

# Hashfunktionen in der Kryptologie

## über ihre Erzeugung und dadurch ermöglichte Attacken

René van Bevern und Robert Bredereck

Friedrich-Schiller-Universität Jena

Blockseminar über Kryptologie, 2008

# Gliederung

- 1 Einführung: Was sind Hashfunktionen?
- 2 Erzeugung von Hashfunktionen
- 3 Attacken

# Gliederung

- 1** Einführung: Was sind Hashfunktionen?
  - Definition und Anwendung
    - Urbildresistenz
    - Kollisionsresistenz
- 2** Erzeugung von Hashfunktionen
- 3** Attacken

# Definition und Anwendungsbeispiele

## Definition

Für ein  $n \in \mathbb{N}$  heißt eine berechenbare Funktion

$H : \{0, 1\}^* \rightarrow \{0, 1\}^n$  *Hashfunktion mit  $n$ -bit-Ausgabe*. Ihr Wert  $H(x)$  an der Stelle  $x \in \{0, 1\}^*$  heißt *Hashwert* von  $x$  (unter  $H$ ).

## Anwendung u.a. als Fingerabdruck größerer Datenmengen

- Signierung von Daten: Wurde der Inhalt manipuliert?
- Prüfsummen: Wurden Daten fehlerfrei übertragen?
- Identifikation größerer Datenmengen mit ihrem Hashwert:
  - in Datenbanken als Schlüssel für Hashtabellen
  - Identifikation von Inhalten in Peer-to-Peer-Tauschbörsen

# Anforderungen an Hashfunktionen im Allgemeinen

Reduktion: Hashwerte deutlich kürzer als ihre Urbilder

Chaos: kleine Änderungen am Urbild bewirken große am Bild

Surjektivität: jeder Hashwert möglich

Effizienz: Hashfunktion ohne großen Zeit- und  
Speicherverbrauch berechenbar

Klar: Hashfunktionen sind i.A. nicht injektiv! Obige Eigenschaften setzen wir voraus, wenn wir von Hashfunktionen sprechen.

# Anforderungen an Hashfunktionen in der Kryptologie

Manipulierte Texte sollen mit geringer Wahrscheinlichkeit zum Hashwert des Originaltextes führen:

- Urbilder schwer zu finden: sonst vorhandene Texte fälschbar
- Kollisionen schwer zu erzeugen: sonst sehr einfaches Spiel, wenn man beide Texte vorformulieren darf

Dies bedingt Gleichverteiltheit der Hashwerte – wird im Weiteren genauer betrachtet

# Gliederung

- 1** Einführung: Was sind Hashfunktionen?
  - Definition und Anwendung
  - **Urbildresistenz**
  - Kollisionsresistenz
- 2** Erzeugung von Hashfunktionen
- 3** Attacken

## Urbilder per Brute Force finden

Es sei  $H$  eine Hashfunktion mit  $n$ -bit-Ausgabe, wie findet man zu einem gegebenen  $y$  ein  $x$ , sodass  $H(x) = y$  gilt?

- Ist die Umkehrfunktion  $H^{-1}$  unbekannt, hilft Brute Force:
  - erzeuge ein  $x$  (z.B. Substitutionsattacke)
  - prüfe, ob  $H(x) = y$
- bei Gleichverteilung: Erfolg nur mit Wahrscheinlichkeit  $1/2^n$ 
  - also:  $2^n$  mal wiederholen: Erfolg mit hoher Wahrscheinlichkeit
- ohne Gleichverteilung,  $x$  wie oben beliebig:
  - existiert  $y$  sodass  $\Pr(H(x) = y) > 1/2^n$
  - für dieses  $y$  ist das Urbild einfacher zu erraten!

# Urbildresistenz

## Definitionen

Es sei  $H$  eine Hashfunktion,  $y$  und  $x'$  gegeben,  $H$  heißt...

...*urbildresistent*, wenn das Lösen der Gleichung  $H(x) = y$  praktisch unmöglich ist.

...*zweiturbildresistent*, wenn das Lösen der Gleichung  $H(x) = H(x')$  praktisch unmöglich ist.

## Bemerkungen

- Idealerweise würden beide Probleme erst nach ca.  $2^n$  Auswertungen der  $n$ -bit-Hashfunktion  $H$  gelöst.
- praktisch unmöglich: heutzutage  $\geq 2^{80}$  Operationen

## Urbildresistenz gibt es nicht geschenkt

Ein Beispiel: Oft stellen sich Schwächen in Hashfunktionen erst später heraus.

Es diene als Hashfunktion  $f(x) = x^2 - 1 \pmod p$ , wobei  $p$  prim sei. Die Funktion  $f$  erzeugt Ausgaben mit  $\log p$ -Länge. Gegeben  $y$ , löse  $f(x) = y$ :

$$\begin{aligned}y &\equiv x^2 - 1 \pmod p \\y + 1 &\equiv x^2 \pmod p \\ \sqrt{y + 1} &\equiv x \pmod p\end{aligned}$$

Aber wie zieht man Wurzeln in endlichen Körpern – sofern sie existieren? Es gibt  $p$  Elemente in  $\mathbb{F}_p$ . Alle zu testen bedürfte  $p = 2^{\log p}$  Auswertungen. Ist  $f$  also urbildresistent für große  $p$ ?

## Urbildresistenz gibt es nicht geschenkt

Ein Beispiel: Die Funktion  $f$  ist jedenfalls nicht (zweit)urbildresistent

Für Primzahlen  $p \equiv 3 \pmod{4}$  lässt sich  $f$  wie folgt umkehren:

$$\begin{aligned}f(x) &\equiv x^2 - 1 \pmod{p} \\f^{-1}(x) &\equiv (x + 1)^{\frac{p+1}{4}} \pmod{p}\end{aligned}$$

Beispiel für  $p = 7$  und  $x = 5$ . Beachte, dass  $2 \equiv -5 \pmod{7}$  sogar ein zweites Urbild ist:

$$\begin{aligned}f(5) &= 5^2 - 1 \pmod{7} = 3 \\f^{-1}(3) &= (3 + 1)^2 \pmod{7} = 2\end{aligned}$$

Die Größe von  $p$  ist dabei unerheblich.

## Zusammenhang zwischen Urbild- und Zweiturbildresistenz

Urbildresistenz impliziert keine Zweiturbildresistenz

Sei  $f : \mathbb{N} \rightarrow \{0, 1\}^n$  eine urbildresistente Hashfunktion. Die Funktion  $g$  auf  $\mathbb{Z}$  sei wie folgt definiert:

$$g(t) = f(|t|)$$

- Die Funktion  $g$  ist auch urbildresistent.
- Aber: für jedes Urbild  $t \neq 0$  ist  $-t$  ein zweites Urbild.

## Zusammenhang zwischen Urbild- und Zweiturbildresistenz

Zweiturbildresistenz impliziert keine Urbildresistenz

Sei  $f$  eine zweiturbildresistente  $n$ -bit-Hashfunktion und sei  $g$  wie folgt definiert:

$$g(x) = \begin{cases} 1 \parallel x, & \text{wenn } |x| = n \\ 0 \parallel f(x), & \text{sonst} \end{cases}$$

- Die Funktion  $g$  ist zweiturbildresistent.
- Aber die Urbilder aller  $y = 1y_1y_2 \dots y_n$  sind sofort bekannt.

# Gliederung

- 1** Einführung: Was sind Hashfunktionen?
  - Definition und Anwendung
  - Urbildresistenz
  - **Kollisionsresistenz**
- 2** Erzeugung von Hashfunktionen
- 3** Attacken

# Kollisionen

## Definition

Eine *Kollision* unter einer Hashfunktion  $H$  sind zwei Nachrichten  $\{x, x'\}$ , sodass  $H(x) = H(x')$  gilt.

## Bemerkung

Nützlich, wenn man verschiedene Texte vorformulieren möchte:  
Gruppen von harmlosen und bösen Texten kollidieren lassen, wie wir später sehen werden.

## Wahrscheinlichkeit von Kollisionen: theoretisch

Hashwerte seien gleichverteilt und  $\bar{p}(d, n)$  sei die Wahrscheinlichkeit, dass unter  $d$  (mit einer  $n$ -bit-Hashfunktion) gehashten Nachrichten *keine* Kollision auftritt:

$$\begin{aligned}\bar{p}(d, n) &= \prod_{i=0}^{d-1} \frac{2^n - i}{2^n} = \prod_{i=0}^{d-1} \left(1 - \frac{i}{2^n}\right) \\ &\leq \prod_{i=0}^{d-1} e^{-i/2^n} = \exp\left(\frac{d \cdot (1-d)}{2^{n+1}}\right)\end{aligned}$$

Diese Abschätzung ermöglicht ein paar weitere Betrachtungen...

## Wahrscheinlichkeit von Kollisionen: konkretisiert

Wie hoch ist die Wahrscheinlichkeit  $\bar{q}_c(n) := \bar{p}(c\sqrt{2^n}, n)$ , unter  $c\sqrt{2^n}$  (mit einer  $n$ -bit-Hashfunktion) gegashten Nachrichten keine Kollision zu finden?

$$\begin{aligned}\bar{q}_c(n) &\leq \exp\left(\frac{2^{n/2}c - 2^n c^2}{2^{n+1}}\right) \\ &= \exp\left(\frac{c}{2^{n/2+1}} - \frac{c^2}{2}\right)\end{aligned}$$

- Für festes, hinreichendes  $c$  ist  $\bar{q}_c(n)$  verschwindend gering.
- Bei zunehmender Bitlänge  $n$  wird  $\bar{q}_c(n)$  noch kleiner.

## Wahrscheinlichkeit von Kollisionen: beispielhaft

Schon für kleine  $c$  ist die Wahrscheinlichkeit, unter  $c\sqrt{2^n}$  ghashten Nachrichten keine Kollision zu finden, sehr gering:

$$\bar{q}_2(16) \leq 0.14$$

$$\bar{q}_4(16) \leq 3.4 \cdot 10^{-4}$$

$$\bar{q}_8(16) \leq 1.29 \cdot 10^{-14}$$

$$\bar{q}_8(1024) \leq 1.27 \cdot 10^{-14}$$

$$\text{6er im Lotto} \geq 7 \cdot 10^{-7}$$

- Hashlänge beeinflusst Ordnung der Wahrscheinlichkeit kaum
- erwarte Kollision nach ca.  $\sqrt{2^n}$  ghashten Nachrichten

# Kollisionsresistenz

## Definition

Eine Hashfunktion heißt *kollisionsresistent*, wenn das Finden einer Kollision nicht praktisch durchführbar ist.

## Bemerkung

Analog zu den anderen Resistenzen sollte das Finden einer Kollision im Optimalfall nicht effizienter als mit Brute Force funktionieren, also  $c\sqrt{2^n}$ ,  $c \geq 2$ , Auswertungen erwarten lassen.

# Kollisionsresistenz und Urbildresistenz

Kollisions-Resistenz impliziert nicht Urbildresistenz

Betrachte erneut die kollisionsresistente  $n$ -bit-Hashfunktion  $f$  und die Hashfunktion  $g$  wie folgt:

$$g(x) = \begin{cases} 1 \parallel x, & \text{wenn } |x| = n \\ 0 \parallel f(x), & \text{sonst} \end{cases}$$

- Die Funktion  $g$  ist kollisionsresistent.
- Aber die Urbilder aller  $y = 1y_1y_2 \dots y_n$  sind sofort bekannt.

## Iterierte Hashfunktionen

Idee: Gegeben sei eine Funktion, welche Eingaben fester Länge verarbeitet, erzeuge eine Hashfunktion, welche Eingaben beliebiger Länge verarbeiten kann.

- Eingabe entsprechend in Blöcke zerlegen
- Blöcke mit gegebener Funktion schrittweise verarbeiten

## Drei Phasen der Berechnung

Gegeben: *Kompressionsfunktion*  $z : \{0, 1\}^n \times \{0, 1\}^m \rightarrow \{0, 1\}^n$

- Die Größe des ersten Parameters entspricht der Ausgabegröße.

Dann läuft die Hashwert-Berechnung in drei Phasen ab:

- 1 Preprocessing
- 2 Processing
- 3 Optionale Output-Transformation

## Phase 1: Preprocessing

Gegeben: *Kompressionsfunktion*  $z : \{0, 1\}^n \times \{0, 1\}^m \rightarrow \{0, 1\}^n$

- Zunächst Eingabezeichenkette verlängern (*engl.* padding)
  - Ihre Länge ist nun ein Vielfaches von  $m$ .
- Zerlegung der aufgefüllten Eingabezeichenkette in Blöcke  $b_1, \dots, b_k$  gleicher, fester Größe  $m$ 
  - Größe  $m$  entspricht Größe des zweiten Parameters von  $z$
  - Somit: modifizierte Eingabe der Länge  $m \cdot k$

## Phase 2: Processing

Blöcke werden wie folgt durch Kompressionsfunktion verarbeitet:

$$h_1 \leftarrow z(IV, b_1)$$

$$h_2 \leftarrow z(h_1, b_2)$$

$\vdots$

$$h_k \leftarrow z(h_{k-1}, b_k)$$

Hierbei stellt  $IV$  den im Allgemeinen bekannten Startwert oder *Initialisierungsvektor* der Hashfunktion dar, der für vergleichbare Ergebnisse fest sein muss.

## Phase 3: optionale Output-Transformation

- eventuell weitere Operationen an  $h_k$  vornehmen
- Trivial: Identität, Konkatenationen, XOR-Verknüpfungen o.ä.
- durch zusätzliche Tricks die Hashfunktion sicherer machen

## Merkle-Damgård-Konstruktion

Entspricht oben beschriebem Verfahren mit Anforderungen an Kompressionsfunktion:

- Sowohl die Eingabe als auch die Ausgabe der Kompressionsfunktion haben eine feste Länge.
- Die Kompressionsfunktion ist kollisionsresistent.

MERKLE und DAMGÅRD [MD89]: Eine Hashfunktion die durch M.-D.-Konstruktion auf einer kollisionsresistenten Kompressionsfunktion entstanden ist, ist ebenfalls kollisionsresistent.

# Gliederung

- 1 Einführung: Was sind Hashfunktionen?
- 2 Erzeugung von Hashfunktionen
- 3 Attacken
  - Gefahr des Blackbox-Paradigmas
  - Schneller  $t$ -Kollisionen finden
  - Konkatenation mehrerer Hashwerte
  - Erzeugung sinnvoller Kollisionen

# Attacken

Warum man Hashfunktionen nicht als Blackboxen betrachten darf

- Die Arbeitsweise einer Blackbox ist unbekannt.
  - Nur vage Laufzeitangaben sind möglich.
  - Ein falscher Eindruck von Sicherheit kann entstehen.
- Kenntnis ihrer inneren Struktur ermöglicht Angriffe, die auf Blackboxes wirkungslos wären.
  - diese Kenntnis haben wir! (M.-D.-Konstruktion)
  - Bringt obere Schranken zum Einsturz
- Blackbox-Paradigma liefert gefährliche Trugschlüsse!

# Gliederung

- 1 Einführung: Was sind Hashfunktionen?
- 2 Erzeugung von Hashfunktionen
- 3 **Attacken**
  - Gefahr des Blackbox-Paradigmas
  - **Schneller  $t$ -Kollisionen finden**
  - Konkatenation mehrerer Hashwerte
  - Erzeugung sinnvoller Kollisionen

# Multikollisionen

## Definition

Sei  $H$  eine Hashfunktion. Eine  $t$ -Kollision ist eine Menge von Nachrichten  $\{x_1, x_2, \dots, x_t\}$ , sodass gilt:

$$H(x_1) = H(x_2) = \dots = H(x_t)$$

## Blackbox-Betrachtung

Nach [JOUX04] ist nach ungefähr  $2^{n \frac{t-1}{t}}$  ghashten Nachrichten eine  $t$ -Kollision erwarten.

## Kollisionen in der Kompressionsfunktion

Man betrachte im Folgenden die Funktionen:

- eine iterierte Hashfunktion  $H : \{0, 1\}^* \rightarrow \{0, 1\}^n$
- ihre Kompressionsfunktion  $z : \{0, 1\}^n \times \{0, 1\}^m \rightarrow \{0, 1\}^n$
- Es gelte  $m > n$ , damit Kollisionen existieren.
- $C : h \mapsto \{B, B'\}$ , derart dass  $z(h, B) = z(h, B')$  gilt

Die Funktion  $C$  findet, gegeben eine Verkettungsvariable  $h$ , zwei verschiedene Nachrichtenblöcke  $B$  und  $B'$ , sodass sie unter  $z$  kollidieren. Das ist nach ca.  $\sqrt{2^n}$  Auswertungen von  $z$  zu erwarten.

## Multikollisionen finden – am Beispiel

Man betrachte mit  $IV$  als Initialisierungsvektor:

- berechne  $C(IV) = \{B_1, B'_1\}$ , s.d.  $z(IV, B_1) = h = z(IV, B'_1)$
- berechne  $C(h) = \{B_2, B'_2\}$ , dann gilt  $z(h, B_2) = z(h, B'_2)$

Dann ist mit  $\{B_1 B_2, B_1 B'_2, B'_1 B_2, B'_1 B'_2\}$  eine 4-Kollision nach nur zwei Aufrufen von  $C$  gefunden, denn es gilt:

$$\begin{aligned} z(z(IV, B_1), B_2) &= z(z(IV, B'_1), B_2) \\ = z(z(IV, B_1), B'_2) &= z(z(IV, B'_1), B'_2) \end{aligned}$$

## Multikollisionen finden – allgemein

### Satz (ANTOINE JOUX)

*Eine  $2^t$ -Kollision für eine iterierte  $n$ -bit-Hashfunktion ist nach ca.  $t\sqrt{2^n}$  Auswertungen ihrer Kompressionsfunktion zu erwarten.*

### Beweis.

- Mit  $t$  Aufrufen von  $C$  werden  $2^t$ -Kollisionen gefunden:
  - gegeben eine  $2^{t-1}$ -Kollision  $k$  mit Wert  $h$
  - berechne  $C(h) = \{B, B'\}$ , dann gilt:  $z(h, B) = z(h, B')$
  - dann erhält man mit  $\{x \parallel B, x \parallel B' : x \in k\}$  eine  $2 \cdot 2^{t-1}$ -Kollision nach nur  $t$  Schritten.
- Von  $C$  werden ca.  $\sqrt{2^n}$  Auswertungen erwartet.



## Eigenschaften der erzeugten Nachrichten

- $2^t$ -Kollision nach  $t$  Schritten im Endergebnis, aber auch:
  - $2^i$ -Kollision in *jedem* Schritt  $i$  der Hashwert-Berechnung
- Die erzeugte Nachrichten sind baumartig verwandt:
  - Der Block  $i$  einer Nachricht ist entweder  $B_i$  oder  $B'_i$ .
  - $2^t$  Nachrichten bilden einen Binärbaum mit  $2^{t+1} - 1$  Knoten.
- Gegeben eine andere iterierte Hashfunktion mit Blockgröße ebenfalls  $m$  und  $O(1)$ -Zeit-Kompressionsfunktion:
  - Die  $2^t$  Hashwerte der  $2^t$  Nachrichten unserer Kollision sind in *Zeit*  $O(2^t)$  berechnbar statt  $O(t2^t)$ .

# Gliederung

- 1 Einführung: Was sind Hashfunktionen?
- 2 Erzeugung von Hashfunktionen
- 3 **Attacken**
  - Gefahr des Blackbox-Paradigmas
  - Schneller  $t$ -Kollisionen finden
  - **Konkatenation mehrerer Hashwerte**
  - Erzeugung sinnvoller Kollisionen

## Stärkere Hashfunktionen aus mehreren schwächeren?

Betrachte folgende iterierte Hashfunktionen:

- iterierte, koll.-resistente Hashfunktion  $f : \{0, 1\}^* \rightarrow \{0, 1\}^{n_f}$
- iterierte, koll.-resistente Hashfunktion  $g : \{0, 1\}^* \rightarrow \{0, 1\}^{n_g}$
- Konkatenation ihrer Hashwerte:  $h = f \parallel g$

Ist  $h$  kollisionsrestistent? Ist für das Finden von Kollisionen in  $h$  ein Aufwand von ca.  $\sqrt{2^{n_f+n_g}}$  zu erwarten?

## Blackbox-Betrachtung täuscht Kollisionsresistenz vor

### Blackbox-Betrachtung

Sind die Ereignisse  $f(x) = y$  und  $g(x') = y$  stochastisch unabhängig, sind tatsächlich ungefähr  $\sqrt{2^{n_f}} \cdot \sqrt{2^{n_g}}$  Auswertungen von  $h$  zu erwarten.

### Satz (ANTOINE JOUX)

*Gilt  $n_f \leq n_g$ , ist eine Kollision in  $h$  schon zu erwarten nach Hashfunktionen-Auswertungen von ungefähr:*

$$n_g \sqrt{2^{n_f}} + \sqrt{2^{n_g}}$$

## Kollisionen für $h$ finden

### Satz (ANTOINE JOUX)

*Gilt  $n_f \leq n_g$ , ist eine Kollision nach ca.  $n_g \sqrt{2^{n_f}} + \sqrt{2^{n_g}}$  Auswertungen zu erwarten.*

### Beweis.

- Erzeuge  $\sqrt{2^{n_g}}$ -Kollision  $k$  für  $f$  mit ca.  $n_g \sqrt{2^{n_f}}$  Aufwand.
- Wende  $g$  auf jede Zeichenkette aus  $k$  an:  $\sqrt{2^{n_g}}$  Auswertungen.
- Dabei ist eine Kollision von  $h$  zu erwarten.



## Bemerkungen zu diesem Ergebnis

- Gilt auch, wenn  $g$  zufällig und damit unabhängig von  $f$  ist:
  - dann natürlich nicht mehr mit baumartiger Berechnung der Werte, sondern in Laufzeit:

$$O\left(n_g \sqrt{2^{n_f}} + n_f \sqrt{2^{n_g}}\right)$$

- Der Beweis funktioniert für  $n_g \leq n_f$  analog, dann in ca.  $\sqrt{2^{n_f}} + n_f \sqrt{2^{n_g}}$  Auswertungen.

# Gliederung

- 1 Einführung: Was sind Hashfunktionen?
- 2 Erzeugung von Hashfunktionen
- 3 **Attacken**
  - Gefahr des Blackbox-Paradigmas
  - Schneller  $t$ -Kollisionen finden
  - Konkatenation mehrerer Hashwerte
  - **Erzeugung sinnvoller Kollisionen**

## Wie sinnvoll sind bisherige Attacken?

Für viele prominente Hashfunktionen können Kollisionen in akzeptabler Zeit gefunden werden. Für iterierte Hashfunktionen sind Multikollisionen sogar schneller als erwartet zu erzeugen. Aber:

- Urbilder sind immer noch schwer zu finden.
- Noch nicht betrachtet: Kollisionen erzeugen, die einem bestimmten Protokoll oder Format genügen.

Sind die entsprechenden Hashfunktionen daher trotzdem noch problemlos einsetzbar?

# Fühle dich niemals zu sicher!

Der Teufel steckt oft im Detail...

DAUM und LUCKS hatten eine interessante Idee:

- Angriff auf weit verbreitetes Dokumentenformat PostScript
- Postscript: sehr mächtige und turingvollständige Sprache
  - Komplexität zum Verstecken von Nachrichten nutzen

Szenario: Der Kassenswart legt einem Vereinspräsidenten ein Dokument zur digitalen Unterschrift vor:

## Fühle dich niemals zu sicher!

Was dir vorgelegt wird...(beitraege.ps)

Liebes Vereinsmitglied,

wir als Vorstand sind leider dieses Jahr durch erhöhte Betriebskosten (rapider Anstieg an Energiekosten und stark erhöhte Käsepreise) dazu gezwungen, den Mitgliedsbeitrag für das kommende Jahr von den bisherigen 40 Euro auf 45 Euro anzuheben.

Mit Freundlichen Grüßen  
Der Vorstand.

# Fühle dich niemals zu sicher!

Was dann passiert...

Signierung des Dokuments durch den Vereinspräsidenten:

- Zur Vereinfachung wird MD5 als Hashfunktion angenommen.
- Die Vereinsmitglieder vertrauen dem Dokument mit der Signatur ihres Chefs.

Aber es wird ein anderes Dokument mit gleichem Hashwert versendet.

## Fühle dich niemals zu sicher!

Was die Mitglieder bekommen...(bank.ps)

Sehr geehrtes Vereinsmitglied,

erfreut können wir bekannt geben, dass wir dieses Jahr doch noch erfolgreich ein Konto bei einer Bank mit wesentlich besseren Konditionen eröffnen können. Durch die niedrigeren Kontoführungsgebühren und höheren Zinsen werden wir in der Lage sein, den Beitrag nur um 5 statt 10 Euro auf 45 Euro anzuheben.

Neue Bankverbindung: Liechtensteiner Staatsbank, BLZ:  
li-5383925, Kto: 4719849823, Kontoinhaber: Kassenwart Alois  
Schummel

Mit Freundlichen Grüßen  
Der Vorstand.

# Fühle dich niemals zu sicher!

## Beobachtungen:

- Die Texte unterscheiden sich erheblich.
- Diese Kollision ist kaum durch Zufall entstanden.
- Erneut: Hashfunktionen nicht als Blackbox betrachten!
- Wie führen PostScript und die Merkle-Damgård-Konstruktion zusammen zu diesem Ergebnis?



# Auflösung der Blackbox

Zur Erinnerung: MD5 basiert wie viele andere Hashfunktionen auf M.-D.-Konstruktion.

- Geschicktes Ausnutzen der Blöcke:
  - erster Block reicht bis » (« vor dem  $X$
  - $X$  ist der zweite Block.
- Finde eine Kollision  $\{B, B'\}$  für den zweiten Block mit dem Hashwert des ersten Blocks als  $IV$ .
- In `beitraege.ps`, setze  $X = Y = B$  auf den gleichen Wert, in `bank.ps`:  $X = B', Y = B$  auf unterschiedliche Werte.
- Kollision in einem Block bei Gleichheit aller anderen Blöcke führt zum gleichen Hashwert.

# Zusammenfassung

- Hashfunktionen oft durch wiederholtes Anwenden von Kompressionsfunktionen erzeugt
- viele Hashfunktionen haben bekannte Struktur  $\neq$  Blackboxes
- Nutzung der Struktur macht sie nichttrivial angreifbar

## Weiterführende Literatur I



Ivan Damgård

A Design Principle for Hash Functions.

Lecture Notes in Computer Science Vol. 435, S. 416-427

Springer, 1989



Antoine Joux

Multicollisions in Iterated Hash Functions

Lecture Notes in Computer Science Vol. 3152, S. 306–316

Springer, 2004



A. Menezes, P. van Oorschot und S. Vanstone

Handbook of Applied Cryptography, Kapitel 9

CRC Press, 1996

<http://www.cacr.math.uwaterloo.ca/hac>

## Weiterführende Literatur II



Ilja Mironov

Hash functions: Theory, attacks, and applications

Technical Report TR-2005-187, Microsoft Research, 2005

<http://research.microsoft.com/users/mironov/>



Mark Stamp, Richard M. Low

Applied Cryptanalysis, S. 253-256

John Wiley & Sons, Inc., 2007