FernUniversität in Hagen

Abschlussarbeit im Studiengang
Bachelor of Science in Informatik

Einfluss von Hashalgorithmen und Zufallsbits auf modifizierte Hashketten

Autor:
Ulrich Viefhaus
Matrikelnummer: 7679963
Ulrich.Viefhaus@FernUni-Hagen.de

Betreuer:
Prof. Dr. Jörg Keller
Lehrgebiet Parallelität und VLSI
Fakultät für Mathematik und Informatik
FernUniversität Hagen

17. Januar 2013

Erklärung zur Abschlussarbeit

Hiermit versichere ich gemäß der Prüfungsordnung, dass ich die vorgelegte Abschlussarbeit für den Studiengang Bachelor of Science in Informatik selbständig und ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt habe. Zitate aus diesen Quellen habe ich als solche kenntlich gemacht.

Nürnberg, den 17. Januar 2013

Ulrich Viefhaus

Zusammenfassung

In dieser Bachelorarbeit geht es um den Einfluss von Hashalgorithmen und Zufallsbits auf die Qualität von Hashketten. Nach einer Einführung in Lamports Hashketten werden einige Hashalgorithmen vorgestellt. In den durchgeführten Versuchen wurden die Wertemengen der Hashfunktionen für kleine Definitionsbereiche mit 10, 12 und 14 Bit sowie 0 bis 4 Zusatzbits generiert und ausgewertet. Dabei wurden auch verschiedene Arten der Reduktion der Bitzahl untersucht. Es hat sich gezeigt, dass es keine signifikanten Unterschiede bei den Hashfunktionen gibt, aber die Verwendung von einem Zusatzbit sinnvoll ist.

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Inhalt und Motivation

In dieser Bachelorarbeit geht es um die Frage, welchen Einfluss die Wahl des Hashalgorithmus und die Anzahl der verwendeten zusätzlichen Zusatzbits auf Hashketten haben. Die Motivation dahinter ist, die Qualität von Hashketten in der Praxis zu verbessern und so die Sicherheit vernetzter Systeme zu erhöhen.

Das Ziel von Hashketten ist es, eine Authentifizierung in unsicheren Umgebungen wie z.B. in öffentlichen Netzwerken zu ermöglichen. Dabei soll weder das Passwort auf dem Server gespeichert noch im Klartext übertragen werden müssen. Durch wiederholte Anwendung einer Hashfunktion wird aus einem Startwert eine Kette von Hashwerten erzeugt, die anschließend als Einmalpasswörter verwendet werden können. Das Verfahren hat aber einige Schwachstellen, die Erweiterungen des ursprünglichen Konzepts nötig machen. Eine davon ist die Erweiterung der Hashkette um zufällige Zusatzbits.

1.2 Fragestellung

Die eigentliche Fragestellung dieser Arbeit ist, ob es Unterschiede in der Qualität der Hashketten gibt, wenn unterschiedliche Hashalgorithmen zum Einsatz kommen und diese mit oder ohne Zusatzbits in den Hashketten eingesetzt werden. Die Qualitätskriterien hierfür werden in Kapitel 5 erklärt. Hierbei interessierte besonders, ob einige der inzwischen als unsicher geltenden Hashalgorithmen messbare Unterschiede im Vergleich zu modernen Algorithmen aufweisen würden. Da die Hashfunktionen sinnvollerweise so lange Ausgaben haben, dass eine vollständige Berechnung aller Ein- und Ausgabepaare nicht in praxisrelevanter Zeit möglich ist, mussten die Ausgabewerte gekürzt werden. Daher wurde auch untersucht, auf welche Weise sich die Hashwerte für das Experiment einfach kürzen ließen und welchen Einfluss die gewählte Art der Reduktion auf die Ergebnisse hat. Zuletzt geht es darum, die Anzahl der Zusatzbits, die eine optimales Verhältnis von zusätzliche, Aufwand und Sicherheitsgewinn bieten, zu ermitteln.

1.3 Ergebnisse

Die Ergebnisse der Arbeit sind wie folgt:

- In dem durchgeführten Experimenten mit gekürzten Hashwerten konnte kein signifikanter Qualitätsunterschied bei den Hashalgorithmen festgestellt werden. Durch seine flexible Gestaltung wird dennoch die Verwendung von KECCAK für Hashketten empfohlen.
- Die Verwendung von zufällig generierten Zusatzbits ist eine sinnvolle Ergänzung des ursprünglichen Konzepts der Hashkette. Am sinnvollsten wird ein Zusatzbit verwendet und bei Bedarf die Länge der Hashwerte weiter vergrößert.
- Während der Erstellung dieser Arbeit wurden weitere Fragen aufgeworfen, die noch nicht geklärt werden konnten.

1.4 Aufbau der Arbeit

Diese Bachelorarbeit ist wie folgt aufgeteilt. Zu Beginn stehen in Kapitel 2 die Grundlagen, Notationen und Definitionen, die für die restliche Arbeit benötigt werden. Danach wird die generelle Funktionalität der Lamport'schen Hashkette in Abschnitt 3.1 erläutert, um den Kontext der durchgeführten Experimente darzustellen. Es werden auch die Vor- und Nachteile der Authentifizierung mit Hashketten behandelt und einige Verbesserungen in Abschnitt 3.2 kurz dargestellt werden. Darauf folgt eine kurze Vorstellung der Hashfunktionen (Abschnitt 4.1), die für die Experimente gewählt wurden. Im Anschluss finden sich die Beschreibung (Kapitel 5) sowie die Resultate (Abschnitt 5.2) der durchgeführten Experimente. Diese werden abschließend interpretiert (Kapitel 6). Im Anhang befindet sich das Literaturverzeichnis (Abschnitt 6.3), die Daten der durchgeführten Experimente (Anhang A) sowie der Inhalt der beigefügten DVD (**??**).

Kapitel 2

Grundlagen und Definitionen

Für die Authentifizierung werden zwei Partner benötigt. Derjenige, der sich gegenüber dem anderen authentisieren will, wird Client genannt. Der Partner, der den anderen authentifiziert, wird Server genannt. Die Kommunikation zwischen einem Server und einem Client wird wie folgt dargestellt:

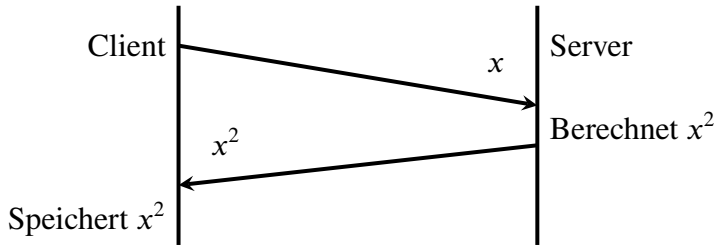

Abbildung 2.1: Beispiel für Kommunikation zwischen Client und Server

Auf der linken Seite der Darstellung ist der Client auf der rechten Seite der Server dargestellt. Jeder diagonale Pfeil stellt eine Nachricht dar, deren Inhalt am Pfeilende abzulesen ist. Auf der Empfängerseite ist notiert, wie dieser die Nachricht verarbeitet. Der zeitliche Ablauf der Kommunikation ist von oben nach unten aufgetragen. In Abbildung 2.1 sendet der Client also zuerst x an den Server. Dieser berechnet x^2 und sendet diesen Wert an den Client. Der Client speichert den empfangenen Wert x^2.

Hashketten sind die Grundlage für diese Arbeit. Daher wird zunächst eine formale Definition dieser benötigt. Dazu muss zunächst geklärt werden, was eine Hashfunktion ist.

Definition 1. Eine Einwegfunktion f ist eine Funktion, die leicht berechenbar aber deren Umkehrfunktion f^- nur mit sehr großem Aufwand berechenbar ist. Siehe auch [1, Seite 100]

Definition 2. Eine Einweghashfunktion h ist eine Einwegfuntkion der Form

$$h : U \to R$$

wobei U ein unendliches Universum ist und R eine endliche Untermenge von U ist. Vergleiche hierzu [2, Seite 1].

Da R eine Untermenge von U ist, kann eine Hashfunktion wiederholt angewandt werden. In der Praxis ist U häufig endlich.

Der Einfachheit halber werden im Folgenden Einweghashfunktion als Hashfunktion bezeichnet. Die Hashlänge in Bits bezeichnet die Anzahl der Stellen des Ausgabewerts einer Hashfunktion in Binärdarstellung. Diese ist bei vielen Hashfunktionen 128 oder 256 Bit und muss daher für das Experiment gekürzt werden.

(Zufällige) Zusatzbits sind Bits, die durch einen Zufallszahlengenerator erzeugt wurden und als Suffix an den Eingabewert einer Hashfunktion angehängt werden.

Für die Hashketten wird ein geheimer Startwert benötigt, aus dem alle anderen Kettenglieder abgeleitet werden. Dieser wird im folgenden (geheimer) Schlüssel genannt und mit K abgekürzt.

Definition 3. Eine Hashkette ist die wiederholte Anwendung einer Hashfunktion auf ein Element der Untermenge $R \subset U$. Sei $K \in R$ und $n \in \mathbb{N}$. Dann sei

$$h_n(K) = h(h_{n-1}(K)) = h(h(h_{n-2}(k))) = \underbrace{h(h(h(\dots(h(K)))))}_{\text{n-mal}}$$

die Hashkette h des Startwertes K der Länge n.

Definition 4. Für eine Untermenge $R \subset U$ einer Hashfunktion gilt:

$$h(R) = \{h(x) | x \in R\}$$

Ferner sei $h_0(R) = R$, $h_1(R) = h(R)$ und $h_{i+1}(R) = h(h_i(R))$, mit $i \in \mathbb{N}$. Statt $h_i(R)$ schreibt man kurz R_i.

Lemma 5. *Für jedes $i \in \mathbb{N}_0$ gilt:*

$$h_{i+1}(R) \subseteq h_i(R)$$

. (siehe [2, S. 3])

Dies bedeutet, dass Hashketten sie Wertemenge nicht vollständig nutzen können. Problematisch wird es dann, wenn die genutzte Untermenge für kryptographische Anwendungen zu klein wird und Brute-Force Angriffe ermöglichen. Um dies genauer analysieren zu können, kann die Menge U und die Hashfunktion h als gerichteter Graph betrachtet werden. Sei $R_0 = U$ und $h(x) : U \to R_0$. Dann kann R_0 als gerichteter Graph interpretiert werden, bei dem jeder Knoten $x \in R$ eine ausgehende Kante nach $h(x)$ hat. Wird jetzt jedes Blatt, also die Knoten mit Eingangsgrad 0, entfernt, erhalten wir den gerichteten Graphen für R_1. Dies kann solange fortgesetzt werden, biss keine Blätter mehr übrig sind. Diese Idee stammt aus [2, S. 5]. Für die Analyse benötigen wir weitere Begriffe aus der Graphen- und Mengentheorie.

Definition 6. Sei D eine Menge und $f : D \to \mathbb{N}$. Eine Multimenge über D ist ein paar $\langle D, f \rangle$.

Definition 7. Ein gerichteter Multigraph $G = (V, E)$ besteht aus einer Knotenmenge V und einer Menge von Kanten E, wobei E eine Multimenge über alle 2-elementigen Teilmengen von V ist.

Im folgenden werde ich der Einfachheit halber gerichteter Graph oder Graph statt gerichteter Multigraph schreiben.

Definition 8. Eine alternierende Folge von Knoten und gerichteten Kanten

$$v_1, \underbrace{(v_1, v_2)}_{e_1}, v_2, \underbrace{(v_2, v_3)}_{e_2}, v_3, \dots, v_{k-1}, \underbrace{(v_{k-1}, v_k)}_{e_{k-1}}, v_k$$

heißt Weg, wenn kein Knoten mehrfach auftritt. Wenn nur Anfangs- und Endknoten übereinstimmen wird die Folge Kreis genannt. (Vergleiche [3, S. 127])

Definition 9. Ein gerichteter Graph $G = (V, E)$ heißt stark zusammenhängend, wenn es für je zwei Knoten v und $w \in V$ mindestens einen Weg von v nach w und mindestens einen Weg von w nach v gibt.

Definition 10. Ein Graph $H = (V', E')$ heißt Teilgraph von $G = (V, E)$, wenn gilt:

$$V' \subseteq V \text{ und } E' \subseteq E$$

Verlgeiche [3, S. 15]

Definition 11. Ein maximaler stark zusammenhängender Teilgraph von $G = (V, E)$ ist eine starke Zusammenhangskomponente von G.

Im folgenden werden starke Zusammenhangskomponenten bei Bedarf mit s.Z. abgekürzt.

Definition 12. Eine schwache Zusammenhangskomponente eines gerichteten Graphen ist eine maximale Teilmenge seiner Knoten, die zusammenhängend wäre, falls die Richtung der Kanten außer Acht gelassen würde.

Definition 13. Die Anzahl der direkten Vorgänger eines Knoten n heißt Eingangsgrad des Knoten n. Analog dazu heißt die Anzahl der direkten Nachfolger eines Knoten n Ausgangsgrad des Knoten n.

Definition 14. Eine Kante, die den Knoten n mit einem direkten Vorgänger verbindet, wird eingehende Kante von n genannt. Analog dazu heißt eine Kante, die den Knoten n mit einem direkten Nachfolger verbindet, ausgehende Kante von n.

Betrachten wir nun einen Graphen für ein Universum U und eine Hashfunktion h. Beginnend bei einer beliebigen Eingabe, dem Schlüssel K, sind die folgenden Hashwerte durch den Weg, der von dem Knoten der Eingabe wegführt, festgelegt. Dieser Weg endet in einer starken Zusammenhangskomponente. Alle weiteren Hashwerte stammen dann aus dieser starken Zusammenhangskomponente, die je nach Struktur des Graphen sehr viel kleiner ist, als die eigentliche Wertemenge.

Werden zufällige Zusatzbits verwendet, entstehen im Graphen zusätzliche Kanten zwischen den Knoten. Jeder Knoten hat nun den Ausgangsgrad 2^z, wobei z die Anzahl der zufälligen Zusatzbits ist. Dadurch werden die starken Zusammenhangskomponenten deutlich größer.

Zur Veranschaulichung zeigt Abbildung 2.2 den Graphen für die Hashfunktion f in Definition 15.

Definition 15. Sei f eine Hashfunktion mit der Definitionsmenge $D = \{a, b, c, d\}$ und der Wertemenge $W = \{a, b, c\}$. f ist wie folgt definiert: $f(a) = b$, $f(b) = c$, $f(c) = b$ und $f(d) = a$.

Obwohl der Wert a in der Wertemenge von f enthalten ist, ist wird er bereits nach der ersten Iteration nicht mehr als Ausgabewert der Hashfunktion f vorkommen, da er nicht in der starken Zusammenhangskomponente b, c liegt. In diesem Beispiel besteht die starke Zusammenhangskomponente nur aus einem Kreis mit den beiden Knoten b und c. In größeren Werteräumen und

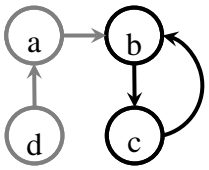

Abbildung 2.2: Darstellung der Funktion f als Graph. Die starke Zusammenhangskomponente ist dunkel dargestellt und besteht aus einem Kreis.

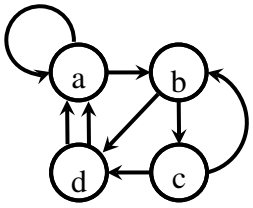

Abbildung 2.3: Darstellung der Funktion f' als Graph. Die starke Zusammenhangskomponente enthält nun alle Knoten und besteht aus mehreren Kreisen.

bei der Hinzunahme von Zusatzbits werden die Strukturen der starken Zusammenhangskomponenten komplexer. Wird f erweitert, so dass Zusatzbits verwendet werden können, erhält man den Graph in Abbildung 2.3.

In der späteren Vorstellung der verwendeten Hashfunktionen werden einige Begriffe der Kryptographie bzw. ihre üblichen Abkürzungen verwendet. Diese werden nun eingeführt. Ein **Brute-Force Angriff** ist ein Angriff, bei dem systematisch alle möglichen Zeichenkombinationen durchprobiert werden, um einen geheimen Schlüssel zu erraten. Ein **Message Authentication Code**, im folgenden **MAC** abgekürzt, ist eine Einweghashfunktion, die von einem zusätzlichen Schlüssel abhängig ist. Nur mit dem Schlüssel kann der Hashwert verifiziert werden (vergleiche [4, S. 455 ff.]). Eine **Key Derivation Function**, im folgenden **KDF** abgekürzt, ist eine Funktion, um aus einem geheimen Passwort und evtl. einer nichtgeheimen Komponente einen geeigneten sicheren Schlüssel abzuleiten (vergleiche [5, S. 5]). Die **wide-pipe** bezeichnet eine Technik, bei der die Hashfunktion intern mit einem wesentlich größeren Wert rechnet, als letztlich als Ausgabe verlangt wird. Dadurch werden interne Kollisionen erschwert (siehe auch [6]). Eine **One-Time-Signature** oder kurz **OTS** ist ein Signaturverfahren, das auf einer Hashfunktion basiert. Es wird ein privater bzw. geheimer und ein öffentlicher Schlüssel benötigt. Diese bilden ein Schlüsselpaar. Der private Schlüssel, auch One Time Secret Key genannt, eines Schlüsselpaares wird mit K_S, der öffentliche Schlüssel, oder auch One Time Public Key, mit K_P bezeichnet. Ein **Second-Preimage-Angriff** auf eine Hashfunktion ist der Versuch, zu einer frei gewählten Nachricht m_1 eine zweite Nachricht m_2 zu finden, so dass beide den gleichen Hashwert besitzen.

Kapitel 3

Authentifikation mit Hashketten

3.1 Lamports ursprüngliche Variante

Lamport schlug in [7] eine Methode vor, um sich als Client in unsicheren Umgebungen gegenüber einem Server zu identifizieren. Er verwendet hierzu einen geheimen Schlüssel K, der als Startwert einer Hashkette genutzt wird. Der Hash $h_i(K)$ sowie der Wert i werden an den Server übertragen und dort gespeichert. Will sich der Client gegenüber diesem authentifizieren, schickt er seinen Namen an den Server. Dieser antwortet mit dem aktuellen Wert des Zählers i. Der Client berechnet nun $h_{i-1}(K)$ und schickt den Hashwert an den Server. Dieser berechnet $h(h_{i-1}(K))$ und vergleicht das Ergebnis mit dem gespeicherten Wert $h_i(K)$. Stimmen beide überein, so war die Authentifizierung erfolgreich, $h_i(K)$ wird durch $h_{i-1}(K)$ ersetzt und der Zähler i wird dekrementiert. Da die Hashfunktion nicht umkehrbar ist, kann nur der Client den Wert $h_{i-1}(K)$ berechnen, da er ihn mit Hilfe der Hashkette aus seinem Schlüssel K generieren kann.

3.1.1 Vorteile

Die Vorteile des Verfahrens nach Lamport sind:

- Keine Übertragung des geheimen Schlüssels K. Somit kann die Authentifizierung über einen unsicheren Kanal übertragen werden.
- Keine Speicherung des Schlüssels auf Serverseite. Ein Diebstahl der Identität nach Erlangen der gespeicherten Daten auf dem Server ist daher nicht möglich.
- Hashketten sind schneller als PublicKey Infrastrukturen.

3.1.2 Nachteile

Allerdings gibt es auch einige Nachteile:

- Die Länge der Hashkette ist aus Effizienzgründen beschränkt, so dass immer wieder eine neue Kette erzeugt und der Hash und der Zähler an den Server übertragen werden müssen.
- Es gibt einen so genannten „small-n“ Angriff auf die Hashketten, bei dem der Angreifer vorgibt der Server zu sein und einen um j niedrigen Wert i anfordert. Der Client schickt dann den passenden Hash $h_{i-1}(K)$. Mit diesem kann der Angreifer sich beim echten Server $i - j$ mal als Client authentifizieren, ohne dass dies bemerkt wird.

- Neben dem *small-n* Angriff ist auch ein herkömmlicher *Man-in-the-Middle* Angriff möglich.
- Bei Verwendung des gleichen geheimen Schlüssels K für Hashketten zur Authentifizierung an verschiedenen Servern, kann sich ein Server A unter bestimmten Vorraussetzungen als Client gegenüber Server B ausgeben. Hierzu muss sich der eigentliche Client i mal am Server A und j mal an Server B angemeldet haben, wobei $i > j$ gelten muss. Aus dem ihm bekannten Kettenglied $h_i(K)$ kann der Server A das alle späteren Kettenglieder und somit auch $h_j(K)$ berechnen, da

  $$h_n = h(h_{n-1}(h_{n-2}(\ldots(h_{n-(j-1)}(h_{n-j}(h_{n-(j+1)}(\ldots(h_{n-(i-1)}(h_{n-i}))))))))$$

  gilt. Mit diesem kann er sich am Server B fälschlicherweise als Client ausgeben.
- Wie in Kapitel 2 erläutert, nutzen Hashketten nur eine Teilmenge der Wertemenge. Falls diese Teilmenge zu klein wird, können Brute-Force Angiffe in realistischer Zeit durchgeführt werden.

3.2 Verbesserungen

Um den oben genannten Problemen beizukommen gibt es einige Verbesserungen und Steigerungen der Effizienz. Im folgenden beschränke ich mich auf die sicherheitsrelevanten Verbesserungen. Für weitere Informationen zur Effizienzsteigerung siehe z.B. [8], [9], [10], [11] und [12].

Um dem *small-n* Angriff zu unterbinden, genügt es, dass sich der Client für jeden Server merkt, welches i zuletzt geschickt wurde (Siehe Abbildung 3.1). Fragt der Server nach dem falschen Wert, handelt es sich offensichtlich nicht um den richtigen Server, sondern um einen Angreifer. *Man-in-the-Middle* Angriffe lassen sich auf diese Weise allerdings nicht unterbinden.

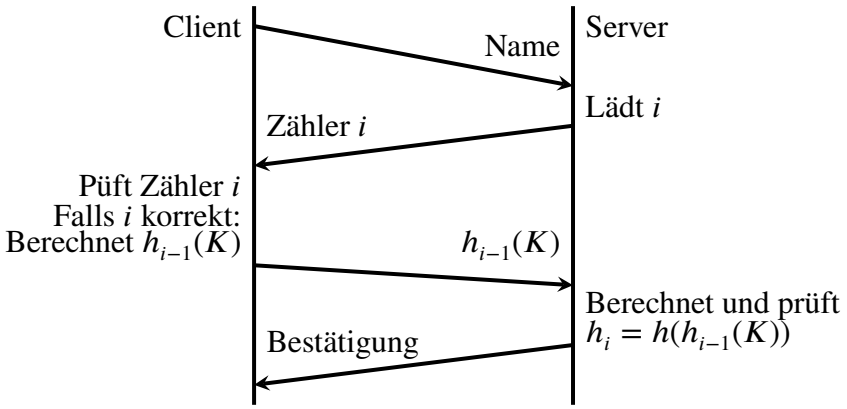

Abbildung 3.1: Authentifizierung mit einer Hashkette und Überprüfung des angeforderten Kettenglieds anhand eines gespeicherten Zählers.

Gegen *Man-in-the-Middle* Angriffe gibt es keine Erweiterungen der ursprünglichen Methode, da hier eine beiderseitige Authentifizierung stattfinden müsste. Diese ist allein mit der Authentifizierung über Hashketten nicht möglich.

Damit jeder Server eine eigene eindeutige Hashkette zugewiesen bekommt, ist es ausreichend, zusätzlich zu dem geheimen Schlüssel K einen Seed S zu benutzen. Ein Kettenglied

wird dann wie folgt berechnet: $h_i = (h(h_{i-1}(K), S)$. Bei der Initialisierung wird der Seed S an den Server geschickt und dort gespeichert. Um Speicherplatz auf dem Client zu sparen, kann der Seed dort gelöscht und vom Server zusammen mit dem aktuellen i übermittelt werden. Der Client berechnet dann h_{i-1} und schickt den Hash an den Server, der $h_i = (h(h_{i-1}(K), S)$ berechnet und auf Korrektheit überprüft. Der Ablauf ist in Abbildung 3.2 zu sehen.

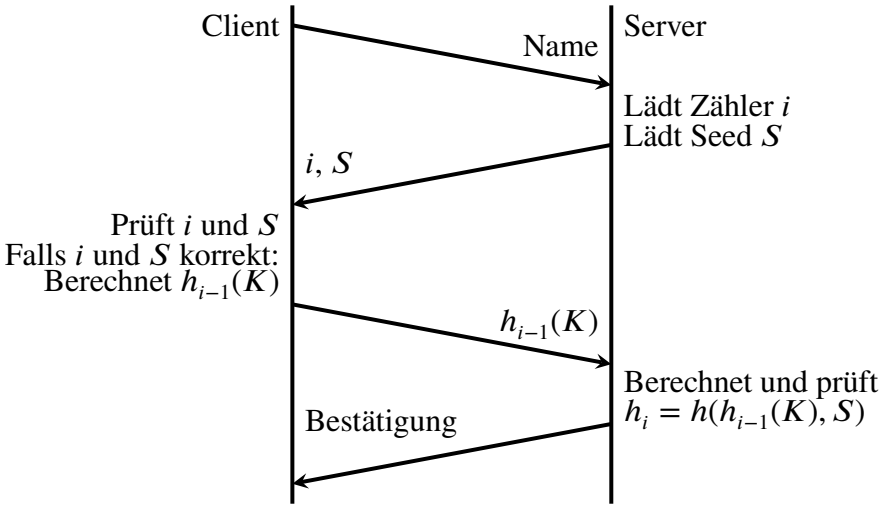

Abbildung 3.2: Authentifizierung mit einer Hashkette mit Seeds und Überprüfung des angeforderten Kettenglieds.

Für die Reinitialisierung der Hashkette wird in [13] die Verwendung einer One-Time-Signature vorgeschlagen. Für weitere Informationen über OTS siehe [14], [15] und [16]. Die Verwendung einer OTS hat den Vorteil, dass die Reinitialisierung automatisch ablaufen und über einen unsicheren Kanal erfolgen kann. Neben der Hashkette der Länge $i - 1$ wird zusätzlich ein One Time Secret Key K_S und ein One Time Public Key K_P für den Client erstellt. Das Ende der Hashkette wird dann wie folgt berechnet: $h_i = h_{i-1}((K), S, h(K_P))$. Hier wird bereits die oben erwähnte Version mit Seed S verwendet. Jetzt kann dieser Hash zusammen mit dem Wert i, dem Seed S und dem Hash des Puplic Keys $h(K_P)$ an den Server übertragen werden. Siehe Abbildung 3.3.

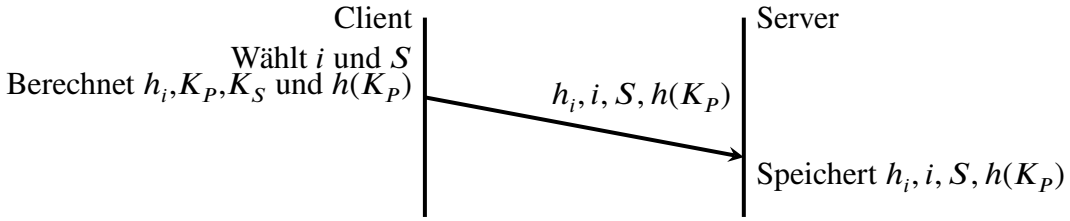

Abbildung 3.3: Initialisierung der Hashkette mit Zusatzinformationen zur automatischen Reinitialisierung.

Will sich nun der Client mit dem Server verbinden, schickt er seinen Namen. Der Server antwortet wie bisher mit dem gespeicherten Wert des Zählers i und dem Seed S und der Client schickt wie gewohnt $h_{i-1}(K)$ an den Server. Wenn es sich dabei um die erste Authentifizierung handelt, muss der Server $h(h_{i-1}(K), S, h(K_P))$ berechnen und mit dem gespeicherten Kettenende vergleichen. Will sich der Client später erneut authentifizieren, berechnet der Server das

nachfolgende Kettenglied wieder mit mit der Formel $h_i = h(h_{i-1}(K), S)$. Der Ablauf ist in Abbildung 3.4 dargestellt.

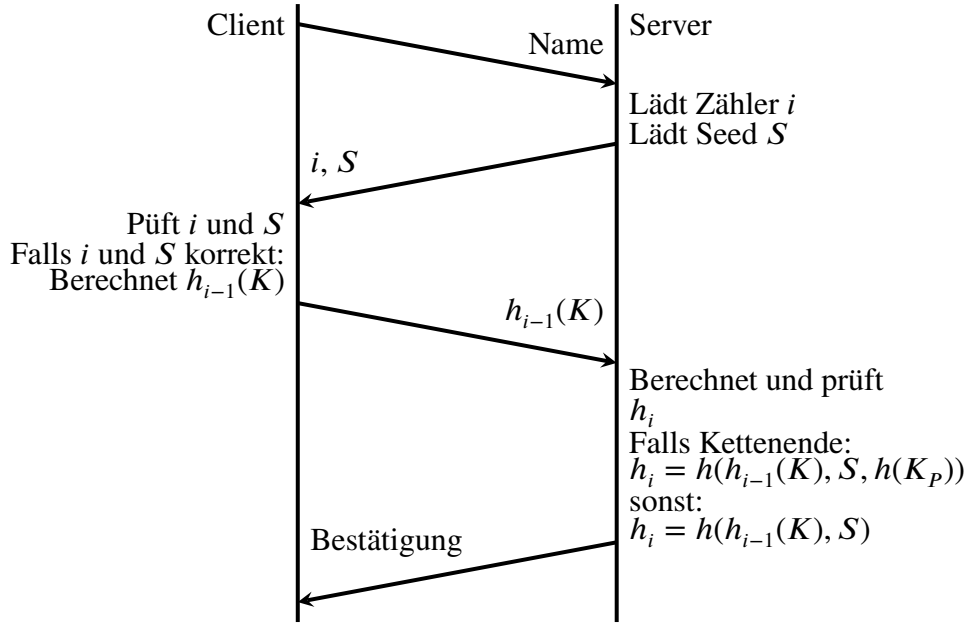

Abbildung 3.4: Nutzung der Kettenglieder zur Authentifizierung.

Nachdem alle Kettenglieder verbraucht sind, wird ein neuer Seed S', sowie ein neues Paar One Time Keys K'_P und K'_S. Nun wird zunächst wieder eine Hashkette der Länge $i - 1$ mit dem neuen Seed berechnet und dann das Ende der Kette: $h'_i = h'_{i-1}(K), S', h(K'_P)$. Dieses wird zusammen mit dem Wert i dem Seed S', dem öffentlichen Schlüssel K_P und mit den Teilen des geheimen Schlüssels K_S, die nötig sind, h'_i gegenüber dem Server zu signieren, verschickt. Der Server kann K_P anhand des in der vorherigen Kette enthaltenen $h(K_P)$) überprüfen. Die Korrektheit des Kettenendes h'_i kann er anhand der preisgegebenen Teile von K_S überprüfen und das Kettenende dann zusammen mit i und dem Seed S' in den Speicher übertragen. Nun können die Kettenglieder wieder wie oben beschrieben genutzt werden.

Eine geringfügig effizientere Variante ist es, die bekanntgegebenen Teile des One Time Secret Key K_S ebenfalls als Teil der Hashkette zu verwenden und zu verbrauchen. Siehe hierzu [13, 5.2].

Es wurde bereits in Kapitel 2 erläutert, dass bei Hashketten nach einer endlichen Anzahl i von Hashberechnungen nur noch Hashwerte aus einer Untermenge R_i der Wertemenge vorkommen. In [2] wurde dies theoretisch erläutert und experimentell veranschaulicht. Die Autoren zeigen, dass die Hashwerte als Knoten eines gerichteten Graphen betrachtet werden können, bei denen durch die Hashfunktion eine gerichtete Kante zwischen der Eingabe und der Ausgabe erzeugt wird.

Diese Strukturen der Graphen werden nun in Kapitel 5 genauer betrachtet. Der Schwerpunkt liegt dabei auf dem Vergleich der Graphen von verschiedenen in der Praxis verwendeten Hashfunktionen, sowie die Verwendung von zufälligen Zusatzbits.

Um zufällige Zusatzbits verwenden zu können, muss ein Pseudozufallszahlengenerator eine entsprechende Anzahl an Bits liefern. Diese müssen dann in die Kette mit einfließen. Allerdings muss der Server den Zustand des Zufallszahlengenerators kennen, um für ein Kettenglied die

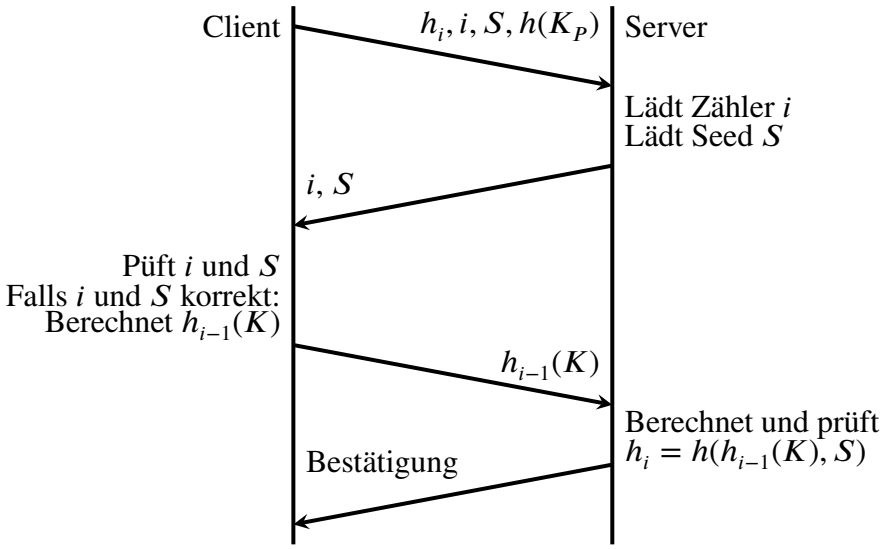

Abbildung 3.5: Automatische Reinitialisierung der Hashkette mittels One-Time-Signature*.*

entsprechenden Zufallsbits in den Hashvorgang einfließen lassen zu können. Für die folgende Betrachtung werden der Einfachheit halber die oben genannten Verbesserungen weggelassen. Es bieten sich hierbei eine einfache Möglichkeiten an, die Zusatzbits zu verwenden. Der Zufallzahlengenerator liefert dabei für eine Hashkette der Länge j $j * z$ Zufallsbits, wobei z_1 den ersten und z_j den letzten ausgegebenen Zufallsbits entspricht. Der Anfang der Hashkette wird dann mit $h_1 = h(K, z_1)$ und alle anderen Kettenglieder mit $h_i = h(h_{i-1}, z_i)$ berechnet. Der Anfangszustand des Zufallszahlengenerators muss zusammen mit dem Ende der Hashkette auf sicherem Wege auf dem Server gespeichert werden. Server und Client generieren dann für jedes Kettenglied die selben z Zufallsbits und berechnen daher die selben Hashwerte. Der Rechenaufwand kann auf Kosten des Speicherplatzes reduziert werden, indem nach einer festgelegten Anzahl an Kettengliedern der Zustand des Zufallszahlengenerators gespeichert wird. Die Berechnung von Zufallsbits für spätere Kettenglieder erfolgt dann von diesem Zwischenstand aus. Die Länge der übertragenen Hashwerte vergrößert sich dadurch nicht, denn die Hashwerte werden auf die gleiche Länge gekürzt, wie die ohne Zusatzbits. Der Sicherheitsgewinn basiert neben den längeren Eingabewerten auf dem Effekt, das die Wertemenge zu größeren Teilen genutzt wird.

Kapitel 4

Hashalgorithmen

In diesem Kapitel werden die Hashalgorithmen, die im Experiment verwendet wurden, kurz vorgestellt. Dabei wird kurz auf ihren Aufbau, Sicherheit und Geschwindigkeit eingegangen. Um die Geschwindigkeit beurteilen zu können wurden verschiedene Quellen verglichen.

4.1 Eine Auswahl an Hashfunktionen

Als erstes werden die 5 SHA-3 Finalisten vorgestellt. Weiterführende Informationen zu diesen vergleichsweise jungen, aber dennoch gut untersuchten Chiffren gibt es unter [17], [18], [19], [20] und [21]. Danach gehe ich kurz auf einige ältere aber verbreitete Chiffren ein.

BLAKE

BLAKE [22] basiert auf **HAIFA** [23], dem **HA**sh **I**terative Fr**A**mework, und ChaCha [24], einer Variation der Salsa20 Stromchiffren Familie [25]. **HAIFA** ersetzt hier beim Entwurf des Hashverfahrens die gängige Merkle-Damgård Konstruktion(siehe z.B. [26]) und ermöglicht explizit die Verwendung von Salts. Außerdem schützt es vor Second-Preimage-Angriffen. Durch die Verwendung einer wide-pipe werden lokale Kollisionen innerhalb der Kompressionsfunktion verhindert. Die BLAKE Chiffrenfamilie unterstüzt außerdem Random Hashing (siehe auch [27]), um die Sicherheit zu erhöhen. Die Autoren heben außerdem die Möglichkeit, BLAKE zu parallelisieren und die Möglichkeit, ihn auf schwacher Hardware einzusetzen hervor [22, S. 4]. Diese Eigenschaften machen ihn für den Einsatz in Hashketten, wie sie in dieser Arbeit untersucht werden, sehr interessant, da durch schnelle Verarbeitung der Hashketten Energie gespart werden kann.

Grøstl

Auch Grøstl [28] verhindert durch eine wide-pipe interne Kollisionen und unterstützt Random Hashing. Der „randomisation mode“ ist dem MAC-Modus sehr ähnlich, ist aber z.B. für Salts vorgesehen (siehe [28, S. 17]) und könnte auch dafür verwendet werden, um Zufallsbits mit in die Berechnung eingehen zu lassen. Der Algorithmus selbst basiert statt auf Blockchiffren auf zwei Permutationen nach der „wide trail design“ (siehe auch [29]) Strategie, die auch in AES Anwendung fand, und S-Boxen, die mit denen aus AES identisch sind. Die Autoren nutzten

daher sehr gut erforschte Techniken, um Grøstl zu entwerfen. Durch das unkomplizierte Design sollte die Familie der Grøstl-Chiffren performant genug für den Einsatz in Hashketten sein.

JH

Wie auch schon Grøstl setzt JH [30] auf die „wide trail design“ Strategie von AES und kombiniert einfache Bauteile zu einer Blockchiffre mit großen Blöcken. Auf diesen baut die Kompressionsfunktion auf. Um dies zu ermöglichen, wurde eine neue Technik zum Ableiten einer Kompressionsfunktion aus bijektiven Funktionen entwickelt. Dadurch entsteht zwar die einfache Möglichkeit, Kollisionen in der Kompressionsfunktion zu erzeugen, allerdings spielt es in realen Anwendungen keine Rolle, da der initiale Wert für die Funktion fix ist. Für Details siehe [30, S. 4f] und [30, S. 26f]. Für die finale Version wurde die Rundenzahl auf 42 erhöht, wodurch die Anzahl der S-Boxen (4x4-bit) auf 10752 gestiegen ist. Trotzdem dürfte JH dank seiner einfachen Struktur gut auf langsamer Hardware implementierbar sein.

KECCAK

KECCAK [19] basiert auf einer Technik die „sponge construction“ (siehe [31]) genannt wird. Mithilfe einer Permutation mit fester Länge und einer Padding Rule wird eine Funktion erzeugt, deren Eingabe und Ausgabe variabler Länge sein können. Dadurch ist KECCAK sehr flexibel und kann den eigenen Sicherheits- und Geschwindigkeitsanforderungen angepasst werden. Gerade diese Andersartigkeit hat ihm kürzlich zum Sieg im SHA3-Wettbewerb verholfen (siehe [32]). Die Möglichkeit, die Funktion einfach so anzupassen, dass sie Hashwerte der gewünschten Länge ausgibt, könnte die Qualität und Geschwindigkeit der Hashketten weiter verbessern, da nicht Bits aus langen Hashwerten extrahiert werden müssen (siehe auch Unterabschnitt 5.2.1).

SKEIN

SKEIN[33] besteht aus drei Komponenten:

- Threefish
- Unique Block Iteration (UBI)
- Optional Argument System

Threefish ist eine Blockchiffre mit einer Blockgröße von 256, 512 oder 1024 Bit. Die Rundenfunktion ist sehr einfach aufgebaut und besteht nur aus XOR, Addition und konstanter Rotation. Dafür werden 72 Runden durchgeführt. UBI basiert auf dem Matyas-Meyer-Oseas Hashverfahren, welches aus einer Eingabe fixer Länge eine Ausgabe fixer Länge erzeugt, und führt nach einer Zerlegung einer beliebig langen Eingabe die Aufrufe von Threefish durch. SKEIN kann durch Übergabe von Parametern als ein MAC oder KDF genutzt werden.

MD5

MD5 [34] wurde 1991 als überarbeitete Version von MD4 von Ron Rivest veröffentlicht, nachdem sich MD4 als zu unsicher herausstellte (siehe [35, S. 215]). MD5 erzeugt wie sein Vorgänger einen 128-Bit Hash. MD5 war sehr weit verbreitet, ist aber ebenfalls unsicher (siehe [36], [37], [38], [39], [40] und [41]). Aus Gründen der Kompatibilität wird MD5 aber noch oft unterstützt und ich werde es mit einbeziehen, um einen bekannten Vergleichswert für die Geschwindigkeit der Chiffren zu haben.

RIPEMD und RIPEMD-160

RIPEMD [42] wurde von Hans Dobbertin, Antoon Bosselaers und Bart Preneel im Rahmen des EU-Projekts RIPE (Race Integrity Primitives Evaluation) entwickelt und steht für RACE Integrity Primitives Evaluation Message Digest. Die Basis für RIPEMD bildet wie bei MD5 MD4. Die Chiffre arbeitet mit Blöcken mit einer Länge von 512 Bit. Nach der Entdeckung mehrerer Schwächen in MD4 und MD5, wurde die Chiffre 1996 durch RIPEMD-160, eine auf 160 Bit erweiterten Version, ersetzt. 1995 fand Dobbertin Kollisionen für die letzten beiden Runden der ursprünglichen RIPEMD-Version (siehe [43]). 2004 wurden Kollisionen für die vollständigen Rundenzahl gefunden[37]. 2011 wurde ein Angriff auf eine schrittreduzierte Version von RIPEMD-160 gefunden (siehe [44]. Der Durchsatz liegt etwa 15% niedriger als bei SHA-1 (siehe [35, S. 219 f.].

SHA-1

SHA-1 (siehe [45]) steht für Secure Hash Algorithm, wobei es sich um eine leicht geänderte Version handelt, was durch die 1 im Namen angedeutet wird. Die ursprüngliche Version SHA-0 (siehe [46]) enthielt eine nicht publizierte Schwäche. In [47] wird ein Angriff zum Finden von Kollisionen bei SHA-0 beschrieben. Beide Versionen wurden vom NIST zusammen mit der NSA entwickelt und arbeiten mit 160-Bit Hashes und basiert ebenfalls auf MD4. Es wird das gleiche Padding wie bei MD4 und RIPEMD verwendet und hat einen ähnlichen Aufbau. 2005 gelang es Xiaoyn Wang, Andrew Yao und Frances Yao effizient, also schneller als mit dem Geburtstagsangriff, Kollisionen für SHA-1 zu erzeugen (siehe [48] und [49]). Auf der Crypto06 wurde schließlich ein Angriff veröffentlicht, bei dem Teile der Nachrichtentexte frei gewählt werden können (siehe [50]).

SHA-2 Familie

Nachdem mögliche Angriffe auf SHA-1 bekannt wurden, wurde die SHA-2 Familie spezifiziert, die aus den Chiffren SHA-256, SHA-224, SHA-512 und SHA-384 besteht (siehe [51]). SHA-256 wurde gegenüber SHA-1 verbessert und die Länge des Hashes erweitert. Bei SHA-224 werden 32 Bit der Ausgabe von SHA-256 einfach verworfen, bei SHA-384 128 Bit der Ausgabe von SHA-512. Laut [35, S. 214] ist SHA-2 nicht anfällig gegen die Angriffe von Wang, Yao und Yao.

4.2 Geschwindigkeitsvergleich

Im folgenden werden die Geschwindigkeiten der vorgestellten Chiffren verglichen. Die Geschwindigkeit spielt eine wichtige Rolle, da eine höhere Geschwindigkeit bei der Ausführung in Software einen niedrigeren Energieverbrauch mit sich bringt. Da Hashketten unter anderem wegen ihrem vergleichsweise geringen Rechenaufwand auch auf sehr kleinen Geräten genutzt werden, stellt der Energieverbrauch einen Faktor dar, der in die Entscheidung für einen Hashalgorithmus einfließt. Hardwareimplementierungen werden hier nicht berücksichtigt, da ihre Kosten und ihr Energieverbrauch von den verwendeten Techniken zur Herstellung abhängen und eine Analyse den Rahmen dieser Arbeit überschreiten würde. A. Mageshwari hat in [52] einige der vielversprechenden Kandidaten der 2. Runde des Wettbewerbs [53] für den SHA-3

Standard der NIST ausgewählt und Implementierungen für einen 8-Bit AVR Microcontroller geschrieben. Von seiner Auswahl sind in der finalen Runde[54] des Wettbewerbs noch BLAKE[22], KECCAK [19] und SKEIN [33] übrig geblieben. S. Heyse u.a. haben in [55] mehrere Kandidaten der 2.Runde untersucht, unter anderem die Finalisten Grøstl [28] und, wie bereits A. Mageshwari, BLAKE. P.-S. Murvay und B. Groza haben in [56] neben den bereits genannten noch den fehlenden Finalisten JH [30] untersucht. [57] liefert Testresultate für Geschwindigkeit, RAM- und ROM-Größe der fünf Kandidaten. Im Folgenden werden die einzelnen Finalisten knapp vorgestellt und auf ihre Eignung für Hashketten auf Microcontrollern geprüft.

Tabelle 4.1: Geschwindigkeit der Hashfunktionen mit AVR-Crypto Lib [58]

Algorithmus	Zyklen pro Byte
BLAKE-32	1114.69
Grøstl-256	8158.12
KECCAK-256	4725.79
Skein-1024-256 in C	8766.66
Skein-1024-256 in ASM	1719.27
Skein-512-256 in C	7594.59
Skein-512-256 in ASM	1444.20
Skein-256-256 in C	7258.91
Skein-256-256 in ASM	1206.44
MD5 in C	666.28
MD5 in ASM	285.28
SHA-1 in C	1183.84
SHA-1 in ASM	578.59
SHA-256 in C	2772.73
SHA-256 in ASM	783.20

Wie in Tabelle 4.1 zu sehen ist, schneidet die Implementierung MD5 erwartungsgemäß sehr gut ab, dient hier wegen der nachgewiesenen Unsicherheit der Chiffre aber nur als Basis für den Vergleich. Das ebenfalls ungenügende SHA-1 wird in seiner C-Implementierung bereits von dem SHA-3 Finalisten BLAKE-32 überholt. Die anderen Finalisten sind in ihrer C-Implementierung langsamer. In ASM kommt SKEIN mit Blockgröße 256 Bit für Threefish und 256 Bit Hash Ausgabewert fast an SHA-1 in C heran. Wenn man die $60,2\%$ Geschwindigkeitssteigerung von SKEIN in ASM zugrunde legt, müssten bei geeigneter Implementierung in ASM auch KECCAK-256 und Grøstl-256 an SHA-256 in C vorbeiziehen.

Tabelle 4.2: Geschwindigkeit der Hashfunktionen laut Spezifikationen

Algorithmus	Zyklen pro Byte	Zeit pro Nachrichtenblock	Plattform
BLAKE laut [22, S. 23]	406	2,6 ms	PIC18F2525
Grøstl laut [28, S. 27]	450 - 550	/	ATmega163
JH laut [30, S. 29]	1344 (geschätzt)	/	/

Schnellere Implementierungen werden von den Autoren der SHA-3 Finalisten in den jeweiligen Spezifikationen verwendet, um ihre Zykel pro Byte zu ermitteln. Es werden aber auch angepasste Versionen der Chiffren verwendet. Wie in Tabelle 4.2 zu sehen ist, liegt die Zahl der Zyklen bei BLAKE und Grøstl sogar unter der der C-Implementierung von MD5 aus Tabelle 4.1. JH schneidet deutlich schlechter ab, allerdings ist der Wert aus den Werten für einen Core2 approximiert.

Tabelle 4.3: Geschwindigkeit der Hashfunktionen laut [52]

Algorithmus	Zyklen pro Byte
BLAKE-32	480
KECCAK	5090
SKEIN-512-256	3970

Auch in Tabelle 4.3 zeigt sich, dass BLAKE-32 eine gute Wahl zu sein scheint, wenn es auf Geschwindigkeit ankommt. Die beiden Assembler-Implementierungen von KECCAK und SKEIN scheinen nicht an die Qualität der AVR-Crypto Lib heranzureichen, da sie deutlich langsamer sind.

Tabelle 4.4: Geschwindigkeit der Hashfunktionen bei langen Nachrichten laut [55]

Algorithmus	Zyklen pro Byte
BLAKE-32	321.1
Grøstl-256	687

In Tabelle 4.4 zeigt sich wieder, dass BLAKE-32 eine gute Wahl ist, falls Geschwindigkeit eine wichtige Rolle spielt. Die Geschwindigkeit von Grøstl erreicht nicht ganz die, die die Autoren in der Spezifikation angeben, ist aber dennoch gut.

Tabelle 4.5: Geschwindigkeit der Hashfunktionen bei Nachrichten der Länge 64 Byte laut [56] [55]

Algorithmus	Zyklen pro 64 Byte	Zyklen pro Byte
MD5	49378	771,5
SHA-1	119537	1887,8
SHA-256	288978	4515,3
BLAKE-32	64887	1013,9
Grøstl-256	242124	3783,2
JH-256	262992	4109,3
KECCAK-256	258129	4033,3
SKEIN-512-256	354395	5537,4

In Tabelle 4.5 sind die optimierten Ergebnisse von [56] für 64 Byte Lange Eingaben zusammengefasst. BLAKE-32 ist wieder der schnellste der SHA-3 Finalisten und hängt sowohl SHA-1, als auch SHA-256 ab. Die anderen Finalisten schneiden nicht so schnell ab.

4.2.1 Schlussfolgerung

Die verschiedenen Arbeiten liefern insgesamt ähnliche Resultate, wobei sich BLAKE-32 beim Tempo deutlich von den anderen sicheren Algorithmen abhebt. KECCAK könnte allerdings einfach an die benötigte Hashlänge angepasst werden, was seiner Geschwindigkeit sehr zuträglich wäre.

Kapitel 5

Experimente mit Hashalgorithmen und Zufallsbits

5.1 Versuchsablauf

Im durchgeführten Experiment berechnet ein Java Programm für die Hashfunktionen alle Hashwerte bei 10, 12 und 14 Bit Hashlänge und erzeugt aus ihnen einen Graphen. Diese Graphen bestehen wiederum aus ein oder mehreren schwachen Zusammenhangskomponenten. In [2] wurde eine Zufallshashfunktion verwendet, um die Graphen zu erzeugen. In dieser Arbeit wurde nun das Experiment abgewandelt und Hashfunktionen aus der Praxis verwendet. Das Programm führt automatisch die Hashfunktionen aus und entfernt dann schrittweise alle Knoten, die nicht in einer starken Zusammenhangskomponente liegen. So bleiben nur diejenigen Knoten übrig, die auch nach beliebig vielen Iterationen der Hashfunktionen übrig bleiben. Dazu werden die Zahlen von 0 bis $2^{10} - 1$ (bzw. $2^{12} - 1$ und $2^{14} - 1$) in BitSet umgewandelt und gegebenenfalls ein BitSet mit 1, 2, 3 oder 4 Bits konkateniert. Diese Zusatzbits würden in einer praxisnahen Anwendung durch einen sicheren Zufallszahlengenerator erzeugt werden. Für die Analyse interessiert aber nicht der konkrete Einzelfall, sondern der gesamte Graph der Hashwerte. Daher werden alle möglichen Kombinationen der Zusatzbits verwendet. Die Eingaben haben somit eine Länge von 10 bis 18 Bit. Diese werden ohne Padding an die Hashfunktionen übergeben, da diese selbst Methoden besitzen, um Eingaben auf die nötige Länge zu strecken oder zu kürzen. Das heißt, dass die Zahl als BitSet für jede mögliche Kombination der Zusatzbits kopiert und um diese erweitert wird. Die Knoten des Graphen haben dann 2^z ausgehende Kanten, wobei z die Anzahl der Zusatzbits ist. Jeder Hashwert der Hashfunktion wird auf den definierten Wertebereich reduziert, indem aus dem Hashwert eine Folge von 10, 12 oder 14 Bit ausgeschnitten wird. Die Versuche wurden jeweils mit den Anfangsbits, Bits aus der Mitte und den Endbits der Hashwerte durchgeführt, um auch der Frage nachzugehen, wie die kurzen Hashwerte zu generieren sind. Hashfunktionen liefern üblicherweise Hashwerte der Länge 128 Bit, 256 Bit und mehr. Eine Veränderung der internen Strukturen könnte großen Einfluss auf die Sicherheit der Hashfunktionen haben. Durch den Lawineneffekt sollten sich bei der Änderung eines Bits der Eingabe der Hashfunktion mindestens 50% der Bits der Ausgabe ändern, wodurch es möglich sein sollte, nur einen Ausschnitt der Ausgabe zu verwenden. In der ersten Runde werden alle Blätter, also Knoten ohne eingehende Kante, mit ihren ausgehenden Kanten aus dem Graphen entfernt. Dieser Prozess wird in jeder Runde wiederholt. Nach x Run-

den bleiben nur Knoten übrig, die sich in starken Zusammenhangskomponenten befinden. Im folgenden werden die Graphen der Hashalgorithmen auf Eigenschaften, die einen Einfluss auf die Sicherheit haben, hin überprüft. Die größte Rolle spielt die Anzahl der Knoten in starken Zusammenhangskomponenten. Weiterhin können die benötigte Rundenzahl x, um alle Knoten außerhalb der starken Zusammenhangskomponenten zu entfernen, sowie die Anzahl und Größe dieser interessant sein.

Ein zweites Javaprogramm führt eine Vorauswertung durch und gibt die starken Zusammenhangskomponenten, ihre Größe und die entfernten Knoten aus. Hier wird der Algorithmus von Tarjan zur Bestimmung starker Zusammenhangskomponenten (siehe [59, S.155 ff.]) verwendet. Dieses Programm benötigt zum Analysieren der Daten viel Speicher, da zum einem jeder Knoten als Objekt existiert und die Rekursionstiefe des Algorithmus von Tarjan gleich der Anzahl der Knoten in der untersuchten starken Zusammenhangskomponente ist. Für die Laufzeit des Algorithmus selbst gilt: $O(\#V + \#E)$ wobei $\#V$ die Anzahl der Knoten und $\#E$ die Anzahl der Kanten ist. Die Vorauswertung wird im Comma Serparated Values Format gespeichert. Eine tabellarische Darstellung der Daten ist in Tabelle A.1, Tabelle A.2 und Tabelle A.3 zu sehen.

Die weitere Verarbeitung der Daten zu Tabellen bzw. Graphen wird mittels LuaLaTeX, eine erweiterte Version von LaTeX, die Luacode ausführen kann, durchgeführt. Dies ermöglicht es, bei Aktualisierung des Datensatzes gleichzeitig die abgeleiteten Werte zu aktualisieren. Außerdem ist es relativ einfach, weitere Daten abzuleiten, falls es dafür Bedarf gibt. Kleinere Tabellen werden im Text eingebettet. Längere befinden sich in Anhang A.

5.2 Resultate

Die Resultate der durchgeführten Versuchsreihen sind auf den folgenden Seiten zusammengefasst. In Tabelle A.1, Tabelle A.2 und Tabelle A.3 sind die generierten Daten im Überblick zu sehen.

Auf die einzelnen Bedeutungen der Resultate wird in den folgenden Abschnitten eingegangen. Dabei wird zunächst geklärt, welchen Einfluss die Art der Gewinnung der Bits aus den Hashwerten hat.

5.2.1 Einfluss der verwendeten Bits

Anzahl der Knoten in starken Zusammenhangskomponenten

Zunächst betrachten wir den Zusammenhang zwischen den gewählten Bits und der Anzahl der Knoten in starken Zusammenhangskomponenten. Es sollten sich möglichst viele Knoten in diesen Komponenten befinden. In Tabelle 5.1 ist die jeweilige durchschnittliche Anzahl der Knoten in starken Zusammenhangskomponenten dargestellt. Außerdem werden die Werte der einzelnen Hashfunktionen in Abbildung 5.1, Abbildung 5.2 und Abbildung 5.3 dargestellt. Generell schwanken die Ergebnisse bei jeder Hashfunktion, so dass es bei keiner Hashfunktion eine klare Empfehlung gibt, wie die gekürzten Hashwerte erzeugt werden sollten.

Allerdings lässt sich folgendes feststellen: Bei 10 Bit Haslänge befinden sich im Durchschnitt mehr Knoten in starken Zusammenhangskomponenten, wenn die Bits aus der Mitte extrahiert werden, als wenn sie am Anfang oder am Ende entnommen werden. Bei 12 Bit liegen die Werte der Bits aus der Mitte und die der Anfangsbits im Schnitt fast gleich auf. Bei 14 Bit liegt die

Hashlänge in Bit	Anzahl der Zusatzbits	Anzahl der Knoten in starken Zusammenhangskomponenten mit Bits vom Anfang	aus der Mitte	vom Ende
10	0	40,56	50,89	26,22
10	1	703,56	690,89	694,44
10	2	952,22	947,44	954,56
10	3	1 019,33	1 018,78	1 020,44
10	4	1 023,89	1 023,89	1 024,00
12	0	102,89	101,67	87,78
12	1	2 778,22	2 761,78	2 775,89
12	2	3 808,78	3 797,22	3 813,67
12	3	4 077,00	4 077,89	4 078,56
12	4	4 096,00	4 096,00	4 095,78
14	0	136,22	155,11	131,56
14	1	11 052,89	11 063,67	11 070,33
14	2	15 189,78	15 193,89	15 210,56
14	3	16 306,89	16 308,56	16 307,67
14	4	16 383,44	16 383,56	16 383,67

Tabelle 5.1: Durchschnittliche Anzahl der Knoten in starken Zusammenhangskomponenten.

durchschnittliche Anzahl der Knoten in starken Zusammenhangskomponenten bei der Verwendung von Bits aus der Mitte wieder deutlich über der der anderen beiden Methoden.

Bei der Verwendung von Zusatzbits ändert sich das Bild. Die durchschnittliche Anzahl der Knoten in starken Zusammenhangskomponenten liegt bei den verschiedenen Hashfunktionen und bei den verschieden Arten zur Erzeugung der kurzen Hashwerte nahe beieinander. Um so mehr Zusatzbits verwendet werden, desto näher.

In Abbildung 5.4 und Abbildung 5.5 ist dies beispielhaft für die Werte der Hashwerte mit der Länge 10 Bit und ein bzw. zwei Zusatzbits zu sehen.

Es scheint insgesamt am sinnvollsten zu sein, die Reduktion so durchzuführen, dass die Bits aus der Mitte der Hashwerte verwendet werden. Im Schnitt erhält man dadurch etwas größere starke Zusammenhangskomponenten. Falls zusätzlich Zufallsbits in die Hashwerte eingehen, spielt die Art der Reduktion nur eine untergeordnete Rolle bzw. ist diese für die Anzahl der Knoten in starken Zusammenhangskomponenten irrelevant.

Starke Zusammenhangskomponenten

Aus den Daten wurde die durchschnittliche Größe der starken Zusammenhangskomponenten berechnet und in Tabelle A.4 zusammengefasst. Generell sind größere Komponenten von Vorteil, da die Hashwerte einer Hashkette ab einem bestimmten Kettenglied nur noch aus einer dieser Komponenten stammen können bzw. jeder Pfad im Graphen nach einer endlichen Zahl an Schritten in einer starken Zusammenhangskomponente endet. Dieser sollte daher möglichst groß sein.

Im Durchschnitt sind die starken Zusammenhangskomponenten in den Graphen größer,

Hashlänge in Bit	Anzahl der Zusatzbits	Durchschnittliche Größe der starken Zusammenhangskomponenten bei Bits vom Anfang	aus der Mitte	vom Ende
10	0	7,84	14,92	7,38
10	1	516,04	536,37	540,33
10	2	952,22	895,17	875,89
10	3	1 019,33	1 018,78	1 020,44
10	4	1 023,89	1 023,89	1 024,00
12	0	23,89	21,61	17,10
12	1	2 240,28	1 989,89	2 466,17
12	2	3 597,00	3 376,67	3 601,28
12	3	4 077,00	4 077,89	4 078,56
12	4	4 096,00	4 096,00	4 095,78
14	0	25,63	31,79	20,93
14	1	7 168,92	6 421,55	6 765,59
14	2	15 189,78	14 350,33	15 210,56
14	3	16 306,89	16 308,56	16 307,67
14	4	16 383,44	16 383,56	16 383,67

Tabelle 5.2: Durchschnittliche Größe starken Zusammenhangskomponenten der Hashfunktionen

wenn die Bits aus der Mitte extrahiert werden. Bei 10 Bit beträgt der Vorsprung fast 100%, bei 14 Bit noch knapp 20%. Bei 12 Bit sind die starken Zusammenhangskomponenten im Schnitt größer, wenn die Bits vom Anfang der Hashwerte genommen werden. Dies ist auf die gesamte Anzahl an Knoten in den starken Zusammenhangskomponenten zurückzuführen, die ja gerade die Gesamtgröße der starken Zusammenhangskomponenten ist. In Tabelle 5.1 aus Abschnitt 5.2.1 ist zu sehen, dass auch die Anzahl der Knoten in starken Zusammenhangskomponenten der Hashfunktionen bei 12 Bit bei Verwendung der Anfangsbits im Schnitt etwas höher liegt, als bei den Bits aus der Mitte. Dies schlägt sich dann auch in der durchschnittlichen Größe der starken Zusammenhangskomponenten nieder.

Aus den Daten ist aber auch ersichtlich, dass pro Graph einige sehr kleine starke Zusammenhanhskomponenten und einige große entstehen. Daher wäre es sinnvoll, die starkren Zusammenhangskomponenten mit der Wahscheinlichkeit, dass eine Hashkette in ihm endet, zu gewichten. Dies würde aber den Rahmen dieser Arbeit überschreiten.

Runden

Die Rundenzahl gibt Auskunft darüber, wie lange es dauert, alle Knoten, die nicht in starken Zusammenhangskomponenten liegen, aus dem Graphen zu entfernen. Bis alle anderen Knoten entfernt wurden, stehen diese noch für die Hashkette zur Verfügung. Es ist daher vorteilhaft, wenn die Rundenzahl höher liegt. Allerdings ist dieser Effekt eher gering, da die Anzahl dieser Knoten im Vergleich zur Größe des Graphen gering ist. In Tabelle 5.3 sind die durchschnittlichen Rundenzahlen der Hashfunktionen aufgelistet.

Keine der Reduktionsarten scheint gegenüber den anderen eine Verbesserung darzustellen.

Hashlänge in Bit	Anzahl der Zusatzbits	Anzahl der Runden mit Bits vom Anfang	aus der Mitte	vom Ende
10	0	46,56	50,67	63,33
10	1	7,89	8,11	8,33
10	2	3,67	3,78	3,33
10	3	2,00	2,22	2,11
10	4	2,00	2,00	2,00
12	0	114,00	104,22	102,33
12	1	8,67	9,00	9,00
12	2	4,44	4,00	4,22
12	3	2,44	2,33	2,22
12	4	2,00	2,00	2,00
14	0	238,67	217,00	255,56
14	1	10,22	10,89	10,22
14	2	5,44	5,00	5,22
14	3	2,89	2,78	2,33
14	4	2,00	2,00	2,00

Tabelle 5.3: Durchschnittliche Rundenanzahl der Hashfunktionen

Die Rangfolge der Reduktionsarten wechselt bei den unterschiedlichen Hashlängen und die Ergebnisse liegen in ähnlichen Wertebereichen.

Die Rundenzahlen der Hashfunktionen mit Zusatzbits liegen so nahe beieinander, dass es keine nennenswerten Unterschiede zwischen den verschiedenen Methoden zur Gewinnung der Bits gibt.

Zusammenfassung

Wie in den letzten Abschnitten gezeigt wurde, ist die Reduktion durch Verwendung der Bits aus der Mitte der Hashwerte am besten geeignet, da hier etwas mehr Knoten in starken Zusammenhangskomponente liegen, als in den beiden anderen Reduktionsarten. Dies spricht dafür, dass die Bits aus der Mitte mehr einer zufälligen Verteilung ähneln als die Anfangs- und Endbits. In den folgenden Abschnitten werden die Untersuchungen daher auf die Resultate dieser Reduktionsart beschränkt.

5.2.2 Hashalgorithmus

Im nun folgenden Abschnitt wir die Frage geklärt, ob es zwischen den untersuchten Hashalgorithmen signifikante Unterschiede in den Graphen gibt und welcher Art diese sind.

Anzahl der Knoten in starken Zusammenhangskomponenten

Zunächst wird wieder der wichtigste Faktor, die Anzahl der Knoten in starken Zusammenhangskomponenten, betrachtet. In Abbildung 5.6 sind die Werte der Hashalgorithmen bei 10 Bit Länge

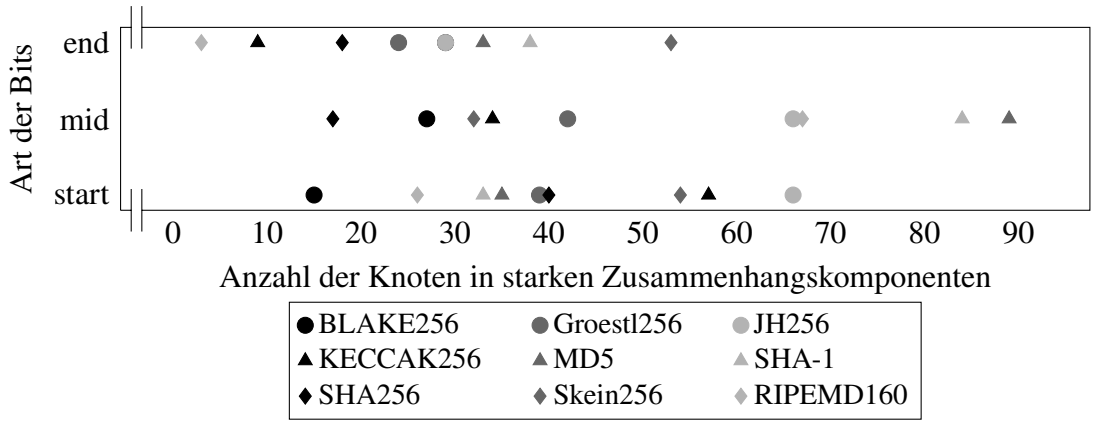

Abbildung 5.1: Vergleich der Anzahl der Knoten in starken Zusammenhangskomponenten in den Graphen der Hashfunktionen bei 10 Bit Hashlänge ohne Zusatzbits bei Erzeugung der Hashwerte aus den Bits vom Anfang, aus der Mitte und vom Ende.

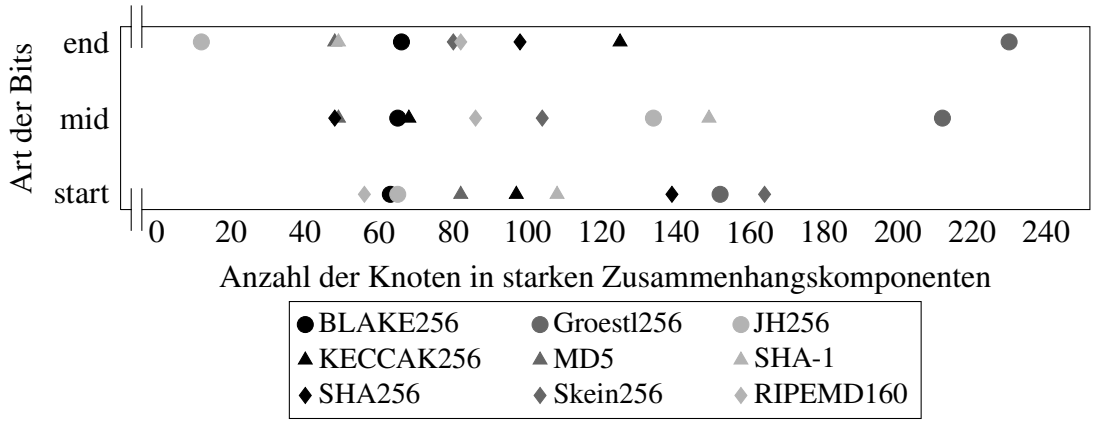

Abbildung 5.2: Vergleich der Anzahl der Knoten in starken Zusammenhangskomponenten in den Graphen der Hashfunktionen bei 12 Bit Hashlänge ohne Zusatzbits bei Erzeugung der Hashwerte aus den Bits vom Anfang, aus der Mitte und vom Ende.

im Vergleich zu sehen. Auffällig ist, dass die Werte bei drei und vier Zusatzbits bei allen Hashalgorithmen fast identisch sind. Hierauf wird in Unterabschnitt 5.2.3 näher eingegangen. Wie in Abbildung 5.7 und Abbildung 5.8 zu sehen ist, tritt dieser Effekt auch bei den Hashlängen 12 und 14 Bit auf. Daher wird hier die Betrachtung zunächst auf die Ergebnisse mit keinen oder wenigen Zusatzbits eingeschränkt.

In Abbildung 5.9 sind die Resultate für die Hashfunktionen ohne Zusatzbits dargestellt. Bei vielen Hashalgorithmen gibt es bei längeren Hashwerten eine größere Anzahl von Knoten in starken Zusammenhangskomponenten. Allerdings tritt bei manchen der gegenteilige Effekt auf. Trotz längerer Hashwerte fällt die Gesamtgröße der starken Zusammenhangskomponenten geringer aus. Diese Verkleinerungen sind wahrscheinlich zufälliger Natur. So ist es möglich, dass trotz längerer Hashwerte mehr Kollisionen auftreten und so weniger Knoten in starken Zusammenhangskomponenten liegen. Zu beachten ist hier, dass nicht automatisch alle Knoten, die bei kürzerer Hashlänge in einer solchen Komponente lagen nicht auch bei längeren Hashwerten in diesen liegen. Auffällig ist bei den getesteten Hashfunktionen, dass die Steigerungen von 10 auf 12 bzw. von 12 auf 14 Bit teils deutlich größer ausfallen, als die erwartete Vervierfachung. Eine

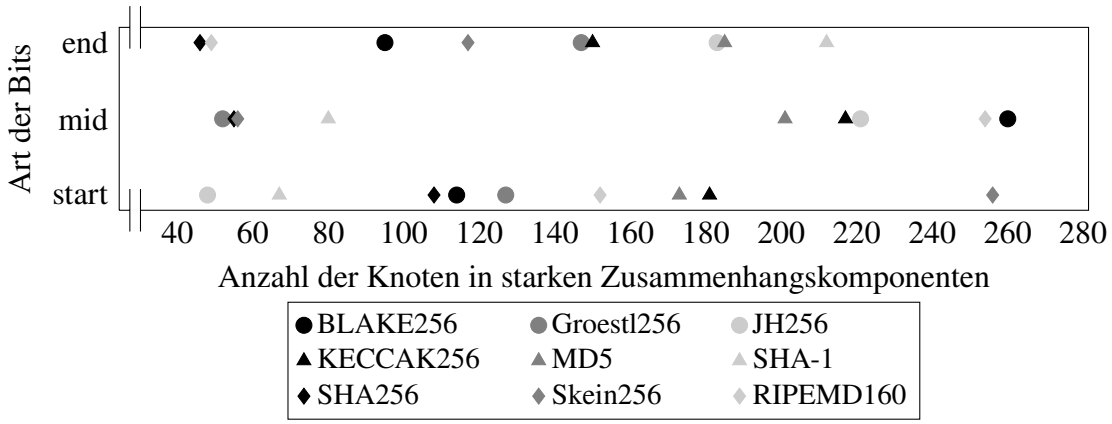

Abbildung 5.3: Vergleich der Anzahl der Knoten in starken Zusammenhangskomponenten in den Graphen der Hashfunktionen bei 14 Bit Hashlänge ohne Zusatzbits bei Erzeugung der Hashwerte aus den Bits vom Anfang, aus der Mitte und vom Ende.

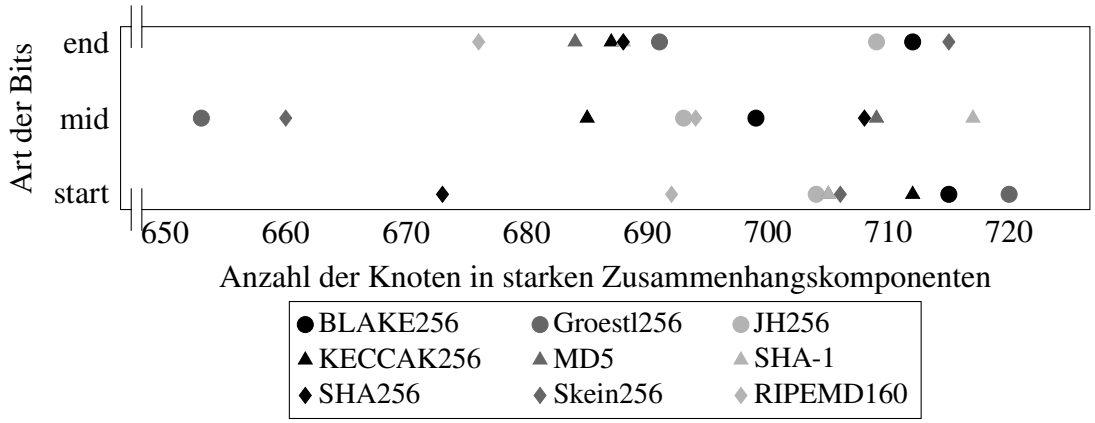

Abbildung 5.4: Vergleich der Anzahl der Knoten in starken Zusammenhangskomponenten in den Graphen der Hashfunktionen bei 10 Bit Hashlänge mit 1 Zusatzbit bei Erzeugung der Hashwerte aus den Bits vom Anfang, aus der Mitte und vom Ende.

Erklärung hierfür könnten zum einen zufällige Effekte sein, so wie sie bei der oben angesprochenen Verkleinerung Ursache sind, und zum anderen könnte ein größeres Teilstück der Hashwerte qualitativ bessere Bits enthalten, als das kürzere. Es ist nicht garantiert, dass alle Teilstücke der Hashwerte gleiche Qualität aufweisen. Wie in Unterabschnitt 5.2.1 untersucht wurde, gibt es zwischen den Teilstücken durchaus messbare Unterschiede. In Tabelle 5.4 wurde die durchschnittliche Anzahl der Knoten in starken Zusammenhangskomponenten für 10, 12 und 14 Bit der Hashalgorithmen ohne Zusatzbits berechnet, da bereits oben erläutert wurde, dass einzelne Werte Schwankungen unterliegen. MD5, JH256 und RIPEMD160 bilden dabei die Spitze im Vergleich der Hashalgorithmen. Am Schluss des Feldes liegt mit großem Abstand SHA256. SHA-1m Skein256, BLAKE256, KECCAK256 und Grøstl256 können sich im Mittelfeld behaupten.

Abbildung 5.10 zeigt die Resultate für Hashfunktionen mit einem Zusatzbit. Die starken Zusammenhangskomponenten sind bereits deutlich größer, als bei den Hashfunktionen ohne Zusatzbits. Die Werte liegen aber auch enger beieinander. Bei allen Hashfunktionen steigt die Größe der starken Zusammenhangskomponenten mit der Länge der Hashwerte an.

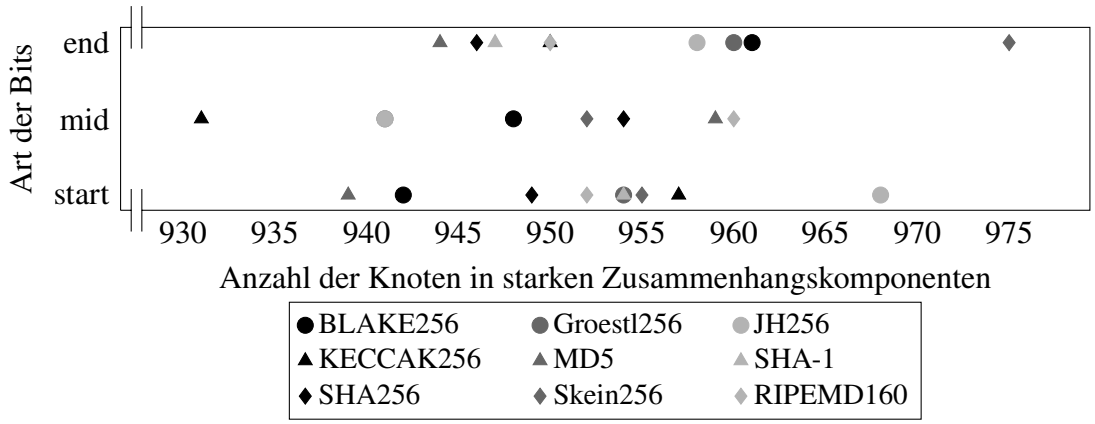

Abbildung 5.5: Vergleich der Anzahl der Knoten in starken Zusammenhangskomponenten in den Graphen der Hashfunktionen bei 10 Bit Hashlänge 2 Zusatzbits bei Erzeugung der Hashwerte aus den Bits vom Anfang, aus der Mitte und vom Ende.

Wie in Abbildung 5.11 zu sehen ist, gilt dies auch für weitere Zusatzbits. Mit größerer Hashlänge nimmt auch die Größe der starken Zusammenhangskomponenten zu. Die Ergebnisse der einzelnen Hashfunktionen liegen bei 2 zusätzlichen Zufallsbits noch näher zusammen, als bei einem. Bei 3 und 4 Zusatzbits sind sie dann nahezu identisch, weswegen hier keine weitere Analyse erfolgt. In Unterabschnitt 5.2.3 wird nochmals ausführlicher darauf eingegangen werden.

Zusammenfassung

Es scheint bei den Hashalgorithmen keine größeren Unterschiede zu geben. Die aufgetretenen Schwankungen bewegen sich im Rahmen des Zufalls. Für genauere Aussagen müssten deutlich mehr Daten erhoben und ausgewertet werden, um die zufälligen Effekte zu eliminieren bzw. zumindest zu minimieren. Wären allerdings deutliche Unterschiede zwischen den Algorithmen aufgetreten, wären auch Unterschiede in der Häufigkeit der Kollision der Algorithmen messbar, was nicht der Fall ist. Welcher Algorithmus nun der beste für die Implementierung einer Hashkette ist, hängt hauptsächlich von den Faktoren der Skalierbarkeit, Geschwindigkeit und Resourcenverbauch ab. KECCAK könnte hier als SHA3 Sieger und durch seine hohe Skalierbarkeit sehr interessant sein. Ansonsten hat der Geschwindigkeitsvergleich in Abschnitt 4.2 gezeigt, dass BLAKE ein interessanter Kandidat für derartige Aufgaben ist.

5.2.3 Zusatzbits

Der letzte untersuchte Faktor ist der Einfluss der Anzahl der Zusatzbits auf die Hashketten bzw. Graphen der Hashfunktionen. Einige Resultate wurden bereits kurz angerissen und werden nun auf den folgenden Seiten vertieft.

Anzahl der Knoten in starken Zusammenhangskomponenten

Zuerst wird wieder der Einfluss auf die Anzahl der Knoten in starken Zusammenhangskomponenten untersucht. In Abbildung 5.6, Abbildung 5.7 und Abbildung 5.8 wurden die Ergebnisse bereits mit anderem Fokus in Unterabschnitt 5.2.2 dargestellt. In Abbildung 5.12, Abbil-

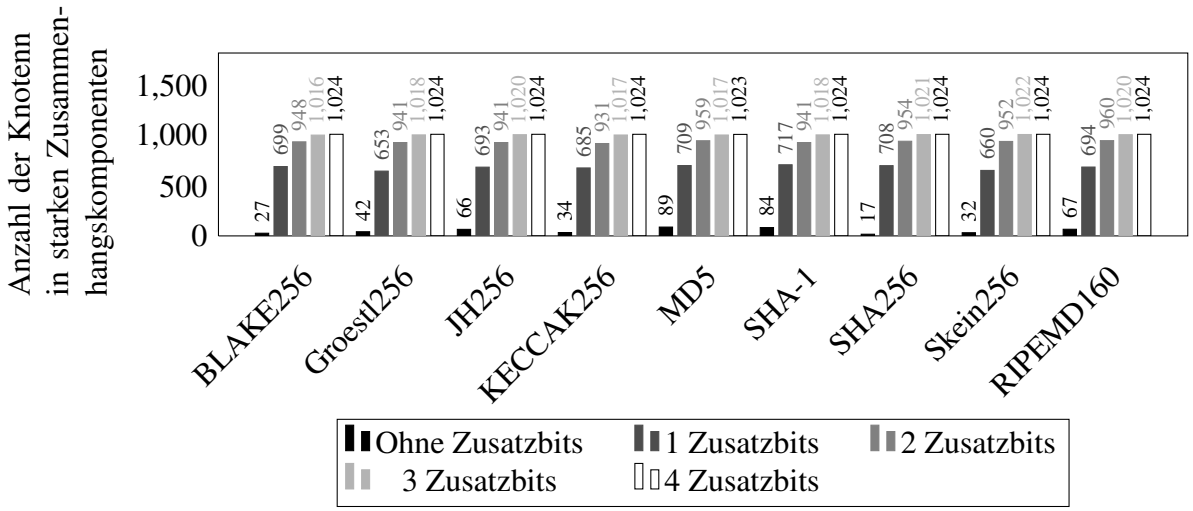

Abbildung 5.6: Vergleich der Anzahl der Knoten in starken Zusammenhangskomponenten der Hashfunktionen bei 10 Bit aus der Mitte der Hashes

Hashalgorithmus	Anzahl der Knoten in starken Zusammenhangskomponenten
BLAKE256	27,00
Groestl256	42,00
JH256	66,00
KECCAK256	34,00
MD5	89,00
SHA-1	50,50
SHA256	17,00
Skein256	32,00
RIPEMD160	67,00

Tabelle 5.4: Durchschnittliche Anzahl der Knoten in starken Zusammenhangskomponenten in den Graphen der Hashalgorithmen mit Hashlänge 10 Bit ohne Zusatbits

dung 5.13 und Abbildung 5.14 ist die Anzahl der Knoten in starken Zusammenhangskomponenten bei 10, 12 und 14 Bit nochmals in Abhängigkeit zu der Anzahl der verwendeten Zusatzbits dargestellt.

Wie man deutlich erkennen kann, nimmt die Gesamtgröße der starken Zusammenhangskomponenten mit dem ersten Zusatzbit sehr stark zu. Die Zunahme wird mit jedem weiteren Zusatzbit geringer. Bei vier Zusatzbits ist das Maximum fast immer erreicht, das heißt der gesamte Graph besteht aus einer starken Zusammenhangskomponente. Eine weitere Steigerung ist daher nicht mehr möglich. Die Verwendung weiterer Zusatzbits ist daher nahezu sinnlos. Bei einigen Kombinationen wäre noch eine minimale Steigerung möglich, da noch ein Knoten außerhalb der starken Zusammenhangskomponente liegt. Dies ist aber zu vernachlässigen.

Zunächst die Betrachtung der Bäume bei 10 Bit Hashlänge: Bei drei Zusatzbits liegen bereits gut 99,22% der Knoten in starken Zusammenhangskomponenten. Diese Werte sind bei allen Hashfunktionen fast identisch. Bei zwei Zusatzbits liegen zwischen 90,92% und 93,75%

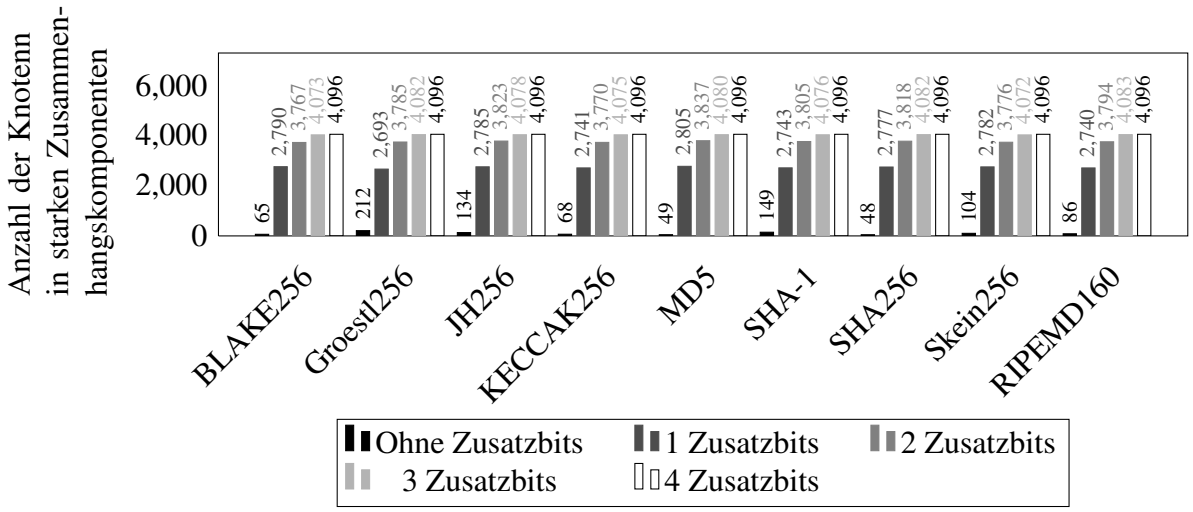

Abbildung 5.7: Vergleich der Anzahl der Knoten in starken Zusammenhangskomponenten der Hashfunktionen bei 12 Bit aus der Mitte der Hashes.

der Knoten in starken Zusammenhangskomponenten. Die Abweichungen zwischen den einzelnen Hashalgorithmen sind weiterhin sehr gering. Bei einem Zusatzbits enthalten die starken Zusammenhangskomponenten zwischen 63,77% (Groestl256) und 70,02% (SHA-1) der Knoten des gesamten Graphen. Demgegenüber liegen bei Hashalgorithmen ohne Zusatzbits zwischen 1,66% (SHA256) und 8,69% (MD5) der Knoten des Graphen in starken Zusammenhangskomponenten.

Nun kommen wir zu 12 Bit Hashlänge: Bei drei Zusatzbits enthalten die starken Zusammenhangskomponenten bereits 99,41% der Knoten des Graphen. Diese Werte sind wieder bei allen Hashfunktionen fast identisch. Bei zwei Zusatzbits enthalten die starken Zusammenhangskomponenten zwischen 91,97% und 93,68% der Knoten. Die Abweichungen zwischen den einzelnen Hashalgorithmen ist weiterhin sehr gering. Bei einem Zusatzbit befinden sich zwischen 65,75% (Groestl256) und 68,48% (MD5) der Knoten des Graphen innerhalb der starken Zusammenhangskomponenten. Die starken Zusammenhangskomponenten der Graphen der Hashalgorithmen ohne Zusatzbits enthalten nur 1,17% (SHA256) bis 5,18% (Groestl256) der Knoten.

Als letztes betrachten wir wieder Hashes mit 14 Bit Länge: Bei drei Zusatzbits enthalten die starken Zusammenhangskomponenten des Graphen 99,47% der gesamten Knoten. Diese Werte sind wieder bei allen Hashfunktionen fast identisch. Bei zwei Zusatzbits enthalten die starken Zusammenhangskomponenten zwischen 92,27% und 93,10% der Knoten des Graphen. Die Abweichungen zwischen den einzelnen Hashalgorithmen sind weiterhin sehr gering. Bei einem Zusatzbit sind zwischen 67,12% (Skein256) und 68,10% (RIPEMD160) der Knoten des Graphen in starken Zusammenhangskomponenten enthalten. Bei Hashalgorithmen ohne Zusatzbits lediglich zwischen 0,32% (Groestl256) und 1,59% (BLAKE256).

Zusammenfassung

Bei den Daten fallen zwei Dinge auf. Zum einem ist die Vergrößerung der starken Zusammenhangskomponenten durch Verwendung von Zusatzbits bei allen Hashlängen gleich. Bei 4 Zusatzbits liegt der Anteil der Knoten, die sich in starken Zusammenhangskomponenten befinden,

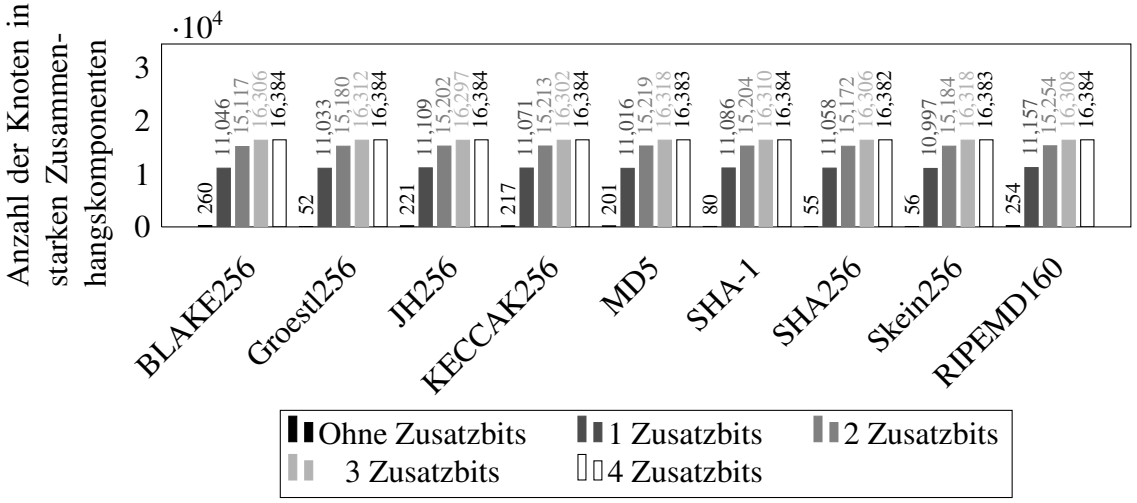

Abbildung 5.8: Vergleich der Anzahl der Knoten in starken Zusammenhangskomponenten der Hashfunktionen bei 14 Bit aus der Mitte der Hashes.

bei fast 100%. Bei 3 Zusatzbits liegt er bei gut 99%. Bei 2 Zusatzbits zwischen 91% und 94% und bei 1 immer noch bei 64% bis 70%. Dies ist bis auf kleine Schwankungen bei allen untersuchten Hashlängen der Fall.

Zum anderen fällt auf, das der prozentuale Anteil der Knoten in starken Zusammenhangskomponenten bei den Hashketten ohne Zusatzbits mit der Hashlänge abnimmt. Bei 10 Bit Hashlänge sind es noch zwischen $1,66\%$ und $8,69\%$. Bei 12 Bit nur noch $1,17\%$ bis $5,18\%$ und bei 14 Bit schließlich nur noch zwischen $0,32\%$ und $1,59\%$. Dieser Trend setzt sich mit zunehmender Hashlänge vermutlich fort. Dies bedeutet, dass die tatsächlich genutzte Teilmenge der Wertemenge nicht im gleichen Maße wächst, wie die Wertemenge selbst.

Die Verwendung von zufälligen Zusatzbits ist daher nicht nur sinnvoll, um den Anteil der Knoten in starken Zusammenhangskomponenten zu vergrößern, sondern auch, um zu verhindern, dass dieser mit zunehmender Hashlänge abnimmt.

1 Zusatzbit bietet die effizienteste Möglichkeit, diese beiden Ziele zu erreichen. Mehr Zusatzbits machen kaum Sinn, da sie den prozentualen Anteil nicht mehr verdoppeln. Stattdessen würden längere Hashwerte die Gesamtzahl der Knoten in starken Zusammenhangskomponenten mehr erhöhen. Bei der Abwägung zwischen 0 und einem Zusatzbit zeigt sich, dass bei Verwendung von einem Zusatzbit und einem Bit längeren Hashwerten $0,6\#V$ statt nur $0,08 * 2\#V = 0,16\#V$ Knoten in starken Zusammenhanskomponenten liegen, wobei $\#V$ die Gesamtzahl der Knoten ist. Bereits bei dem Vergleich von zwei Zusatzbits $(0,9\#V)$ und einem Zusatzbit mit einem 1 Bit längerem Hashwert ist letzteres mit $0,6 * 2 * \#V = 1,2\#V$ Knoten effizienter.

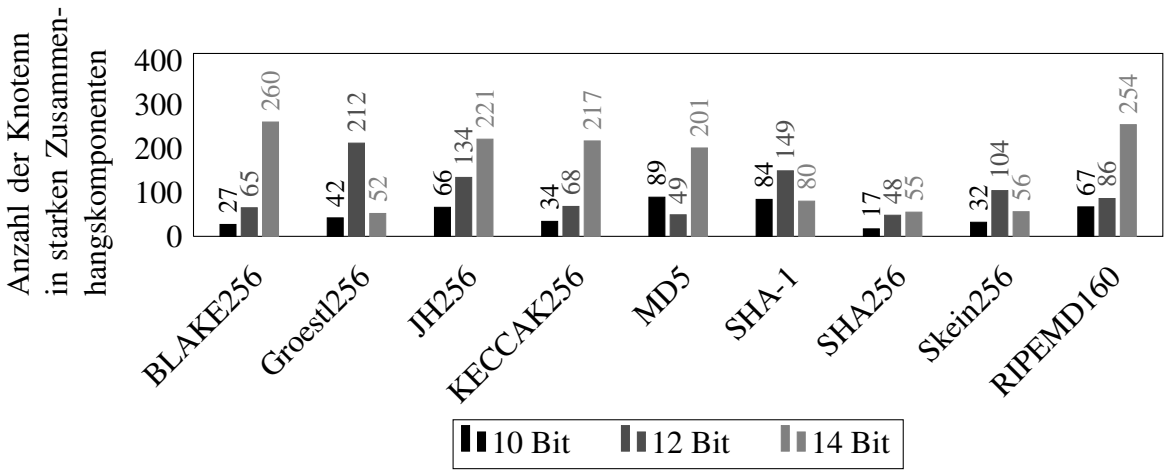

Abbildung 5.9: Vergleich der Anzahl der Knoten in starken Zusammenhangskomponenten der Hashfunktionen ohne Zusatzbits.

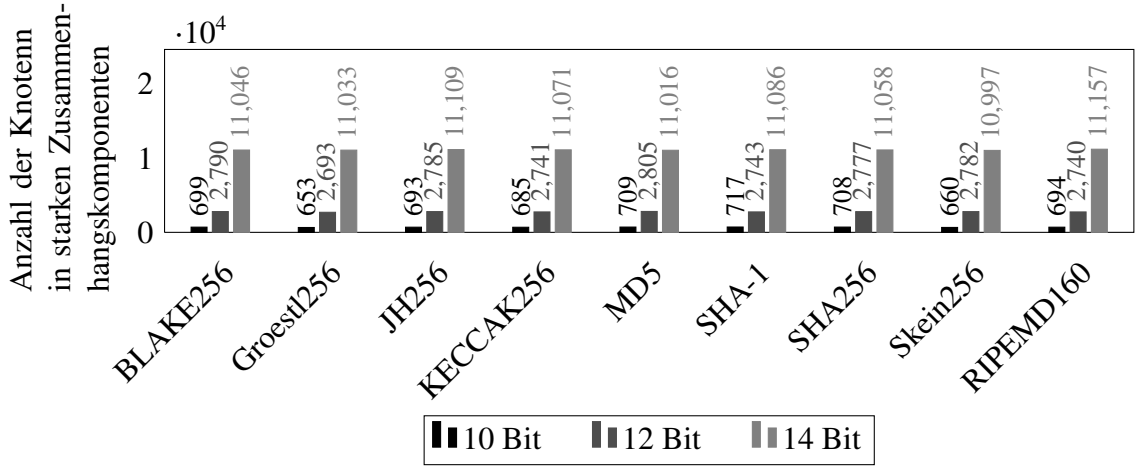

Abbildung 5.10: Vergleich der Anzahl der Knoten in starken Zusammenhangskomponenten der Hashfunktionen mit einem Zusatzbit.

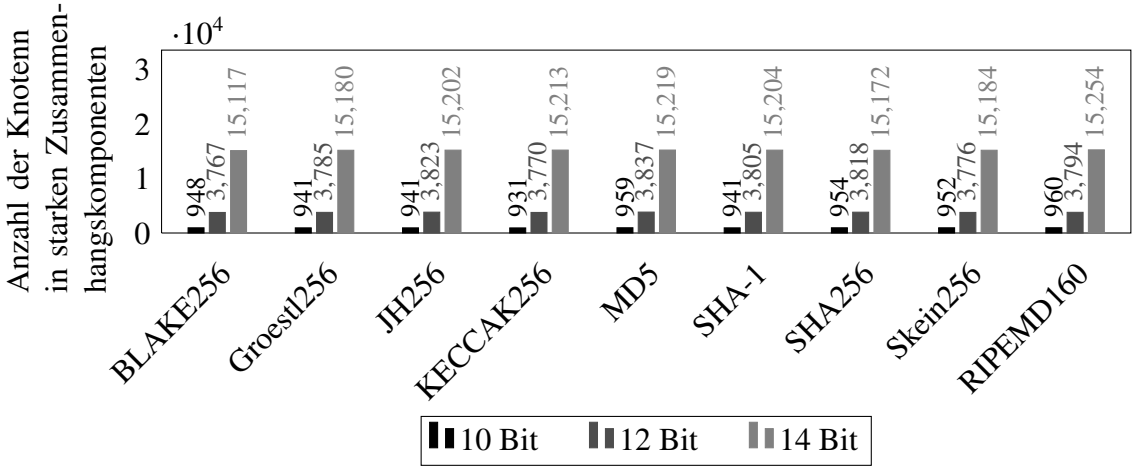

Abbildung 5.11: Vergleich der Anzahl der Knoten in starken Zusammenhangskomponenten der Hashfunktionen mit zwei Zusatzbits.

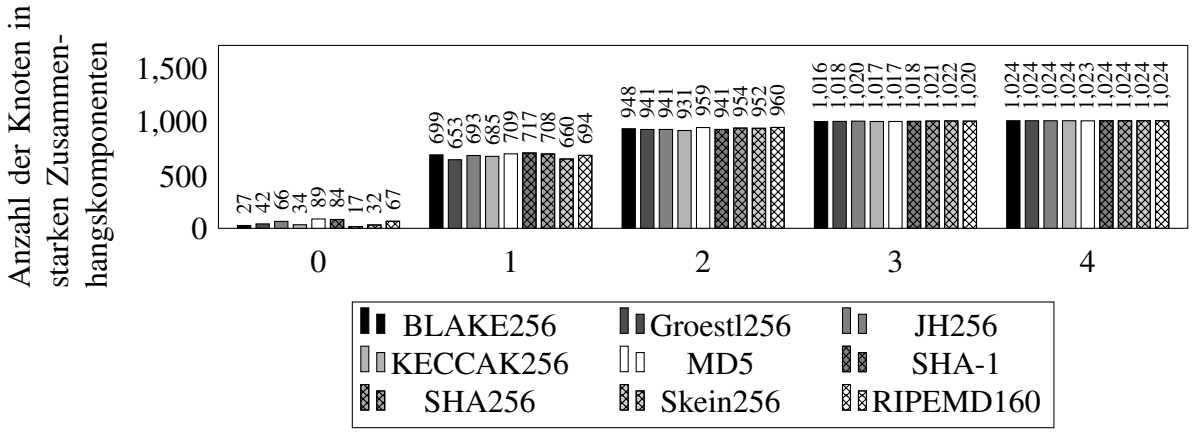

Abbildung 5.12: Einfluss der Zusatzbits auf die Anzahl der Knoten in starken Zusammenhangskomponenten der Hashfunktionen bei 10 Bit aus der Mitte der Hashes.

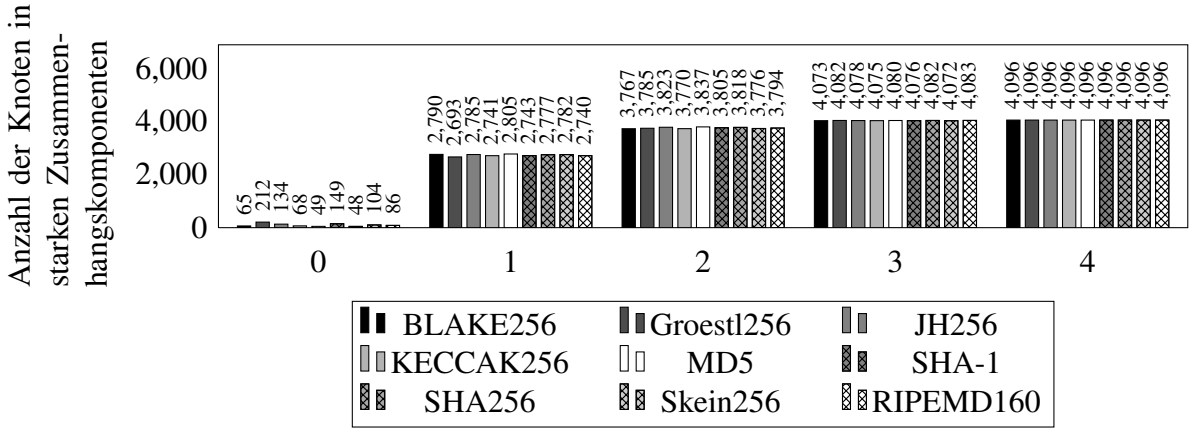

Abbildung 5.13: Einfluss der Zusatzbits auf die Anzahl der Knoten in starken Zusammenhangskomponenten der Hashfunktionen bei 12 Bit aus der Mitte der Hashes.

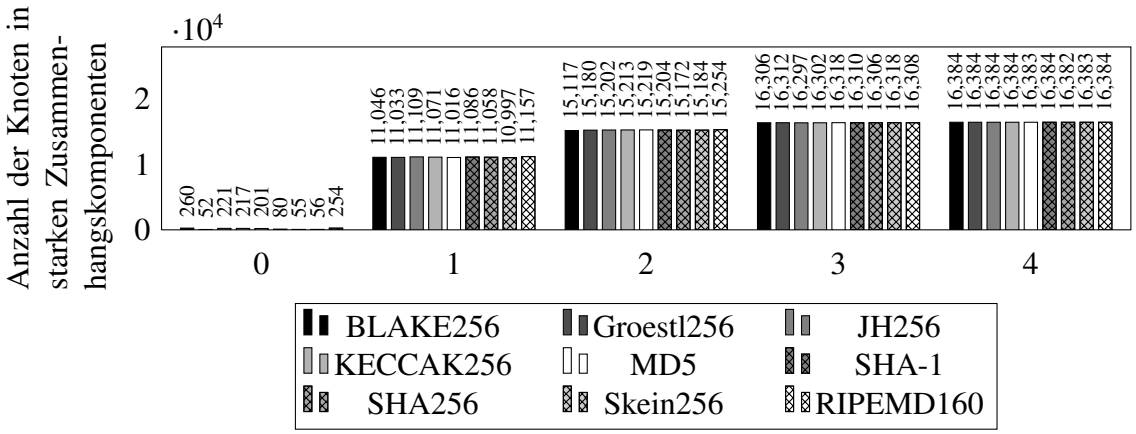

Abbildung 5.14: Einfluss der Zusatzbits auf die Anzahl der Knoten in starken Zusammenhangskomponenten der Hashfunktionen bei 14 Bit aus der Mitte der Hashes.

Kapitel 6

Zusammenfassung

6.1 Empfehlung

In den letzten Abschnitten wurden im Detail auf den Einfluss der Reduktion der Hashwerte auf kürzere Hashwerte, den Hashalgorithmen selbst und die Anzahl der zufälligen Zusatzbits eingegangen. Wie bereits in Unterabschnitt 5.2.1 erklärt, ist es am sinnvollsten die Bits aus der Mitte der Hashwerte auszuschneiden. Allerdings bietet z.B. KECCAK die Möglichkeit, die Funktion so anzupassen, dass sie Hashwerte der gewünschten Länge erzeugt. Ein nachträgliches Kürzen der Hashwerte ist daher nicht mehr nötig. Eine derartige Lösung bietet den Vorteil, dass sie bereits gut untersucht ist und die Qualität der Bits wahrscheinlich höher ist, als die von Teilabschnitten der langen Hashwerte. Da keiner der Graphen der Hashalgorithmen einen signifikanten Unterschied zu den der anderen Algorithmen zeigt (siehe Unterabschnitt 5.2.2), spricht vieles dafür, KECCAK einzusetzen. In der Ankündigung des Gewinners des SHA-3 Wettbewerbs (siehe [32]) wurden die folgenden Punkte hervorgehoben:

- Elegantes Design
- Große Sicherheitsreserven
- Eine gute Gesamtperformance
- Hohe Hardwareeffizienz
- Hohe Flexibilität

Vorallem durch seine hohe Hardwareeffitienz und Flexibilität eignet sich KECCAK gut für die Anwendung in Hashketten, die meist in Sensornetzen und anderen leistungsschwachen Geräten zum Einsatz kommen.

Die Verwendung von zufallsgenerierten Zusatzbits ist sehr empfelenswert. Die starken Zusammenhangskomponenten der Graphen werden damit deutlich größer und verbessern damit auch die Sicherheit der Hashketten (siehe Unterabschnitt 5.2.3). Hier hat sich gezeigt, dass die Verwendung von einem Zusatzbit am sinnvollsten ist und darüber hinaus besser die länge der Hashwerte vergrößert werden sollte.

6.2 Alternative

Es besteht allerdings auch die Möglichkeit, die Verwendung von Zusatzbits zu verzichten und stattdessen die Hashkette rechtzeitig zu reinitialisieren. Rechtzeitig bedeutet hierbei, dass die Teilmenge der Hashwerte, die tatsächlich eingenommen werden noch ausreichend — also den

individuellen Sicherheitsanforderungen entsprechend — groß ist. Hierzu müsste eine Analyse des Graphen der gewünschten Hashfunktion und Hashlänge erfolgen, um den geeignetsten Zeitpunkt zu ermitteln.

6.3 Offene Fragen

Im Laufe der Arbeit wurden einige Fragen aufgeworfen, die nicht mehr beantwortet werden können und Platz für weitere Forschung lassen. Die wichtigsten Fragen sind hier noch einmal zusammengefasst:

- Einige Hashfunktionen (z.B. Grøstl mit Random Hashing) bieten die Möglichkeit, die Hashfunktion zu parametrisieren. Es stellt sich die Frage, ob es Unterschiede in der Qualität und in der Ausführungsgeschwindigkeit gibt, wenn die Zufallsbits nicht zusammen mit dem letzten Hashwert die Eingabe der Hashfunktion bilden, sondern als Parameter an die Hashfunktion übergeben werden.
- Mit leistungstärkeren Systemen könnten Daten von längeren Hashwerten erzeugt und analysiert werden.
- Die Strukturen der Graphen kann genauer analysiert werden. Z.B. kann die Wahrscheinlichkeit berechnet werden, mit der eine Hashkette in eine bestimmte starke Zusammenhangskomponente führt. Da diese deutliche Größenunterschiede aufweisen hat dies Einfluss auf die Sicherheit der Hashketten.
- Falls die Hashkette Reinitialisiert werden soll, bevor ein festgelegter Grenzwert der Größe der Teilmengen der Wertemenge unterschritten wird, muss die Anzahl an Hashschritten berechnet werden, die dafür nötig sind.
- In Hinblick auf eine effiziente Berechnung der Hashketten ist eine Untersuchung der möglichen Pseudozufallzahlengeneratoren wichtig. Dazu zählen z.B. auch Möglichkeiten Zeitbedarf und Speicherplatz gegeneinander abzuwägen.
- Da sich die Ausgabelänge von KECCAK anpassen lässt, ist ein Vergleich der Daten zwischen KECCAK mit Standardlänge und anschließendem Kürzen der Werte, wie im Experiment geschehen, und KECCAK mit kurzer Ausgabe interessant.

Literaturverzeichnis

[1] Ertel, Wolfgang: *Angewandte Kryptographie*. 3., aktualisierte Aufl. München : Hanser, 2007

[2] Dietzfelbinger, Martin ; Keller, Jörg: *On Hash Chains vs. Hash Trees*. 2012. – Unveröffentlichter Artikel

[3] Tittmann, Peter: *Graphentheorie: eine anwendungsorientierte Einführung*. Fachbuchverl. Leipzig im Carl-Hanser-Verl, 2003

[4] Schneier, Bruce: *Applied Cryptography: Protocols, Algorithms, and Source Code in C, Second Edition*. Wiley, 1996

[5] Turan, Meltem S. ; Barker, Elaine ; Burr, William ; Chen, Lily: Recommendation for Password-Based Key Derivation, Part 1: Storage Applications. In: *NIST Special Publication* 800-132 (2012). http://csrc.nist.gov/publications/nistpubs/800-132/nist-sp800-132.pdf

[6] Lucks, Stefan: *Design Principles for Iterated Hash Functions*. Cryptology ePrint Archive, Report 2004/253. http://eprint.iacr.org/2004/253. Version: 2004

[7] Lamport, L.: Password authentication with insecure communication. In: *Communications of the ACM* 24 (1981), Nr. 11, S. 770–772

[8] Fischlin, Marc: Fast Verification of Hash Chains. In: [71], S. 339–352

[9] Hu, Y.C. ; Jakobsson, M. ; Perrig, A.: Efficient constructions for one-way hash chains. In: *Applied Cryptography and Network Security*, Springer, 2005, S. 167–190

[10] Kim, Sung-Ryul: Improved Scalable Hash Chain Traversal. In: [74], S. 86–95

[11] Sella, Yaron: On The Computation-Storage Trade-Offs of Hash Chain Traversal. In: [72], S. 270–285

[12] Yum, Dae H. ; Kim, Jin S. ; Lee, Pil J. ; Hong, Sung J.: On Fast Verification of Hash Chains. In: [73], S. 382–396

[13] Goyal, V.: How to re-initialize a hash chain. In: *The Cryptology ePrint Archive* (2004). http://eprint.iacr.org/2004/097

[14] Merkle, Ralph C.: A Certified Digital Signature. In: [75], S. 218–238

[15] Merkle, Ralph C.: A Digital Signature Based on a Conventional Encryption Function. In: [76], S. 369–378

[16] Bleichenbacher, Daniel ; Maurer, Ueli M.: On the Efficiency of One-Time Digital Signatures. In: [77], S. 145–158

[17] Aumasson, Jean-Philippe ; Henzen, Luca ; Meier, Willi ; Phan, Raphael C.-W.: *SHA-3 proposal BLAKE*. http://www.131002.net/blake/, Abruf: 23.12.2012

[18] Wu, Hongjun: *Hash Function JH*. http://www3.ntu.edu.sg/home/wuhj/research/jh/, Abruf: 28.12.2012

[19] Bertoni, Guido ; Daemen, Joan ; Peeters, Michaël ; Assche, Gilles V.: *The Keccak sponge function family*. http://keccak.noekeon.org/, Abruf: 13.12.2012

[20] Ferguson, Niels ; Lucks, Stefan ; Schneier, Bruce ; Whiting, Doug ; Bellare, Mihir ; Kohno, Tadayoshi ; Callas, Jon ; Walker, Jesse: *The Skein Hash Function Family*. http://www.skein-hash.info/, Abruf: 03.12.2012

[21] Thomsen, Søren S. ; Schläffer, Martin ; Rechberger, Christian ; Mendel, Florian ; Matusiewicz, Krystian ; Knudsen, Lars R. ; Gauravaram, Praveen: *Grøstl – a SHA-3 candidate*. http://www.groestl.info/, Abruf: 04.12.2012

[22] Aumasson, Jean-Philippe ; Henzen, Luca ; Meier, Willi ; Phan, Raphael C.-W.: *SHA-3 proposal BLAKE*. https://131002.net/blake/blake.pdf. Version: 2010, Abruf: 04.12.2012. – Version 1.3

[23] Biham, Eli ; Dunkelman, Orr: *A Framework for Iterative Hash Functions - HAIFA*. Cryptology ePrint Archive, Report 2007/278. http://eprint.iacr.org/2007/278.pdf. Version: 2007

[24] Bernstein, Daniel J.: *ChaCha, a variant of Salsa20*. http://cr.yp.to/papers.html#chacha. Version: 2008, Abruf: 10.12.2012

[25] Bernstein, Daniel J.: *The Salsa20 family of stream ciphers*. http://cr.yp.to/papers.html#salsafamily. Version: 2007

[26] Coron, Jean-Sébastien ; Dodis, Yevgeniy ; Malinaud, Cécile ; Puniya, Prashant: Merkle-Damgård Revisited: How to Construct a Hash Function. In: [60], S. 430–448

[27] Dang, Quynh: Randomized Hashing for Digital Signatures. In: *NIST Special Publication* 800-106 (2009). http://csrc.nist.gov/publications/nistpubs/800-106/NIST-SP-800-106.pdf

[28] Thomsen, Søren S. ; Schläffer, Martin ; Rechberger, Christian ; Mendel, Florian ; Matusiewicz, Krystian ; Knudsen, Lars R. ; Gauravaram, Praveen: *Grøstl – a SHA-3 candidate*. http://www.groestl.info/Groestl.pdf, Abruf: 04.12.2012

[29] Daemen, Joan ; Rijmen, Vincent: Security of a Wide Trail Design. In: [67], S. 1–11

[30] Wu, H.: *The hash function jh*. http://www3.ntu.edu.sg/home/wuhj/research/jh/jh_round3.pdf. Version: 2009, Abruf: 06.12.2012

[31] Bertoni, Guido ; Daemen, Joan ; Peeters, Michaël ; Van Assche, Gilles: *Cryptographic sponge functions*. http://sponge.noekeon.org/CSF-0.1.pdf. Version: 2011, Abruf: 28.12.2012

[32] NIST: *SHA-3 Selection Announcement*. http://csrc.nist.gov/groups/ST/hash/sha-3/sha-3_selection_announcement.pdf, Abruf: 26.12.2012

[33] Ferguson, Niels ; Lucks, Stefan ; Schneier, Bruce ; Whiting, Doug ; Bellare, Mihir ; Kohno, Tadayoshi ; Callas, John ; Walker, Jessie: *The Skein Hash Function Family*. http://www.skein-hash.info/sites/default/files/skein1.1.pdf. Version: 2008, Abruf: 16.12.2012. – Version 1.1

[34] Rivest, R.: *The MD5 Message-Digest Algorithm*. RFC 1321 (Informational). http://www.ietf.org/rfc/rfc1321.txt. Version: April 1992 (Request for Comments). – Updated by RFC 6151

[35] Schmeh, Klaus: *Kryptografie. Verfahren, Protokolle, Infrastrukturen*. 3. Auflage. dpunkt.verlag, 2007 (iX-Edition)

[36] Dobbertin, Hans: Digitale Fingerabdrücke. In: *DuD: Datenschutz und Datensicherheit, Recht und Sicherheit in Informationsverarbeitung und Kommunikation* Jg. 1997 (1997), Nr. 2, S. 82–87

[37] Wang, Xiaoyun ; Feng, Dengguo ; Lai, Xuejia ; Yu, Hongbo: Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD. In: *IACR Cryptology ePrint Archive* 2004 (2004), 199. http://eprint.iacr.org/2004/199

[38] Klíma, Vlastimil: Tunnels in Hash Functions: MD5 Collisions Within a Minute. In: *IACR Cryptology ePrint Archive* 2006 (2006), 105. http://eprint.iacr.org/2006/105

[39] UNBEKANNT: Proof of concept attack further discredits MD5. In: *Network security* 2009; Jg. 2009 (2009), Nr. 1, S. 2–3

[40] Forte, Dario: The death of MD5. In: *Network security* 2009; Jg. 2009 (2009), Nr. 2, S. 18–21

[41] Chen, Shiwei ; Jin, Chenhui: An Improved Collision Attack on MD5 Algorithm. In: [68], S. 343–357

[42] Dobbertin, Hans ; Bosselaers, Antoon ; Preneel, Bart: RIPEMD-160: A Strengthened Version of RIPEMD. In: [69], S. 71–82

[43] Dobbertin, H.: RIPEMD with Two-Round Compress Function Is Not Collision-Free. In: *Journal of cryptology : the journal of the International Association for Cryptologic Research* Jg. 1997 (1997), Nr. 1, S. 51–70

[44] Ohtahara, Chiaki ; Sasaki, Yu ; Shimoyama, Takeshi: Preimage Attacks on the Step-Reduced RIPEMD-128 and RIPEMD-160. In: *IEICE Transactions* 95-A (2012), Nr. 10, S. 1729–1739

[45] Eastlake 3rd, D. ; Jones, P.: *US Secure Hash Algorithm 1 (SHA1)*. RFC 3174 (Informational). http://www.ietf.org/rfc/rfc3174.txt. Version: September 2001 (Request for Comments). – Updated by RFCs 4634, 6234

[46] Eastlake 3rd, D. ; Hansen, T.: *US Secure Hash Algorithms (SHA and HMAC-SHA)*. RFC 4634 (Informational). http://www.ietf.org/rfc/rfc4634.txt. Version: Juli 2006 (Request for Comments). – Obsoleted by RFC 6234

[47] Wang, Xiaoyun ; Yu, Hongbo ; Yin, Yiqun L.: Efficient Collision Search Attacks on SHA-0. In: [60], S. 1–16

[48] Wang, Xiaoyun ; Yin, Yiqun L. ; Yu, Hongbo: Finding Collisions in the Full SHA-1. In: [60], S. 17–36

[49] Wang, X. ; Yao, A. ; Yao, F.: Cryptanalysis on SHA-1. In: *Cryptographic Hash Workshop hosted by NIST*, 2005

[50] Cannière, Christophe D. ; Rechberger, Christian: Finding SHA-1 Characteristics: General Results and Applications. In: [70], S. 1–20

[51] Standard, S. H.: Federal Information Processing Standard Publication# 180. In: *US Department of Commerce, National Institute of Standards and Technology* 56 (1993), 57–71. http://csrc.nist.gov/publications/fips/fips185/fips185.txt

[52] Maheshwari, Arpan: Efficient software implementation of SHA-3 candidates on 8-Bit AVR microcontrollers / Indian Institute of Technology, Kanpur, Indien. 2010. – Forschungsbericht

[53] NIST: *Cryptographic Hash Algorithm Competition*. http://csrc.nist.gov/groups/ST/hash/sha-3/index.html, Abruf: 15.12.2012

[54] NIST: *Third (Final) Round Candidates*. http://csrc.nist.gov/groups/ST/hash/sha-3/Round3/submissions_rnd3.html, Abruf: 15.12.2012

[55] Heyse, Stefan ; Maurich, Ingo von ; Wild, Alexander ; Reuber, Cornel ; Rave, Johannes ; Poeppelmann, Thomas ; Paar, Christof: Evaluation of SHA-3 Candidates for 8-Bit Embedded Processors. In: *2nd SHA-3 Candidate Conference*, 2010

[56] Murvay, Pal-Stefan ; Groza, Bogdan: Performance improvements for SHA-3 finalists by exploiting microcontroller on-chip parallelism. In: *6th International Conference on Risks and Security of Internet and Systems (CRiSIS)*, 2011

[57] LABOR e.V.: *XBX - Platform atmega1284p 16mhz (SHA3 Round 3 Only)*. http://xbx.das-labor.org/trac/wiki/r2011sha3platforms_atmega1284p_16mhz, Abruf: 12.12.2012

[58] LABOR e.V.: *AVR-Crypto-Lib/en - LaborWiki*. http://www.das-labor.org/wiki/Crypto-avr-lib/en#Hashes_2, Abruf: 16.12.2012

[59] Tarjan, Robert E.: Depth-First Search and Linear Graph Algorithms, 1972, S. 146–160

[60] Shoup, Victor (Hrsg.): *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*. Bd. *3621*. Springer, 2005 (Lecture Notes in Computer Science)

[61] Smith, Glen ; Miller, Kyle ; Conway, Scott ; Sullivan, Sean: *opencsv – Frequently Asked Questions*. http://opencsv.sourceforge.net/, Abruf: 16.01.2013

[62] The Apache Software Foundation: *Apache License, Version 2.0*. https://www.apache.org/licenses/LICENSE-2.0.html, Abruf: 16.01.2013

[63] The Legion Of The Bouncy Castle: *Latest Java Releases*. http://www.bouncycastle.org/latest_releases.html, Abruf: 16.01.2013

[64] The Legion Of The Bouncy Castle: *License*. http://www.bouncycastle.org/licence.html, Abruf: 16.01.2013

[65] Stiftung Secure Information and Communication Technologies SIC: *SHA-3 Provider*. http://jce.iaik.tugraz.at/index.php/sic/Products/Core-Crypto-Toolkits/SHA-3_Provider, Abruf: 16.01.2013

[66] Free Software Foundation, Inc.: *GNU GENERAL PUBLIC LICENSE*. https://www.gnu.org/licenses/gpl-3.0-standalone.html, Abruf: 16.01.2013

[67] Menezes, Alfred (Hrsg.) ; Sarkar, Palash (Hrsg.): *Progress in Cryptology - INDOCRYPT 2002, Third International Conference on Cryptology in India, Hyderabad, India, December 16-18, 2002*. Bd. *2551*. Springer, 2002 (Lecture Notes in Computer Science)

[68] Pei, Dingyi (Hrsg.) ; Yung, Moti (Hrsg.) ; Lin, Dongdai (Hrsg.) ; Wu, Chuankun (Hrsg.): *Information Security and Cryptology, Third SKLOIS Conference, Inscrypt 2007, Xining, China, August 31 - September 5, 2007, Revised Selected Papers*. Bd. *4990*. Springer, 2008 (Lecture Notes in Computer Science)

[69] Gollmann, Dieter (Hrsg.): *Fast Software Encryption, Third International Workshop, Cambridge, UK, February 21-23, 1996, Proceedings*. Bd. *1039*. Springer, 1996 (Lecture Notes in Computer Science)

[70] Lai, Xuejia (Hrsg.) ; Chen, Kefei (Hrsg.): *Advances in Cryptology - ASIACRYPT 2006, 12th International Conference on the Theory and Application of Cryptology and Information Security, Shanghai, China, December 3-7, 2006, Proceedings*. Bd. *4284*. Springer, 2006 (Lecture Notes in Computer Science)

[71] Okamoto, Tatsuaki (Hrsg.): *Topics in Cryptology - CT-RSA 2004, The Cryptographers' Track at the RSA Conference 2004, San Francisco, CA, USA, February 23-27, 2004, Proceedings*. Bd. *2964*. Springer, 2004 (Lecture Notes in Computer Science)

[72] Wright, Rebecca N. (Hrsg.): *Financial Cryptography, 7th International Conference, FC 2003, Guadeloupe, French West Indies, January 27-30, 2003, Revised Papers*. Bd. *2742*. Springer, 2003 (Lecture Notes in Computer Science)

[73] Pieprzyk, Josef (Hrsg.): *Topics in Cryptology - CT-RSA 2010, The Cryptographers' Track at the RSA Conference 2010, San Francisco, CA, USA, March 1-5, 2010. Proceedings*. Bd. *5985*. Springer, 2010 (Lecture Notes in Computer Science)

[74] Zhou, Jianying (Hrsg.) ; Yung, Moti (Hrsg.) ; Han, Yongfei (Hrsg.): *Applied Cryptography and Network Security, First International Conference, ACNS 2003. Kunming, China, October 16-19, 2003, Proceedings*. Bd. *2846*. Springer, 2003 (Lecture Notes in Computer Science)

[75] Brassard, Gilles (Hrsg.): *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*. Bd. *435*. Springer, 1990 (Lecture Notes in Computer Science)

[76] Pomerance, Carl (Hrsg.): *Advances in Cryptology - CRYPTO '87, A Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, California, USA, August 16-20, 1987, Proceedings*. Bd. *293*. Springer, 1988 (Lecture Notes in Computer Science)

[77] Kim, Kwangjo (Hrsg.) ; Matsumoto, Tsutomu (Hrsg.): *Advances in Cryptology - ASIACRYPT '96, International Conference on the Theory and Applications of Cryptology and Information Security, Kyongju, Korea, November 3-7, 1996, Proceedings*. Bd. *1163*. Springer, 1996 (Lecture Notes in Computer Science)

Anhang A

Daten

Tabelle A.1

Algo-rithmus	Hash-länge	Anzahl der Zu-satzbits	Runden	Anzahl der Knoten in s.Z.	Anzahl der s.Z.	s.Z. Nummer	Größe der s.Z.
BLAKE256	10	0	72	15	4	1	1
BLAKE256	10	0	72	15	4	2	2
BLAKE256	10	0	72	15	4	3	5
BLAKE256	10	0	72	15	4	4	7
BLAKE256	10	1	6	715	1	1	715
BLAKE256	10	2	5	942	1	1	942
BLAKE256	10	3	2	1020	1	1	1020
BLAKE256	10	4	2	1024	1	1	1024
BLAKE256	12	0	113	63	7	1	1
BLAKE256	12	0	113	63	7	2	1
BLAKE256	12	0	113	63	7	3	1
BLAKE256	12	0	113	63	7	4	13
BLAKE256	12	0	113	63	7	5	15
BLAKE256	12	0	113	63	7	6	16
BLAKE256	12	0	113	63	7	7	16
BLAKE256	12	1	8	2861	1	1	2861
BLAKE256	12	2	4	3811	1	1	3811
BLAKE256	12	3	2	4076	1	1	4076
BLAKE256	12	4	2	4096	1	1	4096
BLAKE256	14	0	366	114	9	1	1
BLAKE256	14	0	366	114	9	2	1
BLAKE256	14	0	366	114	9	3	2
BLAKE256	14	0	366	114	9	4	2
BLAKE256	14	0	366	114	9	5	3
BLAKE256	14	0	366	114	9	6	4
BLAKE256	14	0	366	114	9	7	31

Auf der nächsten Seite fortgesetzt

Tabelle A.1: Resultate der Versuchsreihen mit Bits vom Anfang der Hashwerte

Tabelle A.1 Fortsetzung der vorherigen Seite

Algo-rithmus	Hash-länge	Anzahl der Zu-satzbits	Runden	Anzahl der Knoten in s.Z.	Anzahl der s.Z.	s.Z. Nummer	Größe der s.Z.
BLAKE256	14	0	366	114	9	8	35
BLAKE256	14	0	366	114	9	9	35
BLAKE256	14	1	9	11087	2	1	2
BLAKE256	14	1	9	11087	2	2	11085
BLAKE256	14	2	5	15158	1	1	15158
BLAKE256	14	3	2	16311	1	1	16311
BLAKE256	14	4	2	16383	1	1	16383
Groestl256	10	0	52	39	5	1	1
Groestl256	10	0	52	39	5	2	2
Groestl256	10	0	52	39	5	3	3
Groestl256	10	0	52	39	5	4	15
Groestl256	10	0	52	39	5	5	18
Groestl256	10	1	8	720	1	1	720
Groestl256	10	2	3	954	1	1	954
Groestl256	10	3	2	1017	1	1	1017
Groestl256	10	4	2	1024	1	1	1024
Groestl256	12	0	149	152	4	1	1
Groestl256	12	0	149	152	4	2	1
Groestl256	12	0	149	152	4	3	1
Groestl256	12	0	149	152	4	4	149
Groestl256	12	1	8	2785	2	1	1
Groestl256	12	1	8	2785	2	2	2784
Groestl256	12	2	4	3812	2	1	1
Groestl256	12	2	4	3812	2	2	3811
Groestl256	12	3	2	4086	1	1	4086
Groestl256	12	4	2	4096	1	1	4096
Groestl256	14	0	246	127	7	1	1
Groestl256	14	0	246	127	7	2	1
Groestl256	14	0	246	127	7	3	1
Groestl256	14	0	246	127	7	4	5
Groestl256	14	0	246	127	7	5	7
Groestl256	14	0	246	127	7	6	37
Groestl256	14	0	246	127	7	7	75
Groestl256	14	1	12	11060	2	1	1
Groestl256	14	1	12	11060	2	2	11059
Groestl256	14	2	6	15228	1	1	15228
Groestl256	14	3	3	16316	1	1	16316
Groestl256	14	4	2	16384	1	1	16384
JH256	10	0	40	66	6	1	1

Auf der nächsten Seite fortgesetzt

Tabelle A.1: Resultate der Versuchsreihen mit Bits vom Anfang der Hashwerte

Tabelle A.1 Fortsetzung der vorherigen Seite

Algorithmus	Hashlänge	Anzahl der Zusatzbits	Runden	Anzahl der Knoten in s.Z.	Anzahl der s.Z.	s.Z. Nummer	Größe der s.Z.
JH256	10	0	40	66	6	2	1
JH256	10	0	40	66	6	3	2
JH256	10	0	40	66	6	4	2
JH256	10	0	40	66	6	5	7
JH256	10	0	40	66	6	6	53
JH256	10	1	10	704	4	1	1
JH256	10	1	10	704	4	2	1
JH256	10	1	10	704	4	3	1
JH256	10	1	10	704	4	4	701
JH256	10	2	4	968	1	1	968
JH256	10	3	2	1022	1	1	1022
JH256	10	4	2	1024	1	1	1024
JH256	12	0	116	65	6	1	1
JH256	12	0	116	65	6	2	2
JH256	12	0	116	65	6	3	2
JH256	12	0	116	65	6	4	2
JH256	12	0	116	65	6	5	24
JH256	12	0	116	65	6	6	34
JH256	12	1	8	2732	1	1	2732
JH256	12	2	5	3778	1	1	3778
JH256	12	3	3	4068	1	1	4068
JH256	12	4	2	4096	1	1	4096
JH256	14	0	255	48	4	1	1
JH256	14	0	255	48	4	2	4
JH256	14	0	255	48	4	3	6
JH256	14	0	255	48	4	4	37
JH256	14	1	8	11004	1	1	11004
JH256	14	2	7	15115	1	1	15115
JH256	14	3	3	16303	1	1	16303
JH256	14	4	2	16384	1	1	16384
KECCAK256	10	0	32	57	5	1	1
KECCAK256	10	0	32	57	5	2	1
KECCAK256	10	0	32	57	5	3	5
KECCAK256	10	0	32	57	5	4	22
KECCAK256	10	0	32	57	5	5	28
KECCAK256	10	1	7	712	1	1	712
KECCAK256	10	2	3	957	1	1	957
KECCAK256	10	3	2	1019	1	1	1019
KECCAK256	10	4	2	1024	1	1	1024

Auf der nächsten Seite fortgesetzt

Tabelle A.1: Resultate der Versuchsreihen mit Bits vom Anfang der Hashwerte

Tabelle A.1 Fortsetzung der vorherigen Seite

Algorithmus	Hashlänge	Anzahl der Zusatzbits	Runden	Anzahl der Knoten in s.Z.	Anzahl der s.Z.	s.Z. Nummer	Größe der s.Z.
KECCAK256	12	0	102	97	2	1	3
KECCAK256	12	0	102	97	2	2	94
KECCAK256	12	1	9	2800	2	1	1
KECCAK256	12	1	9	2800	2	2	2799
KECCAK256	12	2	4	3786	1	1	3786
KECCAK256	12	3	3	4076	1	1	4076
KECCAK256	12	4	2	4096	1	1	4096
KECCAK256	14	0	158	181	8	1	1
KECCAK256	14	0	158	181	8	2	1
KECCAK256	14	0	158	181	8	3	1
KECCAK256	14	0	158	181	8	4	5
KECCAK256	14	0	158	181	8	5	11
KECCAK256	14	0	158	181	8	6	25
KECCAK256	14	0	158	181	8	7	30
KECCAK256	14	0	158	181	8	8	107
KECCAK256	14	1	11	10894	4	1	1
KECCAK256	14	1	11	10894	4	2	1
KECCAK256	14	1	11	10894	4	3	1
KECCAK256	14	1	11	10894	4	4	10891
KECCAK256	14	2	5	15127	1	1	15127
KECCAK256	14	3	3	16309	1	1	16309
KECCAK256	14	4	2	16383	1	1	16383
RIPEMD160	10	0	42	26	3	1	1
RIPEMD160	10	0	42	26	3	2	6
RIPEMD160	10	0	42	26	3	3	19
RIPEMD160	10	1	9	692	1	1	692
RIPEMD160	10	2	3	952	1	1	952
RIPEMD160	10	3	2	1018	1	1	1018
RIPEMD160	10	4	2	1023	1	1	1023
RIPEMD160	12	0	118	56	5	1	1
RIPEMD160	12	0	118	56	5	2	4
RIPEMD160	12	0	118	56	5	3	7
RIPEMD160	12	0	118	56	5	4	20
RIPEMD160	12	0	118	56	5	5	24
RIPEMD160	12	1	9	2780	1	1	2780
RIPEMD160	12	2	5	3824	1	1	3824
RIPEMD160	12	3	2	4072	1	1	4072
RIPEMD160	12	4	2	4096	1	1	4096
RIPEMD160	14	0	276	152	4	1	2

Auf der nächsten Seite fortgesetzt

Tabelle A.1: Resultate der Versuchsreihen mit Bits vom Anfang der Hashwerte

Tabelle A.1 Fortsetzung der vorherigen Seite

Algo-rithmus	Hash-länge	Anzahl der Zu-satzbits	Runden	Anzahl der Knoten in s.Z.	Anzahl der s.Z.	s.Z. Nummer	Größe der s.Z.
RIPEMD160	14	0	276	152	4	2	5
RIPEMD160	14	0	276	152	4	3	11
RIPEMD160	14	0	276	152	4	4	134
RIPEMD160	14	1	14	11017	1	1	11017
RIPEMD160	14	2	5	15193	1	1	15193
RIPEMD160	14	3	3	16296	1	1	16296
RIPEMD160	14	4	2	16383	1	1	16383
Skein256	10	0	35	54	7	1	1
Skein256	10	0	35	54	7	2	2
Skein256	10	0	35	54	7	3	4
Skein256	10	0	35	54	7	4	6
Skein256	10	0	35	54	7	5	7
Skein256	10	0	35	54	7	6	16
Skein256	10	0	35	54	7	7	18
Skein256	10	1	8	706	3	1	1
Skein256	10	1	8	706	3	2	1
Skein256	10	1	8	706	3	3	704
Skein256	10	2	4	955	1	1	955
Skein256	10	3	2	1021	1	1	1021
Skein256	10	4	2	1024	1	1	1024
Skein256	12	0	77	164	4	1	7
Skein256	12	0	77	164	4	2	9
Skein256	12	0	77	164	4	3	13
Skein256	12	0	77	164	4	4	135
Skein256	12	1	9	2748	1	1	2748
Skein256	12	2	5	3825	1	1	3825
Skein256	12	3	2	4081	1	1	4081
Skein256	12	4	2	4096	1	1	4096
Skein256	14	0	177	256	5	1	4
Skein256	14	0	177	256	5	2	4
Skein256	14	0	177	256	5	3	27
Skein256	14	0	177	256	5	4	99
Skein256	14	0	177	256	5	5	122
Skein256	14	1	10	11011	1	1	11011
Skein256	14	2	5	15190	1	1	15190
Skein256	14	3	3	16299	1	1	16299
Skein256	14	4	2	16384	1	1	16384
MD5	10	0	48	35	5	1	1
MD5	10	0	48	35	5	2	1

Auf der nächsten Seite fortgesetzt

Tabelle A.1: Resultate der Versuchsreihen mit Bits vom Anfang der Hashwerte

Tabelle A.1 Fortsetzung der vorherigen Seite

Algorithmus	Hashlänge	Anzahl der Zusatzbits	Runden	Anzahl der Knoten in s.Z.	Anzahl der s.Z.	s.Z. Nummer	Größe der s.Z.
MD5	10	0	48	35	5	3	3
MD5	10	0	48	35	5	4	13
MD5	10	0	48	35	5	5	17
MD5	10	1	8	705	2	1	1
MD5	10	1	8	705	2	2	704
MD5	10	2	4	939	1	1	939
MD5	10	3	2	1019	1	1	1019
MD5	10	4	2	1024	1	1	1024
MD5	12	0	119	82	7	1	1
MD5	12	0	119	82	7	2	2
MD5	12	0	119	82	7	3	2
MD5	12	0	119	82	7	4	3
MD5	12	0	119	82	7	5	4
MD5	12	0	119	82	7	6	12
MD5	12	0	119	82	7	7	58
MD5	12	1	8	2758	1	1	2758
MD5	12	2	4	3804	1	1	3804
MD5	12	3	3	4075	1	1	4075
MD5	12	4	2	4096	1	1	4096
MD5	14	0	218	173	6	1	5
MD5	14	0	218	173	6	2	6
MD5	14	0	218	173	6	3	22
MD5	14	0	218	173	6	4	26
MD5	14	0	218	173	6	5	32
MD5	14	0	218	173	6	6	82
MD5	14	1	9	11085	4	1	1
MD5	14	1	9	11085	4	2	1
MD5	14	1	9	11085	4	3	2
MD5	14	1	9	11085	4	4	11081
MD5	14	2	5	15236	1	1	15236
MD5	14	3	3	16305	1	1	16305
MD5	14	4	2	16383	1	1	16383
SHA-1	10	0	68	33	5	1	1
SHA-1	10	0	68	33	5	2	1
SHA-1	10	0	68	33	5	3	3
SHA-1	10	0	68	33	5	4	3
SHA-1	10	0	68	33	5	5	25
SHA-1	10	1	6	705	1	1	705
SHA-1	10	2	3	954	1	1	954

Auf der nächsten Seite fortgesetzt

Tabelle A.1: Resultate der Versuchsreihen mit Bits vom Anfang der Hashwerte

Tabelle A.1 Fortsetzung der vorherigen Seite

Algorithmus	Hashlänge	Anzahl der Zusatzbits	Runden	Anzahl der Knoten in s.Z.	Anzahl der s.Z.	s.Z. Nummer	Größe der s.Z.
SHA-1	10	3	2	1021	1	1	1021
SHA-1	10	4	2	1024	1	1	1024
SHA-1	12	0	163	108	5	1	1
SHA-1	12	0	163	108	5	2	2
SHA-1	12	0	163	108	5	3	3
SHA-1	12	0	163	108	5	4	3
SHA-1	12	0	163	108	5	5	99
SHA-1	12	1	12	2732	4	1	1
SHA-1	12	1	12	2732	4	2	1
SHA-1	12	1	12	2732	4	3	1
SHA-1	12	1	12	2732	4	4	2729
SHA-1	12	2	5	3810	1	1	3810
SHA-1	12	3	3	4082	1	1	4082
SHA-1	12	4	2	4096	1	1	4096
SHA-1	14	0	224	67	6	1	1
SHA-1	14	0	224	67	6	2	2
SHA-1	14	0	224	67	6	3	4
SHA-1	14	0	224	67	6	4	5
SHA-1	14	0	224	67	6	5	10
SHA-1	14	0	224	67	6	6	45
SHA-1	14	1	9	11221	1	1	11221
SHA-1	14	2	5	15232	1	1	15232
SHA-1	14	3	3	16303	1	1	16303
SHA-1	14	4	2	16383	1	1	16383
SHA256	10	0	30	40	6	1	1
SHA256	10	0	30	40	6	2	1
SHA256	10	0	30	40	6	3	1
SHA256	10	0	30	40	6	4	2
SHA256	10	0	30	40	6	5	4
SHA256	10	0	30	40	6	6	31
SHA256	10	1	9	673	2	1	1
SHA256	10	1	9	673	2	2	672
SHA256	10	2	4	949	1	1	949
SHA256	10	3	2	1017	1	1	1017
SHA256	10	4	2	1024	1	1	1024
SHA256	12	0	69	139	6	1	2
SHA256	12	0	69	139	6	2	3
SHA256	12	0	69	139	6	3	9
SHA256	12	0	69	139	6	4	22

Auf der nächsten Seite fortgesetzt

Tabelle A.1: Resultate der Versuchsreihen mit Bits vom Anfang der Hashwerte

Tabelle A.1 Fortsetzung der vorherigen Seite

Algorithmus	Hashlänge	Anzahl der Zusatzbits	Runden	Anzahl der Knoten in s.Z.	Anzahl der s.Z.	s.Z. Nummer	Größe der s.Z.
SHA256	12	0	69	139	6	5	33
SHA256	12	0	69	139	6	6	70
SHA256	12	1	7	2808	1	1	2808
SHA256	12	2	4	3829	1	1	3829
SHA256	12	3	2	4077	1	1	4077
SHA256	12	4	2	4096	1	1	4096
SHA256	14	0	228	108	3	1	22
SHA256	14	0	228	108	3	2	33
SHA256	14	0	228	108	3	3	53
SHA256	14	1	10	11097	3	1	1
SHA256	14	1	10	11097	3	2	1
SHA256	14	1	10	11097	3	3	11095
SHA256	14	2	6	15229	1	1	15229
SHA256	14	3	3	16320	1	1	16320
SHA256	14	4	2	16384	1	1	16384

Ende der Tabelle

Tabelle A.1: Resultate der Versuchsreihen mit Bits vom Anfang der Hashwerte

Tabelle A.2

Algorithmus	Hashlänge	Anzahl der Zusatzbits	Runden	Anzahl der Knoten in s.Z.	Anzahl der s.Z.	s.Z. Nummer	Größe der s.Z.
BLAKE256	10	0	56	27	4	1	1
BLAKE256	10	0	56	27	4	2	1
BLAKE256	10	0	56	27	4	3	2
BLAKE256	10	0	56	27	4	4	23
BLAKE256	10	1	7	699	1	1	699
BLAKE256	10	2	3	948	1	1	948
BLAKE256	10	3	2	1016	1	1	1016
BLAKE256	10	4	2	1024	1	1	1024
BLAKE256	12	0	115	65	2	1	5
BLAKE256	12	0	115	65	2	2	60
BLAKE256	12	1	8	2790	2	1	1
BLAKE256	12	1	8	2790	2	2	2789
BLAKE256	12	2	4	3767	1	1	3767
BLAKE256	12	3	3	4073	1	1	4073
BLAKE256	12	4	2	4096	1	1	4096

Auf der nächsten Seite fortgesetzt

Tabelle A.2: Resultate der Versuchsreihen mit Bits aus der Mitte der Hashwerte

Tabelle A.2 Fortsetzung der vorherigen Seite

Algo-rithmus	Hash-länge	Anzahl der Zu-satzbits	Runden	Anzahl der Knoten in s.Z.	Anzahl der s.Z.	s.Z. Nummer	Größe der s.Z.
BLAKE256	14	0	213	260	5	1	2
BLAKE256	14	0	213	260	5	2	10
BLAKE256	14	0	213	260	5	3	66
BLAKE256	14	0	213	260	5	4	79
BLAKE256	14	0	213	260	5	5	103
BLAKE256	14	1	12	11046	1	1	11046
BLAKE256	14	2	5	15117	1	1	15117
BLAKE256	14	3	3	16306	1	1	16306
BLAKE256	14	4	2	16384	1	1	16384
Groestl256	10	0	55	42	4	1	2
Groestl256	10	0	55	42	4	2	2
Groestl256	10	0	55	42	4	3	15
Groestl256	10	0	55	42	4	4	23
Groestl256	10	1	8	653	1	1	653
Groestl256	10	2	3	941	1	1	941
Groestl256	10	3	2	1018	1	1	1018
Groestl256	10	4	2	1024	1	1	1024
Groestl256	12	0	64	212	4	1	1
Groestl256	12	0	64	212	4	2	20
Groestl256	12	0	64	212	4	3	30
Groestl256	12	0	64	212	4	4	161
Groestl256	12	1	9	2693	1	1	2693
Groestl256	12	2	4	3785	1	1	3785
Groestl256	12	3	2	4082	1	1	4082
Groestl256	12	4	2	4096	1	1	4096
Groestl256	14	0	232	52	4	1	1
Groestl256	14	0	232	52	4	2	2
Groestl256	14	0	232	52	4	3	2
Groestl256	14	0	232	52	4	4	47
Groestl256	14	1	13	11033	1	1	11033
Groestl256	14	2	6	15180	1	1	15180
Groestl256	14	3	3	16312	1	1	16312
Groestl256	14	4	2	16384	1	1	16384
JH256	10	0	33	66	4	1	1
JH256	10	0	33	66	4	2	10
JH256	10	0	33	66	4	3	25
JH256	10	0	33	66	4	4	30
JH256	10	1	6	693	3	1	1
JH256	10	1	6	693	3	2	3

Auf der nächsten Seite fortgesetzt

Tabelle A.2: Resultate der Versuchsreihen mit Bits aus der Mitte der Hashwerte

Tabelle A.2 Fortsetzung der vorherigen Seite

Algorithmus	Hashlänge	Anzahl der Zusatzbits	Runden	Anzahl der Knoten in s.Z.	Anzahl der s.Z.	s.Z. Nummer	Größe der s.Z.
JH256	10	1	6	693	3	3	689
JH256	10	2	4	941	2	1	1
JH256	10	2	4	941	2	2	940
JH256	10	3	2	1020	1	1	1020
JH256	10	4	2	1024	1	1	1024
JH256	12	0	104	134	7	1	1
JH256	12	0	104	134	7	2	2
JH256	12	0	104	134	7	3	8
JH256	12	0	104	134	7	4	13
JH256	12	0	104	134	7	5	26
JH256	12	0	104	134	7	6	32
JH256	12	0	104	134	7	7	52
JH256	12	1	10	2785	1	1	2785
JH256	12	2	4	3823	1	1	3823
JH256	12	3	2	4078	1	1	4078
JH256	12	4	2	4096	1	1	4096
JH256	14	0	215	221	3	1	17
JH256	14	0	215	221	3	2	54
JH256	14	0	215	221	3	3	150
JH256	14	1	9	11109	3	1	1
JH256	14	1	9	11109	3	2	2
JH256	14	1	9	11109	3	3	11106
JH256	14	2	5	15202	1	1	15202
JH256	14	3	2	16297	1	1	16297
JH256	14	4	2	16384	1	1	16384
KECCAK256	10	0	53	34	6	1	1
KECCAK256	10	0	53	34	6	2	1
KECCAK256	10	0	53	34	6	3	2
KECCAK256	10	0	53	34	6	4	4
KECCAK256	10	0	53	34	6	5	8
KECCAK256	10	0	53	34	6	6	18
KECCAK256	10	1	8	685	3	1	1
KECCAK256	10	1	8	685	3	2	2
KECCAK256	10	1	8	685	3	3	682
KECCAK256	10	2	5	931	1	1	931
KECCAK256	10	3	3	1017	1	1	1017
KECCAK256	10	4	2	1024	1	1	1024
KECCAK256	12	0	89	68	4	1	1
KECCAK256	12	0	89	68	4	2	1

Auf der nächsten Seite fortgesetzt

Tabelle A.2: Resultate der Versuchsreihen mit Bits aus der Mitte der Hashwerte

Tabelle A.2 Fortsetzung der vorherigen Seite

Algorithmus	Hashlänge	Anzahl der Zusatzbits	Runden	Anzahl der Knoten in s.Z.	Anzahl der s.Z.	s.Z. Nummer	Größe der s.Z.
KECCAK256	12	0	89	68	4	3	3
KECCAK256	12	0	89	68	4	4	63
KECCAK256	12	1	8	2741	3	1	1
KECCAK256	12	1	8	2741	3	2	1
KECCAK256	12	1	8	2741	3	3	2739
KECCAK256	12	2	3	3770	1	1	3770
KECCAK256	12	3	3	4075	1	1	4075
KECCAK256	12	4	2	4096	1	1	4096
KECCAK256	14	0	163	217	12	1	1
KECCAK256	14	0	163	217	12	2	1
KECCAK256	14	0	163	217	12	3	1
KECCAK256	14	0	163	217	12	4	1
KECCAK256	14	0	163	217	12	5	2
KECCAK256	14	0	163	217	12	6	3
KECCAK256	14	0	163	217	12	7	8
KECCAK256	14	0	163	217	12	8	9
KECCAK256	14	0	163	217	12	9	13
KECCAK256	14	0	163	217	12	10	35
KECCAK256	14	0	163	217	12	11	63
KECCAK256	14	0	163	217	12	12	80
KECCAK256	14	1	9	11071	2	1	1
KECCAK256	14	1	9	11071	2	2	11070
KECCAK256	14	2	5	15213	1	1	15213
KECCAK256	14	3	2	16302	1	1	16302
KECCAK256	14	4	2	16384	1	1	16384
RIPEMD160	10	0	33	67	3	1	3
RIPEMD160	10	0	33	67	3	2	12
RIPEMD160	10	0	33	67	3	3	52
RIPEMD160	10	1	9	694	1	1	694
RIPEMD160	10	2	4	960	1	1	960
RIPEMD160	10	3	2	1020	1	1	1020
RIPEMD160	10	4	2	1024	1	1	1024
RIPEMD160	12	0	167	86	6	1	1
RIPEMD160	12	0	167	86	6	2	1
RIPEMD160	12	0	167	86	6	3	7
RIPEMD160	12	0	167	86	6	4	8
RIPEMD160	12	0	167	86	6	5	23
RIPEMD160	12	0	167	86	6	6	46
RIPEMD160	12	1	12	2740	1	1	2740

Auf der nächsten Seite fortgesetzt

Tabelle A.2: Resultate der Versuchsreihen mit Bits aus der Mitte der Hashwerte

Tabelle A.2 Fortsetzung der vorherigen Seite

Algo-rithmus	Hash-länge	Anzahl der Zu-satzbits	Runden	Anzahl der Knoten in s.Z.	Anzahl der s.Z.	s.Z. Nummer	Größe der s.Z.
RIPEMD160	12	2	4	3794	2	1	1
RIPEMD160	12	2	4	3794	2	2	3793
RIPEMD160	12	3	2	4083	1	1	4083
RIPEMD160	12	4	2	4096	1	1	4096
RIPEMD160	14	0	151	254	4	1	1
RIPEMD160	14	0	151	254	4	2	13
RIPEMD160	14	0	151	254	4	3	21
RIPEMD160	14	0	151	254	4	4	219
RIPEMD160	14	1	10	11157	4	1	1
RIPEMD160	14	1	10	11157	4	2	1
RIPEMD160	14	1	10	11157	4	3	1
RIPEMD160	14	1	10	11157	4	4	11154
RIPEMD160	14	2	5	15254	1	1	15254
RIPEMD160	14	3	3	16308	1	1	16308
RIPEMD160	14	4	2	16384	1	1	16384
Skein256	10	0	55	32	1	1	32
Skein256	10	1	11	660	1	1	660
Skein256	10	2	4	952	1	1	952
Skein256	10	3	2	1022	1	1	1022
Skein256	10	4	2	1024	1	1	1024
Skein256	12	0	64	104	9	1	1
Skein256	12	0	64	104	9	2	1
Skein256	12	0	64	104	9	3	1
Skein256	12	0	64	104	9	4	1
Skein256	12	0	64	104	9	5	2
Skein256	12	0	64	104	9	6	2
Skein256	12	0	64	104	9	7	2
Skein256	12	0	64	104	9	8	8
Skein256	12	0	64	104	9	9	86
Skein256	12	1	8	2782	3	1	1
Skein256	12	1	8	2782	3	2	1
Skein256	12	1	8	2782	3	3	2780
Skein256	12	2	4	3776	2	1	1
Skein256	12	2	4	3776	2	2	3775
Skein256	12	3	3	4072	1	1	4072
Skein256	12	4	2	4096	1	1	4096
Skein256	14	0	310	56	3	1	1
Skein256	14	0	310	56	3	2	2
Skein256	14	0	310	56	3	3	53

Auf der nächsten Seite fortgesetzt

Tabelle A.2: Resultate der Versuchsreihen mit Bits aus der Mitte der Hashwerte

Tabelle A.2 Fortsetzung der vorherigen Seite

Algo-rithmus	Hash-länge	Anzahl der Zu-satzbits	Runden	Anzahl der Knoten in s.Z.	Anzahl der s.Z.	s.Z. Nummer	Größe der s.Z.
Skein256	14	1	11	10997	2	1	1
Skein256	14	1	11	10997	2	2	10996
Skein256	14	2	5	15184	2	1	1
Skein256	14	2	5	15184	2	2	15183
Skein256	14	3	3	16318	1	1	16318
Skein256	14	4	2	16383	1	1	16383
MD5	10	0	43	89	4	1	1
MD5	10	0	43	89	4	2	3
MD5	10	0	43	89	4	3	10
MD5	10	0	43	89	4	4	75
MD5	10	1	10	709	1	1	709
MD5	10	2	4	959	1	1	959
MD5	10	3	3	1017	1	1	1017
MD5	10	4	2	1023	1	1	1023
MD5	12	0	101	49	3	1	3
MD5	12	0	101	49	3	2	19
MD5	12	0	101	49	3	3	27
MD5	12	1	7	2805	3	1	2
MD5	12	1	7	2805	3	2	3
MD5	12	1	7	2805	3	3	2800
MD5	12	2	4	3837	1	1	3837
MD5	12	3	2	4080	1	1	4080
MD5	12	4	2	4096	1	1	4096
MD5	14	0	223	201	10	1	1
MD5	14	0	223	201	10	2	2
MD5	14	0	223	201	10	3	3
MD5	14	0	223	201	10	4	4
MD5	14	0	223	201	10	5	5
MD5	14	0	223	201	10	6	5
MD5	14	0	223	201	10	7	11
MD5	14	0	223	201	10	8	33
MD5	14	0	223	201	10	9	41
MD5	14	0	223	201	10	10	96
MD5	14	1	11	11016	7	1	1
MD5	14	1	11	11016	7	2	1
MD5	14	1	11	11016	7	3	1
MD5	14	1	11	11016	7	4	1
MD5	14	1	11	11016	7	5	2
MD5	14	1	11	11016	7	6	3

Auf der nächsten Seite fortgesetzt

Tabelle A.2: Resultate der Versuchsreihen mit Bits aus der Mitte der Hashwerte

Tabelle A.2 Fortsetzung der vorherigen Seite

Algorithmus	Hashlänge	Anzahl der Zusatzbits	Runden	Anzahl der Knoten in s.Z.	Anzahl der s.Z.	s.Z. Nummer	Größe der s.Z.
MD5	14	1	11	11016	7	7	11007
MD5	14	2	5	15219	1	1	15219
MD5	14	3	3	16318	1	1	16318
MD5	14	4	2	16383	1	1	16383
SHA-1	10	0	41	84	6	1	1
SHA-1	10	0	41	84	6	2	2
SHA-1	10	0	41	84	6	3	2
SHA-1	10	0	41	84	6	4	3
SHA-1	10	0	41	84	6	5	6
SHA-1	10	0	41	84	6	6	70
SHA-1	10	1	8	717	1	1	717
SHA-1	10	2	3	941	1	1	941
SHA-1	10	3	2	1018	1	1	1018
SHA-1	10	4	2	1024	1	1	1024
SHA-1	12	0	88	149	8	1	1
SHA-1	12	0	88	149	8	2	1
SHA-1	12	0	88	149	8	3	1
SHA-1	12	0	88	149	8	4	3
SHA-1	12	0	88	149	8	5	14
SHA-1	12	0	88	149	8	6	21
SHA-1	12	0	88	149	8	7	27
SHA-1	12	0	88	149	8	8	81
SHA-1	12	1	9	2743	1	1	2743
SHA-1	12	2	4	3805	1	1	3805
SHA-1	12	3	2	4076	1	1	4076
SHA-1	12	4	2	4096	1	1	4096
SHA-1	14	0	209	80	6	1	3
SHA-1	14	0	209	80	6	2	4
SHA-1	14	0	209	80	6	3	7
SHA-1	14	0	209	80	6	4	7
SHA-1	14	0	209	80	6	5	9
SHA-1	14	0	209	80	6	6	50
SHA-1	14	1	10	11086	1	1	11086
SHA-1	14	2	4	15204	1	1	15204
SHA-1	14	3	3	16310	1	1	16310
SHA-1	14	4	2	16384	1	1	16384
SHA256	10	0	87	17	4	1	1
SHA256	10	0	87	17	4	2	1
SHA256	10	0	87	17	4	3	1

Auf der nächsten Seite fortgesetzt

Tabelle A.2: Resultate der Versuchsreihen mit Bits aus der Mitte der Hashwerte

Tabelle A.2 Fortsetzung der vorherigen Seite

Algorithmus	Hashlänge	Anzahl der Zusatzbits	Runden	Anzahl der Knoten in s.Z.	Anzahl der s.Z.	s.Z. Nummer	Größe der s.Z.
SHA256	10	0	87	17	4	4	14
SHA256	10	1	6	708	3	1	1
SHA256	10	1	6	708	3	2	1
SHA256	10	1	6	708	3	3	706
SHA256	10	2	4	954	1	1	954
SHA256	10	3	2	1021	1	1	1021
SHA256	10	4	2	1024	1	1	1024
SHA256	12	0	146	48	4	1	1
SHA256	12	0	146	48	4	2	1
SHA256	12	0	146	48	4	3	5
SHA256	12	0	146	48	4	4	41
SHA256	12	1	10	2777	1	1	2777
SHA256	12	2	5	3818	1	1	3818
SHA256	12	3	2	4082	1	1	4082
SHA256	12	4	2	4096	1	1	4096
SHA256	14	0	237	55	4	1	1
SHA256	14	0	237	55	4	2	9
SHA256	14	0	237	55	4	3	20
SHA256	14	0	237	55	4	4	25
SHA256	14	1	13	11058	2	1	1
SHA256	14	1	13	11058	2	2	11057
SHA256	14	2	5	15172	1	1	15172
SHA256	14	3	3	16306	1	1	16306
SHA256	14	4	2	16382	1	1	16382

Ende der Tabelle

Tabelle A.2: Resultate der Versuchsreihen mit Bits aus der Mitte der Hashwerte

Tabelle A.3

Algorithmus	Hashlänge	Anzahl der Zusatzbits	Runden	Anzahl der Knoten in s.Z.	Anzahl der s.Z.	s.Z. Nummer	Größe der s.Z.
BLAKE256	10	0	55	29	6	1	1
BLAKE256	10	0	55	29	6	2	1
BLAKE256	10	0	55	29	6	3	1
BLAKE256	10	0	55	29	6	4	2
BLAKE256	10	0	55	29	6	5	2
BLAKE256	10	0	55	29	6	6	22

Auf der nächsten Seite fortgesetzt

Tabelle A.3: Resultate der Versuchsreihen mit Bits vom Ende der Hashwerte

Tabelle A.3 Fortsetzung der vorherigen Seite

Algo-rithmus	Hash-länge	Anzahl der Zu-satzbits	Runden	Anzahl der Knoten in s.Z.	Anzahl der s.Z.	s.Z. Nummer	Größe der s.Z.
BLAKE256	10	1	9	712	1	1	712
BLAKE256	10	2	3	961	1	1	961
BLAKE256	10	3	2	1020	1	1	1020
BLAKE256	10	4	2	1024	1	1	1024
BLAKE256	12	0	70	66	3	1	1
BLAKE256	12	0	70	66	3	2	13
BLAKE256	12	0	70	66	3	3	52
BLAKE256	12	1	11	2736	1	1	2736
BLAKE256	12	2	4	3829	1	1	3829
BLAKE256	12	3	2	4081	1	1	4081
BLAKE256	12	4	2	4095	1	1	4095
BLAKE256	14	0	232	95	7	1	1
BLAKE256	14	0	232	95	7	2	2
BLAKE256	14	0	232	95	7	3	5
BLAKE256	14	0	232	95	7	4	5
BLAKE256	14	0	232	95	7	5	12
BLAKE256	14	0	232	95	7	6	29
BLAKE256	14	0	232	95	7	7	41
BLAKE256	14	1	12	10920	3	1	1
BLAKE256	14	1	12	10920	3	2	2
BLAKE256	14	1	12	10920	3	3	10917
BLAKE256	14	2	5	15213	1	1	15213
BLAKE256	14	3	3	16315	1	1	16315
BLAKE256	14	4	2	16382	1	1	16382
Groestl256	10	0	51	24	4	1	3
Groestl256	10	0	51	24	4	2	3
Groestl256	10	0	51	24	4	3	4
Groestl256	10	0	51	24	4	4	14
Groestl256	10	1	7	691	1	1	691
Groestl256	10	2	3	960	1	1	960
Groestl256	10	3	2	1021	1	1	1021
Groestl256	10	4	2	1024	1	1	1024
Groestl256	12	0	97	230	7	1	1
Groestl256	12	0	97	230	7	2	4
Groestl256	12	0	97	230	7	3	5
Groestl256	12	0	97	230	7	4	14
Groestl256	12	0	97	230	7	5	17
Groestl256	12	0	97	230	7	6	32
Groestl256	12	0	97	230	7	7	157

Auf der nächsten Seite fortgesetzt

Tabelle A.3: Resultate der Versuchsreihen mit Bits vom Ende der Hashwerte

Tabelle A.3 Fortsetzung der vorherigen Seite

Algo-rithmus	Hash-länge	Anzahl der Zu-satzbits	Runden	Anzahl der Knoten in s.Z.	Anzahl der s.Z.	s.Z. Nummer	Größe der s.Z.
Groestl256	12	1	8	2786	2	1	1
Groestl256	12	1	8	2786	2	2	2785
Groestl256	12	2	4	3802	1	1	3802
Groestl256	12	3	2	4087	1	1	4087
Groestl256	12	4	2	4096	1	1	4096
Groestl256	14	0	320	147	6	1	4
Groestl256	14	0	320	147	6	2	5
Groestl256	14	0	320	147	6	3	8
Groestl256	14	0	320	147	6	4	20
Groestl256	14	0	320	147	6	5	35
Groestl256	14	0	320	147	6	6	75
Groestl256	14	1	10	11061	1	1	11061
Groestl256	14	2	6	15238	1	1	15238
Groestl256	14	3	3	16309	1	1	16309
Groestl256	14	4	2	16384	1	1	16384
JH256	10	0	67	29	2	1	1
JH256	10	0	67	29	2	2	28
JH256	10	1	7	709	1	1	709
JH256	10	2	3	958	1	1	958
JH256	10	3	2	1021	1	1	1021
JH256	10	4	2	1024	1	1	1024
JH256	12	0	179	12	1	1	12
JH256	12	1	10	2806	1	1	2806
JH256	12	2	4	3810	1	1	3810
JH256	12	3	2	4068	1	1	4068
JH256	12	4	2	4095	1	1	4095
JH256	14	0	330	183	4	1	2
JH256	14	0	330	183	4	2	23
JH256	14	0	330	183	4	3	53
JH256	14	0	330	183	4	4	105
JH256	14	1	10	11140	2	1	1
JH256	14	1	10	11140	2	2	11139
JH256	14	2	5	15138	1	1	15138
JH256	14	3	2	16311	1	1	16311
JH256	14	4	2	16384	1	1	16384
KECCAK256	10	0	63	9	3	1	1
KECCAK256	10	0	63	9	3	2	2
KECCAK256	10	0	63	9	3	3	6
KECCAK256	10	1	6	687	2	1	1

Auf der nächsten Seite fortgesetzt

Tabelle A.3: Resultate der Versuchsreihen mit Bits vom Ende der Hashwerte

Tabelle A.3 Fortsetzung der vorherigen Seite

Algorithmus	Hashlänge	Anzahl der Zusatzbits	Runden	Anzahl der Knoten in s.Z.	Anzahl der s.Z.	s.Z. Nummer	Größe der s.Z.
KECCAK256	10	1	6	687	2	2	686
KECCAK256	10	2	4	950	1	1	950
KECCAK256	10	3	2	1019	1	1	1019
KECCAK256	10	4	2	1024	1	1	1024
KECCAK256	12	0	90	125	7	1	1
KECCAK256	12	0	90	125	7	2	2
KECCAK256	12	0	90	125	7	3	9
KECCAK256	12	0	90	125	7	4	9
KECCAK256	12	0	90	125	7	5	19
KECCAK256	12	0	90	125	7	6	27
KECCAK256	12	0	90	125	7	7	58
KECCAK256	12	1	8	2783	1	1	2783
KECCAK256	12	2	4	3814	1	1	3814
KECCAK256	12	3	3	4081	1	1	4081
KECCAK256	12	4	2	4096	1	1	4096
KECCAK256	14	0	141	150	5	1	1
KECCAK256	14	0	141	150	5	2	2
KECCAK256	14	0	141	150	5	3	10
KECCAK256	14	0	141	150	5	4	23
KECCAK256	14	0	141	150	5	5	114
KECCAK256	14	1	9	11060	3	1	1
KECCAK256	14	1	9	11060	3	2	1
KECCAK256	14	1	9	11060	3	3	11058
KECCAK256	14	2	4	15215	1	1	15215
KECCAK256	14	3	2	16308	1	1	16308
KECCAK256	14	4	2	16384	1	1	16384
RIPEMD160	10	0	88	3	2	1	1
RIPEMD160	10	0	88	3	2	2	2
RIPEMD160	10	1	9	676	1	1	676
RIPEMD160	10	2	3	950	1	1	950
RIPEMD160	10	3	2	1017	1	1	1017
RIPEMD160	10	4	2	1024	1	1	1024
RIPEMD160	12	0	120	82	7	1	1
RIPEMD160	12	0	120	82	7	2	1
RIPEMD160	12	0	120	82	7	3	10
RIPEMD160	12	0	120	82	7	4	10
RIPEMD160	12	0	120	82	7	5	17
RIPEMD160	12	0	120	82	7	6	20
RIPEMD160	12	0	120	82	7	7	23

Auf der nächsten Seite fortgesetzt

Tabelle A.3: Resultate der Versuchsreihen mit Bits vom Ende der Hashwerte

Tabelle A.3 Fortsetzung der vorherigen Seite

Algo-rithmus	Hash-länge	Anzahl der Zu-satzbits	Runden	Anzahl der Knoten in s.Z.	Anzahl der s.Z.	s.Z. Nummer	Größe der s.Z.
RIPEMD160	12	1	7	2768	1	1	2768
RIPEMD160	12	2	4	3826	1	1	3826
RIPEMD160	12	3	2	4076	1	1	4076
RIPEMD160	12	4	2	4096	1	1	4096
RIPEMD160	14	0	225	49	7	1	1
RIPEMD160	14	0	225	49	7	2	1
RIPEMD160	14	0	225	49	7	3	2
RIPEMD160	14	0	225	49	7	4	2
RIPEMD160	14	0	225	49	7	5	4
RIPEMD160	14	0	225	49	7	6	4
RIPEMD160	14	0	225	49	7	7	35
RIPEMD160	14	1	11	11030	3	1	1
RIPEMD160	14	1	11	11030	3	2	2
RIPEMD160	14	1	11	11030	3	3	11027
RIPEMD160	14	2	6	15196	1	1	15196
RIPEMD160	14	3	2	16299	1	1	16299
RIPEMD160	14	4	2	16384	1	1	16384
Skein256	10	0	67	53	4	1	1
Skein256	10	0	67	53	4	2	8
Skein256	10	0	67	53	4	3	17
Skein256	10	0	67	53	4	4	27
Skein256	10	1	8	715	2	1	1
Skein256	10	1	8	715	2	2	714
Skein256	10	2	3	975	1	1	975
Skein256	10	3	2	1021	1	1	1021
Skein256	10	4	2	1024	1	1	1024
Skein256	12	0	94	80	4	1	1
Skein256	12	0	94	80	4	2	1
Skein256	12	0	94	80	4	3	26
Skein256	12	0	94	80	4	4	52
Skein256	12	1	9	2766	1	1	2766
Skein256	12	2	4	3823	2	1	1
Skein256	12	2	4	3823	2	2	3822
Skein256	12	3	3	4078	1	1	4078
Skein256	12	4	2	4096	1	1	4096
Skein256	14	0	282	117	10	1	1
Skein256	14	0	282	117	10	2	1
Skein256	14	0	282	117	10	3	2
Skein256	14	0	282	117	10	4	2

Auf der nächsten Seite fortgesetzt

Tabelle A.3: Resultate der Versuchsreihen mit Bits vom Ende der Hashwerte

Tabelle A.3 Fortsetzung der vorherigen Seite

Algo-rithmus	Hash-länge	Anzahl der Zu-satzbits	Runden	Anzahl der Knoten in s.Z.	Anzahl der s.Z.	s.Z. Nummer	Größe der s.Z.
Skein256	14	0	282	117	10	5	4
Skein256	14	0	282	117	10	6	5
Skein256	14	0	282	117	10	7	10
Skein256	14	0	282	117	10	8	16
Skein256	14	0	282	117	10	9	30
Skein256	14	0	282	117	10	10	46
Skein256	14	1	10	11215	2	1	1
Skein256	14	1	10	11215	2	2	11214
Skein256	14	2	5	15222	1	1	15222
Skein256	14	3	2	16307	1	1	16307
Skein256	14	4	2	16384	1	1	16384
MD5	10	0	74	33	7	1	1
MD5	10	0	74	33	7	2	1
MD5	10	0	74	33	7	3	1
MD5	10	0	74	33	7	4	3
MD5	10	0	74	33	7	5	5
MD5	10	0	74	33	7	6	11
MD5	10	0	74	33	7	7	11
MD5	10	1	12	684	2	1	1
MD5	10	1	12	684	2	2	683
MD5	10	2	5	944	4	1	1
MD5	10	2	5	944	4	2	1
MD5	10	2	5	944	4	3	1
MD5	10	2	5	944	4	4	941
MD5	10	3	2	1022	1	1	1022
MD5	10	4	2	1024	1	1	1024
MD5	12	0	82	48	8	1	1
MD5	12	0	82	48	8	2	1
MD5	12	0	82	48	8	3	2
MD5	12	0	82	48	8	4	2
MD5	12	0	82	48	8	5	3
MD5	12	0	82	48	8	6	5
MD5	12	0	82	48	8	7	8
MD5	12	0	82	48	8	8	26
MD5	12	1	10	2789	2	1	1
MD5	12	1	10	2789	2	2	2788
MD5	12	2	4	3821	1	1	3821
MD5	12	3	2	4079	1	1	4079
MD5	12	4	2	4096	1	1	4096

Auf der nächsten Seite fortgesetzt

Tabelle A.3: Resultate der Versuchsreihen mit Bits vom Ende der Hashwerte

Tabelle A.3 Fortsetzung der vorherigen Seite

Algorithmus	Hashlänge	Anzahl der Zusatzbits	Runden	Anzahl der Knoten in s.Z.	Anzahl der s.Z.	s.Z. Nummer	Größe der s.Z.
MD5	14	0	259	185	8	1	1
MD5	14	0	259	185	8	2	2
MD5	14	0	259	185	8	3	7
MD5	14	0	259	185	8	4	7
MD5	14	0	259	185	8	5	8
MD5	14	0	259	185	8	6	9
MD5	14	0	259	185	8	7	59
MD5	14	0	259	185	8	8	92
MD5	14	1	10	11117	2	1	1
MD5	14	1	10	11117	2	2	11116
MD5	14	2	5	15201	1	1	15201
MD5	14	3	2	16308	1	1	16308
MD5	14	4	2	16384	1	1	16384
SHA-1	10	0	45	38	3	1	6
SHA-1	10	0	45	38	3	2	9
SHA-1	10	0	45	38	3	3	23
SHA-1	10	1	7	688	2	1	2
SHA-1	10	1	7	688	2	2	686
SHA-1	10	2	3	947	1	1	947
SHA-1	10	3	2	1022	1	1	1022
SHA-1	10	4	2	1024	1	1	1024
SHA-1	12	0	116	49	7	1	2
SHA-1	12	0	116	49	7	2	3
SHA-1	12	0	116	49	7	3	3
SHA-1	12	0	116	49	7	4	4
SHA-1	12	0	116	49	7	5	5
SHA-1	12	0	116	49	7	6	12
SHA-1	12	0	116	49	7	7	20
SHA-1	12	1	9	2810	1	1	2810
SHA-1	12	2	6	3823	1	1	3823
SHA-1	12	3	2	4079	1	1	4079
SHA-1	12	4	2	4096	1	1	4096
SHA-1	14	0	254	212	10	1	1
SHA-1	14	0	254	212	10	2	2
SHA-1	14	0	254	212	10	3	4
SHA-1	14	0	254	212	10	4	5
SHA-1	14	0	254	212	10	5	5
SHA-1	14	0	254	212	10	6	10
SHA-1	14	0	254	212	10	7	10

Auf der nächsten Seite fortgesetzt

Tabelle A.3: Resultate der Versuchsreihen mit Bits vom Ende der Hashwerte

Tabelle A.3 Fortsetzung der vorherigen Seite

Algo-rithmus	Hash-länge	Anzahl der Zu-satzbits	Runden	Anzahl der Knoten in s.Z.	Anzahl der s.Z.	s.Z. Nummer	Größe der s.Z.
SHA-1	14	0	254	212	10	8	46
SHA-1	14	0	254	212	10	9	61
SHA-1	14	0	254	212	10	10	68
SHA-1	14	1	12	10978	1	1	10978
SHA-1	14	2	5	15263	1	1	15263
SHA-1	14	3	3	16306	1	1	16306
SHA-1	14	4	2	16383	1	1	16383
SHA256	10	0	60	18	3	1	1
SHA256	10	0	60	18	3	2	1
SHA256	10	0	60	18	3	3	16
SHA256	10	1	10	688	1	1	688
SHA256	10	2	3	946	1	1	946
SHA256	10	3	3	1021	1	1	1021
SHA256	10	4	2	1024	1	1	1024
SHA256	12	0	73	98	4	1	1
SHA256	12	0	73	98	4	2	5
SHA256	12	0	73	98	4	3	9
SHA256	12	0	73	98	4	4	83
SHA256	12	1	9	2739	1	1	2739
SHA256	12	2	4	3775	1	1	3775
SHA256	12	3	2	4078	1	1	4078
SHA256	12	4	2	4096	1	1	4096
SHA256	14	0	257	46	4	1	1
SHA256	14	0	257	46	4	2	1
SHA256	14	0	257	46	4	3	8
SHA256	14	0	257	46	4	4	36
SHA256	14	1	8	11112	1	1	11112
SHA256	14	2	6	15209	1	1	15209
SHA256	14	3	2	16306	1	1	16306
SHA256	14	4	2	16384	1	1	16384

Ende der Tabelle

Tabelle A.3: Resultate der Versuchsreihen mit Bits vom Ende der Hashwerte

Tabelle A.4

Reduktions-art	Algo-rithmus	Länge	Anzahl der Zusatzbits	Durchschnittliche Größe der starken Zusammen-hangskomponenten
Ende	BLAKE256	10	0	4,83
Ende	BLAKE256	10	1	712,00
Ende	BLAKE256	10	2	961,00
Ende	BLAKE256	10	3	1 020,00
Ende	BLAKE256	10	4	1 024,00
Ende	BLAKE256	12	0	22,00
Ende	BLAKE256	12	1	2 736,00
Ende	BLAKE256	12	2	3 829,00
Ende	BLAKE256	12	3	4 081,00
Ende	BLAKE256	12	4	4 095,00
Ende	BLAKE256	14	0	13,57
Ende	BLAKE256	14	1	3 640,00
Ende	BLAKE256	14	2	15 213,00
Ende	BLAKE256	14	3	16 315,00
Ende	BLAKE256	14	4	16 382,00
Ende	Groestl256	10	0	6,00
Ende	Groestl256	10	1	691,00
Ende	Groestl256	10	2	960,00
Ende	Groestl256	10	3	1 021,00
Ende	Groestl256	10	4	1 024,00
Ende	Groestl256	12	0	32,86
Ende	Groestl256	12	1	1 393,00
Ende	Groestl256	12	2	3 802,00
Ende	Groestl256	12	3	4 087,00
Ende	Groestl256	12	4	4 096,00
Ende	Groestl256	14	0	24,50
Ende	Groestl256	14	1	11 061,00
Ende	Groestl256	14	2	15 238,00
Ende	Groestl256	14	3	16 309,00
Ende	Groestl256	14	4	16 384,00
Ende	JH256	10	0	14,50
Ende	JH256	10	1	709,00
Ende	JH256	10	2	958,00
Ende	JH256	10	3	1 021,00
Ende	JH256	10	4	1 024,00
Ende	JH256	12	0	12,00
Ende	JH256	12	1	2 806,00
Ende	JH256	12	2	3 810,00
Ende	JH256	12	3	4 068,00

Auf der nächsten Seite fortgesetzt

Tabelle A.4: Durchschnittsgröße der starken Zusammenhangskomponenten

Tabelle A.4 Fortsetzung der vorherigen Seite

Reduktions-art	Algo-rithmus	Länge	Anzahl der Zusatzbits	Durchschnittliche Größe der starken Zusammen-hangskomponenten
Ende	JH256	12	4	4 095,00
Ende	JH256	14	0	45,75
Ende	JH256	14	1	5 570,00
Ende	JH256	14	2	15 138,00
Ende	JH256	14	3	16 311,00
Ende	JH256	14	4	16 384,00
Ende	KECCAK256	10	0	3,00
Ende	KECCAK256	10	1	343,50
Ende	KECCAK256	10	2	950,00
Ende	KECCAK256	10	3	1 019,00
Ende	KECCAK256	10	4	1 024,00
Ende	KECCAK256	12	0	17,86
Ende	KECCAK256	12	1	2 783,00
Ende	KECCAK256	12	2	3 814,00
Ende	KECCAK256	12	3	4 081,00
Ende	KECCAK256	12	4	4 096,00
Ende	KECCAK256	14	0	30,00
Ende	KECCAK256	14	1	3 686,67
Ende	KECCAK256	14	2	15 215,00
Ende	KECCAK256	14	3	16 308,00
Ende	KECCAK256	14	4	16 384,00
Ende	RIPEMD160	10	0	1,50
Ende	RIPEMD160	10	1	676,00
Ende	RIPEMD160	10	2	950,00
Ende	RIPEMD160	10	3	1 017,00
Ende	RIPEMD160	10	4	1 024,00
Ende	RIPEMD160	12	0	11,71
Ende	RIPEMD160	12	1	2 768,00
Ende	RIPEMD160	12	2	3 826,00
Ende	RIPEMD160	12	3	4 076,00
Ende	RIPEMD160	12	4	4 096,00
Ende	RIPEMD160	14	0	7,00
Ende	RIPEMD160	14	1	3 676,67
Ende	RIPEMD160	14	2	15 196,00
Ende	RIPEMD160	14	3	16 299,00
Ende	RIPEMD160	14	4	16 384,00
Ende	Skein256	10	0	13,25
Ende	Skein256	10	1	357,50
Ende	Skein256	10	2	975,00

Auf der nächsten Seite fortgesetzt

Tabelle A.4: Durchschnittsgröße der starken Zusammenhangskomponenten

Tabelle A.4 Fortsetzung der vorherigen Seite

Reduktions-art	Algo-rithmus	Länge	Anzahl der Zusatzbits	Durchschnittliche Größe der starken Zusammen-hangskomponenten
Ende	Skein256	10	3	1 021,00
Ende	Skein256	10	4	1 024,00
Ende	Skein256	12	0	20,00
Ende	Skein256	12	1	2 766,00
Ende	Skein256	12	2	1 911,50
Ende	Skein256	12	3	4 078,00
Ende	Skein256	12	4	4 096,00
Ende	Skein256	14	0	11,70
Ende	Skein256	14	1	5 607,50
Ende	Skein256	14	2	15 222,00
Ende	Skein256	14	3	16 307,00
Ende	Skein256	14	4	16 384,00
Ende	MD5	10	0	4,71
Ende	MD5	10	1	342,00
Ende	MD5	10	2	236,00
Ende	MD5	10	3	1 022,00
Ende	MD5	10	4	1 024,00
Ende	MD5	12	0	6,00
Ende	MD5	12	1	1 394,50
Ende	MD5	12	2	3 821,00
Ende	MD5	12	3	4 079,00
Ende	MD5	12	4	4 096,00
Ende	MD5	14	0	23,13
Ende	MD5	14	1	5 558,50
Ende	MD5	14	2	15 201,00
Ende	MD5	14	3	16 308,00
Ende	MD5	14	4	16 384,00
Ende	SHA-1	10	0	12,67
Ende	SHA-1	10	1	344,00
Ende	SHA-1	10	2	947,00
Ende	SHA-1	10	3	1 022,00
Ende	SHA-1	10	4	1 024,00
Ende	SHA-1	12	0	7,00
Ende	SHA-1	12	1	2 810,00
Ende	SHA-1	12	2	3 823,00
Ende	SHA-1	12	3	4 079,00
Ende	SHA-1	12	4	4 096,00
Ende	SHA-1	14	0	21,20
Ende	SHA-1	14	1	10 978,00

Auf der nächsten Seite fortgesetzt

Tabelle A.4: Durchschnittsgröße der starken Zusammenhangskomponenten

Tabelle A.4 Fortsetzung der vorherigen Seite

Reduktions-art	Algo-rithmus	Länge	Anzahl der Zusatzbits	Durchschnittliche Größe der starken Zusammen-hangskomponenten
Ende	SHA-1	14	2	15 263,00
Ende	SHA-1	14	3	16 306,00
Ende	SHA-1	14	4	16 383,00
Ende	SHA256	10	0	6,00
Ende	SHA256	10	1	688,00
Ende	SHA256	10	2	946,00
Ende	SHA256	10	3	1 021,00
Ende	SHA256	10	4	1 024,00
Ende	SHA256	12	0	24,50
Ende	SHA256	12	1	2 739,00
Ende	SHA256	12	2	3 775,00
Ende	SHA256	12	3	4 078,00
Ende	SHA256	12	4	4 096,00
Ende	SHA256	14	0	11,50
Ende	SHA256	14	1	11 112,00
Ende	SHA256	14	2	15 209,00
Ende	SHA256	14	3	16 306,00
Ende	SHA256	14	4	16 384,00
Mitte	BLAKE256	10	0	6,75
Mitte	BLAKE256	10	1	699,00
Mitte	BLAKE256	10	2	948,00
Mitte	BLAKE256	10	3	1 016,00
Mitte	BLAKE256	10	4	1 024,00
Mitte	BLAKE256	12	0	32,50
Mitte	BLAKE256	12	1	1 395,00
Mitte	BLAKE256	12	2	3 767,00
Mitte	BLAKE256	12	3	4 073,00
Mitte	BLAKE256	12	4	4 096,00
Mitte	BLAKE256	14	0	52,00
Mitte	BLAKE256	14	1	11 046,00
Mitte	BLAKE256	14	2	15 117,00
Mitte	BLAKE256	14	3	16 306,00
Mitte	BLAKE256	14	4	16 384,00
Mitte	Groestl256	10	0	10,50
Mitte	Groestl256	10	1	653,00
Mitte	Groestl256	10	2	941,00
Mitte	Groestl256	10	3	1 018,00
Mitte	Groestl256	10	4	1 024,00
Mitte	Groestl256	12	0	53,00

Auf der nächsten Seite fortgesetzt

Tabelle A.4: Durchschnittsgröße der starken Zusammenhangskomponenten

Tabelle A.4 Fortsetzung der vorherigen Seite

Reduktions-art	Algo-rithmus	Länge	Anzahl der Zusatzbits	Durchschnittliche Größe der starken Zusammen-hangskomponenten
Mitte	Groestl256	12	1	2 693,00
Mitte	Groestl256	12	2	3 785,00
Mitte	Groestl256	12	3	4 082,00
Mitte	Groestl256	12	4	4 096,00
Mitte	Groestl256	14	0	13,00
Mitte	Groestl256	14	1	11 033,00
Mitte	Groestl256	14	2	15 180,00
Mitte	Groestl256	14	3	16 312,00
Mitte	Groestl256	14	4	16 384,00
Mitte	JH256	10	0	16,50
Mitte	JH256	10	1	231,00
Mitte	JH256	10	2	470,50
Mitte	JH256	10	3	1 020,00
Mitte	JH256	10	4	1 024,00
Mitte	JH256	12	0	19,14
Mitte	JH256	12	1	2 785,00
Mitte	JH256	12	2	3 823,00
Mitte	JH256	12	3	4 078,00
Mitte	JH256	12	4	4 096,00
Mitte	JH256	14	0	73,67
Mitte	JH256	14	1	3 703,00
Mitte	JH256	14	2	15 202,00
Mitte	JH256	14	3	16 297,00
Mitte	JH256	14	4	16 384,00
Mitte	KECCAK256	10	0	5,67
Mitte	KECCAK256	10	1	228,33
Mitte	KECCAK256	10	2	931,00
Mitte	KECCAK256	10	3	1 017,00
Mitte	KECCAK256	10	4	1 024,00
Mitte	KECCAK256	12	0	17,00
Mitte	KECCAK256	12	1	913,67
Mitte	KECCAK256	12	2	3 770,00
Mitte	KECCAK256	12	3	4 075,00
Mitte	KECCAK256	12	4	4 096,00
Mitte	KECCAK256	14	0	18,08
Mitte	KECCAK256	14	1	5 535,50
Mitte	KECCAK256	14	2	15 213,00
Mitte	KECCAK256	14	3	16 302,00
Mitte	KECCAK256	14	4	16 384,00

Auf der nächsten Seite fortgesetzt

Tabelle A.4: Durchschnittsgröße der starken Zusammenhangskomponenten

Tabelle A.4 Fortsetzung der vorherigen Seite

Reduktions-art	Algo-rithmus	Länge	Anzahl der Zusatzbits	Durchschnittliche Größe der starken Zusammen-hangskomponenten
Mitte	RIPEMD160	10	0	22,33
Mitte	RIPEMD160	10	1	694,00
Mitte	RIPEMD160	10	2	960,00
Mitte	RIPEMD160	10	3	1 020,00
Mitte	RIPEMD160	10	4	1 024,00
Mitte	RIPEMD160	12	0	14,33
Mitte	RIPEMD160	12	1	2 740,00
Mitte	RIPEMD160	12	2	1 897,00
Mitte	RIPEMD160	12	3	4 083,00
Mitte	RIPEMD160	12	4	4 096,00
Mitte	RIPEMD160	14	0	63,50
Mitte	RIPEMD160	14	1	2 789,25
Mitte	RIPEMD160	14	2	15 254,00
Mitte	RIPEMD160	14	3	16 308,00
Mitte	RIPEMD160	14	4	16 384,00
Mitte	Skein256	10	0	32,00
Mitte	Skein256	10	1	660,00
Mitte	Skein256	10	2	952,00
Mitte	Skein256	10	3	1 022,00
Mitte	Skein256	10	4	1 024,00
Mitte	Skein256	12	0	11,56
Mitte	Skein256	12	1	927,33
Mitte	Skein256	12	2	1 888,00
Mitte	Skein256	12	3	4 072,00
Mitte	Skein256	12	4	4 096,00
Mitte	Skein256	14	0	18,67
Mitte	Skein256	14	1	5 498,50
Mitte	Skein256	14	2	7 592,00
Mitte	Skein256	14	3	16 318,00
Mitte	Skein256	14	4	16 383,00
Mitte	MD5	10	0	22,25
Mitte	MD5	10	1	709,00
Mitte	MD5	10	2	959,00
Mitte	MD5	10	3	1 017,00
Mitte	MD5	10	4	1 023,00
Mitte	MD5	12	0	16,33
Mitte	MD5	12	1	935,00
Mitte	MD5	12	2	3 837,00
Mitte	MD5	12	3	4 080,00

Auf der nächsten Seite fortgesetzt

Tabelle A.4: Durchschnittsgröße der starken Zusammenhangskomponenten

Tabelle A.4 Fortsetzung der vorherigen Seite

Reduktions-art	Algo-rithmus	Länge	Anzahl der Zusatzbits	Durchschnittliche Größe der starken Zusammen-hangskomponenten
Mitte	MD5	12	4	4 096,00
Mitte	MD5	14	0	20,10
Mitte	MD5	14	1	1 573,71
Mitte	MD5	14	2	15 219,00
Mitte	MD5	14	3	16 318,00
Mitte	MD5	14	4	16 383,00
Mitte	SHA-1	10	0	14,00
Mitte	SHA-1	10	1	717,00
Mitte	SHA-1	10	2	941,00
Mitte	SHA-1	10	3	1 018,00
Mitte	SHA-1	10	4	1 024,00
Mitte	SHA-1	12	0	18,63
Mitte	SHA-1	12	1	2 743,00
Mitte	SHA-1	12	2	3 805,00
Mitte	SHA-1	12	3	4 076,00
Mitte	SHA-1	12	4	4 096,00
Mitte	SHA-1	14	0	13,33
Mitte	SHA-1	14	1	11 086,00
Mitte	SHA-1	14	2	15 204,00
Mitte	SHA-1	14	3	16 310,00
Mitte	SHA-1	14	4	16 384,00
Mitte	SHA256	10	0	4,25
Mitte	SHA256	10	1	236,00
Mitte	SHA256	10	2	954,00
Mitte	SHA256	10	3	1 021,00
Mitte	SHA256	10	4	1 024,00
Mitte	SHA256	12	0	12,00
Mitte	SHA256	12	1	2 777,00
Mitte	SHA256	12	2	3 818,00
Mitte	SHA256	12	3	4 082,00
Mitte	SHA256	12	4	4 096,00
Mitte	SHA256	14	0	13,75
Mitte	SHA256	14	1	5 529,00
Mitte	SHA256	14	2	15 172,00
Mitte	SHA256	14	3	16 306,00
Mitte	SHA256	14	4	16 382,00
Anfang	BLAKE256	10	0	3,75
Anfang	BLAKE256	10	1	715,00
Anfang	BLAKE256	10	2	942,00

Auf der nächsten Seite fortgesetzt

Tabelle A.4: Durchschnittsgröße der starken Zusammenhangskomponenten

Tabelle A.4 Fortsetzung der vorherigen Seite

Reduktions-art	Algo-rithmus	Länge	Anzahl der Zusatzbits	Durchschnittliche Größe der starken Zusammen-hangskomponenten
Anfang	BLAKE256	10	3	1 020,00
Anfang	BLAKE256	10	4	1 024,00
Anfang	BLAKE256	12	0	9,00
Anfang	BLAKE256	12	1	2 861,00
Anfang	BLAKE256	12	2	3 811,00
Anfang	BLAKE256	12	3	4 076,00
Anfang	BLAKE256	12	4	4 096,00
Anfang	BLAKE256	14	0	12,67
Anfang	BLAKE256	14	1	5 543,50
Anfang	BLAKE256	14	2	15 158,00
Anfang	BLAKE256	14	3	16 311,00
Anfang	BLAKE256	14	4	16 383,00
Anfang	Groestl256	10	0	7,80
Anfang	Groestl256	10	1	720,00
Anfang	Groestl256	10	2	954,00
Anfang	Groestl256	10	3	1 017,00
Anfang	Groestl256	10	4	1 024,00
Anfang	Groestl256	12	0	38,00
Anfang	Groestl256	12	1	1 392,50
Anfang	Groestl256	12	2	1 906,00
Anfang	Groestl256	12	3	4 086,00
Anfang	Groestl256	12	4	4 096,00
Anfang	Groestl256	14	0	18,14
Anfang	Groestl256	14	1	5 530,00
Anfang	Groestl256	14	2	15 228,00
Anfang	Groestl256	14	3	16 316,00
Anfang	Groestl256	14	4	16 384,00
Anfang	JH256	10	0	11,00
Anfang	JH256	10	1	176,00
Anfang	JH256	10	2	968,00
Anfang	JH256	10	3	1 022,00
Anfang	JH256	10	4	1 024,00
Anfang	JH256	12	0	10,83
Anfang	JH256	12	1	2 732,00
Anfang	JH256	12	2	3 778,00
Anfang	JH256	12	3	4 068,00
Anfang	JH256	12	4	4 096,00
Anfang	JH256	14	0	12,00
Anfang	JH256	14	1	11 004,00

Auf der nächsten Seite fortgesetzt

Tabelle A.4: Durchschnittsgröße der starken Zusammenhangskomponenten

Tabelle A.4 Fortsetzung der vorherigen Seite

Reduktions-art	Algo-rithmus	Länge	Anzahl der Zusatzbits	Durchschnittliche Größe der starken Zusammen-hangskomponenten
Anfang	JH256	14	2	15 115,00
Anfang	JH256	14	3	16 303,00
Anfang	JH256	14	4	16 384,00
Anfang	KECCAK256	10	0	11,40
Anfang	KECCAK256	10	1	712,00
Anfang	KECCAK256	10	2	957,00
Anfang	KECCAK256	10	3	1 019,00
Anfang	KECCAK256	10	4	1 024,00
Anfang	KECCAK256	12	0	48,50
Anfang	KECCAK256	12	1	1 400,00
Anfang	KECCAK256	12	2	3 786,00
Anfang	KECCAK256	12	3	4 076,00
Anfang	KECCAK256	12	4	4 096,00
Anfang	KECCAK256	14	0	22,63
Anfang	KECCAK256	14	1	2 723,50
Anfang	KECCAK256	14	2	15 127,00
Anfang	KECCAK256	14	3	16 309,00
Anfang	KECCAK256	14	4	16 383,00
Anfang	RIPEMD160	10	0	8,67
Anfang	RIPEMD160	10	1	692,00
Anfang	RIPEMD160	10	2	952,00
Anfang	RIPEMD160	10	3	1 018,00
Anfang	RIPEMD160	10	4	1 023,00
Anfang	RIPEMD160	12	0	11,20
Anfang	RIPEMD160	12	1	2 780,00
Anfang	RIPEMD160	12	2	3 824,00
Anfang	RIPEMD160	12	3	4 072,00
Anfang	RIPEMD160	12	4	4 096,00
Anfang	RIPEMD160	14	0	38,00
Anfang	RIPEMD160	14	1	11 017,00
Anfang	RIPEMD160	14	2	15 193,00
Anfang	RIPEMD160	14	3	16 296,00
Anfang	RIPEMD160	14	4	16 383,00
Anfang	Skein256	10	0	7,71
Anfang	Skein256	10	1	235,33
Anfang	Skein256	10	2	955,00
Anfang	Skein256	10	3	1 021,00
Anfang	Skein256	10	4	1 024,00
Anfang	Skein256	12	0	41,00

Auf der nächsten Seite fortgesetzt

Tabelle A.4: Durchschnittsgröße der starken Zusammenhangskomponenten

Tabelle A.4 Fortsetzung der vorherigen Seite

Reduktions-art	Algo-rithmus	Länge	Anzahl der Zusatzbits	Durchschnittliche Größe der starken Zusammen-hangskomponenten
Anfang	Skein256	12	1	2 748,00
Anfang	Skein256	12	2	3 825,00
Anfang	Skein256	12	3	4 081,00
Anfang	Skein256	12	4	4 096,00
Anfang	Skein256	14	0	51,20
Anfang	Skein256	14	1	11 011,00
Anfang	Skein256	14	2	15 190,00
Anfang	Skein256	14	3	16 299,00
Anfang	Skein256	14	4	16 384,00
Anfang	MD5	10	0	7,00
Anfang	MD5	10	1	352,50
Anfang	MD5	10	2	939,00
Anfang	MD5	10	3	1 019,00
Anfang	MD5	10	4	1 024,00
Anfang	MD5	12	0	11,71
Anfang	MD5	12	1	2 758,00
Anfang	MD5	12	2	3 804,00
Anfang	MD5	12	3	4 075,00
Anfang	MD5	12	4	4 096,00
Anfang	MD5	14	0	28,83
Anfang	MD5	14	1	2 771,25
Anfang	MD5	14	2	15 236,00
Anfang	MD5	14	3	16 305,00
Anfang	MD5	14	4	16 383,00
Anfang	SHA-1	10	0	6,60
Anfang	SHA-1	10	1	705,00
Anfang	SHA-1	10	2	954,00
Anfang	SHA-1	10	3	1 021,00
Anfang	SHA-1	10	4	1 024,00
Anfang	SHA-1	12	0	21,60
Anfang	SHA-1	12	1	683,00
Anfang	SHA-1	12	2	3 810,00
Anfang	SHA-1	12	3	4 082,00
Anfang	SHA-1	12	4	4 096,00
Anfang	SHA-1	14	0	11,17
Anfang	SHA-1	14	1	11 221,00
Anfang	SHA-1	14	2	15 232,00
Anfang	SHA-1	14	3	16 303,00
Anfang	SHA-1	14	4	16 383,00

Auf der nächsten Seite fortgesetzt

Tabelle A.4: Durchschnittsgröße der starken Zusammenhangskomponenten

Tabelle A.4 Fortsetzung der vorherigen Seite

Reduktions-art	Algo-rithmus	Länge	Anzahl der Zusatzbits	Durchschnittliche Größe der starken Zusammen-hangskomponenten
Anfang	SHA256	10	0	6,67
Anfang	SHA256	10	1	336,50
Anfang	SHA256	10	2	949,00
Anfang	SHA256	10	3	1 017,00
Anfang	SHA256	10	4	1 024,00
Anfang	SHA256	12	0	23,17
Anfang	SHA256	12	1	2 808,00
Anfang	SHA256	12	2	3 829,00
Anfang	SHA256	12	3	4 077,00
Anfang	SHA256	12	4	4 096,00
Anfang	SHA256	14	0	36,00
Anfang	SHA256	14	1	3 699,00
Anfang	SHA256	14	2	15 229,00
Anfang	SHA256	14	3	16 320,00
Anfang	SHA256	14	4	16 384,00

Ende der Tabelle

Tabelle A.4: Durchschnittsgröße der starken Zusammenhangskomponenten

Anhang B

DVD Inhalt

Auf der beigefügten DVD befinden sich folgende Dateien:

- Eine digitale Version dieser Bachelorarbeit im PDF Format mit dem Namen *bachelorarbeit_ulrich_viefhaus_7679963.pdf*. Sie befindet sich im Wurzelverzeichniss der DVD.
- Die beiden Testprogramme als ausführbare *.jar*. *ImageMaker.jar* berechnet die Hashwerte. *ImageViewer.jar* führt die Analysen durch. Die Programme wurden unter Debian GNU/Linux mit Eclipse SDK 3.5.2 und dem Java 7 OpenJDK entwickelt und getestet. Unter Java 6 stehen einige der verwendeten Funktionen der Standardbibliotheken nicht zu Verfügung. Obwohl Java auf vielen Plattformen lauffähig ist, wurde nicht getestet, ob das Progamm auf anderen Plattformen als die oben genannte identisch funktioniert. Die Programme benötigen je nach Konfiguration mehr Speicher, als die JRE standardmäßig zur Verfügung stelle. Der ImageMaker wurde in den Versuchen mit den Parametern *-Xms1024m -Xmx3096m* gestartet. Für den ImageViewer wurden die Parameter *-Xms1024m -Xmx4096m -Xss1024m* genutzt. An den ImageMaker kann über die Kommandozeile ein Liste mit Konfigurationsdateien oder Ordnern, die diese enthalten, übergeben werden. Ansonsten sucht er nach den Dateien im Ordner *conf.d*. An den ImageViewer kann eine Liste mit *.csv* Dateien übergeben werden, die analysiert werden sollen. Ansonsten durchsucht er den Ordner *data* nach Dateien, wobei nur die lexikographisch letzte Datei aus jedem Ordner verwendet wird. Normalerweise ist dies die Datei, in denen der Graph nur noch starke Zusammenhangskomponenten enthält.
- Der Quellcode der Testprogramme. Dieser befindet sich in dem Ordner */src/*.
- Die Javadocs der Testprogramme befinden sich in dem Ordner */doc/*.
- Die verwendeten freien Bibliotheken, auf die die Testprogramme zurückgreifen. Diese befinden sich in dem Ordner */lib/*. Die Bibliotheken sind:
  - „OpenCSV“ (siehe [61]) ist eine Javabibliothek, die die einfache Nutzung des Comma Separated Values Format ermöglicht. Sie wurde von der Bytecode Pty Ltd. entwickelt. OpenCSV steht unter der Apache 2.0 Lizenz (siehe [62]).
  - „Bouncy Castle Crypto APIs for Java“ (siehe [63]) enthält einen Provider für viele der etwas älteren Hashalgorithmen. Der Quellcode ist unter einer abgewandelten Version der MIT X11 Licence (siehe [64]) verfügbar.
  - Der „IAIK SHA3 Provider“ (siehe [65]) stellt Implementierungen der SHA-3 Kandidaten zur Verfügung. Entwickelt wird der Provider von der Stiftung Secure Information and Communication Technologies SIC. Er steht unter der GNU General Public License Version 3 (siehe [66]), ist aber nur im Quellcode erhältlich. Der gezippte

Quellcode liegt in */lib/*. Die entpackten Javaklassen liegen in */src/iaik/sha3/*.

- Sämtliche generierten Daten des Experiments. Diese befinden sich in dem Ordner */data/*. In dem Ordner befindet sich die Datei *summary.csv*, die die Zusammenfassung der Resultate enthält, wie sie in den Tabellen Tabelle A.1, Tabelle A.2 und Tabelle A.3 dargestellt ist. Die einzelnen Dateien für die Graphen und Analysen befinden sich in den Unterordnern von */data/*. Die Struktur der Unterordner ist wie folgt. Auf der ersten Ebene liegen die Ordner für die Reduktionsart der Hashwerte. Auf der folgenden Ebene liegen die Hashfunktionen, danach die Hashlänge und schließlich die Anzahl an Zusatzbits. Z.B. liegen alle Daten zu dem Graph von SHA1 mit 10 Bit Hashlänge und 1 Zusatzbit, wobei die Bits für die Hashwerte aus der Mitte genommen wurden in dem Ordner */data/from_mid/sha/10bits/1bits_prg/*. Es gibt zwei Dateitypen in den Ordnern. Die Dateien mit der Endung *.csv* enthalten jeweils eine der Runden, wie sie in Kapitel 5 beschrieben werden. Dabei ist der erste Wert jeder Zeile der Hashwert, der als Eingabe für die Hashfunktion dient und der zweite Wert die Ausgabe der Hashfunktion. Wurden Zusatzbits verwendet, gibt es für jeden Eingabewert mehrere Zeilen mit je einem Ausgabewert. Die Dateien mit der Endung *.csv.analysis* enthalten die Analyse des Graphen der entsprechenden Runde. In dem Experiment wurden nur die Graphen der jeweils letzten Runde analysiert, da diese nur noch starke Zusammenhangskomponenten enthalten.
- Die Konfigurationsdateien für die Testprogramme, mit denen die mitgelieferten Daten generiert werden können. Im Ordner */conf.d/* befinden sich Unterordner für jeden verwendeten Hashalgorithmus. In diesen befinden sich die Konfigurationsdateien für den jeweiligen Algorithmus mit verschiedenen Einstellungen. Die Namen der Dateien enthalten die Hashlänge und die Anzahl der Zusatzbits und enden auf *.properties*. Die Konfigurationsdateien sind selbsterklärend kommentiert.