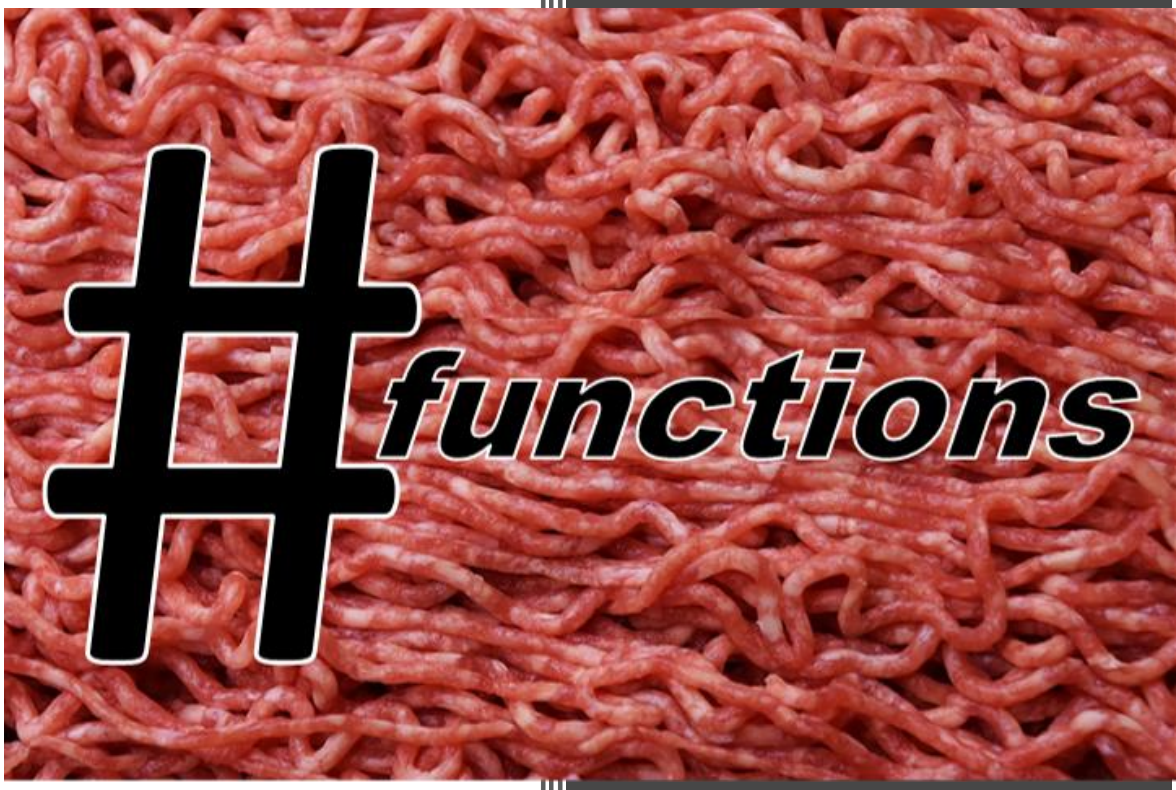
Kryptographische Hashfunktionen



r.uhlmann (2023)

INHALT

Der Hashwert

1 EINLEITUNG

In einer Datenbank nach einem bestimmten Dokument zu suchen, von dem es vielleicht auch noch mehrere, leicht voneinander abweichende Versionen gibt, ist sehr zeitaufwändig. Dies ähnelt der Suche nach einer bestimmten Person in der Datenbank der Kriminalpolizei. Als hilfreich erweist sich dort die AFIS-Datenbank, in der in Österreich mehr als eine Million Fingerabdrücke digital gespeichert sind.

Abbildung 1.1

Fingerabdrücke können einen Menschen eindeutig identifizieren. Zumindest sind bis heute keine zwei Menschen mit gleichem Fingerabdruck bekannt. Sogar bei eineiigen Zwillingen sind sie unterschiedlich ausgeprägt. [Wikipedia]

Bei der Suche nach einem Dokument wäre so ein „Fingerabdruck“, also ein kurzer Zahlenstring als Label zur eindeutigen Identifizierung, ebenfalls wünschenswert. Dieses Label ist als Hashwert bekannt. Die Funktion dazu ist die sogenannte Hashfunktion. Sie macht aus dem Dokument „Hackfleisch“ (□) (Faschiertes) und komprimiert dies gleichzeitig auf die geforderte Länge des Hashwerts. Je nach Art der verwendeten Hash-Funktion hat der Hashwert danach eine Länge von 128, 160, 192, 224 oder 256 Bit.

So erhält der Text

> *„Lorem ipsum dolor sit amet, consectetur adipisici elit, sed eiusmod tempor incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquid ex ea commodi consequat. Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.”*

durch die **MD5**-Hashfunktion den Hashwert

> *hex: „355cc021818c8a794653037e2b005113“ bzw.*
> *bin: „11010101011100110000000010000110000001100011001000101001111001010001100101001100000011011111100010101100000000010100010001001 1“*

und das Wort

> *„Cicero“*

den **MD5**-Hashwert

> *hex: „c6dd4ad42eda4a72ed9944fb183c407b“ bzw.*
> *bin: „11000110110111010100101011010100001011101101101001001010011100101110110110010001010001001111101100011000001111000100000000111101 1“*

Eine Hashfunktion erzeugt immer Hashwerte der gleichen Länge. Bei **MD5** (**Message-Digest Algorithm 5**) sind diese 128 Bit lang und werden binär berechnet aber üblicherweise hexadezimal angeschrieben.

Weitere Anwendungen von Hashwerten sind die Signatur von Dokumenten und die Verifizierung von Software oder Passwörtern.

2 DIE HASHFUNKTION

Die Hashfunktion erzeugt zu einem beliebig langen digitalen Input einen für diesen Input charakteristischen Output mit festgelegter Länge, dem sogenannten Hashwert.

Definition 2.1 (BITSTRING)
Als digitaler In- bzw. Output sind hier **Bitstrings** (**Bitvektoren**) zu verstehen. Dies sind Zeichenketten bestehend aus Nullen und Einsern, welche als Elemente der Produktmenge $\{0\,,1\}^n\ ,n \in \mathbb{N}^+$ aufgefasst werden. Ein Beispiel wäre $011101 \in \{0\,,1\}^6$.
Mit $\{0\,,1\}^*$ bezeichnet man die Menge aller (beliebig) endlich langer Bitstrings.

Bemerkung 1: $\mathbb{N}^+ \coloneqq \mathbb{N}/\{0\} = \{1\,,2\,,\dots\}$

Bemerkung 2: $|\square|$ steht für die Bitstring-Länge.

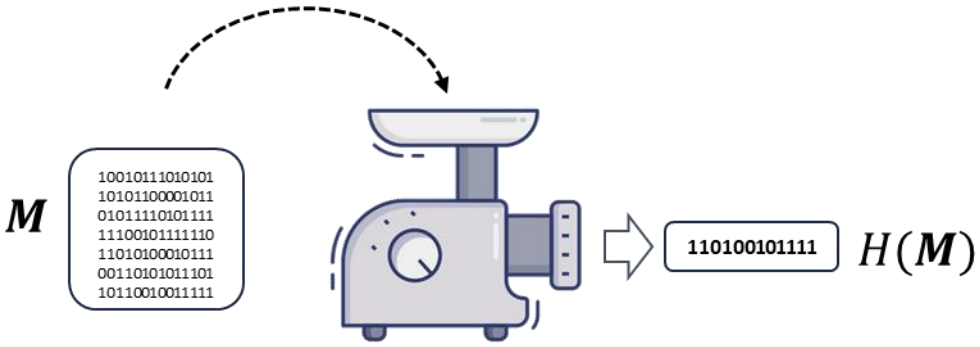


Abbildung 2.1: Hash

Definition 2.2 (HASHFUNKTION)

$$\begin{array}{ccccc} H\colon & \{0\,,1\}^* & \longrightarrow & \{0\,,1\}^l\,, l \in \mathbb{N}^+ \\ \square & \boldsymbol{M} & \longmapsto & H(\boldsymbol{M}) \end{array}$$

wobei der Bitstring $\boldsymbol{M}$ als „Message“ (Nachricht) und $H(\boldsymbol{M})$ als „Hashwert“ der Message (Nachricht) bezeichnet wird.

Eine Hashfunktion sollte folgende **Eigenschaften** besitzen:

1) $H(\boldsymbol{M})$ soll effizient (schnell) berechenbar und relativ kurz sein.
2) Die Hashwerte sollen gleichmäßig in der Wertemenge $\{0\,,1\}^l$ verteilt sein.
3) Unabhängig von der Länge der Nachrichten $|\boldsymbol{M}_1| \neq |\boldsymbol{M}_2|$, haben die Hashwerte die gleiche Länge $l = |H(\boldsymbol{M}_1)| = |H(\boldsymbol{M}_2)|$.
4) Kleinste Unterschiede im Input $\boldsymbol{M}_1 \approx \boldsymbol{M}_2\,, \boldsymbol{M}_1 \neq \boldsymbol{M}_2$ sollen völlig unterschiedliche Hashwerte $H(\boldsymbol{M}_1) \neq H(\boldsymbol{M}_2)$ als Output generieren.

So liefert die Hashfunktion **MD5** für obiges „Lorem ipsum … est laborum.“ bei fehlendem Schlusspunkt („Lorem ipsum … est laborum“) den hexadezimal geschriebenen Hashwert „*94004a32a1bba603fb39beb85980ba4a*“, welcher ein gänzlich anderer ist.

Die Zuordnung Person ↦ Fingerabdruck scheint bijektiv zu sein. Dies kann mathematisch für Hashfunktionen nicht gelten. Es gibt theoretisch abzählbar unendlich viel mögliche Dokumente, Passwörter etc., für die es nur endlich viele (bei **MD5** sind das 2^{128}) binäre Hashwerte gibt. Es muss also auch den Fall geben, dass die Hashfunktion für zwei unterschiedliche Bitstrings den gleichen Hashwert errechnet. Man nennt dies eine **Kollision**.

Definition 2.3 (KOLLISION)
Seien $\boldsymbol{M}_1, \boldsymbol{M}_2 \in \{0,1\}^*$ zwei Bitstrings (Messages). Als Kollision bezeichnet man die Eigenschaft, dass

$$\boldsymbol{M}_1 \neq \boldsymbol{M}_2 \ \wedge H(\boldsymbol{M}_1) = H(\boldsymbol{M}_2).$$

Kollisionen sind verständlicherweise unerwünscht, jedoch „part of the game“.

3 KRYPTOGRAPHISCHE HASHFUNKTIONEN

Hashfunktionen werden sehr häufig in sicherheitsrelevanten Bereichen eingesetzt. Im Besonderen beim Nachweis der Integrität und Authentizität von Daten. Dies umfasst die Verifikation von Passwörtern und Signaturen bzw. bei E-Mails, Banküberweisungen sowie Softwareupdates für Computer, Smartphones etc. Ohne die verwendeten Hashfunktionen wäre der dafür nötige Rechenaufwand und Datenverkehr um vieles größer. Da die jeweils verwendete Hashfunktion öffentlich gekannt ist, muss in solch sensiblen Bereichen eine Hashfunktion noch weitere Eigenschaften besitzen.

5) **Einwegfunktion:** Aus dem Hashwert darf praktisch nicht auf die ursprüngliche Nachricht geschlossen werden können. „Praktisch nicht“ bedeutet, dass dies nur mit „zeitlich immensen Rechenaufwand oder durch Zufall“ gelänge.

$$H(\boldsymbol{M}) \nRightarrow \boldsymbol{M}$$

6) **Kollisionsresistenz:** Die meist unvermeidbaren Kollisionen können insofern nicht ausgenützt werden, als die Hashfunktion keine Rückschlüsse auf den möglichen Kollisionspartner zulässt. Man unterscheidet schwache und starke Kollisionsresistenz (siehe unten).

Definition 3.1 (STARKE UND SCHWACHE HASHFUNKTION)

Eine Hashfunktion mit den Eigenschaften 1) bis 6) ist eine kryptographische Hashfunktion.

Eine kryptographische Hashfunktion heißt **schwach kollisionsresistent**, wenn bei Kenntnis von $\boldsymbol{M}_1, H(\boldsymbol{M}_1)$ es praktisch (zeitlich und ressourcenmäßig) unmöglich ist, Rückschlüsse auf einen möglichen Kollisionspartner $\boldsymbol{M}_2, H(\boldsymbol{M}_2) = H(\boldsymbol{M}_1)$ zu ziehen.

Eine kryptographische Hashfunktion heißt **stark kollisionsresistent**, wenn es praktisch (zeitlich und ressourcenmäßig) unmöglich ist, überhaupt Kollisionspaare $\boldsymbol{M}_1 \neq \boldsymbol{M}_2, H(\boldsymbol{M}_1) = H(\boldsymbol{M}_2)$ zu finden.

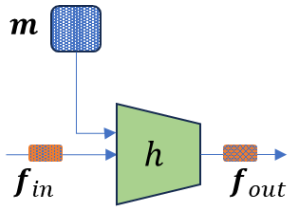
Ein Beispiel dazu: Wüsste man, dass die Freigabe einer Banküberweisung von 12.50 € den gleichen Hashwert wie eine von 130.90 € hat, dann könnte eine Betrügerin in einem Webshop ein Produkt um 12.50 € anbieten. Da im Datentransfer zwischen Kunde und Bank ja nie die echten Daten, sondern

lediglich deren Hashwerte kommuniziert werden, kann eine Bankverbindung vorgetäuscht und der Bank eine Überweisung von 13.90 € im Namen des Kunden angewiesen werden.

Bei schwachen Hashfunktionen kann man zwar zu einem bestimmten $\boldsymbol{M}$ und $H(\boldsymbol{M})$ den Kollisionspartner nicht finden, sie lassen aber zu, mit hinnehmbarem Rechenaufwand beliebige Kollisionen zu finden und diese in einer Datenbank öffentlich zu machen.

4 Die Merkle – Damgård – Konstruktion (M/D)

Viele Hashfunktionen H beruhen auf der sogenannten Merkle – Damgård – Konstruktion. Dabei wird eine Nachricht, ein Dokument oder welche digitale Zeichenkette ($\boldsymbol{M}$) auch immer, häppchenweise ($\boldsymbol{m}_i$) durch eine Kompressionsfunktion h iterativ ineinander verrechnet, bis die ganze Nachricht zu einem Hashwert verarbeitet ist.



4.1 Die Kompressionsfunktion

Zentraler Bestandteil des Algorithmus ist die **Kompressionsfunktion** h. Sie ist selbst eine Hashfunktion und verarbeitet die Verkettung zweier Bitstrings $\boldsymbol{f_{in}}$ und $\boldsymbol{m}$ zu einem Bitstring $\boldsymbol{f}_{out}$, wobei $|\boldsymbol{f}_{out}| = |\boldsymbol{f_{in}}|$. Z.B.: $\underbrace{110}_{\boldsymbol{f_{in}}}\ \underbrace{00101}_{\boldsymbol{m}} \longrightarrow \underbrace{010}_{\boldsymbol{f_{out}}}$

$\boldsymbol{m}$ ist ein Teilstring der Nachricht $\boldsymbol{M}$, dessen Länge sich nach der Kapazität der Kompressionsfunktion h richtet. Eine Kompressionsfunktion, die einen 16 Bit Input in einen 6 Bit Output komprimiert, kann 10 Bit ($\boldsymbol{m}$) der Nachricht ($\boldsymbol{M}$) verarbeiten. Die Funktion arbeitet dabei ähnlich einem Fleischwolf, der jede Eingabe zu Hackfleisch verarbeitet. Hackfleisch wird in Österreich als „Faschiertes“ bezeichnet, deshalb auch die Wahl des Buchstaben f in $\boldsymbol{f_{in}}$ und $\boldsymbol{f_{out}}$. Die Kompressionsfunktion h verarbeitet bereits faschiertes Daten-Hack ($\boldsymbol{f_{in}}$) zusammen mit dem Teilstring $\boldsymbol{m}$ wieder zu faschiertem und komprimierten Daten-Hack ($\boldsymbol{f_{out}}$) mit $|\boldsymbol{f}_{out}| = |\boldsymbol{f_{in}}|$.

Definition 4.1 (Kompressionsfunktion)
Seien $\boldsymbol{f} \in \{0,1\}^l$ und $\boldsymbol{m} \in \{0,1\}^k$ zwei Bitstrings der Länge l und k. Dann ist

$$\begin{array}{rccc} h\colon & \{0,1\}^l \times \{0,1\}^k & \longrightarrow & \{0,1\}^l \\ & (\boldsymbol{f};\boldsymbol{m}) & \longmapsto & h(\boldsymbol{f};\boldsymbol{m}) \end{array};\ l,k \in \mathbb{N}^+$$

eine Kompressionsfunktion h im Sinne einer Merkle – Damgård – Konstruktion.

Die Kompressionsfunktion h ist dabei selbst eine Hashfunktion, die Bitstrings der Länge $l + k$ (Input) auf die Länge l (Output) komprimiert.

Die Qualität der M/D-Konstruktion steht und fällt mit der Qualität der Kompressionsfunktion.

Beispiel 4.1 (Kompressionsfunktion MDβ)

Bleiben wir bei dem Beispiel mit 16 Bit → 6 Bit. Beispielsweise sei

$\boldsymbol{f}_{in} = 110000$ (6 Bit) sowie $\boldsymbol{m} = 0011011010$ (10 Bit),

also $(\boldsymbol{f}_{in};\boldsymbol{m}) = 110000\,0011011010$ (16 Bit)

Die Aufgabe der Kompressionsfunktion besteht aus $(\boldsymbol{f}_{in};\boldsymbol{m}) \rightarrow \boldsymbol{f}_{out}$, wobei $\boldsymbol{f}_{out}$ ebenfalls 6 Bit umfassen muss.

DIE KONSTRUKTION (*frei erfunden*): Bildung von 6 Bit-Strings aus den Eingangsdaten. Dazu Teilung der Nachricht $\boldsymbol{m}$ in zwei gleichlange Teile 00110-11010 und Ergänzung durch jeweils ein Paritätsbit. Die zwei so entstandenen 6 Bit-Strings werden zuletzt mit $\boldsymbol{f}_{in}$ mod2 addiert (XOR-Verknüpfung).

$\bar{\boldsymbol{m}}$	0	0	1	1	0	**0**
$\bar{\bar{\boldsymbol{m}}}$	1	1	0	1	0	**1**
$\boldsymbol{f}_{in}$	1	1	0	0	0	0
⬚	– –	– –	– –	– –	– –	– –
Summe mod_2 : $\boldsymbol{f}_{out}$	0	0	1	0	0	1

Ergebnis des Iterationsschrittes:

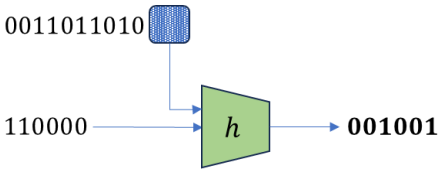


Abbildung 4.1

4.2 DIE MERKLE-DAMGÅRD-HASHFUNKTION

Die eigentliche Hashfunktion besteht aus der wiederholten Anwendung der Kompressionsfunktion. Dazu muss die Nachricht Blöcke $\boldsymbol{m}_i$ der Länge k (siehe **Definition *4.1***) zerlegt werden. Für unsere erfundene Hashfunktion (Bezeichnung $\mathbf{MD}_{\beta}$) sind das 10er-Blöcke. Der letzte Block bleibt dabei meist unvollständig und muss irgendwie aufgefüllt werden. Diesen Vorgang nennt man „**Padding**“. Dies wird im nächsten Abschnitt noch näher erklärt.

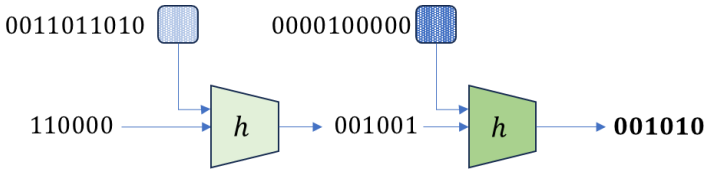


Abbildung 4.2

Startwert: Die Iteration startet mit einem beliebig gewählten, aber öffentlichen Hashwert $\boldsymbol{f}_0$ (initial value $\boldsymbol{i.v.} = \boldsymbol{f_0}$) und dem ersten Teil $\boldsymbol{m}_1$ der Nachricht $\boldsymbol{M}$.

Iteration: Der daraus entstehende erste Hashwert $\boldsymbol{f}_1$ bildet gemeinsam mit dem nächsten Abschnitt $\boldsymbol{m}_2$ der Nachricht den Input der darauf folgenden Iteration.

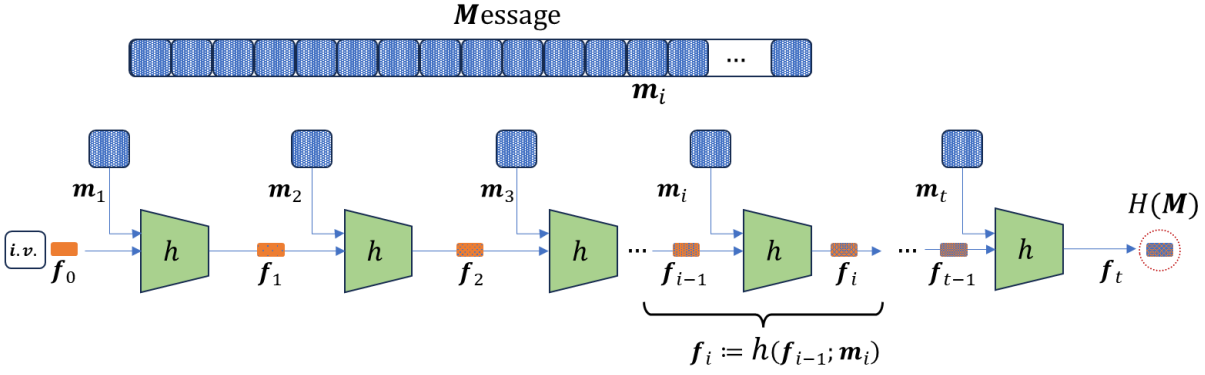


Abbildung 4.3

Das Flussdiagramm (Abbildung 4.3) zeigt diesen iterativen Prozess. Der letzte Output ist der gesuchte Hashwert des Dokuments etc.

Hashwert: $\boldsymbol{f}_t = H(\boldsymbol{M})$

4.3 Das Padding

Die Kompressionsfunktion h unserer Hashfunktion **MD$_\beta$** aus Beispiel 4.1 verarbeitet in jedem Iterationsschritt 10 Bits von $\boldsymbol{M}$ und erzeugt daraus einen 6 Bit langen Hashwert $H(\boldsymbol{M})$. Der zu hashende String muss also in Blöcke $\boldsymbol{m}_i$ zu 10 Bits zerlegt werden, was sich im gegebenen Fall nicht perfekt ausgeht. Der letzte Block ist möglicherweise unvollständig und muss im Prozess des Paddings irgendwie aufgefüllt werden.

„[0100000101] … [0101011100] [1001 XXXXXX]“

In diesem Bereich des Paddings wird auch noch die **Länge von** $|\boldsymbol{M}|$ eingefügt. Das führt dazu, dass das Hinzufügen oder Weglassen von auch nur einem Bit einem veränderten Hashwert erzeugt.

Beispiel 4.2 (Brutus)

Der Text „Auch du mein Sohn Brutus“ lautet binär:

„010000010111010101100011011010000010000001100100011101010010000001101101011001010110100101101110000100000010100110110111101101000011011100010000001000010011100100111010101110100011101010111001 1“

Das sind 24 Zeichen (inkl. Leerzeichen) zu je 8 Bits. Somit hat $\boldsymbol{M}$ eine Länge von 192 Bits. Der zu hashende String inklusive Padding muss wieder in Blöcke $\boldsymbol{m}_i$ zu 10 Bits zerlegt werden.

„[0100000101] [1101010110] [0011011010] [0000100000] [0110010001] [1101010010] [0000011011] [0101100101] [0110100101] [1011100010] [0000010100] [1101101111] [0110100001] [1011100010] [0000010000] [1001110010] [0111010101] [1101000111] [0101011100] [11 XXXXXXXX]“

Der Abschnitt des Paddings **beginnt mit einer 1** (das hat auch seinen Grund) **und schließt mit einem Block** (z.B. 9 Bit)**, der für die Bitstringlänge reserviert ist**. Dazwischen stehen so viele 0 als es für ein ganzzahliges Vielfaches der Blocklänge braucht.

Da $|\boldsymbol{M}| = 192_{dez.} = 11000000_{bin.} = \overbrace{011000000}^{9Bit}{}_{bin.}$, lautet der fertige Bitstring inklusive Padding

„[0100000101] [1101010110] [0011011010] [0000100000] [0110010001] [1101010010] [0000011011] [0101100101] [0110100101] [1011100010] [0000010100] [1101101111] [0110100001] [1011100010] [0000010000] [1001110010] [0111010101] [1101000111] [0101011100] [11**1**0000000] [0**011000000**]“

Mit $\boldsymbol{f}_0 \coloneqq [000000]$ ergibt die Iteration

$$H(\boldsymbol{M}) = 001100$$

Eine Änderung in der Nachricht ergibt andere Hashwerte:
„Auch du meyn Sohn Brutus“ $\rightarrow H(\boldsymbol{M}) = 001001$
„Auch du, mein Sohn Brutus“ $\rightarrow H(\boldsymbol{M}) = 100010$

Bemerkung: Diese Hashfunktion erfüllt ihren didaktischen Zweck durch Verwendung kleiner Zahlen, ist aber genau deshalb unbrauchbar. Sie liefert lediglich $2^8 = 256$ verschiedene Hashwerte, Kollisionsresistenz sieht anders aus □.

4.4 Die Kollisionsresistenz der Merkle-Damgård-Konstruktion

Hacker sind an Kollisionen ($H(\boldsymbol{M}_1) = H(\boldsymbol{M}_2)$ obwohl $\boldsymbol{M}_1 \neq \boldsymbol{M}_2$) interessiert. Ist es möglich, in annehmbarer Zeit Kollisionen zu finden, dann gilt die Hashfunktion als unsicher. Die schon öfter zitierte Hashfunktion **MD5** gilt seit längerer Zeit als unsicher und wird nicht mehr empfohlen. Es gibt aber noch Anwendungsbereiche, wo Kollisionsresistenz nicht so wichtig ist.

Für die Einschätzung, inwiefern eine M/D-Hashfunktion H kollisionsresistent ist, gilt der

Satz 4.1

Die Verwendung einer kollisionsresistenten Kompressionsfunktion h garantiert die Kollisionsresistenz der gesamten M/D-Hashfunktion H.

Die Rolle der Kompressionsfunktion h

Um die obige Behauptung zu begründen, benötigen wir zuvor noch einen Hilfssatz (Lemma). In seinem Beweis steckt die ganze Argumentation für den Hauptsatz (Satz 4.1), der lediglich die Kontraposition des Lemmas ausdrückt.

Satz 4.2 (Lemma)

Sei H eine M/D-Hashfunktion mit integrierter Kompressionsfunktion h, so gilt:

Jede Kollision von H geht mit einer Kollision in h einher.

$$H(\boldsymbol{M}) = H(\boldsymbol{M}') \;\wedge\; \boldsymbol{M} \neq \boldsymbol{M}' \quad \Longrightarrow \quad \exists(\boldsymbol{m}_i; \boldsymbol{f}_{i-1}) \neq (\boldsymbol{m}'_j; \boldsymbol{f}'_{j-1}) \;\wedge\; h(\boldsymbol{m}_i; \boldsymbol{f}_{i-1}) = h(\boldsymbol{m}'_j; \boldsymbol{f}'_{j-1})$$

Beweis

Annahme einer Kollision von H: $H(\boldsymbol{M}) = H(\boldsymbol{M}') \;\wedge\; \boldsymbol{M} \neq \boldsymbol{M}'$ (siehe Abbildung 4.4)

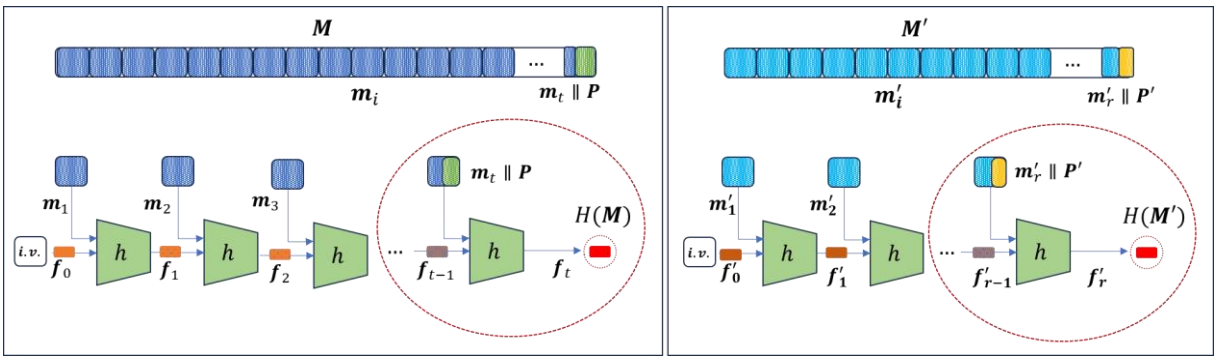


Abbildung 4.4: Flussdiagramm einer Kollision – Zwei unterschiedliche Nachrichten haben den gleichen Hashwert.

Es stellt sich die Frage, wie dieser gleiche Hashwert aus unterschiedlichen Nachrichten $\boldsymbol{M} \neq \boldsymbol{M}'$ entstanden sein konnte. Der Prozess startet bei beiden Nachrichten mit dem gleichen Startwert $\boldsymbol{i.v.} = \boldsymbol{f}'_0$, aber spätestens dort, wo sich die Nachrichten $\boldsymbol{M}$ und $\boldsymbol{M}'$ unterscheiden, laufen die Ergebnisse auseinander. Auch die Rundenanzahl t und r ist bei gegebenenfalls unterschiedlicher Länge der Nachrichten verschieden. Da H laut Annahme eine Kollision aufweist, gleichen sich die Werte am Schluss allerdings wieder: $\boldsymbol{f}_t = \boldsymbol{f}'_r$

Betrachten wir das genauer, und zwar vom Ende her:

$$H(\boldsymbol{M}) = H(\boldsymbol{M}') \;\Leftrightarrow\; \boldsymbol{f}_t = \boldsymbol{f}'_r$$

Der Input der letzten Runde ist das Ende der jeweiligen Nachricht $\boldsymbol{m}_t$ bzw. $\boldsymbol{m}'_r$ inklusive Padding $\boldsymbol{P}$ bzw. $\boldsymbol{P}'$ sowie der Hash $\boldsymbol{f}_t$ bzw. $\boldsymbol{f}'_r$ der Runde davor: $\left(\underbrace{\boldsymbol{f}_{t-1}}\,;\underbrace{\boldsymbol{m}_t \parallel \boldsymbol{P}}\right)$ bzw. $\left(\underbrace{\boldsymbol{f}'_{r-1}}\,;\underbrace{\boldsymbol{m}'_r \parallel \boldsymbol{P}'}\right)$

$$H(\boldsymbol{M}) = \boldsymbol{f}_t = h\left(\underbrace{\boldsymbol{f}_{t-1}}\,;\underbrace{\boldsymbol{m}_t \parallel \boldsymbol{P}}\right) = h\left(\underbrace{\boldsymbol{f}'_{r-1}}\,;\underbrace{\boldsymbol{m}'_r \parallel \boldsymbol{P}'}\right) = \boldsymbol{f}'_r = H(\boldsymbol{M}')$$

Der Output der letzten Runde und damit der eigentliche Hashwert hängt somit von drei Bitstrings ab:

- Dem jeweiligen Nachrichtenende $\boldsymbol{m}_t, \boldsymbol{m}'_r$
- dem jeweiligen Paddingblock $\boldsymbol{P}, \boldsymbol{P}'$
- dem jeweiligen Hash der vorhergehenden Runde $\boldsymbol{f}_{t-1}, \boldsymbol{f}'_{r-1}$

Von daher unterscheiden wir zwei Möglichkeiten:

Fall 1: Die Argumente von h in der letzten Runde unterscheiden sich.

$$\underbrace{\boldsymbol{f}_{t-1}; \boldsymbol{m}_t \parallel \boldsymbol{P}} \neq \underbrace{\boldsymbol{f}'_{r-1}; \boldsymbol{m}'_r \parallel \boldsymbol{P}'}$$

Dies stellt eine **Kollision von $\boldsymbol{h}$** dar (Argumente verschieden, Funktionswerte trotzdem gleich).

$$\underbrace{\boldsymbol{f}_{t-1}; \boldsymbol{m}_t \parallel \boldsymbol{P}} \neq \underbrace{\boldsymbol{f}'_{r-1}; \boldsymbol{m}'_r \parallel \boldsymbol{P}'} \;\wedge\; h(\boldsymbol{f}_{t-1}; \boldsymbol{m}_t \parallel \boldsymbol{P}) = h(\boldsymbol{f}'_{r-1}; \boldsymbol{m}'_r \parallel \boldsymbol{P}')$$

Genau das wird im Lemma behauptet! Wenn H eine Kollision hat, dann auch ihre Kompressionsfunktion h.

Fall 2: Die Argumente von h in der letzten Runde gleichen sich.

$$\underbrace{\boldsymbol{f}_{t-1}; \boldsymbol{m}_t \parallel \boldsymbol{P}} = \underbrace{\boldsymbol{f}'_{r-1}; \boldsymbol{m}'_r \parallel \boldsymbol{P}'}$$

Dann müssen alle drei Teile des Argumentes von h gleich sein:

$$\boldsymbol{m}_t = \boldsymbol{m}'_r \;\wedge\; \boldsymbol{P} = \boldsymbol{P}' \;\wedge\; \boldsymbol{f}_{t-1} = \boldsymbol{f}'_{r-1}$$

Insbesondere $\boldsymbol{P} = \boldsymbol{P}'$! Hier liegt der Clou! Da die Hashfunktion so konstruiert ist, dass im Paddingblock die Länge der Nachricht codiert ist, folgt aus

$$\boldsymbol{P} = \boldsymbol{P}' \Longrightarrow |\boldsymbol{M}| = |\boldsymbol{M}'|$$

Der Fall 2 kann also nur in Nachrichten gleicher Länge auftreten. Daraus folgt auch, dass die Anzahl t und r der Blöcke $\boldsymbol{m}_i$ gleich ist:

$$t = r$$

Damit schreibt sich obige Bedingung

$$\boldsymbol{m}_t = \boldsymbol{m}'_t \;\wedge\; \boldsymbol{P} = \boldsymbol{P}' \;\wedge\; \boldsymbol{f}_{t-1} = \boldsymbol{f}'_{t-1} \;.$$

$\boldsymbol{f}_{t-1} = \boldsymbol{f}'_{t-1}$ bedeutet, dass die beiden vorletzten Kompressionswerte auch gleich sind:

$$\boldsymbol{f}_{t-1} = \boldsymbol{f}'_{t-1} = h(\boldsymbol{f}_{t-2}; \boldsymbol{m}_{t-1}) = h(\boldsymbol{f}'_{t-2}; \boldsymbol{m}'_{t-1})$$

Ab hier wiederholt sich die Argumentation.

Entweder

$$\boldsymbol{m}_{t-1} \neq \boldsymbol{m}'_{t-1} \;\vee\; \boldsymbol{f}_{t-2} \neq \boldsymbol{f}'_{t-2} \;,$$

dann zeigt h auf dieser Stufe eine Kollision, denn

$$h(\boldsymbol{f}_{t-2}; \boldsymbol{m}_{t-1}) = h(\boldsymbol{f}'_{t-2}; \boldsymbol{m}'_{t-1}), \text{ aber} \boldsymbol{f}_{t-2}; \boldsymbol{m}_{t-1} \neq \boldsymbol{f}'_{t-2}; \boldsymbol{m}'_{t-1} \,.$$

Oder

$$\boldsymbol{m}_{t-1} = \boldsymbol{m}'_{t-1} \;\wedge\; \boldsymbol{f}_{t-2} = \boldsymbol{f}'_{t-2} \;,$$

dann führt die Argumentation wieder auf die Kompressionsrunde davor. Dass dieser „Fall 2“ durchgeht bis zum Anfang der M/D-Konstruktion kann aber auch nicht sein, denn dann wäre $\boldsymbol{M} = \boldsymbol{M}'$.

Die Annahme war eine Kollision von H, also $\boldsymbol{M} \neq \boldsymbol{M}'$. Somit muss zwangsläufig in der M/D-Konstruktion einmal Fall 1 auftreten, in dem die Kompressionsfunktion h selbst eine Kollision aufweist. ■*q.e.d.*

Aus dem Lemma (Satz 4.2) folgt durch Kontraposition die oben schon erwähnte Einschätzung der Kollisionsresistenz einer M/D-Hashfunktion H. $(A \Longrightarrow B) \Leftrightarrow (\neg B \Longrightarrow \neg A)$

Das Lemma kann nämlich auch umgekehrt formuliert werden. Bezüglich zweier Nachrichten $\boldsymbol{M}$ und $\boldsymbol{M}'$ gilt dann:

(Kollision in $H \Longrightarrow$ Kollision in h) $\Leftrightarrow$ (Keine Kollision in $h \Longrightarrow$ Keine Kollision in H)

Das wiederum bedeutet: **Weist die kleine Kompressionsfunktion $\boldsymbol{h}$ selten Kollisionen auf, dann auch die ganze M/D-Hashfunktion $\boldsymbol{H}$.**

$\rightarrow$ (SATZ 4.1)
h ist (stark) kollisionsresistent $\Longrightarrow$ H ist (stark) kollisionsresistent.

Dies ist ein großer Vorteil aller Hashfunktionen, welche auf der Merkle-Damgård-Konstruktion aufbauen. Es *genügt*, die als Kompressionsfunktion dienende *kleine* Hashfunktion h kollisionssicher zu gestalten, dann ist dies auch die eigentliche Hashfunktion.

5 SICHERHEIT UND BITLÄNGE

Unsere Hashfunktion **MD$_\beta$** erzeugt, wie schon erwähnt, lediglich 64 unterschiedliche Hashwerte. Allgemein sind es 2^l, wenn $l := |H(\boldsymbol{M})|$ die Bitlänge der Hashwerte ist. Gibt es mehr als 2^l Nachrichten, sind Kollisionen unvermeidbar. Kurze Hashwerte verbieten sich daher von selbst, lange haben den Nachteil längerer Rechenzeit. Wie lange sollen Hashwerte also sein?

So kurz wie möglich, so lang wie nötig.

Bei Angriffen auf Hashfunktionen werden drei typische Arten unterschieden:

- **Collision attack** ($\rightarrow$ Auffinden von Kollisionen)
- **Preimage attack** ($\rightarrow$ Auffinden von Urbildern)
- **Second preimage attack** ($\rightarrow$ Auffinden von zweiten Urbildern)

In einem **Kollisionsangriff** wird versucht, zwei Nachrichten mit gleichem Hashwert zu finden:

Gegeben: $H \rightarrow$ *Gesucht*: $\boldsymbol{M}_1 \neq \boldsymbol{M}_2\ (beliebig) \wedge H(\boldsymbol{M}_1) = H(\boldsymbol{M}_2)$

Werden Kollisionen gefunden, so ist die Hashfunktion für gewisse kryptografische Verfahren unbrauchbar geworden.

In einem **Preimage-Angriff** (Pre-Image) wird versucht, zu einem gegebenen Hashwert eine dazu passende Nachricht, also das bzw. ein Urbild zu finden:

Gegeben: $H(\boldsymbol{M}) \rightarrow$ *Gesucht*: $\boldsymbol{M}$

Zum Beispiel aus dem Passwort-Hash das Passwort zu extrahieren.

In einem **Second preimage-Angriff** wird versucht, zu einer gegebenen Nachricht eine zweite dazu passende Nachricht mit gleichem Hashwert zu finden:

Gegeben: $\boldsymbol{M}$, $H(\boldsymbol{M}) \rightarrow$ *Gesucht*: $\boldsymbol{M}' \wedge H(\boldsymbol{M}') = H(\boldsymbol{M})$

Man hat ein Passwort über dessen Hashwert geknackt und möchte weitere Passwörter zur getarnten Verwendung finden.

DEFINITION 5.1 (n-BIT SICHERHEIT)

In der Kryptographie bedeutet eine n-Bit-Sicherheit, dass der Angreifer 2^n Operationen ausführen müsste, um die Verschlüsselung zu knacken.

Ein wesentlicher Faktor ist dabei die Bitlänge.

Eine Hashfunktion hat bezüglich der genannten Angriffsarten unterschiedliche Werte der Sicherheit (Resistenzwerte).

5.1 KOLLISIONSANGRIFF

Nehmen wir an, der Angriff erfolgt durch einfaches systematisches Durchprobieren. Es stellt sich dabei die Frage, wie groß die Wahrscheinlichkeit ist, dass die Hashfunktion hält. Das heißt, dass bei einem Angriff in annehmbarer Zeit es unwahrscheinlich ist, Kollisionen zu entdecken. Ein Maß dafür ist die Anzahl der Versuche, die „durchschnittlich“ dafür benötigt werden. Dies lässt sich abschätzen.

Als konkretes Beispiel dient wieder eine 6 Bit-Hashfunktion.

Beispiel				Allgemein			
H_6:	$\{0,1\}^*$	$\longrightarrow$	$\{0,1\}^6$	H:	$\{0,1\}^*$	$\longrightarrow$	$\{0,1\}^l, l \in \mathbb{N}^+$
⬚	$\boldsymbol{M}$	$\longmapsto$	$H(\boldsymbol{M})$	⬚	$\boldsymbol{M}$	$\longmapsto$	$H(\boldsymbol{M})$

Es gibt unendlich viele Nachrichten $\boldsymbol{M} \in \{0,1\}^*$, da das Asteriskzeichen $(*)$ für jede beliebige natürliche Zahl steht. H_6 ordnet jeder einen der 64 Hashwerte zu. Wir gehen dabei von einer gleichmäßigen Verteilung dieser Hashwerte innerhalb $\{0,1\}^6$ aus. So sollte der Hashwert z.B. 001011 nicht öfter in $\{0,1\}^6$ auftreten als z.B. 101001 etc.

BEZEICHNUNGEN:
$P(n; K)$ … Wahrscheinlichkeit, bei n Versuchen Kollisionen zu entdecken.
$P(n; \neg K)$ … Wahrscheinlichkeit, bei n Versuchen keine Kollisionen zu entdecken.

Wir konzentrieren uns auf die Wahrscheinlichkeit $P(n; \neg K)$, keine Kollision zu entdecken. Für die Gegenwahrscheinlichkeit $P(n; K)$, mindestens eine Kollision zu entdecken, gilt dann

$$P(n; K) = 1 - P(n; \neg K).$$

Für die Entdeckung einer Kollision sind mindestens 2 Versuche nötig.

Der 1. Versuch mit einem beliebigen Bitstring $\boldsymbol{M}_1 \in \{0,1\}^*$ ergibt den ersten von 64 Hashwerten.

Im 2. Versuch mit einem anderen Bitstring $\boldsymbol{M}_2 \in \{0,1\}^* \backslash \{\boldsymbol{M}_1\}$ beträgt die Wahrscheinlichkeit, dass dieser verschieden zum ersten ist, ${}^{63}/_{64} \approx 0{,}98 = 98\%$.

Im 3. Versuch mit $\boldsymbol{M}_3 \in \{0,1\}^* \backslash \{\boldsymbol{M}_1, \boldsymbol{M}_2\}$ beträgt die Wahrscheinlichkeit, wieder keine Übereinstimmung unter den drei schon „gezogenen“ zu finden, nur mehr ${}^{63}/_{64} \cdot {}^{62}/_{64} \approx 0{,}95 = 95\%$

(1) $P(\mathrm{n} = 1; \neg K) = 1$

(2) $P(\mathrm{n} = 2; \neg K) = {}^{63}/_{64}$

(3) $P(\mathrm{n} = 3; \neg K) = {}^{63}/_{64} \cdot {}^{62}/_{64}$

(4) $P(\mathrm{n} = 4; \neg K) = {}^{63}/_{64} \cdot {}^{62}/_{64} \cdot {}^{61}/_{64}$

(5) $$P(\mathrm{n}; \neg K) = \frac{64}{64} \cdot \frac{63}{64} \cdot \frac{62}{64} \cdot \frac{61}{64} \cdot \ldots \cdot \frac{64-(n-1)}{64} = \frac{64 \cdot 63 \cdot \ldots \cdot 1}{(64-n) \cdot \ldots \cdot 1} \cdot \frac{1}{64^n} = \frac{64!}{(64-n)! \cdot 64^n}$$

Die Wahrscheinlichkeit, bei einer Bitlänge l der Hashwerte nach n Versuchen noch immer keine Kollision gefunden zu haben, beträgt also allgemein

$$P(\mathrm{n}; \neg K) = \frac{2^l!}{(2^l - n)! \cdot (2^l)^n}$$

Mit der Stirlingformel " $n! \cong \sqrt{2\,\pi\,n} \cdot \left(\frac{n}{e}\right)^n$ " lässt sich das folgendermaßen abschätzen:

$$P(\mathrm{n}; \neg K) \cong \left(\frac{2^l}{2^l - n}\right)^{2^l - n + 0{,}5} \cdot e^{-n} = \left(\frac{1}{1 - \frac{n}{2^l}}\right)^{2^l - n + 0{,}5} \cdot e^{-n}$$

Die Wahrscheinlichkeit, mit n Versuchen doch mindestens eine Kollision zu finden, beträgt

$$P(\mathrm{n};K) = 1 - \frac{2^l!}{(2^l-n)!\cdot(2^l)^n} \cong 1 - \left(\frac{2^l}{2^l-n}\right)^{2^l-n+0{,}5} \cdot e^{-n}$$

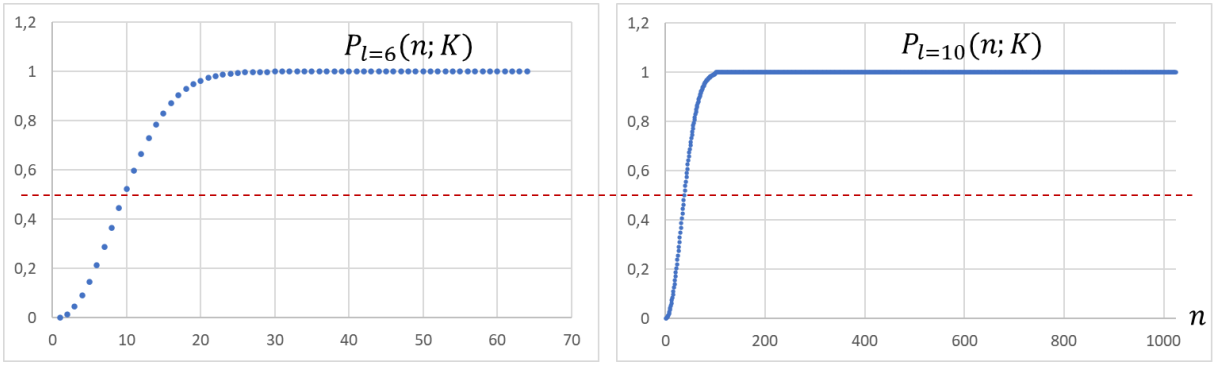


Abbildung 5.1: Wahrscheinlichkeit, nach n Versuchen mindestens eine Kollision entdeckt zu haben. Links bei 6 Bit-Hashwertlänge, rechts bei 10 Bit. Die Wahrscheinlichkeit von 50% wird relativ bald überschritten.

Wie in Abbildung 5.1 zu sehen, steigt die Wahrscheinlichkeit, mindestens eine Kollision zu finden, ‚relativ bald' und ‚relativ rasch' auf nahezu 100%. Ein markanter Punkt ist derjenige, an dem die Wahrscheinlichkeit die 50% übersteigt.

Schätzen wir also ab, wie viele Versuche für eine 50%ige Wahrscheinlichkeit nötig sind, mindestens eine Kollision zu finden (Abbildung 5.1, rote Linie).

$$P(n;K) = \mathbf{0,5} \cong 1 - \left(\frac{2^l}{2^l-n}\right)^{2^l-n+0{,}5} \cdot e^{-n}$$

$$\left(\frac{2^l}{2^l-n}\right)^{2^l-n+0{,}5} \cdot e^{-n} \cong \mathbf{0,5} \quad \text{bzw.} \quad \left(\frac{1}{1-\frac{n}{2^l}}\right)^{2^l-n+0{,}5} \cdot e^{-n} \cong \mathbf{0,5}$$

Die Anzahl der nötigen Versuche n ist offensichtlich klein im Vergleich zu 2^l, der Anzahl der zu testenden Hashwerte. Der Vergleich der Taylorreihen von

$$\frac{1}{1-x} = 1 + x + x^2 + x^3 + \cdots \quad \text{und} \quad e^x = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \cdots$$

ergibt für kleine Zahlen $\frac{1}{1-x} \cong e^x$ und somit

$$\left(e^{\frac{n}{2^l}}\right)^{2^l-n+0{,}5} \cdot e^{-n} \cong \mathbf{0,5} \quad \text{bzw.} \quad e^{\frac{-n^2+0{,}5n}{2^l}} \cong \mathbf{0,5}$$

$$e^{\frac{-n^2+0{,}5n}{2^l}} \cong \mathbf{0,5} \quad | \; \boldsymbol{ln}()$$

$$n^2 - 0{,}5n \cong -2^l \cdot \ln(\mathbf{0,5}) = 2^l \cdot \ln\left(\frac{1}{\mathbf{0,5}}\right)$$

$$\left(n - \frac{1}{4}\right)^2 - \frac{1}{16} \cong 2^l \cdot \ln\left(\frac{1}{\mathbf{0,5}}\right)$$

$$n \cong \frac{1}{4} + \sqrt{\frac{1}{16} + 2^l \cdot \ln\left(\frac{1}{\mathbf{0,5}}\right)} \cong \sqrt{2^l \cdot \ln\left(\frac{1}{\mathbf{0,5}}\right)} = 0{,}83 \cdot \sqrt{2^l} \cong \sqrt{2^l}$$

Somit lautet die grobe Abschätzung

$$P(K) \geq 0{,}5 \quad \Leftrightarrow \quad n \gtrsim 2^{l/2}$$

Für $l = 6$ Bit liegt die Grenze bei $n \cong 2^3 = 8$, für $l = 10$ Bit bei $n \cong 2^5 = 32$. Die korrekten Werte sind $n = 10$ bzw. $n = 38$.

$n \cong 0{,}83 \cdot \sqrt{2^l}$ hat eine Bitlänge von etwa $l/2$. ($ld\left(0{,}83 \cdot \sqrt{2^l}\right) = l/2 - 0{,}2688 \cong l/2$)

Verlangt man 90%ige Sicherheit ($P(n; K) \leq 10\%$), so ändert sich auch nicht viel:

$$n \cong \sqrt{2^l \cdot \ln\left(\frac{1}{\mathbf{0,1}}\right)} = 1{,}52 \cdot \sqrt{2^l},$$

was auch nur maximal 1 Binärstellen mehr bedeutet: ($ld\left(1{,}52 \cdot \sqrt{2^l}\right) = l/2 + 0{,}6$)

Die Abschätzung zeigt, dass eine Hashfunktion H, die Hashwerte der Länge l erzeugt, lediglich eine „Sicherheit“ bzw. „**Kollisionsresistenz**“ von etwa „$2^{l/2}$“ besitzt.
Eine Bitlänge von 256 Bit hat „nur“ eine Sicherheit von 128 Bit. Das heißt, ein Angreifer benötigt 2^{128} Versuche für eine 50%-Chance, mindestens eine Kollision zu errechnen. Das sind $3{,}4 \ldots \cdot 10^{38}$ Versuche bzw. Rechenoperationen. Eine CPU oder GPU, die z.B. 10^9 Rechenoperationen/s schafft, braucht dazu länger als das geschätzte Alter unseres Universums. Wenn eine Hashfunktion geknackt wird, dann mit intelligenteren Methoden.

SATZ 5.1 (KOLLISIONSRESISTENZ)

Sei H eine Hashfunktion und $|H(\boldsymbol{M})| = l$ die Bitlänge ihrer Hashwerte. Es gibt somit 2^l mögliche Hashwerte.

Um darin mit über 50%iger Wahrscheinlichkeit eine Kollision

$$H(\boldsymbol{M}) = H(\boldsymbol{M}') \quad , \quad \boldsymbol{M} \neq \boldsymbol{M}'$$

zu finden, sind etwa $2^{l/2}$ Versuche nötig.

Eine $\boldsymbol{l}$-Bit Hashfunktion besitzt eine Kollisionsresistenz von $\boldsymbol{2^{l/2}}$.

BEWEIS siehe oben!

5.2 PREIMAGE-ANGRIFFE

Dieselbe Herangehensweise wie vorhin. Gegeben ist ein Hashwert H_0. Gesucht wird eine Nachricht (ein Urbild) $\boldsymbol{M}_1$ mit $H(\boldsymbol{M}_1) = H_0$. (Gilt im Wesentlichen auch für Second Preimage Angriffe)

$P(n; H_0)$ … Wahrscheinlichkeit, bei n Versuchen Übereinstimmungen mit H_0 zu entdecken.
$P(n; \neg H_0)$ … Wahrscheinlichkeit, bei n Versuchen keine Übereinstimmung mit H_0 zu entdecken.

Wir konzentrieren uns vorerst wieder auf die Wahrscheinlichkeit $P(n; \neg H_0)$, keine Übereinstimmung zu entdecken.

Der 1. Versuch mit einem beliebigen Bitstring $\boldsymbol{M}_1 \in \{0,1\}^*$ ergibt den ersten von $2^6 = 64$ Hashwerten. Die Wahrscheinlichkeit keiner Übereinstimmung mit H_0 beträgt somit $^{63}/_{64}$.

Im 2. Versuch mit einem anderen Bitstring $\boldsymbol{M}_2 \in \{0,1\}^* \backslash \{\boldsymbol{M}_1\}$ beträgt die Wahrscheinlichkeit, dass dieser wieder nicht H_0 ergibt erneut, $^{63}/_{64}$, somit zusammen $^{63}/_{64} \cdot {}^{63}/_{64}$.

Die Wahrscheinlichkeit nach n Versuchen noch immer keinen Erfolg zu haben, beträgt

$$P(\mathrm{n}; \neg H_0) = \left(\frac{63}{64}\right)^n = \left(\frac{2^l - 1}{2^l}\right)^n .$$

Die uns interessierende Gegenwahrscheinlichkeit beträgt

$$P(n; H_0) = 1 - \left(\frac{2^l - 1}{2^l}\right)^n .$$

Es sei die Grenze des Akzeptablen wieder $P(n; H_0) = {}^1/_2 = 50\%$.

$$\frac{1}{2} = 1 - \left(\frac{2^l - 1}{2^l}\right)^n$$

$$\frac{1}{2} = \left(\frac{2^l - 1}{2^l}\right)^n \quad | \ln(\square)$$

$$\ln\left(\frac{1}{2}\right) = n \cdot \ln\left(\frac{2^l - 1}{2^l}\right)$$

$$n = \frac{\ln\left(\frac{1}{2}\right)}{\ln\left(\frac{2^l - 1}{2^l}\right)} = \frac{-0{,}693}{\ln\left(\frac{2^l - 1}{2^l}\right)} = \frac{-0{,}693}{\ln\left(1 - \frac{1}{2^l}\right)}$$

Dies lässt sich noch weiter vereinfachen, wenn wir lediglich an einer Abschätzung interessiert sind. Da 2^l in der Praxis eine ziemlich große Zahl ist (z.B. $2^{512} \cong 1{,}34 \cdot 10^{154}$), ist $1 - {}^1/_{2^l} \cong 1$. Wir ersetzen deshalb den Logarithmus durch die ersten beiden Terme der Taylorreihe ($\ln(x) = x - 1$, was der Tangente entspricht) und erhalten

$$n = \frac{-0{,}693}{\ln\left(1 - \frac{1}{2^l}\right)} \cong \frac{-0{,}693}{\left(1 - \frac{1}{2^l}\right) - 1} = 0{,}693 \cdot 2^l .$$

Approximieren wir $0{,}693 \approx 0{,}5$, so erhalten wir den in der Literatur genannten Wert 2^{l-1} für die Sicherheit bzw. Resistenz bezüglich eines Preimage-Angriffs.

SATZ 5.2 (PREIMAGE-RESISTENZ)

Sei H eine Hashfunktion, $|H(\boldsymbol{M})| = l$ die Bitlänge ihrer Hashwerte und H_0 ein bestimmter Hashwert.

Um darin mit über 50%iger Wahrscheinlichkeit ein Urbild

$$\boldsymbol{M} \; mit \; H(\boldsymbol{M}) = H_0$$

zu finden, sind etwa 2^{l-1} Versuche nötig.

Eine $\boldsymbol{l}$-Bit Hashfunktion hat bezüglich Preimage (und Second-Preimage) eine Resistenz von $\boldsymbol{2^{l-1}}$.

6 AUFGABEN

1 *PADDING*

Nachricht: „$\overbrace{00101101}\ \overbrace{11010111}\ \overbrace{00000101}\ \overbrace{00010000}\ \overbrace{00010011}\ \overbrace{10101011}\ 111001\underbrace{\ldots\ldots\ldots\ldots}_{Padding}$“

AUFGABENSTELLUNG:

Ergänze den Paddingteil (8 Bit für die Bitstringlänge)!

Lsg.: $\overbrace{00101101}\ \overbrace{11010111}\ \overbrace{00000101}\ \overbrace{00010000}\ \overbrace{00010011}\ \overbrace{10101011}\ \overbrace{111001\mathbf{10}}\ \overbrace{\mathbf{00110110}}$

2 *HELLO!*

Das Wort „Hello!“ ist binär codiert als „0100100001100101011011000110110001101111100100001“.

Gegeben sei die Merkle-Damgård-Hashfunktion MDχ, mit folgender **Kompressionsfunktion** h_χ:

$$\begin{array}{ccccc} h_\chi: & \{0,1\}^{10}\times\{0,1\}^{14} & \longrightarrow & \{0,1\}^{10} & \\ & (\boldsymbol{f_i};\boldsymbol{m_{i+1}}) & \mapsto & \boldsymbol{f_{i+1}}=h(\boldsymbol{f_i};\boldsymbol{m_{i+1}}) & ;\boldsymbol{f_0}=1010101010 \end{array}$$

Der Bitstring $(\boldsymbol{f_i};\boldsymbol{m_{i+1}})$ wird in 3 gleich lange Teilstrings $(\boldsymbol{f_i};\boldsymbol{m_{i+1}})=(\boldsymbol{A};\boldsymbol{B};\boldsymbol{C})$ zerlegt und danach folgendermaßen verrechnet:

$$(A;00)xor(0;B;0)xor(00;C)$$

Beispiel:

$\underbrace{1010101010}_{f_0}\underbrace{0100100001100 1}_{m_1}$ [01011 ……..00001]

$\overbrace{10101010}^{\boldsymbol{A}}\overbrace{10010010}^{\boldsymbol{B}}\overbrace{00011001}^{\boldsymbol{C}}$ [01011 ……..00001]

$$\begin{array}{rl} & 1010101\mathbf{00} \\ xor & \mathbf{0}100100\mathbf{10} \\ xor & \mathbf{00}00011001 \\ = & \mathbf{1110010101} \end{array}$$

Padding: 6 Bit für die Bitstringlänge.

AUFGABENSTELLUNG:

a) Wie viele Zeichen kann der Text maximal umfassen?
b) Ergänze den Bitstring um das Padding!
c) Berechne den Hashwert MDχ(*Hello!*)!

[Lsg: a) 63 Zeichen

b) 0100100001100 1 01011011000110 11000110111100 100001**10110000** → 1111010010]

3 *SHAKESPEARE*

Die Kompressionsfunktion h einer M/D-Hashfunktion komprimiert Bitstrings der Länge 128 Bit auf 80 Bit.

AUFGABENSTELLUNG:

Wie viele Runden werden für das folgende 240 Bit lange Passwort benötigt?

„tobeornottobethatisthequestion”

“01110100011011110110001001100101011011110111001001101110011011110111010001110100
01101111011000100110010101110100011010000110000101110100011010010111001101110100
01101000011001010111000101110101011001010111001101110100011010010110111101101110”

Lsg.: $|(f_{i-1}; m_i)| = 128 \Longrightarrow |m_i| = 48 \Longrightarrow 240/48 + \underbrace{1}_{Padding} = 6$

$$(f_0; m_1 = 01110100011011110110001001100101011011110111001 0) \longrightarrow f_1$$
$$(f_1; m_2 = 01101110011011110111010001110100011011110110001 0) \longrightarrow f_2$$
$$(f_2; m_3 = 01100101011101000110100001100001011101000110100 1) \longrightarrow f_3$$
$$(f_3; m_4 = 01110011011101000110100001100101011100010111010 1) \longrightarrow f_4$$
$$(f_4; m_5 = 01100101011100110111010001101001011011110110111 0) \longrightarrow f_5$$

Padding:

$$(f_5; m_6 = 10000000000000000000000000000000000000000011110000) \longrightarrow H(M)$$

4 *STIRLING*

Für die Abschätzung der Wahrscheinlichkeit, bei einer Bitlänge l der Hashwerte nach n Versuchen noch immer keine Kollision gefunden zu haben, wurde die Stirlingformel " $n! \cong \sqrt{2\,\pi\,n} \cdot \left(\frac{n}{e}\right)^n$ " verwendet.

AUFGABENSTELLUNG:

Die Aufgabenstellung bezieht sich darauf, die Abschätzung

$$\frac{2^l!}{(2^l - n)! \cdot (2^l)^n} \cong \left(\frac{1}{1 - \frac{n}{2^l}}\right)^{2^l - n + 0{,}5} \cdot e^{-n}$$

nachzuvollziehen.

5 *ABSCHÄTZUNG*

Die Abschätzung, wie viele Versuche nötig sind, dass die Wahrscheinlichkeit, mindestens eine Kollision zu finden, die 50%-Marke **übersteigt**, lautet

$$P(n; K) = \mathbf{0,5} \leq 1 - \left(\frac{2^l}{2^l - n}\right)^{2^l - n + 0{,}5} \cdot e^{-n}$$

AUFGABENSTELLUNG:

Kennzeichne in weiterer Folge, welche Relationszeichen (≤ , ≥) das richtige ist.

$$\left(\frac{2^l}{2^l - n}\right)^{2^l - n + 0{,}5} \cdot e^{-n} \leq \geq \mathbf{0,5} \quad \text{bzw.} \quad \left(\frac{1}{1 - \frac{n}{2^l}}\right)^{2^l - n + 0{,}5} \cdot e^{-n} \leq \geq \mathbf{0,5}$$

Die Anzahl der nötigen Versuche n ist offensichtlich klein im Vergleich zu 2^l, der Anzahl der zu testenden Hashwerte. Der Vergleich der Taylorreihen von

$$\frac{1}{1-x} = 1 + x + x^2 + x^3 + \cdots \quad \text{und} \quad e^x = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \cdots$$

ergibt für kleine Zahlen $\frac{1}{1-x} \leq\geq e^x$ (Tangente!) und somit

$$\left(e^{\frac{n}{2^l}}\right)^{2^l - n + 0{,}5} \cdot e^{-n} \leq\geq \mathbf{0,5} \quad \text{bzw.} \quad e^{\frac{-n^2+0{,}5n}{2^l}} \leq\geq \mathbf{0,5}$$

$$e^{\frac{-n^2+0{,}5n}{2^l}} \leq\geq \mathbf{0,5} \quad | \; \boldsymbol{ln}()$$

$$n^2 - 0{,}5n \leq\geq -2^l \cdot \ln(\mathbf{0,5}) = 2^l \cdot \ln\left(\frac{1}{\mathbf{0,5}}\right)$$

$$\left(n - \frac{1}{4}\right)^2 - \frac{1}{16} \leq\geq 2^l \cdot \ln\left(\frac{1}{\mathbf{0,5}}\right)$$

$$n \leq\geq \frac{1}{4} + \sqrt{\frac{1}{16} + 2^l \cdot \ln\left(\frac{1}{\mathbf{0,5}}\right)} \leq\geq \sqrt{2^l \cdot \ln\left(\frac{1}{\mathbf{0,5}}\right)} = 0{,}83 \cdot \sqrt{2^l} \leq\geq \sqrt{2^l}$$