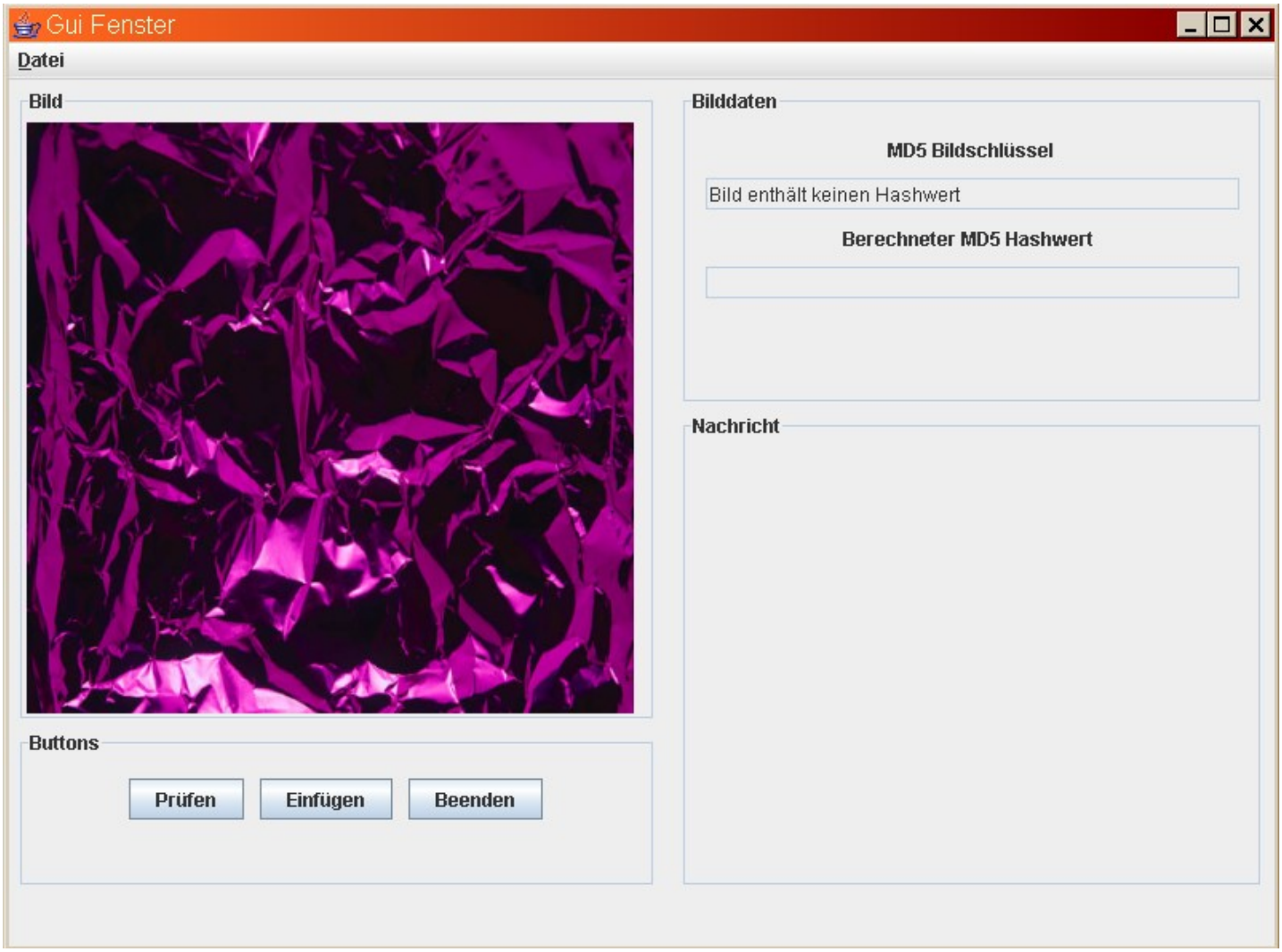


Abb. 9: Bild ohne Hashwert

Nach einfügen des passwortgeschützten Hashwertes (Abb. 10) kann die Integrität des Bildes getestet werden.

Abb. 10: Passwort eingeben

Dann, nach dem Klicken auf „Prüfen“ und erfolgreicher Wieder-Eingabe des Passworts quittiert das Programm das Übereinstimmen der Hashwerte mit einem lachenden Smiley und einem lustigen Geräusch. Bei Fehleingabe erscheinen / ertönen die negativen Pendants.

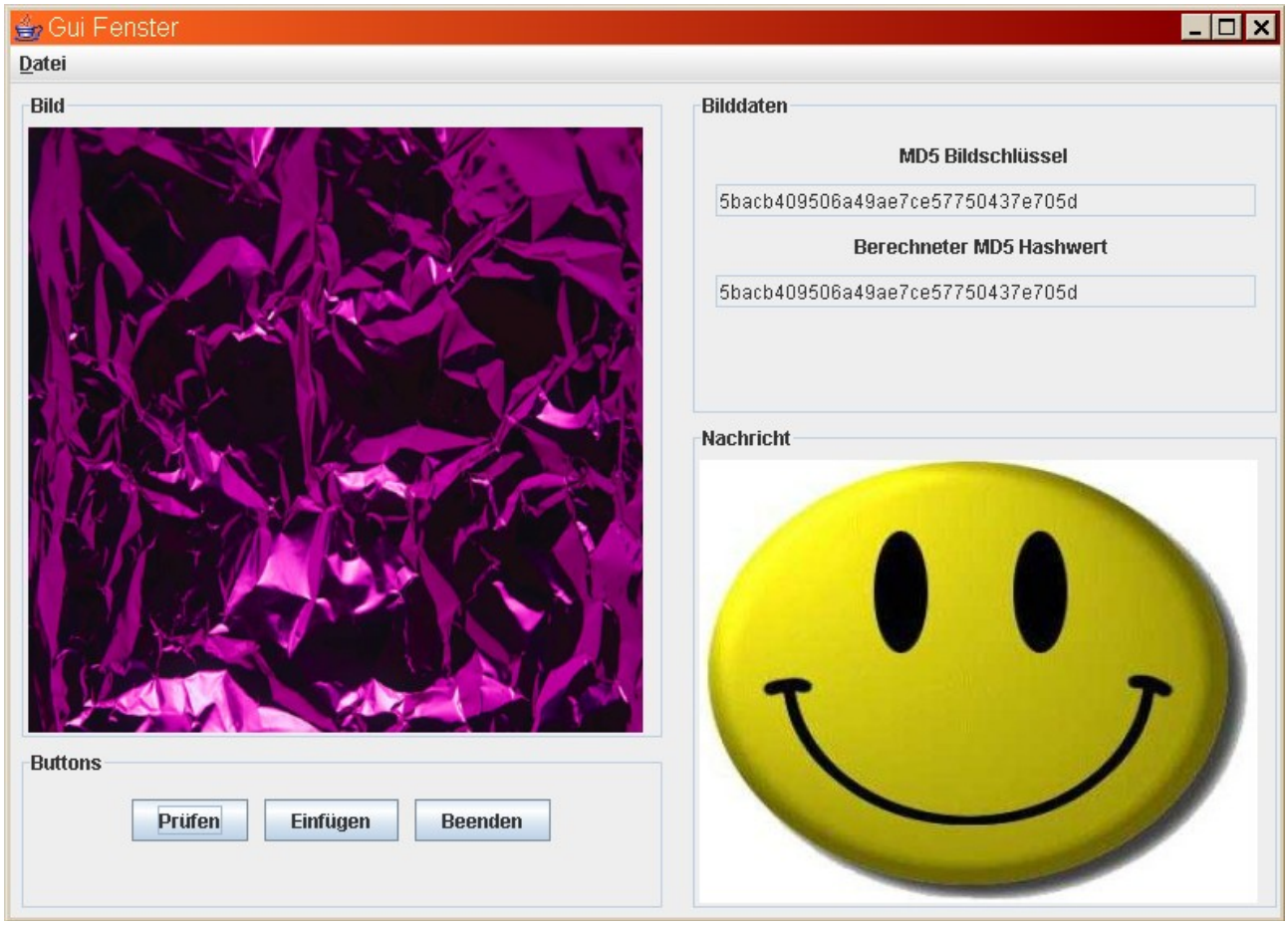


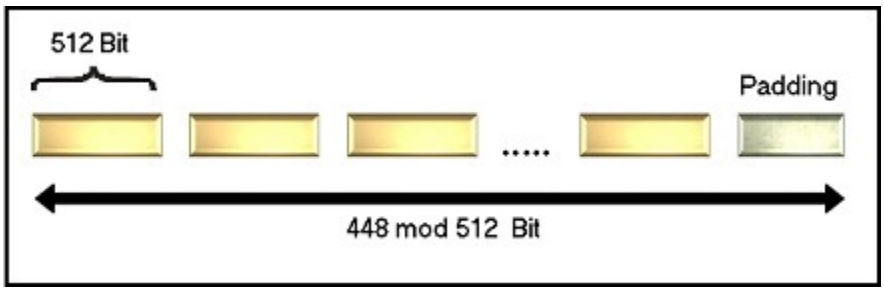
Abb. 11: Erfolgreicher Vergleich der Hashwerte

Asysmptotische Analyse von MD5

Der Algorithmus lässt sich in fünf Teilschritte gliedern (s. RFC1321):

1. Padding-Bits anhängen
2. Länge der Eingabe anhängen
3. MD5 Buffer initialisieren
4. Eingabe in 512 Bit-Blöcken verarbeiten
5. Ausgabe

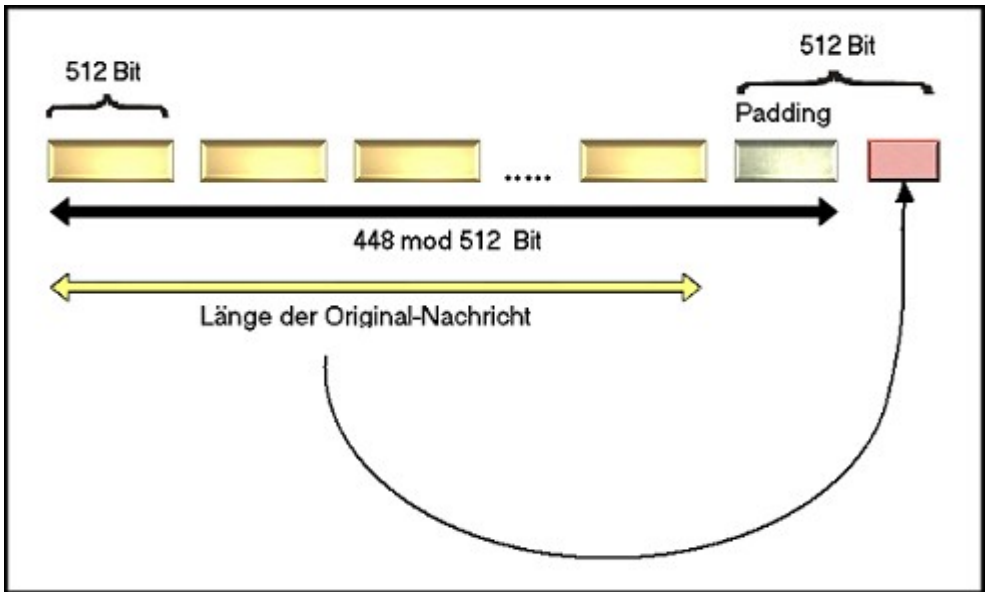
Das Anhängen der Padding-Bits



Erfordert je nach Datentyp eine konstante Anzahl an Operationen. In der Beschreibung des Algorithmus´ (RFC1321) ist angegeben, daß in jedem Fall ein Padding an die Eingabenachricht angehangen wird, auch, falls die Länge der Nachricht bereits einen Divisionsrest von 448 hat (siehe Abschnitt 2). Im schlechtesten Fall besitzt das Padding eine Länge von 512 Bit, was 16 Operationen auf einer 32 Bit Maschine erfordern würde.

Damit sei **a = 16**

Das Anhängen der Nachrichtenlänge



Erfordert zwei Operationen (64 Bit), d.h. **b = 2**

Das Initialisieren des MD5-Buffers erfordert nochmal **c = 4** Operationen (vier Register).

Die Schritte 1,2,3 und 5 sind alle in Konstanter Zeit **C** ausführbar.

Der Verarbeitungsschritt ist jedoch von der Nachrichtenlänge abhängig.

Wenn die Laufzeit für eine Rundenoperation „`[abcd k s i]`“ den Betrag **a** hat, hat ein Schleifendurchlauf die Laufzeit **64a**, da diese Operation pro Durchlauf 64 mal ausgeführt wird. Hinzu kommen pro Durchlauf **24** Initialisierungen und **4** Additionen, deren Laufzeit mit **b** und **c** bezeichnet sei. Dann ist die Gesamtlaufzeit für eine Eingabenachricht der Länge **n**:

$$\left(\frac{n}{16}-1\right)\cdot(64a+24b+4c)+C \quad \in \quad O(n)$$