



- Seminararbeit -

Sicherheit der Merkle-Damgård-Konstruktion

Marc Oelsner

Dozenten: Herr Prof. Köbler, Herr Kuhnert

30. November 2010

Fachbereich Informatik
der Humboldt-Universität zu Berlin

Inhaltsverzeichnis

1	Einführung	1
2	Schlüssellose Hashverfahren	2
2.1	Sicherheitsanforderungen	2
2.2	Das Zufallsorakelmodell	4
2.3	Iterierte Hashfunktionen	8
3	Die Merkle-Damgård-Konstruktion	9
4	Fazit	15
5	Literaturverzeichnis	16

1 Einführung

Die digitale Nachrichtenübermittlung der heutigen Zeit ist vielen Gefahren ausgesetzt. Darunter fallen vor allem die Möglichkeiten, Nachrichten abzufangen, sie zu modifizieren und die veränderte Nachricht an den eigentlichen Empfänger weiterzuleiten. Ohne technische Hilfsmittel könnte der Empfänger keine Manipulation feststellen, sofern der Absender nicht nach dem korrekten Inhalt befragt wird.

Um dieser Gefahr entgegenzuwirken, bedient man sich kryptografischer Hashverfahren. Diese Maßnahmen sollen für die Beteiligten einer Nachrichtenübermittlung Schutzfunktionen bieten. Sie sollen die Integrität von Nachrichten bzw. jeglicher Art digitaler Daten sicherstellen. Des Weiteren können mit Hilfe dieser Hashverfahren Vertraulichkeiten, wie bspw. die Anonymität der Personen oder die Geheimhaltung der Nachricht, bewahrt werden.

Durch die Anwendung eines kryptografischen Hashverfahrens h auf eine beliebige Nachricht x wird eine sogenannte Prüfsumme (der *Hashwert*) berechnet. Diese Prüfsumme wird als $h(x)$ bezeichnet. Sie steht repräsentativ für die Identifikation einer Nachricht x und wird gelegentlich als digitaler Fingerabdruck bezeichnet, da sie ähnlich wie der menschliche Fingerabdruck durch besondere Merkmale ausgezeichnet ist.

Mit Hilfe eines solchen Verfahrens können der Absender und der Empfänger die beiden Hashwerte der Nachrichten berechnen. Dabei sollten die Prüfsummen der Originalnachricht und der empfangenen Nachricht übereinstimmen. Andernfalls kann von einer Manipulation während der Übertragung der Nachricht ausgegangen werden.

Die kryptografischen Hashverfahren können ganz allgemein in zwei verschiedene Klassen eingeteilt werden (vgl. Abb. 1). Zum Einen wären dies die sogenannten *schlüssellosen* Verfahren und zum Anderen die Hashverfahren, bei denen ein geheimer Schlüssel verwendet wird.

In dieser Arbeit soll jedoch nur auf einen Repräsentanten der schlüssellosen Klasse eingegangen werden. Um den Begriff eines schlüssellosen Hashverfahrens zu verstehen, soll dieser zunächst näher betrachtet werden.

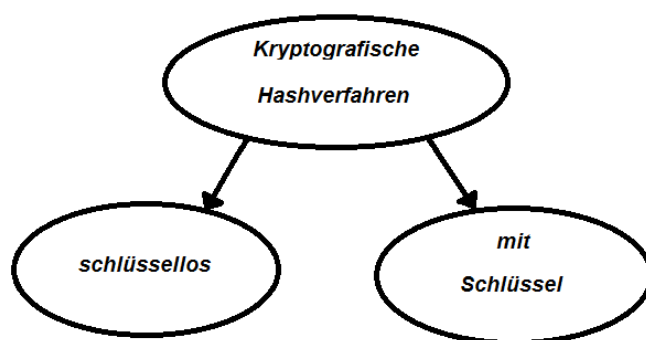


Abbildung 1: Grobe Einteilung der Hashverfahren in zwei Klassen

2 Schlüssellose Hashverfahren

Zu den schlüssellosen Hashverfahren gehören solche, die der Integrität von Nachrichten dienen sollen. Dieses Schutzziel besagt, dass Daten über einen konkreten Zeitraum unverändert und vollständig sein sollen. Eine Veränderung von Daten könnte aufgrund eines technischen Fehlers, einer unabsichtlichen oder einer gar absichtlichen Modifikation eintreten. Die Integrität umfasst also die *Datensicherheit* (= Schutz vor Verlust) und die *Fälschungssicherheit* (= Schutz gegen vorsätzliche Veränderung). Die Integrität ist also genau dann gewährleistet, wenn eine Nachricht x vom Absender unverändert und vollständig den Empfänger erreicht.

Auf die Nennung weiterer schlüsselloser Hashverfahren wird an dieser Stelle verzichtet.

Die Hashverfahren, die der Erkennung vorsätzlicher Manipulationen dienen, werden als *MDC* bezeichnet (engl. für *Modification Detection Code*).

Um die Integrität einer beliebigen Nachricht x sicherzustellen, dient das folgende typische Beispiel als Anschauung (vgl. Abb. 2).

Angenommen Bob möchte eine Nachricht x über einen ungesicherten Kanal an Alice senden (bspw. eine Webanwendung). Die bei Alice angekommene Nachricht werde als x' bezeichnet. Bob berechnet den Hashwert $h(x)$ seines Originaltextes und schickt diesen über einen sicheren Kanal an Alice.

Alice muss nun prüfen, ob die Berechnung des Hashwertes ihrer Nachricht, dem Wert des Hashwertes von Bob gleicht – ist also $h(x) = h(x')$?

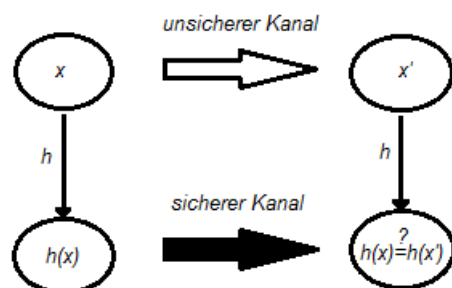


Abbildung 2: Beispiel für die Anwendung eines MDC

2.1 Sicherheitsanforderungen

Die schlüssellosen Hashverfahren oder auch Hashfunktionen sind weitestgehend bekannt. Dennoch finden sie im täglichen Gebrauch Anwendung und müssen aus diesem Grund konkreten Sicherheitsanforderungen genügen.

Im Folgenden sollen drei Sicherheitskriterien an eine Hashfunktion vorgestellt werden.

Nehmen wir an, dass $h : \mathcal{X} \rightarrow \mathcal{Y}$ eine öffentliche, also schlüssellose Hashfunktion ist. Sei $x \in \mathcal{X}$ eine beliebige Nachricht und sei $y := h(x)$. Ein Paar $(x, y) \in \mathcal{X} \times \mathcal{Y}$ heißt gültig für h , falls $y = h(x)$. Im Allgemeinen ist es wünschenswert, dass die Paare (x, y) nur auf dem Weg konstruiert werden, dass zu einer gegebenen Nachricht x der Wert y berechnet wird, indem die Funktion h auf x angewendet wird.

Ein Paar (x, x') wird als Kollisionspaar für h bezeichnet, falls $h(x) = h(x')$.

Mit $m := |\mathcal{Y}|$ sei die Anzahl der Hashwerte bezeichnet. Ist ferner der Textraum \mathcal{X} endlich, mit $|\mathcal{X}| =: n$, so fordern wir oft, dass $n \geq 2m$. Wenn die Textlänge größer ist als die Länge des berechneten Hashwertes, so wird h eine (n, m) -Kompressionsfunktion genannt.

Das Problem der öffentlichen Hashfunktionen h besteht nun darin, dass es sehr einfach ist, zu einer gegebenen Nachricht x den zugehörigen Hashwert $y = h(x)$ zu berechnen. Auf diese Weise lassen sich gültige Paare (x, y) schnell erzeugen.

Für kryptografische Anwendungen soll es weiterhin schwer sein, Informationen zu gewinnen, wenn der Hashwert y bereits bekannt ist. Wir erhalten ein erstes Kriterium.

Problem 1: Die Bestimmung eines Urbildes

Gegeben: Eine Hashfunktion $h : \mathcal{X} \rightarrow \mathcal{Y}$ und ein Hashwert $y \in \mathcal{Y}$.

Gesucht: Eine Nachricht $x \in \mathcal{X}$ mit $h(x) = y$.

Das Urbildproblem besagt also, dass zu einem gegebenen Hashwert y eine Nachricht x gesucht wird mit $h(x) = y$. Wenn ein Urbild x zu einem Hashwert y gefunden wird, so ist das entstehende Paar (x, y) gültig.

Falls das Finden eines solchen Elements x jedoch einen enormen Aufwand erfordert, so wird h als Einweg-Hashfunktion oder auch als urbildresistent bezeichnet (engl. *one-way* oder *preimage resistant*).

Dieses Verfahren wird beispielsweise bei Anmeldungen durch Passwörter benötigt.

Die Hashwerte der Passwörter werden dabei z. B. in einer Datenbank gespeichert. Der Hashwert des vom Anwender eingegebenen Passwortes wird durch die Hashfunktion h berechnet und es wird geprüft, ob ein solcher Wert in der Datenbank vorliegt. Umgekehrt soll das Auslesen der Passwörter jedoch nicht möglich sein.

Ein weiteres Kriterium ist das sogenannte „Zweite Urbildproblem“

Problem 2: Die Bestimmung eines zweiten Urbildes

Gegeben: Eine Hashfunktion $h : \mathcal{X} \rightarrow \mathcal{Y}$ und eine Nachricht $x \in \mathcal{X}$.

Gesucht: Eine Nachricht $x' \in \mathcal{X}$ mit $x' \neq x$, sodass $h(x') = h(x)$.

Das zweite Urbildproblem besagt, dass zu einer gegebenen Nachricht x eine weitere Nachricht x' mit $x \neq x'$ gesucht wird, sodass $h(x') = h(x)$. Infolgedessen wäre das Paar $(x', h(x))$

als gültig einzustufen. Eine Hashfunktion heißt schwach kollisionsresistent (engl. *second preimage resistant*), wenn das Finden einer solchen Nachricht x' nur unter großem Aufwand möglich ist. Diese Eigenschaft wird in Abbildung 2 benötigt, wobei der Angreifer eine Nachricht finden muss, sodass die Hashwerte übereinstimmen.

Angenommen einem Angreifer ist nur die Hashfunktion h bekannt, so steht er vor folgendem Problem.

Problem 3: Finden einer Kollision

Gegeben: Eine Hashfunktion $h : \mathcal{X} \rightarrow \mathcal{Y}$.

Gesucht: Nachrichten $x, x' \in \mathcal{X}$, sodass $x' \neq x$ und $h(x') = h(x)$.

Das Finden zweier verschiedener Nachrichten x, x' sodass $h(x') = h(x)$ wird als Kollisionsproblem bezeichnet. Angenommen (x, y) wäre ein gültiges Paar und x, x' eine Kollision, dann ist auch (x', y) ein gültiges Paar.

Hashfunktionen, die einen immensen Aufwand erfordern, eine solche Kollision zu finden, werden als (*stark*) kollisionsresistent bezeichnet (engl. *collision resistant*).

2.2 Das Zufallsorakelmodell

Ausgehend von einem Angriff auf eine Hashfunktion $h : \mathcal{X} \rightarrow \mathcal{Y}$ benutzen wir das Zufallsorakelmodell (ZOM) dazu, die Effizienz dieser Angriffe abzuschätzen.

Nehmen wir an, dass die Menge der Nachrichten \mathcal{X} sowie die Menge der Hashwerte \mathcal{Y} bekannt seien. Dann können wir ausgehend von diesen beiden Mengen eine Funktion so konstruieren, dass im theoretischen Sinn zu jedem $x \in \mathcal{X}$ ein zufälliges $y \in \mathcal{Y}$ liefert, sodass also $h(x) = y$ ist.

Eine ähnliche Konstruktion ist die folgende. Nehmen wir an, dass die Menge \mathcal{H} die Menge aller Hashfunktionen ist, die von \mathcal{X} nach \mathcal{Y} abbilden. Aus dieser Menge wählen wir nun zufällig eine Funktion $h \in \mathcal{H}$ aus, mit deren Hilfe die Hashwerte berechnet werden.

Dieses Vorgehen ist ab einer bestimmten Größe für den Angreifer nicht mehr effizient genug. Der Aufwand wäre zu groß, um Informationen über die Funktion h zu erhalten. Aus diesem Grund ähnelt dieses theoretische Modell der Befragung eines Orakels. Eine gute kryptografische Hashfunktion h besitzt weiterhin die Eigenschaft, dass auch die Berechnung von Hashwerten schwer vorhersagbar ist, wenn bereits einige dieser Werte von Nachrichten bekannt sind.

Der folgende Satz liefert uns eine Aussage darüber, mit welcher Wahrscheinlichkeit die Berechnung des Hashwertes $h(x)$ einer Nachricht x einem beliebigen $y \in \mathcal{Y}$ gleicht.

Satz 1

Sei $\mathcal{X}_0 = \{x_0, x_1, \dots, x_k\}$ eine beliebige Menge von k verschiedenen Nachrichten aus \mathcal{X} und seien weiter $y_1, \dots, y_k \in \mathcal{Y}$. Dann gilt für eine zufällig ausgewählte Funktion $h \in \mathcal{H}$ und jedes Paar $(x, y) \in \{\mathcal{X} - \mathcal{X}_0\} \times \mathcal{Y}$:

$$\Pr [h(x) = y] = \frac{1}{m}.$$

Dabei war $m = |\mathcal{Y}|$ die Anzahl der Hashwerte. Folglich ist dies der Ausdruck einer stochastisch unabhängigen Gleichverteilung.

Diese Eigenschaft ist oft der Ansatz für Beweise, in denen nach Komplexitäten gefragt wird.

Sei folgender Algorithmus¹ zur Bestimmung eines Urbildes gegeben:

Algorithm 1: FINDPREIMAGE(h, y, q)

```

choose any  $\mathcal{X}_0 \subseteq \mathcal{X}$ ,  $q = |\mathcal{X}_0|$ 
  for each  $x \in \mathcal{X}_0$ 
    do { if  $h(x) = y$ 
         then return ( $x$ )
      }
  return (failure)

```

Um nun eine Aussage über die obere Komplexitätsschranke zu treffen, betrachten wir diesen Algorithmus. Die Erfolgswahrscheinlichkeit zu einem Hashwert y das Urbild x zu finden, ist durch den folgenden Satz gegeben.

Satz 2

Der **Algorithmus 1** liefert (bei zufälliger Wahl von \mathcal{X}_0) mit einer Erfolgswahrscheinlichkeit von $\epsilon = 1 - (1 - \frac{1}{m})^q$ ein Urbild des Hashwertes y .

Beweis: Sei $y \in \mathcal{Y}$ fixiert und sei $\mathcal{X}_0 = \{x_0, x_1, \dots, x_q\}$. Sei für $1 \leq i \leq q$ E_i das Ereignis „ $h(x) = y$ “. Nach Satz 1 sind diese Ereignisse stochastisch unabhängig und ihre Wahrscheinlichkeit ist $\Pr [E_i] = \frac{1}{m}$. Also ist danach

$$\Pr [E_1 \cup \dots \cup E_q] = 1 - \Pr [\bar{E}_1 \cap \dots \cap \bar{E}_q] = 1 - \left(1 - \frac{1}{m}\right)^q$$

□

¹D. R. Stinson: Cryptography, Theory and Practice, 2002 Chapman & Hall/CRC, S. 121

Anschließend lässt sich an den Algorithmus zur Bestimmung eines Urbildes analog ein Algorithmus zur Bestimmung eines zweiten Urbildes konstruieren.

Algorithm 2: FINDSECONDPREIMAGE(h, x, q)

```

 $y \leftarrow h(x)$ 
choose  $\mathcal{X}_0 \subseteq \mathcal{X} \setminus \{x\}$ ,  $|\mathcal{X}_0| = q - 1$ 
  for each  $x_0 \in \mathcal{X}_0$ 
    do { if  $h(x_0) = y$ 
         then return ( $x_0$ )
       }
  return (failure)

```

Dazu erhalten wir folgenden Satz.

Satz 3

Der **Algorithmus 2** liefert mit einer Erfolgswahrscheinlichkeit von $\epsilon = 1 - (1 - \frac{1}{m})^{q-1}$ ein zweites Urbild mit $x_0 \neq x$ des Hashwertes $y = h(x)$.

Der Beweis ist analog zu dem Beweis von Satz 2.

Als nächstes wird ein elementarer Algorithmus zum Finden einer Kollision betrachtet.

Algorithm 3: FINDCOLLISION(h, q)

```

choose  $\mathcal{X}_0 \subseteq \mathcal{X} \setminus \{x\}$ ,  $|\mathcal{X}_0| = q$ 
for each  $x \in \mathcal{X}_0$ 
  do  $y_x \leftarrow h(x)$ 
if  $y_x = y_{x'}$  for some  $x' \neq x$ 
  then return ( $x, x'$ )
else return (failure)

```

Anhand dieses Algorithmus ergibt sich der Satz 4.

Satz 4

Der **Algorithmus 3** liefert mit einer Erfolgswahrscheinlichkeit von

$$\epsilon = 1 - \frac{(m-1)(m-2) \cdots (m-q+1)}{m^{q-1}}$$

ein Kollisionspaar (x, x') für h .

Der Beweis soll an dieser Stelle nicht geführt werden. Er findet sich aber im *Stinson, Cryptography Theory and Practice, 2002* auf Seite 123f. wieder.

Ein Beispiel eines Kollisionsproblems ist das sogenannte Geburtstagsparadoxon. Man kann berechnen, wie viele Personen in einem Raum nötig sind, sodass zwei von ihnen mit einer Wahrscheinlichkeit von 50% am selben Tag Geburtstag haben. Die Antwort auf diese Frage ist verblüffend, denn es sind lediglich 23 Personen nötig (genauer: 22,3 Personen). Denn mit $m = 365$ und $q = 23$ erhalten wir $\epsilon \approx 0.507$.

Innerhalb des Zufallsorakelmodells ist es also einfacher eine Kollision zu finden, als ein Urbild bzw. ein zweites Urbild. Es stellt sich nun die Frage, ob das Kollisionsproblem die beiden anderen Probleme impliziert.

Sei nun folgender Algorithmus gegeben.

Algorithm 4: COLLISIONTOSECONDPREIMAGE(h)

external ORACLE2NDPREIMAGE

choose $x \in \mathcal{X}$ uniformly at random

if (Oracle2ndPreimage(h, x) = x') **and** $x' \neq x$ **and** ($h(x) = h(x')$)

then return (x, x')

else return (failure)

Angenommen ORACLE2NDPREIMAGE ist ein (ϵ, q) -Algorithmus, der das zweite Urbildproblem für ein ϵ , ein festes q und für eine feste Hashfunktion h löst. Dann ist **Algorithmus 4** ein $(\epsilon, q+2)$ -Algorithmus zur Berechnung des Kollisionsproblems bei gleicher Hashfunktion h (mit gleichem ϵ, q). Dieser Algorithmus ist unabhängig von der Funktion h durchführbar. Das bedeutet, dass die Hashfunktion h keine Voraussetzungen erfüllen muss.

Es lässt sich nun schlussfolgern, dass die Kollisionsresistenz impliziert, dass das Finden eines zweiten Urbilds schwierig ist.

Die interessantere Frage ist, ob die Kollisionsresistenz auch die Urbildresistenz impliziert. Zeigen wir also, dass sich das Kollisionsproblem auf das Urbildproblem reduzieren lässt.

Satz 5

Sei $h : \mathcal{X} \rightarrow \mathcal{Y}$ eine (n, m) -Hashfunktion mit $n \geq 2m$. Dann ist das Problem P3, ein Kollisionspaar für h zu bestimmen, auf das Problem P1, ein Urbild zu bestimmen, reduzierbar.

Beweis: Sei \mathcal{A} ein Invertierungsalgorithmus für h , d.h. \mathcal{A} berechnet für jeden Hashwert y in $W(h) = \{h(x) \mid x \in \mathcal{X}\}$ ein Urbild x mit $h(x) = y$. Betrachte den folgenden Las-Vegas Algorithmus \mathcal{B} .

Algorithm 5: LAS-VEGAS-ALGORITHM

```

choose  $x \in \mathcal{X}$  randomly
let  $y := h(x)$ 
let  $x' := \mathcal{A}(y)$ 
if ( $x \neq x'$ )
    then return ( $x, x'$ )
else return (failure)

```

Sei nun $\mathcal{C} = \{h^{-1}(y) \mid y \in \mathcal{Y}\}$. Dann hat \mathcal{B} eine Erfolgswahrscheinlichkeit von

$$\sum_{C \in \mathcal{C}} \frac{\|C\|}{\|\mathcal{X}\|} \cdot \frac{\|C\| - 1}{\|C\|} = \frac{1}{n} \cdot \sum_{C \in \mathcal{C}} (\|C\| - 1) = \frac{n - m}{n} \geq \frac{1}{2}.$$

□

2.3 Iterierte Hashfunktionen

In diesem Kapitel soll nun überlegt werden, wie eine kollisionsresistente Kompressionsfunktion $k : \mathcal{X}_0 \rightarrow \mathcal{Y}$ zu einer kollisionsresistenten Hashfunktion $h : \mathcal{X} \rightarrow \mathcal{Y}$ erweitert werden kann. Dabei seien nun der Definitionsbereich \mathcal{X} und der (Hash-)Wertebereich \mathcal{Y} in binärer Darstellung. Also ist $k : \{0, 1\}^q \rightarrow \{0, 1\}^p$ und auch $h : \{0, 1\}^* \rightarrow \{0, 1\}^p$, wobei $p, q \in \mathbb{N}$ (und meist $q \geq p$).

Eine auf diesem Weg konstruierte Funktion h wird iterative Hashfunktion genannt.

Im Folgenden schreiben wir $x||y$ für die Konkatenation zweier Bitstrings x und y .

Sei die Funktion $k : \{0, 1\}^{m+t} \rightarrow \{0, 1\}^m$ eine Kompressionsfunktion, wobei $t \geq 1$ ist. Auf der Grundlage einer Kompressionsfunktion erhalten wir in drei Schritten eine iterative Hashfunktion h .

Zunächst wird das sogenannte „Preprocessing“ durchgeführt. Dieser Vorgang soll eine beliebige Nachricht x , mit $|x| \leq m + t + 1$, in einen neuen String $y = y_1||y_2||\dots||y_r$ überführt werden, sodass $|y_i| \equiv 0 \pmod{t}$ ist. Die Länge der r Blöcke sei $|y_i| = t$ für $i = 1, \dots, r$.

Nach dem *Preprocessing* wird das „Processing“ angewendet. Sei IV ein beliebiger Initialwert in Form eines Bitstrings der Länge m . Dann berechnen wir iterativ Werte mit

$$\begin{aligned}
z_0 &\leftarrow IV \\
z_1 &\leftarrow k(z_0||y_1) \\
z_2 &\leftarrow k(z_1||y_2) \\
&\vdots \\
z_r &\leftarrow k(z_{r-1}||y_r)
\end{aligned}$$

Abschließend an das *Processing* kann nun noch die Ausgabe (engl. *optional output transformation*) des finalen Wertes z_r modifiziert werden.

Sei $g : \{0, 1\}^m \rightarrow \{0, 1\}^l$ eine beliebige öffentliche Funktion. Dann definieren wir unseren Hashwert mit $h(x) = g(z_r)$.

Die oben iterierte Hashfunktion ist $h : \bigcup_{i=m+t+1}^{\infty} \{0, 1\}^i \rightarrow \{0, 1\}^l$.

Ein weiterer üblicher Schritt im *Preprocessing* ist durch die Bildung des Bitstrings y mit

$$y = x || \text{pad}(x)$$

gegeben, wobei pad eine *padding*-Funktion ist. In der Regel wird an die Nachricht x die Länge $|x|$ angehängt. Damit y dennoch ein Vielfaches von t bleibt, werden beim Padding führende Nullen an die Länge $|x|$ konkateniert.

Das Preprocessing muss die Injektivität von $x \mapsto y$ gewährleisten. Andernfalls ist die Kollisionsresistenz nicht gegeben. Des Weiteren ist nun $|y| = r \cdot t \leq |x|$ wegen der Forderung nach Injektivität.

Ein wichtiger Repräsentant zur Konstruktion iterativer Hashfunktionen ist die Merkle-Damgård-Konstruktion, die im nächsten Kapitel vorgestellt wird. Diese Konstruktionsart bietet uns die Möglichkeit, Aussagen über die Sicherheit zu treffen.

3 Die Merkle-Damgård-Konstruktion

Die besondere Eigenschaft der Merkle-Damgård-Konstruktion ist, dass sie die gewünschten Sicherheiten eines kryptografischen Hashverfahrens erfüllt, sofern die zu Grunde liegende Kompressionsfunktion kollisionsresistent ist.

Sei k eine kollisionsresistente Kompressionsfunktion mit $k : \{0, 1\}^{m+t} \rightarrow \{0, 1\}^m$ und $t \geq 1$. Im Folgenden wollen wir nun mit Hilfe von k eine kollisionsresistente Hashfunktion $h : \mathcal{X} \rightarrow \{0, 1\}^m$ konstruieren, mit

$$\mathcal{X} = \bigcup_{i=m+t+1}^{\infty} \{0, 1\}^i.$$

Zunächst betrachten wir den Fall für $t \geq 2$.

Seien $x \in \mathcal{X}$ Bitstrings mit $|x| = n \geq m + t + 1$. Dieser Bitstring lässt sich, ähnlich wie zuvor betrachtet, in Blöcke gleicher Länge zerlegen, von denen der letzte, k -te Block mit Nullen aufgefüllt wird. Dann ist also

$$x = x_1 || x_2 || \dots || x_k$$

mit $|x_1| = |x_2| = \dots = |x_{k-1}| = t - 1$ und $|x_k| = t - 1 - d$, wobei d , mit $0 \leq d \leq t - 2$, die Anzahl der angehängten Nullen angibt. Damit erhalten wir also die Anzahl der Blöcke mit

$$k = \left\lceil \frac{n}{t-1} \right\rceil.$$

Algorithm 6: MERKLEDAMGÅRD(x)

```

external  $k$ 
comment:  $k : \{0, 1\}^{m+t} \rightarrow \{0, 1\}^m$ , where  $t \geq 2$ 
 $n \leftarrow |x|$ 
 $k \leftarrow \lceil n/(t-1) \rceil$ 
 $d \leftarrow k \cdot (t-1) - n$ 
for  $i \leftarrow 1$  to  $k-1$ 
  do  $y_i \leftarrow x_i$ 
 $y_k \leftarrow x_k \parallel 0^d$ 
 $y_{k+1} \leftarrow$  the binary representation of  $d$ 
 $z_1 \leftarrow 0^{m+1} \parallel y_1$ 
 $g_1 \leftarrow k(z_1)$ 
for  $i \leftarrow 1$  to  $k$ 
  do  $\begin{cases} z_{i+1} \leftarrow g_i \parallel 1 \parallel y_{i+1} \\ g_{i+1} \leftarrow k(z_{i+1}) \end{cases}$ 
 $h(x) \leftarrow g_{k+1}$ 
return ( $h(x)$ )

```

Es ist also

$$y(x) = y_1 \parallel y_2 \parallel \cdots \parallel y_{k+1}.$$

Man bemerke hierbei, dass der Block y_k durch den von d Nullen aufgefüllten Block x_k konstruiert wird. Ferner ist y_{k+1} durch führende Nullen und der binären Darstellung von d gebildet worden. Auf diese Weise erhalten wir für $1 \leq i \leq k+1$: $|y_i| = t-1$.

Wir hashen also x , indem zunächst $y(x)$ konstruiert wird. Es muss darauf geachtet werden, dass die Konstruktion von y_{k+1} die Injektivität von $x \mapsto y(x)$ gewährleistet.

Diese Eigenschaft wurde gefordert, damit die iterative Funktion kollisionsresistent ist.

Der folgende Satz liefert uns die Vererbung der Kollisionsresistenz der Kompressionsfunktion auf eine Hashfunktion.

Satz 6

Sei $k : \{0, 1\}^{m+t} \rightarrow \{0, 1\}^m$ eine kollisionsresistente Kompressionsfunktion mit $t \geq 2$.

Dann ist die Funktion

$$h : \bigcup_{i=m+t+1}^{\infty} \{0, 1\}^i \rightarrow \{0, 1\}^m,$$

so wie im Algorithmus 6 konstruiert, eine kollisionsresistente Hashfunktion.

Beweis: Angenommen, wir haben zwei Nachrichten $x \neq x'$ mit $h(x) = h(x')$. Wir werden zeigen, dass wir in polynomieller Zeit eine Kollision finden.

Wir schreiben $y(x) = y_1 \parallel y_2 \parallel \dots \parallel y_{k+1}$ und $y(x') = y'_1 \parallel y'_2 \parallel \dots \parallel y'_{l+1}$, wobei x, x' mit d bzw. d' Nullen gepaddet wurden. Weiter schreiben wir für die im Algorithmus berechneten Werte der g_i als g_1, \dots, g_{k+1} bzw. g'_1, \dots, g'_{l+1} .

Dabei betrachten wir nun zwei Fälle: Entweder ist $|x| \equiv |x'| \pmod{t-1}$ oder nicht.

Fall 1: $|x| \not\equiv |x'| \pmod{t-1}$.

Hier ist $d \neq d'$ und $y_{k+1} \neq y'_{l+1}$. Wir haben mit

$$\begin{aligned} k(g_k \parallel 1 \parallel y_{k+1}) &= g_{k+1} \\ &= h(x) \\ &= h(x') \\ &= g'_{l+1} \\ &= k(g'_l \parallel 1 \parallel y'_{l+1}) \end{aligned}$$

eine Kollision für h , da $y_{k+1} \neq y'_{l+1}$.

Fall 2: $|x| \equiv |x'| \pmod{t-1}$.

Es ist hier günstig diesen Fall noch einmal zu trennen, sodass wir zwei Teilbetrachtungen durchführen.

Also ist **Fall 2a:** $|x| = |x'|$.

Hier haben wir $k = l$ und $y_{k+1} = y'_{l+1}$. Wir beginnen wie im ersten Fall mit

$$\begin{aligned} k(g_k \parallel 1 \parallel y_{k+1}) &= g_{k+1} \\ &= h(x) \\ &= h(x') \\ &= g'_{k+1} \\ &= k(g'_k \parallel 1 \parallel y'_{k+1}). \end{aligned}$$

Falls $g_k \neq g'_k$ ist, dann finden wir eine Kollision für unsere Kompressionsfunktion k . Nehmen wir also an, dass $g_k = g'_k$ ist, dann ist

$$\begin{aligned} k(g_{k-1} \parallel 1 \parallel y_k) &= g_k \\ &= g'_k \\ &= k(g'_{k-1} \parallel 1 \parallel y'_k). \end{aligned}$$

Entweder finden wir nun eine Kollision für k oder es ist $g_{k-1} = g'_{k-1}$ und $y_k = y'_k$. Finden wir keine Kollision, arbeiten wir solange weiter, bis wir Folgendes betrachten können.

$$\begin{aligned} k(0^{m+1} \parallel y_1) &= g_1 \\ &= g'_1 \\ &= k(0^{m+1} \parallel y'_1). \end{aligned}$$

Ist $y_1 \neq y'_1$, dann finden wir eine Kollision für k . Nehmen wir also an, dass $y_1 = y'_1$ ist. Dann ist aber $y_i = y'_i$ für $1 \leq i \leq k+1$ und damit $y(x) = y(x')$. Das impliziert jedoch, dass $x = x'$ sein müsste, da $x \mapsto y(x)$ eine injektive Abbildung ist. Da wir jedoch vorausgesetzt haben, dass $x \neq x'$ ist, haben wir einen Widerspruch.

Fall 2b: Es ist $|x| \neq |x'|$.

O.B.d.A. sei $|x'| > |x|$ und damit $l > k$. Wir gehen nun ähnlich wie in Fall 2a vor. Nehmen wir an, dass wir für k keine Kollisionen finden, so erreichen wir letztlich folgende Situation, mit

$$\begin{aligned} k(0^{m+1} \parallel y_1) &= g_1 \\ &= g'_{l-k+1} \\ &= k(g'_{l-k} \parallel 1 \parallel y'_{l-k+1}). \end{aligned}$$

Jedoch ist das $(m+1)$ -ste Bit von

$$0^{m+1} \parallel y_1$$

eine 0 und das erste Bit von

$$g'_{l-k} \parallel 1 \parallel y'_{l-k+1}$$

eine 1. Also finden wir eine Kollision für k . Dies ist jedoch ein Widerspruch zur Annahme.

Wir haben nun alle möglichen Fälle betrachtet und kommen zu dem Schluss, dass die gewünschte Eigenschaft - die Vererbung der Kollisionsresistenz von k auf die Hashfunktion h - bewiesen ist.

□

Für den Fall $t = 1$ benötigen wir im weiteren Vorgehen eine etwas andere Konstruktion zur Bestimmung einer kollisionsresistenten Hashfunktion h . Sei nun $|x| = n \geq m + 2$. Anders als im Algorithmus 6 wenden wir auf x eine andere Funktion f an. Diese arbeitet bitweise den String von x entlang mit $f(0) := 0$ und $f(1) := 01$.

Algorithm 7: MERKLEDAMGÅRD2(x)

external k

comment: $k : \{0, 1\}^{m+1} \rightarrow \{0, 1\}^m$

$n \leftarrow |x|$

$y \leftarrow 11 \parallel f(x_1) \parallel f(x_2) \parallel \dots \parallel f(x_n)$

denote $y = y_1 \parallel y_2 \parallel \dots \parallel y_k$, where $y_i \in \{0, 1\}, 1 \leq i \leq k$

$g_1 \leftarrow k(0^m \parallel y_1)$

for $i \leftarrow 1$ **to** $k - 1$

do $g_{i+1} \leftarrow k(g_i \parallel y_{i+1})$

return (g_k)

Satz 7

Sei $k : \{0, 1\}^{m+1} \rightarrow \{0, 1\}^m$ eine kollisionsresistente Kompressionsfunktion. Dann ist die Funktion

$$h : \bigcup_{i=m+2}^{\infty} \{0, 1\}^i \rightarrow \{0, 1\}^m,$$

so wie im Algorithmus 7 konstruiert, eine kollisionsresistente Hashfunktion.

Beweis: Nehmen wir an, wir finden $x \neq x'$ mit $h(x) = h(x')$. Wir schreiben

$$y(x) = y_1 y_2 \dots y_k$$

und

$$y(x') = y'_1 y'_2 \dots y'_l.$$

Wir betrachten dabei zwei Fälle.

Fall 1: Es ist $k = l$.

Analog zu Satz 6 finden wir entweder eine Kollision für k oder aber wir bekommen den Fall, dass $y = y'$ ist. Dies würde aber $x = x'$ implizieren. Also Widerspruch.

Fall 2: Es ist $k \neq l$.

O.B.d.A. sei $l > k$. Ähnlich wie in Satz 6 nehmen wir dann an, dass wir für k keine Kollisionen finden. Daraus folgen jedoch die Gleichheiten mit

$$\begin{aligned} y_k &= y'_l \\ y_{k-1} &= y'_{l-1} \\ &\vdots \\ y_1 &= y'_{l-k+1}. \end{aligned}$$

Dies widerspricht jedoch der oben geforderten Eigenschaft.
Also ist h kollisionsresistent.

□

Aus diesen beiden Sätzen lässt sich nun der folgende abschließende Satz herleiten.

Satz 8

Sei $k : \{0, 1\}^{m+t} \rightarrow \{0, 1\}^m$ eine kollisionsresistente Kompressionsfunktion mit $t \geq 1$.
Dann existiert eine kollisionsresistente Hashfunktion

$$h : \bigcup_{i=m+t+1}^{\infty} \{0, 1\}^i \rightarrow \{0, 1\}^m .$$

Die Anzahl, wie oft die Kompressionsfunktion k während der Berechnung von h angewendet wird, ist dabei

$$\begin{aligned} 1 + \left\lceil \frac{n}{t-1} \right\rceil & , \quad \text{falls } t \geq 2 \\ 2n + 2 & , \quad \text{falls } t = 1, \end{aligned}$$

wobei $|x| = n$.

4 Fazit

Ausgehend von der gewählten Kompressionsfunktion k erhalten wir also eine Hashfunktion h mit den gleichen Sicherheitseigenschaften von k . Es ist also ratsam, bei der Verwendung des Hashverfahrens mehr Zeit und Aufwand in das Finden der richtigen Kompressionsfunktion zu legen. Optional kann natürlich diese öffentliche Methode etwas abstrahiert werden, indem auf den resultierenden Hashwert nach der Merkle-Damgård-Konstruktion noch eine private Funktion g , wie in Kapitel 2.3 erwähnt, angewendet wird. Diese Funktion sollte dabei auch injektiv sein.

Die Paddingfunktion, so wie sie in dieser Arbeit vorgestellt wurde, erfordert, dass die Länge einer Nachricht bekannt ist. Es sollte zwar ein Input beliebiger Länge betrachtet werden, der jedoch für das Padding berechenbar sein müsste.

Des Weiteren gibt es auch Funktionen, die in jedem Iterationsschritt padden. Dies würde jedoch eine enorme Rechenzeit nach sich ziehen.

5 Literaturverzeichnis

1. Alaoui, Diplomarbeit: Konstruktion von Hashfunktionen, 2007
2. Köbler, Vorlesungsskript Kryptologie II, 2010
3. Stinson, Cryptography Theory and Practice, Chapman & Hall/CRC, 2002