RWTH Aachen University
Research Group IT-Security
http://itsec.rwth-aachen.de

The most efficient SHA-1 attacks

Seminar: Selected Topics in Mobile Security

Johannes Gilger, 266 377
Betreut von Georg Neugebauer

17. März 2010

Inhaltsverzeichnis

1 Einleitung

Hashfunktionen, auch "Message Digest" genannt, sind aus der heutigen Telekommunikation und Kryptographie nicht mehr wegzudenken. Aus ihrer Eigenschaft, selbst minimale Änderungen an Datenblöcken zu erkennen, ergeben sich zwei wichtige Anwendungsbereiche die sich in die bewusste Manipulation von Daten und technische Übertragungsfehler unterteilen lassen. Auf die Verwendung von Hashfunktionen zur Addressierung wird hier nicht näher eingangen.

In dieser Ausarbeitung beschäftige ich mich mit der momentan am weitesten verbreiteten kryptografisch sicheren Hashfunktion, SHA-1 [12]. Ich möchte zunächst die Anwendungsgebiete und Angriffsmöglichkeiten von Hashfunktionen aufzeigen. Danach wird die Funktionsweise von SHA-0 und SHA-1 erklärt, um mit diesem Wissen eine Einleitung in die bisherigen Angriffe auf den SHA-1 (die auf den SHA-0-Angriffen aufbauen) zu ermöglichen. Das Fazit der bisherigen Angriffe verdeutlicht warum inzwischen vor dem weiteren Neueinsatz von SHA-1 abgeraten wird.

Obwohl auch das ältere MD5 noch häufig auftaucht, hat es seinen Status als kryptografisch sichere Hashfunktion schon lange verloren und gilt im praktischen Sinne als gebrochen. Es wird in dieser Ausarbeitung nicht behandelt.

1.1 Definition

Eine Hashfunktion ist eine Funktion $h : D \rightarrow B$ die gewöhnlicherweise einen Definitionsbereich D hat der mächtiger ist als der Bildbereich B. Eine allgemeingültige Anforderung an Hashfunktionen ist, dass sie sich schnell berechnen lassen und effizient in Hard- und Software implementiert werden können. Kryptografisch sichere Hashfunktionen sind Einwegsfunktionen, auch "Falltürfunktion" genannt, da sich das Urbild eines Hashwerts nicht berechnen lässt.

Kryptografisch sichere Hashfunktionen haben zusätzliche Sicherheitsanforderungen die hier vorgestellt werden.

- *Preimage resistance:* Für den Ausgabewert einer Hashfunktion $y = h(x)$ sollte es praktisch unmöglich sein einen Eingabewert x zu finden.
- *Second preimage resistance:* Für einen gegebenen Eingabewert x_1 und $h(x_1)$ ist es schwer einen zweiten Wert $x_2 \neq x_1$ zu finden, so dass $h(x_1) = h(x_2)$ gilt.
- *Collision resistance:* Es sollte schwer sein zwei Werte $x_1 \neq x_2$ zu finden, so dass $h(x_1) = h(x_2)$ gilt.

Diese Anforderungen ziehen einen Angreifer in Betracht der gewillt ist Sicherheitsverfahren zu brechen die auf solchen Hashfunktionen basieren. Die Anforderungen unterscheiden kryptografisch sichere Hashfunktionen von den gewöhnlichen Hashfunktionen, die zur Adressierung oder zur Fehlerkontrolle verwendet werden, welche entsprechend weniger anspruchsvoll ausfallen.

1.2 Schwachstellen von Hashfunktionen

Wie bei der Verschlüsselung von Daten mit symmetrischen Schlüsseln, muss bei Hashfunktionen zwischen zwei unterschiedlichen Arten von Schwachstellen unterschieden werden. Die erste, und offensichtlichste, Schwachstelle ergibt sich aus der Schlüssellänge, bzw. der Länge der Ausgabe bei Hashes. Kleinere Wertebereiche bedeuten dass die Wahrscheinlichkeit steigt durch einfaches Ausprobieren zwei Eingabewerte mit dem gleichen Hashwert zu finden.

Für den Hashalgorithmus MD5 ist durch steigende Rechenkapazitäten und geringe Speicherkosten inzwischen der Ansatz über Rainbow-Tables stark verbreitet. Rainbow-Tables sind vorberechnete Datenstrukturen die einen Kompromiss zwischen Speicher- und Rechenaufwand eingehen, indem sie Paare (x, y) aus Eingabewerten x und Hashes y speichern zwischen denen mehrere Schritte von abwechselnder Hashanwendung h und Reduktion r liegen. Die Hashfunktion h wird auf den Eingabewert angewendet, und die

Reduktionsfunktion r wird auf $h(x)$ angewendet, um aus $h(x)$ jeweils wieder einen neuen Eingabewert für die nächste Iteration zu generieren. Sucht man den Eingabewert für einen Hash $h(a)$ so führt man auf diesem Wert die Schritte r und h solange aus bis der Ausgabewert einer der gespeicherten Ausgabewerte $y_1 \in \{(x_i, y_i)\}$ entspricht. Man führt die Schritt h und r auf dem zugehörigen Eingabewert x_1 aus bis man wieder bei $h(a)$ angelegt ist, wobei der Wert a an der vorigen Position steht (siehe Abb. 1).

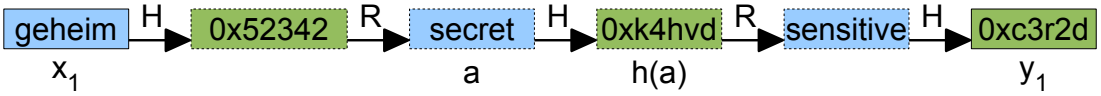

Abbildung 1: Eine Rainbow-Table mit 3 Iterationen

Dieses Verfahren Kollisionen zu finden ist besonders da interessant, wo der Eingabewert von keinem Interesse ist, also z.B. bei Passwortauthentifikation in Computersystemen. Um sich gegen Rainbow-Tables zu schützen hilft der Einsatz eines sog. “Salts”, also eines Werts der vor der Anwendung der Hashfunktion dem Eingabewert hinzugefügt wird. Dieser kann systemweit oder auch zufallsgeneriert pro Benutzer generiert werden und hat zur Folge, dass die Berechnung einer Rainbow-Table für jeden Benutzer einzeln durchgeführt werden muss.

Der zweite, und interessantere, Angriffsvektor sind algebraische Schwachstellen in der Konstruktion von Hashfunktionen. Hierbei versucht man durch Kryptanalyse der Verfahren einen Weg zu finden mit dem Kollisionen effizienter auffindbar sind als durch sequentielles Ausprobieren aller möglichen Eingabewerte. Während man, durch das Geburtstagsparadoxon, davon ausgehen kann bei SHA-1 mit einer Ausgabelänge von 160 Bits nach 2^{80} Rechenschritten ein Kollisionspaar zu finden, gibt es inzwischen Ansätze die weit unter diesem theoretischen Maximum liegen. Im den folgenden Abschnitten wird gezeigt wie man solche Verfahren findet und welche Auswirkungen auf den praktischen Einsatz von SHA-1 sie zum jetzigen Zeitpunkt haben.

1.3 Praktischer Nutzen von Kollisionen

Inwieweit kryptanalytische Ergebnisse relevant für praktische Angriffe auf Hashfunktionen sind hängt wiederum stark vom Einsatzgebiet ab. Betrachtet man nur die letzten beiden Eigenschaften kryptografischer Hashfunktionen, so ist es einleuchtend, dass mit dem Brechen der Second Preimage-Resistance auch die Collision-Resistance gebrochen ist. Glücklicherweise liegen diese beiden Eigenschaften komplexitätstheoretisch gesehen weit auseinander (mittels Brute Force braucht man theoretisch 2^{160} Hashoperationen um ein zweites Dokument mit dem gleichen Hashwert für ein bereits bestehendes Dokument zu finden), und alle bisherigen Angriffe auf Funktionen der SHA-Familie beziehen sich nur auf die leichter zu berechnende Aufgabenstellung überhaupt zwei kollidierende Eingabewerte zu finden.

Ein Szenario ist das Signieren von Zertifikaten (z.B. für SSL-gesicherte Webseiten). Um eine Kollision auszunutzen muss ein Angreifer zwei kollidierende Zertifikate finden (die beide ”Sinn” ergeben) und das erste, harmlos erscheinende, an die Certificate Authority (CA) zum unterschreiben (signieren) senden. Nur wenn die CA dieses erste Zertifikat *unverändert* unterschreibt kann der Angreifer dem Opfer dann sein zweites, falsche Zertifikat unterjubeln, auf das die Signatur ebenfalls zutrifft. Dies funktioniert da die Signaturen in Public-Key Kryptosystemen aus Effizienzgründen nur auf einen Digest, also den Hash, der Nachricht angewendet werden. Praktisch sind Angriffe dieser Art inzwischen für MD5 bekannt geworden. So wurde von Forschern anlässlich des 25. Chaos Communication Congress eine Attacke präsentiert bei der, mittels der Chosen-Prefix-Methode, sogar eine Rogue CA erstellt wurde [15]. Das bedeutet dass die Angreifer in der Lage waren eine legitim aussehende Certificate Authority zu erzeugen, die wiederum von einer CA signiert war, deren Root-Certificat in allen populären Webbrowsern vorinstalliert ist. Die Erzeugung der CA war möglich obwohl die Signatur sich auf ein Webseiten-Zertifikat und nicht auf ein CA-Zerfikat bezog. Mit dieser falschen Certificate Authority können beliebige gefälschte Zertifikate ausgestellt werden ohne dass einem Benutzer die Täuschung auffällt, da die Zertifikatskette von seinem Webbrowser einwandfrei verifiziert wird. Als Reaktion auf diese Ergebnisse wurde der Ausgabestop von MD5-signierten Zertifikaten in kürzester Zeit von den betroffenen Providern umgesetzt.

Angriffe basierend auf mangelnder Collision Resistance können einfach unwirksam oder zumindest schwieriger gemacht werden: Als signierende Stelle sollte man in keinem Fall ein Zertifikat unverändert unterschreiben sondern stattdessen eine kleine Änderung, die für den Inhalt irrelevant ist, am Dokument vornehmen. Der Angreifer müsste dann in der Lage sein die Second preimage resistance zu brechen um einen Angriffsvektor zu haben. Im Fall der o.g. MD5-Kollision gab es zwar auch einen variablen Teil der gehashed wurde, jedoch konnten die Angreifer durch Beobachtung und wiederholtes Ausprobieren schliesslich diesen Teil abpassen.
Glücklicherweise gilt, dass neue und effizientere Ergebnisse bei der Suche nach Kollisionen (zum Brechen der Collision-Resistance) keine Auswirkungen auf bereits signierte Daten haben. Bereits signierte Zertifikate sind erst dann in Gefahr gefälscht zu werden, wenn die Second Preimage-Resistance einer verwendeten Hashfunktion nicht mehr sichergestellt ist.

Um das Schadenspotential von neuen Veröffentlichungen einzuschätzen ist es notwendig die angegebenen Komplexitätsabschätzungen für Kollisionen genau zu betrachten. Hier können schnell Fehler beim Vergleich der Wahrscheinlichkeiten unterlaufen. So beziehen sich bei manchen Ergebnissen die Komplexitätsangaben von O^n nicht auf einzelne Hashoperationen sondern auf Paare von Eingabewerten die jeweils gehashed werden. Das ist insofern erklärbar als dass durch geschickte Abbruchbedingungen der ansonsten unterschlagene Faktor 2 wieder ausgeglichen wird und die Komplexität hier, wie in vielen Angriffen, verglichen zur vollen Version des SHA-1 angegeben wird. Allerdings sollte man sich bewusst machen dass unterschiedliche Hashalgorithmen sich teilweise stark in der Laufzeit unterscheiden. Das Gleiche gilt natürlich auch für die anvisierte Größe des zu kollidierenden Dokuments, wenn man ein konkretes Szenario betrachtet und nicht nur allgemein Kollisionen sucht. Laufzeitkomplexitäten für Kollisionsmethoden auf rundenreduzierte Hashverfahren beziehen sich eben auf diese Rundenzahl, so dass die Berechnung oft um den Faktor 2 schneller ist als der volle Algorithmus. Da alle Komplexitäten nur Abschätzungen sind (und im Fall von SHA-1 bisher nicht überprüft werden können) können den Kryptanalysen Fehler passieren die u.U. erst später auftauchen, nachdem die Ankündigung einer neuen Untergrenze zur Kollisionsfindung schon weite Verbreitung in nichtwissenschaftlichen Kreisen erfahren hat [8].

1.4 Auswirkungen auf die Einsatzgebiete

Öffentliche Behörden, private Firmen und Forscher verfolgen die Entwicklungen bei der Kryptanalyse von verbreiteten Hashfunktionen, da diese ein lohnender Angriffsvektor sind mit dessen praktisch erfolgreicher Kryptanalyse automatisch eine Reihe wichtiger Systeme unsicher würde. Wie auch bei symmetrischen Verschlüsselungsverfahren wie DES ist eine Umstellung spätestens dann anzuraten, wenn die zuständigen US-Behörden vom weiteren Neueinsatz der Verfahren abraten. In diesem Zusammenhang hört man oft den Merksatz: ”Attacks never get worse, they only get better”.

Während die Nutzung einer neuen Hashfunktion bei informellen Protokollen und eigener Software vergleichsweise einfach ist (z.B. beim manuellen Verifizieren von SSH und PGP-Fingerprints oder bei der Integritätskontrolle von Softwarepaketen bei Linux-Distributionen) und der Übergang fliessend gestaltet werden kann, gibt es leider viele Einsatzbereiche in denen eine Umstellung kompliziert und teuer ist, da Erweiterungen, oder teilweise sogar komplette Neuentwicklungen bereits standartisierter Kommunikationsprotokolle nötig sind. Selbst nichtstandardisierte Projekte stehen vor einem Problem, wenn sie in großen Teilen ihrer Codebasis SHA-1 fest einprogrammiert haben, oder zumindest von einem Hash der Länge 160 Bit ausgehen.

Die Mängel an bestehenden Protokollen und Standards (z.B. TLS oder S/MIME) wurden im Fall von SHA-1 früh genug erkannt und sind zum Zeitpunkt des Schreibens schon durch Revisionen an den Standards behoben [1]. Um auch in Zukunft einfach die verwendete Hashfunktion austauschen zu können ist eine Möglichkeit dabei die Hashlänge variabel zu lassen um für neuere Hashfunktionen mit längerer Ausgabe kompatibel zu sein. Zusätzlich wird häufig die verwendete Hashfunktion direkt mit der Signatur gespeichert, so dass an dieser Stelle keine Uneindeutigkeiten enstehen.

Der weitere Einsatz von gebrochenen Hashfunktionen in HMACs (Hash-based Message Authentication Code) stellt kein Problem da, solange die Sicherheit des symmetrischen Schlüssels vor einem Angreifer garantiert werden kann.

2 SHA-0 und SHA-1

2.1 Geschichte

Der erste SHA-Algorithmus (”Secure Hash Algorithm”) wurde vom NIST (National Institute of Standards and Technology) in Zusammenarbeit mit der NSA (National Security Agency) im Jahr 1993 veröffentlicht ([10]). Kurz darauf wurde diese, als SHA-0 standardisierte Hashfunktion von der NSA aufgrund nicht näher begründeter Sicherheitsbedenken revidiert und durch eine minimal veränderte Version ersetzt [11], heute als SHA-1 bekannt. Später wurde mit [12] der SHA-1 Algorithmus standardisiert und zusätzlich die Algorithmen SHA-224, SHA-256, SHA-384, und SHA-512 der SHA-2-Familie eingeführt. Auf dem letzten Stand befindet sich [13]. Im aktuellsten Standard ist ein Verweis auf [6] enthalten, welches aktuelle kryptanalytische Ergebnisse in Betracht zieht und von der Benutzung von SHA-1 in neuen Systemen abrät.

2.2 Funktionsweise

SHA-0 und SHA-1 sind Hashfunktionen die auf einer Eingabe von max. $2^{64}-1$ Bit arbeiten und einen 160 Bit grossen Hashwert generieren. Der Algorithmus bereitet die Ausgangswerte vor, indem zunächst eine Paddingfunktion auf die Eingabe angewendet wird. Diese bringt die Eingabelänge auf ein Vielfaches von 512 Bit, die Größe der Blöcke mit denen SHA-1 arbeitet.

Um die Funktionen zu beschreiben müssen noch ein paar Konventionen zur Notation getroffen werden:

- Bitweise Operatoren: $\wedge, \vee, \oplus$ und $\neg$.
- Addition modulo 2^{32}: $x + y = (x + y) \bmod 2^{32}$
- Zyklische Linksrotation um n Stellen: $ROTL^n(x)$
- Konkatenation: $\|$

1. **Padding und Aufteilung:** Sei M der Eingabewert mit Länge ℓ Bit. Dann wird zunächst eine 1 und k 0-Bits an die Nachricht angehängt so dass für die neue Länge gilt: $\ell + 1 + k \equiv 488 \bmod 512$. Die letzten 64 Bit sind die Binärdarstellung der Nachrichtenlänge ℓ. Die gepaddete Nachricht kann nun in Blöcke der Größe 512 Bit aufgeteilt werden, wobei hier allgemein von sechzehn 32-Bit Worten $M_0^{(i)} \cdots M_{15}^{(i)}$ die Rede ist.

   Beispiel für die ASCII-Nachricht ”SHA” (ohne terminierendes Null-Byte oder Newline):

   $$\underbrace{\texttt{01010011}}_{\text{”S”}}\ \underbrace{\texttt{01001000}}_{\text{”H”}}\ \underbrace{\texttt{01000001}}_{\text{”A”}}\ \texttt{1}\ \overbrace{\texttt{00...00}}^{423\text{ Bit}}\ \overbrace{\texttt{00...0}\underbrace{\texttt{11000}}_{\ell=24}}^{64\text{ Bit}}$$

2. **Initiale Hashwerte:** Da die SHA-Funktion blockweise arbeitet und die 160 Bit Ausgabe des vorherigen Blocks als Eingabeparameter für den aktuellen Block benutzt braucht man festgelegte Anfangswerte um den ersten Block zu berechnen. Diese werden im Standard als folgende 32-Bit Worte spezifiert (in hexadezimaler Darstellung):

   $$H_0^{(0)} = \texttt{0x67452301}$$
   $$H_1^{(0)} = \texttt{0xefcdab89}$$
   $$H_2^{(0)} = \texttt{0x98badcfe}$$
   $$H_3^{(0)} = \texttt{0x10325476}$$
   $$H_4^{(0)} = \texttt{0xc3d2e1f0}$$

Nun beginnt die blockweise Berechnung der Eingabe, auch Merkle-Damgård Konstruktion genannt. Jeder Block $M^{(i)}, i = 1 \dots n$ wird folgendermaßen bearbeitet:

1. **Expansion:** Jeder Block (512 Bit) wird in einen Wert W, auch "Message Schedule", expandiert (2560 Bit) so dass W aus 80 32-Bit Worten besteht:

$$W_t = \begin{cases} M_t & 0 \leq t \leq 15 \\ ROTL^1(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) & 16 \leq t \leq 79 \end{cases}$$

Unterschied: Die Linksrotation für die Schritte $16 \leq t \leq 79$ der Expansion ist das einzige Merkmal welches SHA-1 im Gegensatz zu SHA-0 besitzt, ansonsten sind beide Hashfunktionen identisch.

2. **Hashwerte:** Wie schon erwähnt werden die Hashwerte des vorherigen Blocks $M^{(i-1)}$ als Eingabe in die Variablen $\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}$ und $\mathbf{e}$ ("Chaining-Variablen") übernommen:

$$a = H_0^{(i-1)} \quad b = H_1^{(i-1)} \quad c = H_2^{(i-1)} \quad d = H_3^{(i-1)} \quad e = H_4^{(i-1)}$$

3. **Hashrunden:** Es folgen *4 Runden* mit *je 20 Schritten*, insgesamt $t = 0 \dots 79$. Diesen Schritt nennt man auch "Kompressionsfunktion" da sie aus den 2560 Bit von W einen 160 Bit großen Hashwert erzeugt:

$$\begin{aligned} a_{t+1} &= ROTL^5(a_t) + f_t(b_t, c_t, d_t) + e_t + K_t + W_t \\ b_{t+1} &= a_t \\ c_{t+1} &= ROTL^{30}(b_t) \\ d_{t+1} &= c_t \\ e_{t+1} &= d_t \end{aligned}$$

Dieser Schritt ist in Abbildung 2 dargestellt. Man sieht wie eine Änderung in einem Wort W sich durch die Kompressionsfunktion propagiert und in jeder Variable einmal auftaucht.

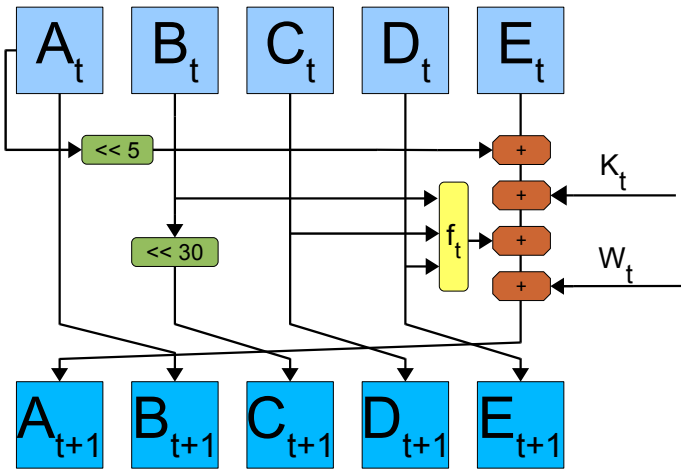

Abbildung 2: Ein Schritt der SHA-1 Rundenfunktion

Die Funktion f_t und die Konstante K_t sind hierbei abhängig von der aktuellen Runde:

$0 \leq t \leq 19$	$f_t(x,y,z) = IF(x,y,z) = (x \wedge y) \vee (\neg x \wedge z)$	$K_t =$ `0x5a827999`
$20 \leq t \leq 30$	$f_t(x,y,z) = XOR(x,y,z) = x \oplus y \oplus z$	$K_t =$ `0x6ed9eba1`
$40 \leq t \leq 59$	$f_t(x,y,z) = MAJ(x,y,z) = (x \wedge y) \vee (x \wedge z) \vee (y \wedge z)$	$K_t =$ `0x8f1bbcdc`
$60 \leq t \leq 79$	$f_t(x,y,z) = XOR(x,y,z) = x \oplus y \oplus z$	$K_t =$ `0xca62c1d6`

Die Konstanten werden durch die Berechnung von $2^{30} \cdot \sqrt{x}, x \in \{2, 3, 5, 10\}$ gewonnen.

4. **Blockhashwert:** Zum Schluss wird der Hashwert für den Nachrichtenblock berechnet:

$$H_0^{(i)} = a + H_0^{(i-1)}$$
$$H_1^{(i)} = b + H_1^{(i-1)}$$
$$H_2^{(i)} = c + H_2^{(i-1)}$$
$$H_3^{(i)} = d + H_3^{(i-1)}$$
$$H_4^{(i)} = e + H_4^{(i-1)}$$

Der 160 Bit lange Hash besteht dann aus der Ausgabe des letzten Blocks, also für eine Eingabenachricht die aus n Blöcken besteht:

$$H_0^{(n)} \| H_1^{(n)} \| H_2^{(n)} \| H_3^{(n)} \| H_4^{(n)}$$

2.3 Beispiel-Hash

Ein Beispiel wie sich durch das Verändern eines einzigen Zeichens ein komplett anderer Hashwert ergibt sieht wie folgt aus:

SHA-1(SHA-0 generiert 160 Bit große Hashwerte.) = `2437f11ccdc1bd804afad073e24f188498294ea2`
SHA-1(SHA-**1** generiert 160 Bit große Hashwerte.) = `3102273221604d0e31662b60e9e2e9f17fffa0ec`

3 Kryptanalyse

Das Geburtstagsparadoxon sagt aus, dass die Wahrscheinlichkeit zweier beliebiger Werte den gleichen Hashwert zu haben $1^{-\frac{160}{2}}$ ist. Die Kryptanalyse sucht nach Methoden Kollisionen effizienter zu finden als es das Geburtstagsparadox zulässt. Wird ein solches Ergebnis entdeckt, bezeichnet man die Hashfunktion als "gebrochen", was jedoch noch keinerlei Rückschluss auf die praktische Durchführbarkeit der neuen Attacke gibt.

Im Zusammenhang mit Hashfunktionen und symmetrischen Verschlüsselungsverfahren spricht man häufig von differentieller Kryptanalyse bzw. differentiellen Pfaden durch die Hashfunktion. Unter differentieller Kryptanalyse versteht man das Vorgehen, Änderungen in der Nachricht durch die Funktion zu verfolgen und entsprechende Unterschiede in der Ausgabe auf diese Änderungen zurückzuführen. Da man Änderungen an frei wählbaren Nachrichten vornimmt und diese als Eingabe für die Funktion verwendet, fallen die Angriffe in die Kategorie der "Chosen-plaintext attacks". Für den SHA bedeutet das, dass man ein allgemeingültiges Änderungsmuster für die Nachrichtenwörter findet welches die Hashfunktion auf einen anderen Weg bringt, der mit nicht-trivialer Wahrscheinlichkeit zum gleichen Hashwert führen soll. Hat man einen solchen differentiellen Pfad konstruiert wird er in Form eines Störvektors (siehe Abb. 5) veröffentlicht und kann, zusammen mit unterschiedlichen Techniken zur Optimierung und frühzeitigen Abbruchbedingungen, zur Kollisionssuche verwendet werden. Das bedeutet dass man die Kollisionen an den im Störvektor angegebenen Stellen startet indem man eine bitweise Negation in den Nachrichtenwörter an den entsprechenden Stellen vornimmt. Um diese Änderungen auszugleichen und zum gleichen Hashwert zu kommen sind in den folgenden Runden Korrekturen nötig, welche sich ebenfalls direkt aus dem Störvektor ergeben. Die Suche nach Kollisionen ist probabilistisch, eine Kollision wird also mit einer bestimmten Wahrscheinlichkeit gefunden. Diese Wahrscheinlichkeit ergibt sich aus dem Aufbau des Störvektors, wobei die Anzahl der gestarteten Kollisionen und deren Positionen in der rundenbasierte Kompressionsfunktion ausschlaggebend sind, da nicht jede Nachricht auf die der Störvektor angewendet wird die richtigen Eigenschaften besitzt um auch wirklich zu einer Kollision zu führen.

Die Grundlage für Angriffe auf Hashalgorithmen sind lokale Kollisionen. Diese Kollisionen beziehen sich auf wenige Schritte des gesamten Hashverfahrens (pro Block) und ergeben sich aus dem Aufbau der rundenbasierten Kompressionsfunktion. Im Fall von SHA-0 und SHA-1 bildet eine 6-schrittige Kollision die

Grundlage. Obwohl diese Kollisionen leicht zu verstehen und herbeizuführen sind repräsentieren sie nur den ersten Schritt der Kryptanalyse, da es nicht möglich ist einzelne Kollisionen auf Verfahren wie SHA-1 einzuführen. Allerdings kann durch die Verkettung lokaler Kollisionen eine Kollision auf dem vollen Hashalgorithmus gefunden werden. Kollisionen auf rundenreduzierten Versionen eines Algorithmus dienen gleichzeitig auch als Verifikation der Ergebnisse für den vollen Algorithmus, besonders wenn die Wahrscheinlichkeit für Kollisionen auf der vollen Rundenzahl zu klein ist um empirisch überprüft zu werden.

4 Angriffe auf SHA-0

Die Kryptanalyse von SHA-0 begann 1998 mit der Konstruktion einer 6-schrittigen lokalen Kollision [4], die den Grundbaustein für die Kollision auf dem vollen Algorithmus bildet. Dazu wurden zunächst abgeschwächte Versionen des SHA-0 betrachtet, die sich darin einschränkten für die Rundenfunktion f_t nur das lineare XOR zu benutzen und zur Berechnung von a_t ebenfalls XOR statt der Addition modulo 32 einzusetzen. Es ist nicht bekannt ob dieses Ergebnisse der NSA ebenfalls vorlagen als sie sich dazu entschloss, SHA-1 als Verbesserung von SHA-0 zu veröffentlichen.

Bereits 1997 hatte Wang die gleiche Kollision für SHA-0 unabhängig von Chabaud und Joux gefunden [16]. Die Ergebnisse die für die Kollisionsfindung für Hashalgorithmen wie dem (ähnlich strukturierten) MD4/MD5, HAVAL und RIPEMD benutzt wurden halfen bei der Analyse von SHA-0. Die nächsten Fortschritte waren die Konzepte der neutralen Bits und der Nachrichtenmodifikation die in einer sehr effizienten Methode für Beinahe-Kollisionen (142 von 160 Bit) resultierte [2]. Beinahe-Kollisionen haben nicht nur rein akademischen Wert, man denke an Szenarien in denen Hashes von Menschen verifiziert werden oder wo von der Ausgabe der Hashfunktion nur ein Teil benutzt wird. Beinahe-Kollisionen mehrerer Blöcke wurden darüberhinaus bei späteren Angriffen benutzt um Kollisionen zu finden. Für Beinahe-Kollisionen ist die Anforderung 1 (Tab. 1) überflüssig, da nicht mehr gefordert wird dass angefangene Kollisionen bis zur letzten Runde aufgelöst werden können. Daher können Masken mit kleinerem Hamming-Gewicht gefunden werden.

Die Wahrscheinlichkeit eine Kollision zu finden stieg danach auf 2^{-58}, dann auf 2^{-51} ([3]) und konnte schließlich auf 2^{-39} ([17], 2005) angehoben werden. Der effizienteste differentielle Pfad über 80 Schritte wurde zum Zeitpunkt des Schreibens auf 2^{36} geschätzt ([9], 2006). Für den vollen SHA-0 wurden schon echte Kollisionen (beginnend 2004, konstruiert über Beinahe-Kollisionen auf 4 Blöcken) erzeugt, ein Beispiel ist in Abbildung 3 gegeben. Kollisionen müssen für den SHA-1 erst noch gefunden werden. Das bisherige Ausbleiben vergleichbarer Resultate für den SHA-1 zeigt den Effekt den die zusätzliche Linksrotation bei der Nachrichtenexpansion hat.

4.1 SHA-0 Kollisionen nach Chabaud / Joux

Chabaud und Joux legten den Grundstein für die weitere Kryptanalyse des SHA-0 und SHA-1-Algorithmus. Ihr Vorgehen kann in 3 Schritte unterteilt werden:

1. Wahl einer linearen Approximation von SHA-0, in der die nicht-linearen Funktionen (+, IF, MAJ) durch möglichst "gleiche" lineare Funktionen ersetzen werden.
2. Das Finden von Kollision bzw. Störvektoren für diese lineare Approximation von SHA-0.
3. Wiedereinsetzen der nicht-linearen Funktionen und berechnen der Wahrscheinlichkeit dass diese sich wie die lineare Approximation verhalten.

Die Nachrichtendifferenzen auf den Wörtern W^t werden, der Einfachheit halber, zunächst ohne Berücksichtigung der Anforderung durch die Nachrichtenexpansion betrachtet, es ist also möglich alle 80 Wörter $W^0, \ldots, W^{79}$ direkt zu verändern. Eine einzelne 6-schrittige lokale Kollision ist dann einfach mit einem Störvektor (eng: "perturbance/disturbance vector") möglich (siehe Abb. 4). Die 6 Schritte sind mindestens nötig da man durch die Propagation der Änderung durch die Variablen $a, \ldots, e$ 6 Schritte braucht um diese wieder auszugleichen, da sich eine Änderung in W_j^{t-1} zuerst in a_t, dann in $b_{t+1}, c_{t+2}, d_{t+3}$ und zuletzt in

Nachricht:

```
a766a602 b65cffe7 73bcf258 26b322b3 d01b1a97 2684ef53 3e3b4b7f 53fe3762
24c08e47 e959b2bc 3b519880 b9286568 247d110f 70f5c5e2 b4590ca3 f55f52fe
effd4c8f e68de835 329e603c c51e7f02 545410d1 671d108d f5a4000d cf20a439
4949d72c d14fbb03 45cf3a29 5dcda89f 998f8755 2c9a58b1 bdc38483 5e477185
f96e68be bb0025d2 d2b69edf 21724198 f688b41d eb9b4913 fbe696b5 457ab399
21e1d759 1f89de84 57e8613c 6c9e3b24 2879d4d8 783b2d9c a9935ea5 26a729c0
6edfc501 37e69330 be976012 cc5dfe1c 14c4c68b d1db3ecb 24438a59 a09b5db4
35563e0d 8bdf572f 77b53065 cef31f32 dc9dbaa0 4146261e 9994bd5c d0758e3d
```

Modifizierte Nachricht:

```
a766a602 b65cffe7 73bcf258 26b322b1 d01b1ad7 2684ef51 be3b4b7f d3fe3762
a4c08e45 e959b2fc 3b519880 39286528 a47d110d 70f5c5e0 34590ce3 755f52fc
6ffd4c8d 668de875 329e603e 451e7f02 d45410d1 e71d108d f5a4000d cf20a439
4949d72c d14fbb01 45cf3a69 5dcda89d 198f8755 ac9a58b1 3dc38481 5e4771c5
796e68fe bb0025d0 52b69edd a17241d8 7688b41f 6b9b4911 7be696f5 c57ab399
a1e1d719 9f89de86 57e8613c ec9e3b26 a879d498 783b2d9e 29935ea7 a6a72980
6edfc503 37e69330 3e976010 4c5dfe5c 14c4c689 51db3ecb a4438a59 209b5db4
35563e0d 8bdf572f 77b53065 cef31f30 dc9dbae0 4146261c 1994bd5c 50758e3d
```

Hashwert: `c9f16077 7d4086fe 8095fba5 8b7e20c2 28a4006b`

Abbildung 3: SHA-0 Kollision (4 Blöcke, 2048 Bit) nach Biham/Chen [3]

e_{t+4} zeigt. Diese Unterschiede müssen durch einen Unterschied in den Nachrichtenwörtern $W^t, \ldots, W^{t+4}$ ausgeglichen werden damit die Berechnung von $a_t, \ldots, a_{t+4}$ für beide Nachrichten zum gleichen Ergebniss führt. Diese Kollision kann beliebig oft parallel auf unterschiedlichen Bits auftreten.

	Anforderung	*Zweck*
1	$x_t = 0 : t \in \{75, \ldots, 79\}$	Kollision bis zum letzten Schritt fertig haben
2	$x_t = 0 : t \in \{-5, \ldots, -1\}$	Teilkollisionen in den ersten Schritten vermeiden
3	Keine benachbarten 1-Bits in den ersten 17 Variablen	Unmöglichen Kollisionspfad wegen der IF-Funktion vermeiden

Tabelle 1: Die Anforderungen an den Störvektor

Im Fall von SHA-0 betrachtet man einen 80-Bit langen Vektor x, den sog. Störvektor. Die Elemente des Vektors beziehen sich im einfachsten Beispiel auf das jeweilige zweite Bit der Nachrichtenwörter $W^0, \ldots, W^{79}$, da durch die Rotation im dritten Schritt das geänderte Bit an die Stelle des höchstwertigsten Bits rotiert wird, und somit ein Übertrag durch Addition in späteren Schritten vermieden wird. Der Vektor gibt die Schritte an in denen eine lokale Kollision starten soll, also die 32 Bit großen Nachrichtenwörter in denen Unterschiede zwischen den Nachrichten M und M' auftreten. Diese Unterschiede sollen jeweils nach 6 Schritten mit einer lokalen Kollision wieder aufgelöst werden. Da man den Unterschied in einem Schritt aufgrund der Rundenfunktion 5 Schritte lang in den Arbeitsvariablen mitnimmt müssen auch die nächsten 5 Schritte, und damit die nächsten 5 Nachrichtenwörter, Korrektur-Unterschiede enthalten. Aus dem Vektor x (auch Maske genannt) können mittels Rotationen direkt die 5 korrektiven Masken für die Folgewörter generiert werden, z.B. durch Linksrotation um 5 Bits und Übersetzung um einen Schritt für die erste Maske.

Da SHA-0 die Nachricht expandiert kann ein Angreifer allerdings nur die Wörter $W^0, \ldots, W^{15}$ unmittelbar bearbeiten, während die restlichen Wörter von der Expansionsfunktion abhängig sind. Zieht man die Nachrichtenexpansion also wieder mit in Betracht so muss zusätzlich für x gelten:

$$x_t = x_{t-3} \oplus x_{t-8} \oplus x_{t-14} \oplus x_{t-16}, \forall i \in \{16, \ldots, 79\}.$$

Mit dieser Maske kann dann den erforderlichen modifizierten Eingabeblock durch die Umkehrfunktion der linearen Expansion berechnen. Anhand Abbildung 3 kann man anschaulich sehen wie sich die Störungen

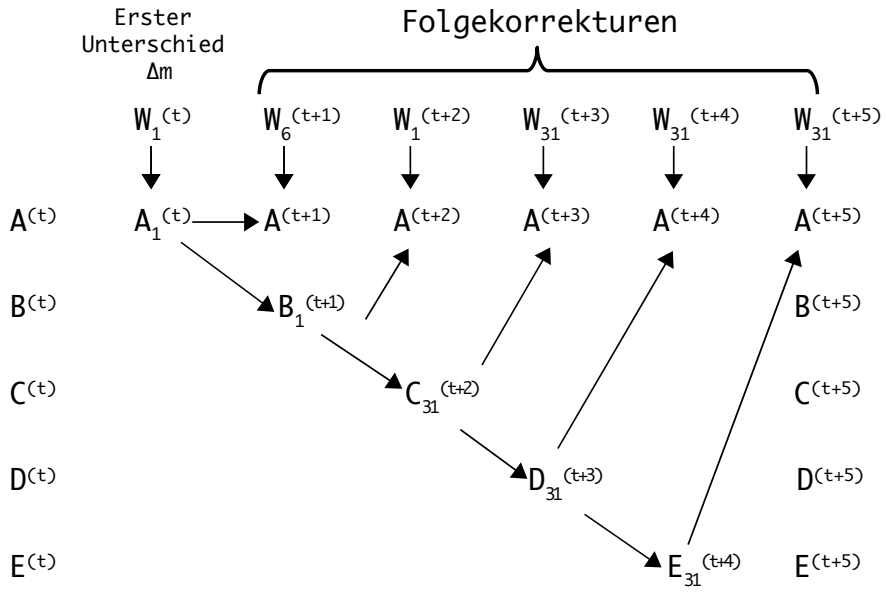

Abbildung 4: Die SHA-0 6-Schritt Kollision nach Chabaud/Joux [4]

auf der Nachricht verteilen. Durch die fehlende Rotation der Expansionsfunktion sieht man deutlich dass sich die Unterschiede auf Bits 1, 6 und 31 der 32-Bit Wörter $(0, \ldots, 31)$ beschränken und nach Korrekturen auf dem ersten Bit jeweils die typischen Korrekturen der lokalen 6-Schritt-Kollision im 32-Bit-Abstand folgen (1-6-1-31-31-31). In der letzten Zeile wurden lokale Kollisionen nicht abgeschlossen, da in diesem Beispiel Beinahe-Kollisionen auf den einzelnen Blöcken erzeugt werden.

Im zweiten Teil seiner Analyse zieht Chabaud die vorgesehenen Rundenfunktionen f_t wieder mit in Betracht und untersucht mit einer Fallunterscheidung mit welcher Wahrscheinlichkeit und unter welchen Voraussetzungen sich die nicht-linearen Rundenfunktionen (IF und MAJ) wie das lineare XOR verhalten (Tabelle 2). Dieses Verhalten ist nötig damit eine Änderung in den Nachrichtenwörtern durch die Funktion f_t durchgegeben wird und die Korrekturen in den folgenden Schritten auf diese Änderung angewendet werden können. Anforderung 3 in Tabelle 1 ergibt sich aus der Wahrscheinlichkeit für die IF-Funktion bei gleichzeitiger Änderung der Werte c^t und d^t immer eine Änderung hervorzurufen während die XOR-Funktion nie eine Änderung ausgibt.

	Wahrscheinlichkeit	
Unterschied	*IF*	*MAJ*
$b'^{(t)} \oplus b^{(t)} = 2^1$	$1/4$	$1/2$
$c'^{(t)} \oplus c^{(t)} = 2^{31}$ oder $d'^{(t)} \oplus d^{(t)} = 2^{31}$	$1/2$	$1/2$
$c'^{(t)} \oplus c^{(t)} = 2^{31}$ und $d'^{(t)} \oplus d^{(t)} = 2^{31}$	0	1

Tabelle 2: Wahrscheinlichkeiten für das Verhalten wie XOR

Danach wird die Addition für a_t, die ebenfalls eine nicht-lineare Funktion ist, unter dem Gesichtspunkt des Übertrags für sich betrachtet. Die ersten beiden Schritte einer lokalen Kollision können zum Übertrag führen, da hier Bits an den Stelle 1 und 6 manipuliert werden. Im folgenden Schritt befindet sich das geänderte Bit durch Rotation an Position 31 und Überträge sich nicht mehr möglich. Die Wahrscheinlichkeit dass während der Addition kein Übertrag auftritt trägt als Faktor $\frac{1}{4}$ zur Gesamtwahrscheinlichkeit pro Kollision bei und gilt auch während der XOR-Runden.

B	C	D	IF	MAJ
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	1	1
1	0	0	0	0
1	0	1	0	1
1	1	0	1	1
1	1	1	1	1

Tabelle 3: Verhalten von IF und MAJ

Zum Schluss werden die Erkenntnisse auf den vollen SHA-0 angewendet. Es werden zwei Muster gefunden die mit Wahrscheinlichkeiten von 2^{-68} bzw. 2^{-69} (siehe Abb. 5) zu Kollisionen auf dem vollen SHA-0 führen. Die Wahrscheinlichkeit gibt also an, in wievielen Fällen sich *alle* Rundenfunktionen und die Addition gleichzeitig linear verhalten. Die führenden 5 Bits sind nötig, da Korrekturen der 5 Nachfolgewörter durch Bit-Rotation und Runden-Übersetzungen geschehen und für die ersten 5 Wörter bereits angefangen Kollisionen erst einen bestimmten Schritt erreicht haben können.

Wörter $-5, \ldots, 79$ v.l.n.r Eine 1 bedeutet einen Unterschied in Bit 2 des jeweiligen Worts.

```
00000  00100010000000101111 01100011100000010100
       01000100100100111011 00110000111110000000
```

Abbildung 5: Differentieller Pfad für SHA-0 mit Erfolgswahrscheinlichkeit 2^{-69}, $hw(x) = 30$ nach [4]

Diese Wahrscheinlichkeiten ergeben sich aus den schrittweise Wahrscheinlichkeiten dass sich die Störung für die nicht-linearen Rundenfunktionen wie gewünscht (also wie das XOR) verhält (siehe Tab. 3 und Tab. 2). Die Wahrscheinlichkeiten für einen Störvektor können in einer Tabelle notiert und anschliessend zur gesamten Wahrscheinlichkeit multipliziert werden. Durch geschickte Implementierung kann die Wahrscheinlichkeit Kollisionen auf der gesamten Rundenzahl mittels des Vektors in Abb. 5 zu finden schliesslich von 2^{-69} auf 2^{-61} angehoben werden. Eine Folge von fünf Nullen im Muster erlaubt es den SHA-0 nach der bis dahin ausgeführten Zahl von Schritten zu beenden um nach Kollisionen auf der schrittreduzierten Version zu suchen. Im obigen Beispiel ist dies nach Runde 35 der Fall, für die Kollisionen mit einer Wahrscheinlichkeit von 2^{-14} gefunden wurden.

Die Wahrscheinlichkeit das gewünschte Verhalten auf der gesamten Hashfunktion zu haben sinkt mit jeder Störung. Daher ist es offensichtlich warum man Störvektoren mit möglichst kleinem Hamming-Gewicht, insbesondere in den Runden mit nicht-linearen Rundenfunktionen, finden will.

4.2 Verbesserungen von Biham/Chen (2004), Wang (2005) und Naito (2006)

Biham und Chen führten die Möglichkeit ein, die Kollisionssuche für SHA-0 ab einer höheren Runde starten zu lassen [2]. Da damit die Faktoren der übersprungenen Schritte nicht mehr zur gesamten Wahrscheinlichkeit beitragen, lassen sich Kollisionen auf diese Weise effizienter finden. Dazu werden Nachrichtenpaare M und M' konstruiert für die sich die nicht-linearen Funktionen bis zu einem bestimmten Schritt r wie gewünscht verhalten. Die neutralen Bits dieser Nachrichten sind jene Bits, die in M und M' identisch gekippt werden können, ohne dass sich die Nachrichtenpaare in den ersten r Schritten nicht mehr linear verhalten. Biham und Chen geben einen Algorithmus an, der, für einzelne Nachrichtenpaare Mengen von neutralen Bits findet, die mit einer Wahrscheinlichkeit von $\frac{1}{8}$ die Linearität beibehalten. In ihrem Beispiel können sie auf diese Weise die Kollisionssuche effektiv ab Schritt 22 starten lassen. Da sich die Ergebnisse in dieser Ausarbeitung auf eine 82-Schritt-Version von SHA-0 beziehen (diese ist leichter anzugreifen als die 80 Schritte), sind die Kollisionen die für den vollen SHA-0 gefunden werden nur Beinahe-Kollisionen.

Mit der Entdeckung der neutralen Bits in den Nachrichtenwörtern sowie der Technik der Multi-Block-Kollisionen [3], die auf den SHA-0 übertragen werden konnten, stellte Prof. Wang, gleichzeitig mit der Veröffentlichung des ersten differentiellen Kollisionspfad durch den vollen SHA-1, ihre verbesserte Kollisionsmethode für den SHA-0 vor [17]. Multi-Block-Kollisionen funktionieren indem man zwei (oder mehr) Nachrichtenblöcke betrachtet und die Forderung einer Kollision nur für den letzten der Blöcke macht. Da jeder Nachrichtenblock mit dem Hashwert des vorherigen Blocks initialisiert wird können Beinahe-Kollisionen vorheriger Blöcke leicht in dem darauffolgenden Schritt aufgelöst werden. Diese Kollisionsmethode ist besonders dann relevant, wenn echte Kollisionsverfahren für den vollen Algorithmus zu ineffizient sind.

Mithilfe von Techniken zur Nachrichtenmodifikation kann die erste Runde komplett vorberechnet, also deterministisch behandelt werden. Das bedeutet natürlich dass man im folgenden Störvektoren mit geringem Hamming-Gewicht in den Runden 2-4 finden will, insbesondere für die dritte Runde (MAJ). Nachrichtenmodifikation bedeutet dass man, für einen bestimmten Störvektor, die Nachrichtenwörter so manipuliert dass sie sich in der ersten Runde immer wie gewünscht verhalten. Dieses Vorgehen ähnelt den Neutralen Bits von Biham/Chen und ist im Fall von SHA-0 zunächst nicht effektiver als dieses, da man mittels der Nachrichtenmodifikation nach Wang nur die ersten 20 Schritte systemmatisch erfüllen kann.
Ein Team um Y. Naito erweiterte die Nachrichtenmodifikation auf die Schritte 21-24 [9], womit sie wiederrum effektiver als die Neutralen Bits ist. Damit konnte die Wahrscheinlichkeit Kollisionen zu finden auf 2^{-36} verbessert werden.

5 Angriffe auf SHA-1

Nach Angriffen auf rundenreduzierte Versionen des SHA-1 [14] folgte die erste Attacke auf den vollen SHA-1 im Jahr 2005. Einer Gruppe um Prof. Wang (bereits durch ihre Kryptanalyse von SHA-0 und früheren Algorithmen wie MD4/MD5/RIPEMD-160/HAVAL bekannt) gelang es Kollisionen für den vollen SHA-1 mit einer Komplexität von weniger als 2^{69} Hashoperationen zu finden [18]. Wang gelang es ebenfalls Komplexität weiter auf 2^{63} zu senken, allerdings wurden zu diesem Ergebniss bisher keine wissenschaftliche Ausarbeitung veröffentlicht [5] und es wurde auf der Crypto 2006 Konferenz nur stellvertretend von A. Shamir während der Rump Session vorgestellt, da Prof. Wang keine Einreisegenehmigung erhalten hatte.

Auf der Rump-Session der Eurocrypt 2009 kündigten McDonald, Hawkes und Pieprzyk an, Kollisionen mit 2^{52} Hashoperationen finden zu können [8]. Dabei hatten sie sich auf einen neu entdeckten Störvektor gestützt, der bei der gesonderten Analyse und Bewertung von SHA-1-Störvektoren gefunden wurde. Dieser Vektor versprach für die Runden 2-4 eine Komplexität von 2^{57} und wurde zusammen mit einer sog. “Boomerang”-Attacke dazu genutzt die Gesamtkomplexität zu erreichen. Kurz darauf wurde festgestellt, dass bei der Abschätzung der Wahrscheinlichkeiten für den Störvektor Fehler unterlaufen waren, und das Ergebniss wurde vorerst zurückgezogen. Zum aktuellen Zeitpunkt lag keine überarbeitete Fassung der Ergebnisse vor.

5.1 SHA-1 Kollisionen nach Wang (2005)

Das Suchen nach differentiellen Pfaden durch den SHA-1 unterscheidet sich von dem Herangehen an den SHA-0. Die Linksrotation in der Nachrichtenexpansion bei SHA-1 ist der Unterschied der verhindert dass die kryptanalytischen Ergebnisse vom SHA-0 direkt übernommen werden können. Statt einem 80-Bit Vektor der jeweils Bits an der gleichen Position in den Nachrichtenwörtern identifiziert muss man nun auf 80 32-Bit-Worten rechnen, da Nachrichtendifferenzen sich über die expandierten Wörter verteilen. Naiv würde das bedeuten, dass sich der Suchraum für 16 aufeinanderfolgende Wörter von 2^{16} auf 2^{512} vergrössert, jedoch werden Heuristiken benutzt um nur einen kleinen relevanten Teil hiervon in Betracht zu ziehen.

Das wichtigste Kriterium bei der Suche nach Störvektoren ist das Hamming-Gewicht des Vektors, also die Anzahl der 1-Bits. Für den vollen SHA-1 muss dieses echt kleiner als 27 sein, da die Wahrscheinlichkeit, dass der gefundene differentielle Pfad zu einer Kollision führt, sonst unter 2^{-80} fällt, also jener Wahrscheinlichkeit die man durch das Geburtstagsparadoxon gegeben hat. Diese Eigenschaft ergibt sich

daraus dass sich im Schnitt jede Änderung mit einer Wahrscheinlichkeit von $\frac{1}{8}$ wie gewünscht verhält, daher $(\frac{1}{8})^{26} > 2^{-80} > (\frac{1}{8})^{27}$.

Um leichtere Vektoren finden zu können werden zunächst die zweite und dritte Anforderung an den Störvektor (Tabelle 1) aufgehoben, und es wird zusätzlich nur verlangt dass der Vektor der Nachrichtenexpansion entspricht. Wang nimmt, basierend auf Beobachtungen der Störvektoren, vom Suchraum, der einer 80×32-Matrix mit 0/1-Einträgen entspricht, die ersten zwei Bits einer Spalte in Betracht und iteriert über die zwei Spalten der Länge 2^{16}. Es sind also 64 Spalten möglich für die jeweils 2^{32} Einträge berechnet werden $(64 \cdot 2^{32} = 2^{38})$. Dies kann man machen da 16 zusammenhängende Einträge auch die restlichen Vektoren festlegen. In diesem Fall können Störungen nur in den ersten beiden Spalten auftreten. Der prinzipiell große Suchraum beim SHA-1 enthält darüberhinaus viele Störvektoren die durch einfache Rotationen auseinander entstehen. Der beste von Wang gefundene differentielle Pfad der eine Kollision ermöglichte hatte allerdings ein zu großes Hamming-Gewicht ($hw(x) = 31$), was eine Wahrscheinlichkeit von 2^{-93} Kollisionen zu finden entspricht, viel schlechter noch als das Geburtstagsparadox es zulässt. Für den SHA-1 auf 75 Schritte reduziert findet sich ein Vektor mit Hamminggewicht 26.

Deshalb wird von Wang die Herangehensweise über Beinahe-Kollisionen auf mehreren Blöcken gewählt, da man hier auch die erste Bedingung aufheben kann um so Störvektoren mit kleinerem Hamming-Gewicht zu finden. Der Suchraum wird mittels der Nachrichtenexpansion weiter berechnet als Schritt 80, und die Suche bezieht sich auf ein beliebiges 80-Schritt-Interval. Wang findet direkt mehrere Störvektoren mit ausreichend kleinem Hamming-Gewicht. Ein gewählter Vektor mit $hw(x) = 25$ kann benutzt werden um Beinahe-Kollisionen für den vollen SHA-1 mit einer Wahrscheinlichkeit von 2^{-68} finden. Wenn eine Beinahe-Kollision für einen Nachrichtenblock gefunden wurde, hat die Suche nach einem folgenden Nachrichtenblock der genau entgegengesetzt Differenzen aufweist, die gleiche Komplexität wie für den ersten Block, so dass nur der Faktor 2 in die Laufzeit mit einfliesst. Hieraus ergibt sich die Gesamtkomplexität von 2^{69}. Dank Eigenschaften der IF-Funktion und Methoden zur Nachrichtenmodifikation können die Anforderungen an die Arbeitsvariablen für die ersten 22 Schritte wie bei SHA-0 alle im Vorraus erfüllt werden [2] und der probabilistische Teil, und damit die ganze Suchkomplexität, besteht nur noch aus den Runden 2-4, für die der Störvektor ein entsprechend geringes Hamminggewicht haben sollte.

6 Die Zukunft der SHA-Familie

6.1 Angriffe auf SHA-2

Bisher wurden keine Angriffe auf die volle Rundenzahl einer der SHA-2 Algorithmen veröffentlicht. Bekannt wurden Kollisionen auf den SHA-256 und SHA-512 die auf 22 von 80 (SHA-512) bzw 24 von 64 (SHA-256) Runden reduziert sind. Das kryptanalytische Interesse an diesen Hashfunktionen dürfte im Anbetracht der weniger verbreiteten Verwendung und der absehbaren Zeitspanne bis zur Standardisierung von SHA-3 entsprechend geringer sein, nicht zuletzt da sich Ergebnisse kaum oder gar nicht auf den SHA-3 übertragen lassen werden.

6.2 Der nächste Standard - SHA-3

Da bisherige Hashalgorithmen immer nur für einen kurzen Zeitraum sicher waren machte sich das NIST daran für die nächste Generation von sicheren Hashalgorithmen vorzusorgen. So wurde im Jahr 2007 die "NIST hash function competition" ausgerufen um einen neuen, sicheren Hashstandard zu finden [7]. Dieser Aufruf ähnelt dem Vorgehen beim AES, dem Advanced Encryption Standard, der seinerzeit den veralteten DES ersetzte. Neben dem gesteigerten akademischen Interesse an den Hashalgorithmen und ihrer Kryptanalyse werden auf diese Weise auch Vorwürfe entkräftet, dass bei der Entwicklung von Hashalgorithmen unter Ausschluss der Öffentlichkeint innerhalb von US-Behörden intentionelle Schwachstellen eingefügt werden könnten. Schon die Wahl der Rundenkonstanten $K^{(t)}$ für SHA-1 zeugt von der Einstellung Zweifel an der Wahl bestimmter Parameter zu zerstreuen, wie sie nach der Änderungen der S-Boxen von DES durch die NSA massiv laut wurden.

Der Zeitrahmen der SHA-3-Competition zeigt mit welcher Sorgfalt an die Konstruktion eines kryptografisch sicheren Hashverfahrens herangegangen wird. Zwischen dem Zeitpunkt der Ausschreibung (November 2007) und dem angepeilten Zeitpunkt der Standardisierung (2012) liegen fast 4 Jahre, die vom NIST in mehrere Runden aufgeteilt werden. Dabei wurden von den 51 für die erste Runde zugelassenen Algorithmen nur 14 Algorithmen in die zweite Runde aufgenommen. Für die ausgeschiedenen Algorithmen wurden zu großen Teilen Kollisionsattacken gefunden die (teilweise signifikant) unter dem theoretischen Maximum lagen. In manchen Fällen wurden auch Komplexitätsabschätzungen oder sogar Beispiele für Preimage und Second-Preimage-Attacken bekannt.

Von den Algorithmen die es in Runde 2 geschafft haben wird nach einem Jahr nochmal eine Teilmenge gebildet die dann als letzte Runde vor der Bekanntgabe des Gewinners geprüft werden. Kriterien für die Evaluation der Kandidaten sind neben der Sicherheit auch die Geschwindigkeit in Hard- und Software, der Speicherverbrauch, Flexibilität von Parametern wie Ausgabelänge (um alte Funktionen zu ersetzen) und Rundenzahl und der Verständlichkeit von Referenzimplementationen. Da alle Algorithmen offen und lizenzfrei sein müssen kann man die Entwicklung der Algorithmen während des Wettbewerbs verfolgen und kommentieren. Änderungen dürfen z.B. bei den geschwindigkeitsoptimierten Implementationen der Designer gemacht werden. Obwohl die Algorithmen und die Kommentare zu möglichen Problemen öffentlich sind liegt die endgültige Entscheidung für den Gewinner beim NIST.

7 Fazit

Wir haben gesehen wie SHA-0 und SHA-1 aufgebaut sind und mit welchen Mitteln versucht wird Unterschiede in Nachrichten möglichst schnell und weit über den Ausgabewert zu verteilen. Bei der Kryptanalyse der Algorithmen haben sich die nicht-linearen Rundenfunktionen als größte Hürde dargestellt, auch wenn die Frage bleibt ob andere Funktionen oder die Ersetzung der XOR-Runden die erfolgreiche Kryptanalyse nicht noch weiter verzögert hätten.

Auf SHA-1 bezogen zeigt sich im Verlauf der Veröffentlichungen, dass neue Ergebnisse immer einen gemeinsamen Teil bei der Konstruktion von Kollisionen aufweisen. Der wichtige Faktor um effizientere Pfade zu finden unterscheidet sich jedoch stark und ist einzig von der Kreativität der Autoren abhängig. Eine allgemeingültige und übersichtliche Methode zur Kollisionsfindung und der involvierten Techniken aufzuzeigen gibt es schlichtweg nicht. Daher ist es nicht vorauszusehen mit welcher Geschwindigkeit bessere Verfahren entwickelt werden oder wo der nächste Schritt von der Komplexität gesehen her anzusiedeln ist. Eine untere Grenze anzugeben ist ebenfalls nicht möglich. Wie auch bei älteren Algorithmen beobachtet ist zu erwarten, dass neue Ergebnisse für den SHA-0/SHA-1 auch bei der Analyse neuerer Hashalgorithmen gefunden werden, sofern diese Komponenten besitzen die sich ähnlich verhalten. Es bleibt zu erwarten ob die letzten Ankündigungen zur Kollisionswahrscheinlichkeit ([8], [5]) noch überarbeitet werden und ob die Wahrscheinlichkeit infolgedessen steigt oder sinkt. Die momentan bekannten Angriffe auf SHA-1 sind noch weit vom praktisch berechenbaren entfernt.

Literatur

[1] BELLOVIN, S.M., E.K. RESCORLA und N. RESONANCE: *Deploying a New Hash Algorithm.* 2006.

[2] BIHAM, E. und R. CHEN: *Near-collisions of SHA-0.* Lecture Notes in Computer Science, Seiten 290–305, 2004.

[3] BIHAM, E., R. CHEN, A. JOUX, P. CARRIBAULT, C. LEMUET und W. JALBY: *Collisions of SHA-0 and Reduced SHA-1.* In: *Advances in Cryptology–EUROCRYPT*, Band 3494, Seiten 36–57. Springer, 2005.

[4] CHABAUD, F. und A. JOUX: *Differential collisions in SHA-0.* Lecture Notes in Computer Science, Seiten 56–71, 1998.

[5] COCHRAN, M.: *Notes on the Wang et al.* 2^{63} *SHA-1 Differential Path.* Cryptology ePrint Archive, Report 2007/474, 2007. `http://eprint.iacr.org/`.

[6] DANG, Q.: *Recommendation for Applications Using Approved Hash Algorithms*. NIST Special Publication, 800:107, 2009.

[7] FEDERAL REGISTER: *Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA-3) Family*. 72:62213–62220, 2007-11-02.

[8] MCDONALD, C., P. HAWKES und J. PIEPRZYK: *Differential Path for SHA-1 with complexity $O(2^{52})$*. –withdrawn–.

[9] NAITO, Y., Y. SASAKI, T. SHIMOYAMA, J. YAJIMA, N. KUNIHIRO und K OHTA: *Improved collision search for SHA-0*. Lecture Notes in Computer Science, 4284:21, 2006.

[10] NIST: *Federal Information Processing Standards Publication 180*. Secure Hash Standard (SHS), 1993.

[11] NIST: *Federal Information Processing Standards Publication 180-1*. Secure Hash Standard (SHS), 1995.

[12] NIST: *Federal Information Processing Standards Publication 180-2*. Secure Hash Standards (SHS), 2002.

[13] NIST: *Federal Information Processing Standards Publication 180-3*. Secure Hash Standards (SHS), 2008.

[14] RIJMEN, V. und E. OSWALD: *Update on SHA-1*. Technischer Bericht, Springer, 2005.

[15] STEVENS, M., A. SOTIROV, J. APPELBAUM, A. LENSTRA, D. MOLNAR, D.A. OSVIK und B. DE WEGER: *Short chosen-prefix collisions for md5 and the creation of a rogue ca certificate*. In: *Proceedings of the 29th Annual International Cryptology Conference on Advances in Cryptology*, Seite 69. Springer, 2009.

[16] WANG, X.: *The Collision attack on SHA-0*. Chinese, not translated yet, 1997.

[17] WANG, X., Y.L. YIN und H. YU: *Efficient collision search attacks on SHA-0*. Lecture Notes in Computer Science, 3621:1–16, 2005.

[18] WANG, X., Y.L. YIN und H. YU: *Finding collisions in the full SHA-1*. Lecture Notes in Computer Science, 3621:17–36, 2005.