Bewertung von Hash-Funktionen

Achim Jung
Fachbereich Mathematik
Technische Hochschule Darmstadt
Schloßgartenstraße 7
6100 Darmstadt

February 3, 1996

1 Einführung

Eine *Hash-Funktion* ist eine berechenbare Funktion, die einem beliebigen, insbesondere beliebig langen Text (über einem fest gewählten Alphabet Σ) als Wert eine Zeichenkette fester Länge zuordnet. Als Alphabet wählt man üblicherweise die Menge $\{0, 1\}$, beziehungsweise davon abgeleitete Zeichensätze wie etwa ASCII. Man kann dann Texte aus Σ^* auch als im Binärsystem dargestellte Zahlen auffassen. Die Definition einer Hash-Funktion liest sich dann so:

- Eine *Hash-Funktion* h ist eine Abbildung von $\mathbb{N}_0$ in $[0, s] \subset \mathbb{N}_0$.

Im folgenden wollen wir diese beiden Sichtweisen von Texten und Hash-Funktionen gleichzeitig verwenden, um sowohl die Operationen auf Σ^* (wie Konkatenation) als auch die arithmetischen Operationen zur Verfügung zu haben.

Das Ergebnis $h(M)$ der Anwendung der Hash-Funktion h auf den Text M bezeichnen wir als *Komprimat*. In der mir vorliegenden, durchweg Englischsprachigen, Literatur finden sich hierfür die Ausdrücke *message digest, fingerprint, hash-code, residue, check-sum, imprint.* Hash-Funktionen hingegen werden fast durchgängig als *hash-functions* bezeichnet.

Da eine Hash-Funktion beliebig lange Texte akzeptiert, die Ausgabe jedoch von fester Länge ist, muß sie verschiedenen Texten das gleiche Komprimat zuweisen. Ein Paar (M, M') mit $h(M) = h(M')$ nennt man eine *Kollision*. Wie gesagt, die Existenz von Kollisionen liegt in der Natur der Sache, wir wollen jedoch Hash-Funktionen konstruieren, deren Kollisionen nur mit großem Rechenaufwand zu finden sind. Eine solche Funktion wollen wir (dem üblichen Gebrauch entsprechend) *kollisionsfrei* nennen. Diesen Begriff genauer zu durchleuchten ist das Hauptanliegen dieser Arbeit.

Hash-Funktionen sind zwar essentieller Bestandteil von Kryptosystemen, die Literatur zum Thema ist gleichwohl spärlich. Die 'Großen Namen' der Kryptographie fehlen ganz. Ich vermute, der Grund hierfür ist, daß man anfänglich die Konstruktion kollisionsfreier Hash-Funktionen als trivial ansah und daß theoretische und praktische Fortschritte auf diesem Gebiet wenig spektakulär sind.

2 Hash-Funktionen als Baustein in der Kryptographie

Wozu werden Hash-Funktionen verwendet? Ich möchte hier auf drei typische Anwendungen kurz eingehen.

Zum ersten kann man Hash-Funktionen zur Berechnung einer Manipulationssicherung verwenden. Hierzu verlängert man eine Nachricht M um das Komprimat $h(M)$. Ist bei der Übertragung die Nachricht zufällig oder böswillig geändert worden, so kann dies vom Empfänger überprüft werden, indem er das Komprimat der empfangenen Nachricht mit dem mitgesandten $h(M)$ vergleicht. Übertragungsfehler können allerdings nur erkannt, nicht jedoch korrigiert werden, hierfür ist die Theorie fehlerkorrigierender Kodes zuständig. Vor böswilligen Manipulationen andererseits ist man nur geschützt, wenn die Hash-Funktion Geheimnis von Sender und Empfänger bleibt. In diesem Fall hat man bereits eine einfache Form einer elektronischen Unterschrift (Die aber Dritten gegenüber nicht glaubhaft gemacht werden kann.). Im Gegensatz hierzu wollen wir uns hier mit öffentlich zugänglichen, gar normierten Hash-Funktionen befassen.

Solche öffentlichen Hash-Funktionen finden Verwendung in der Generierung elektronischer Unterschriften. Man unterschreibt in diesem Fall nicht die Nachricht selbst, sondern nur das Komprimat. Dadurch ergibt sich in der Regel ein erheblicher Zeitgewinn und das ist es was Hash-Funktionen für den Chipkarteneinsatz unentbehrlich macht. Eine Sicherheitseinbuße möchte man sich damit jedoch nicht einhandeln, deshalb muß nicht nur das Kryptosystem sondern auch die Hash-Funktion sicher gegen Manipulationsversuche sein. Was in Bezug auf Hash-Funktionen 'Sicherheit' heißt soll weiter unten untersucht werden.

In der Tat kann sich sogar ein Sicherheitsgewinn ergeben. So hat zum Beispiel das RSA-Verfahren die Eigenschaft Produkte von Zahlen auf die Produkte der individuellen Unterschriften abzubilden. Dies kann auf verschiedene Weisen in der Kryptanalyse verwendet werden. Hat eine Hash-Funktion diese Homomorphie-Eigenschaft nicht, so verstärkt ihre Anwendung das RSA-System. Kollisionsfreiheit der Hash-Funktion impliziert weiterhin eine wünschenswerte Ein-Weg-Eigenschaft des Signaturverfahrens. Hierzu ein konkretes Beispiel: Die Schlüssel individueller Benutzer könnte man so zertifizieren, daß man ihnen die Unterschrift eines 'vertrauenswürdigen Zentrums' beifügt. Betrügerische Dritte könnten fiktive aber zertifizierte Benutzer erfinden, indem sie eine beliebige Zahl als Unterschrift des Zentrums einsetzen und sich die hierzu gehörigen Benutzerschlüssel durch (ja von jedem mögliche) Authentifikation dieser Unterschrift verschaffen. Durch Verwendung einer Hash-Funktion wird dies verhindert: Der Angreifer kann nur das Komprimat der Schlüssel berechnen, woraus er die Schlüssel selbst nur mit unverhältnismäßig hohem Aufwand rekonstruieren kann.

Diesen drei praktischen Anwendungen sei als Kuriosum ein theoretisches Ergebnis angefügt: Es wurde gezeigt (Rompel 1990), daß man jedes Signaturverfahren zur Konstruktion einer kollisionsfreien Hash-Funktion verwenden kann und daß man umgekehrt aus einer kollisionsfreien Hash-Funktion vollwertige elektronische Unterschriften ableiten kann. Die erste Implikation liegt nahe, denn ein Verfahren, das Text und Schlüssel in Unterschrift transformiert, kann man ja unmittelbar als Komprimierung verstehen. Um die kryptographische Sicherheit zu beweisen, muß man jedoch für beide Richtungen komplizierte Transformationen verwenden, was eine praktische Nutzung verhindert.

3 Zwei typische Hash-Funktionen und zwei typische krytanalytische Angriffe

Hash-Funktionen kann man sich fast immer als nach dem folgenden Schema aufgebaut denken: Gegeben ist eine *Komprimierungsfunktion* f, die Texte fester Länge l in Texte kürzerer Länge m transformiert. Texte beliebiger Länge werden dann in Blöcke der Länge $l - m$ zerlegt und das Komprimat durch fortgesetzte Anwendung von f berechnet. Bildlich dargestellt ergibt sich folgende Situation:

Zwei schlechte aber instruktive Hash-Funktionen erhält man auf folgende Weise:
s-square-mod-N: Sei N ein große zusammengesetzte Zahl der Länge n. Die Komprimierungsfunktion f reduziert von $2n$ auf n durch folgende Operation: Linke Hälfte und rechte Hälfte werden per 'Exklusiv-oder' aufaddiert und das Resultat modulo N quadriert.
s-DES: Die Komprimierungsfunktion f reduziert von $64 + 56$ Bit auf 64 Bit indem das DES-Verschlüsselungsverfahren auf die ersten 64 Bit mit dem Rest

als Schlüssel angewandt wird.

Ein Angriff auf *s-square-mod-N* ist auf folgende Weise möglich: Ist eine Nachricht M mit Komprimat $h(M)$ gegeben, so kann man für eine beliebige weitere Nachricht M' durch Anhängen eines einzigen Blockes B von Länge n erreichen, daß $h(M) = h(M' \parallel B)$ gilt, die Komprimate also übereinstimmen. Für den anzuhängenden Block B gibt es hierbei nur eine Möglichkeit, denn es muß gelten: $h(M') \;\; \mathsf{xor} \;\; B = h_{k-1}(M)$, also $B = h_{k-1}(M) \;\; \mathsf{xor} \;\; h(M')$. Dabei bezeichne $h_{k-1}(M)$ das Zwischenergebnis vor der letzten Quadratur in der Berechnung des Komprimates von M. Man erkennt an dieser Darstellung, daß B sich in eindeutiger Weise aus zwei Komprimaten errechnet, mit hoher Wahrscheinlichkeit ist B daher eine 'unsinnige' Textfolge.

Der zweite Angriff, den wir hier betrachten wollen, ist etwas komplizierter. Er bezieht sich auf *s-DES*. Gegeben sei wieder eine Nachricht M und es wird eine weitere Nachricht M' mit gleichem Komprimat konstruiert. Hierzu stellt man sich eine Vielzahl, etwa eine Milliarde, verschiedene und böswillig sinnvolle linke bzw. rechte Nachrichtenhälften zusammen und berechnet für linke Hälften jeweils das Komprimat und für rechte Hälften das 'inverse' Komprimat. Letzteres ergibt sich, wenn man in der Definition von *s-DES* die Verschlüsselungsfunktion durch die zugehörige Entschlüsselungsfunktion ersetzt. Dabei nimmt man als Startwert $h(M)$. Die Wahrscheinlichkeitstheorie lehrt uns, daß es mit hoher Wahrscheinlichkeit zu einer Koinzidenz kommt: Ein Komprimat ist gleich einem inversen Komprimat. Die Verkettung der beiden zugehörigen Nachrichtenhälften hat dann das gleiche Komprimat wie M.

Wir halten fest, daß die Schwäche von *s-DES* eine Folge der Kürze der Nachrichtenblöcke ist.

4 Die Sicherheit von Hash-Funktionen

Wir wollen hier die Sicherheit von Hash-Funktionen nach drei verschiedenen Aspekten untersuchen: Ergebnis der Kryptanalyse, Art der Kryptanalyse, und Möglichkeiten des Kryptanalytikers.

4.1 Ergebnis der Kryptanalyse

Hier unterscheiden wir zwei Fälle.

- A1: *Durch Angriff auf die Hash-Funktion können nur Unterschriften zu solchen Nachrichten synthetisiert werden, die mit hoher Wahrscheinlichkeit einen sinnlosen Teil enthalten.*

- A2: *Durch Angriff auf die Hash-Funktion kommt man nur mit unverhältnismäßig hohem Rechenaufwand zu neuen Unterschriften.*

Die erste schwächere Alternative haben wir bei *s-square-mod-N* kennegelernt. Die neuen Nachrichten M' enthalten alle mindestens einen Block, dessen Wert von Zwischenergebnissen in der Berechnung der Komprimate abhängt. Es sei hier einmal versucht abzuschätzen, wie groß die Wahrscheinlichkeit ist, daß ein zufällig gewählter Block sinnvoll ist. Als 'sinnvoll' wollen wir ihn hier bereits dann ansehen, wenn er aus alphanumerischen Zeichen besteht. Diese machen etwa $1/4$ aller ASCII-Zeichen (von Byte-Größe) aus. Realistische Blocklängen für *s-square-mod-N* beginnen bei 512 Bit gleich 64-Byte. Es ergibt sich eine Wahrscheinlichkeit von etwa 10^{-38}. Die richtige Definition von 'sinnvollem Block' hängt natürlich von der Anwendung ab, aber man kann sagen, daß der beschriebene Angriff keine Gefahr birgt, falls nur alphanumerische Zeichen unterschrieben werden sollen. Werden zusätzlich fest vorgegebene Formate, wie etwa im Datenverkehr zwischen Bankinstituten, verwendet, so ist nach diesen Überlegungen *s-square-mod-N* schon als ausreichend sicher anzusehen. Ist andererseits die Anwendungssituation für die Hash-Funktion nicht fest vorgegeben, so darf man sich mit Sicherheitslevel A1 nicht zufrieden geben.

s-DES verletzt bereits die minimale Anforderung von A1. Dies nicht aus prinzipiellen Schwächen, sondern lediglich, weil DES mit einer Datenlänge von nur 64 Bit arbeitet.

In den nun folgenden Betrachtungen wollen wir die Eigenschaft A1 stets als erfüllt ansehen.

4.2 Art der Kryptanalyse

Wir unterscheiden folgende Angriffe auf Hash-Funktionen.

- B1 *Zu gegebener Nachricht M werden weitere Nachrichten M' mit gleichem Komprimat gesucht.*

- B2 *Es werden Nachrichtenpaare M, M' gesucht, die gleiches Komprimat haben.*

- B3 *Es werden Nachrichtenmengen* $M_1, M_2, \ldots, M_n$ *gesucht, derart, daß die Unterschrift zu* M_n *sich (im Umweg über die Hash-Funktion) aus den Unterschriften zu* $M_1, M_2, \ldots, M_{n-1}$ *synthetisieren läßt.*

Es fällt auf, daß der Unterschied zwischen dem zweiten und dem dritten Angriff gering ist, der zwischen den beiden ersten jedoch erheblich. In der Tat, der erste Angriff ist wohl der naheliegendste: Einem Angreifer sind unterschriebene Texte in die Hände gefallen und er versucht die zugehörigen elektronischen Unterschriften für ein ihm günstiges M' wiederzuverwenden. Ein Angriff der zweiten, bzw. dritten Art ist erst sinnvoll, falls der Angreifer die Möglichkeit hat, die unterschreibende Instanz dazu zu bewegen, für ihn die Nachrichten M, bzw. M_1 bis M_{n-1} zu signieren. Andererseits kann man sich auf den Standpunkt stellen (und insbesondere der Anwender wird dies tun), daß es unnütz ist, zwischen den Angriffen B2 und B3 zu unterscheiden. Es ist aber tatsächlich so, daß der überwältigende Teil der Literatur sich mit dem zweiten Angriff beschäftigt (eine Hash-Funktion, die ihm widersteht heißt eben gerade *kollisionsfrei* bzw. *kollisionsresistent*), während Angriffe der dritten Art ignoriert werden. Diese Problematik ist durch die Analyse der CCITT-Normfunktion X.509 (die auf *s-square-mod-N* aufbaut) schärfstens deutlich geworden: Ohne je eine Kollision zu finden, gelingt es Don Coppersmith, aus alten Unterschriften unter von ihm geeignet gewählte Dokumente, Unterschriften unter neue Texte selbst zu erzeugen.

Daß Angriffe nach B2 bzw. B3 überhaupt eine Gefahr darstellen können, mag bezweifelt werden, denn wenn jemand von ihm gewählte Texte signieren lassen kann, warum dann nicht schon jeden beliebigen? und wenn andererseits die unterschreibende Instanz die Texte auf ihre Semantik hin überprüft, bevor sie signiert, wieso fällt ihr dann nicht auf, daß die Texte einen unsinnigen weil zufälligen Block enthalten? Ich möchte daher kurz eine Situation beschreiben, in der ein solcher Angriff zur Gefahr werden würde. Nehmen wir an, ein Softwarehaus vermarktet 'virensichere' Softwareprodukte. Die Sicherung soll dabei auf folgende Weise geschehen: Vor Ausführung eines solchermaßen gesicherten Programmes wird die gleichfalls mitgelieferte elektronische Unterschrift in einem eigenen Chip überprüft. Nur falls diese Überprüfung erfolgreich ist, wird das Programm zur Ausführung freigegeben. Ein betrügerischer Angestellter des Softwarehauses könnte nun so vorgehen. Er schreibt zwei Programme, das eine korrekt, das andere mit krimineller Nebenfunktion. Das korrekte Programm legt er, natürlich im Quelltext, zur Prüfung vor und es wird auch eine elektronische Unterschrift erhalten.

Allerdings muß die Unterschrift zum übersetzten Programm und nicht zum Quelltext passen. Dem Maschinenkode jedoch hat der Programmierer einen Block angehängt, der das Programm nicht stört, ihm hingegen ermöglicht, auch sein kriminelles Programm so zu verlängern, daß ihm die gleiche Unterschrift zukommt. Der Unterschied zwischen den beiden Programmen ist nun vom Sicherheitschip im Kundenrechner nicht mehr feststellbar.

4.3 Möglichkeiten des Kryptanalytikers

Hier kann man, in aufsteigender Folge, drei Personenkreise unterscheiden: C1: *Außenstehende*, C2: *Insider* und C3: *Eigentümer*.

Ein Außenstehender hat nur die Möglichkeit mit ihm in die Hände gefallenem Material zu arbeiten, er kann die unterschreibende Instanz nicht dazu bewegen, ihm genehme Texte zu signieren. Angriffe der zweiten und der dritten Art bleiben ihm daher verschlossen. Ein Insider hat das Vertrauen der unterschreibenden Instanz, allerdings wird auch ihm nicht jeder Wunsch erfüllt. Der Eigentümer schließlich hat die Möglichkeit mit Hash-Funktion und Signierverfahren nach Gutdünken zu verfahren.

Warum nun sollte ein Eigentümer seine eigenen Unterschriften fälschen? Nun, er könnte etwa stets nur Unterschriften zu solchen Texten leisten, zu denen er sich einen weiteren Alternativtext mit gleicher Unterschrift zurechtgelegt hat. In einem späteren Konflikt könnte er dann behaupten, die Unterschrift unter den Originaltext sei gefälscht, in Wirklichkeit habe er den Alternativtext unterschrieben und der Kontrahent habe seine Unterschrift gefälscht. Ein gewichtigerer Grund, die Hash-Funktion auch vor Eigentümern zu sichern, ist, daß, falls die geheime Information des Benutzers auf irgend eine Weise öffentlich wird, nicht befürchtet werden muß, daß neue Dokument mit scheinbar gültiger Unterschrift auftauchen. Etwas verkürzt könnte man sagen: Die Hash-Funktion gehört zum *öffentlichen* Teil des Signaturverfahrens, jeder muß sie benutzen können, warum sollte es also überhaupt irgend jemanden geben, der sie vorteilhafter für sich verwenden kann als andere?

Wir haben nun in drei Dimensionen jeweils zwei, bzw. drei Sicherheitsstufen unterschieden. Zusammengenommen ergeben sich damit $2 \cdot 3 \cdot 3 = 18$ verschiedene Sicherheitsklassen. Diese verwirrend große Zahl reduziert sich glücklicherweise in der Praxis doch beträchtlich, denn gewisse Kombinationen sind entweder sinnlos oder uninteressant. So hat der Schreiber dieser

Zeilen in einem demnächst erscheinenden GMD-Bericht aus diesen 18 Klassen die fünf wichtigsten herausgearbeitet. Mit der hier entwickelten Terminologie sind dies die Klassen (A1,B1,C1), (A2,B1,C1), (A2,B2,C2), (A2,B3,C2) und (A2,B3,C3).

5 Mathematische Beweise zur Sicherheit von Hash-Funktionen

Kryptographische Sicherheit präzise zu formalisieren ist keine leichte Aufgabe, andererseits aber unverzichtbare Grundlage eines jeden Beweises. Als Methoden stehen zur Verfügung:

Informationtheorie: Ein System ist *unbeschränkt sicher*, wenn man zeigen kann, daß ein Angreifer nie genügend Information sammeln kann, um eine korrekte Kryptanalyse durchführen zu können. Diese Sicherheit gibt es nur in Ausnahmefällen. Für Hash-Funktionen und elektronische Unterschriften steht sie nicht zur Verfügung.

Komplexitätstheorie: Es wird stets der (erhoffte!) Unterschied zwischen P und NP ausgenutzt, da die Überprüfung einer Signatur bzw. das Berechnen eines Komprimats auf einfache und schnelle Weise möglich sein soll. Aber auch hier lauern Fallen. In der Komplexitätstheorie betrachtet man überwiegend das Extremverhalten von Algorithmen ('worst-case-analysis'). Dieses aber kann zeitaufwendig sein auch wenn die große Mehrzahl der Probleme sich auf einfache Weise lösen läßt. Klassisches Beispiel ist das Laufzeitverhalten des Simplexalgorithmus in der Linearen Programmierung: Es hat viele Jahre gedauert, bis man durch Angabe von absurd unrealistischen Beispielen hat zeigen können, daß der Simplexalgorithmus exponentiell viel Zeit verbrauchen kann. In der Praxis hatte man durchweg hervorragendes Laufzeitverhalten festgestellt.

Ein weiteres Problem ist, daß die Komplexitätstheorie stets Aussagen über eine unendliche Problemklasse macht, der Kryptanalytiker hingegen nur ein einziges konkretes Kryptosystem angreifen will. Auf eine griffige Formel gebracht: Faktorisieren ist im Allgemeinen schwierig, trotzdem kann ich die Zahl 15 zerlegen. Deshalb macht man in Bezug auf Hash-Funktionen nicht eine Aussage über eine einzelne Funktion sondern stets über parametrisierte Funktionsfamilien. Und deshalb ist es auch müßig zu hoffen, daß es eine einzelne beweisbar sichere Hash-Funktion je geben könnte.

Kryptanalyse: Dies ist die älteste Methode. Verfahren, die lange Zeit intensiver Kryptanalyse standgehalten haben, gewinnen des Vertrauen der Anwender.

Der Stand der Forschung stellt sich mir gegenwärtig so dar: Im allgemeinen ist die Eigenschaft A1 ('Mit hoher Wahrscheinlichkeit unbrauchbare Kryptanalyseresultate') unmittelbar an der Definition der Hash-Funktion ablesbar, so daß es wenig interessant erscheint hierüber Sätze zu beweisen. Im Zweifelsfall helfen statistische Tests, die man an der Komprimierungsfunktion f durchführt.

Aus der zweiten Skala wird heute meist das Problem der Kollisionen, B2, (seltener B1) angegangen, dies in Verknüpfung mit A2 und C2. Es gibt allerdings kein einziges vollständiges Resultat zu berichten. Die Hash-Funktionen, die beweisbar die Sicherheitsstufe (A2,B2,C2) erreichen, sind indiskutabel langsam. Ein Lichtblick ist das kürzlich bewiesene 'Damgårdsche Prinzip': Eine Hash-Funktion ist schon dann kollisionsfrei, wenn die zugehörige Komprimierungsfunktion kollisionsfrei ist.

Stufe B3 zu beweisen, erscheint unmöglich. Es läßt sich wohl kaum eine Aussage über 'alle möglichen Berechnungsmethoden' beweisen. Es ergibt sich damit die paradoxe Situation, daß für den Anwender eine Verletzung von B3 genauso schwer wiegt, wie ein Verstoß gegen B2, der Mathematiker jedoch nur für B2 eine umfassende Garantie je wird geben können.

6 Starke Hash-Funktionen

Es soll hier nur erwähnt werden, daß sowohl *s-square-mod-N* wie auch *s-DES* in vielversprechender Weise weiterentwickelt wurden. Aus Platzgründen will ich jedoch auf eine detaillierte Beschreibung verzichten. Für *square-mod-N* verweise ich auf

```
Achim Jung: The Strength of the CCITT/ISO Hash Function
            GMD Bericht (im Druck)
```

Eine DES-basierte Hash-Funktion findet man in

```
W.L.Price: Hash Functions - a Tutorial and Status Report
           NPL Report DITC 151/89
```

Des weiteren beschäftigen sich derzeit viele Kryptographen mit einem Verfahren von Ronald Rivest, über das sich jedoch noch keine Aussagen machen lassen.

7 Zusammenfassung

- Hash-Funktionen lassen sich unter drei Gesichtspunkten bewerten:

  Ergebnis der Analyse, 2 Stufen A1, A2,

  Art der Analyse, 3 Stufen B1, B2, B3,

  Möglichkeiten des Kryptanalytikers, 3 Stufen C1, C2, C3.

- Die Sicherheitsstufe A1 ist im allgemeinen gewährleistet.

- Höhere Sicherheitstufen werden beweisbar nur dann, wenn statt einer einzelnen Hash-Funktion eine parametrisierte Familie von Hash-Funktionen betrachtet wird. Verfahren, die nicht parametrisierbar sind, sind somit prinzipiell benachteiligt.

- Für die kombinierte Sicherheitsstufe (A2,B2,C3) kann man in absehbarer Zeit mit mathematischen Beweisen rechnen, für (A2,B3,C3) jedoch nicht.

- Da aus Sicht des Benutzers die Stufen B2 und B3 äquivalent sind, muß stets geprüft werden ob in der konkreten Anwendung Insiderattacken möglich sind.