fhwedel
UNIVERSITY OF APPLIED SCIENCES

FACHHOCHSCHULE WEDEL

Seminararbeit

in der Fachrichtung

Wirtschaftsingenieurswesen

Thema:

Kryptographische Hash - Funktionen

Eingereicht von:	Niklas Jensen Mat-Nr: 101636 E-Mail: wing101636@fh-wedel.de
Erarbeitet im:	6. Semester
Vortrag gehalten am:	14.05.2018
Abgegeben am:	28.05.2018
Betreuer:	Prof. Dr. Michael Anders Fachhochschule Wedel Feldstraße 143 22880 Wedel

Inhalt

1. Einleitung

Meldet man sich beispielsweise von Zuhause oder in der Firma mit seinem Benutzerkonto in bestimmten Programm an, werden der Benutzername und das Passwort abgefragt. Dabei wird das Passwort, welches man zu Anfang festlegen durfte, nicht an sich abgefragt, sondern der Hash – Wert zu diesen Passwort wird mit dem zugehörigen in einer Datenbank gespeicherten Hash – Wert verglichen.

Jedes Passwort wird zunächst unter Verwendung einer Hash - Funktion verarbeitet und danach in einer bestimmten Passwortdatei des Systems gespeichert. Da Hash – Funktionen Einwegfunktionen sind und die Berechnung von A nach B funktioniert, jedoch nicht von B nach A, lässt sich aus dem Hash – Wert des Passwortes das Passwort selber nicht wiederherstellen. Beim Einloggen wird das eingegebene Passwort abgebildet und mit dem gespeicherten Hashwert verglichen. Sind diese gleich, so stimmen die Passwörter überein und es wird der Zugriff auf das gewünschte Programm gewährt. Erhält ein Angreifer Zugriff auf die Datenbank mit den Hash – Werten kann er dennoch kein Originalpasswort rekonstruieren (vgl. Abbildung 1).

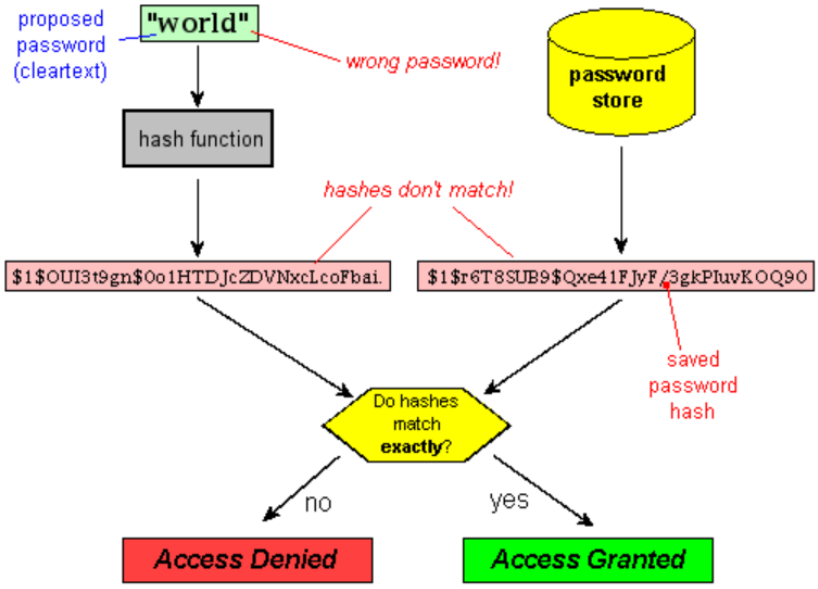


Abbildung 1 Passwortschutz anhand von Hash – Funktionen[1]

[1] Vgl. http://www.fastweb.it/internet/password-hashing-cos-e-e-cosa-serve/

Ein weiteres Anwendungsbeispiel von kryptographischen Hash – Funktionen ist der digitale Fingerabdruck von Nachrichten, die eine Person A einer anderen Person B schickt, um sich zu vergewissern, dass die Nachricht auf dem Weg nicht manipuliert wurde. In Abbildung 2 möchte Alice ihrem Bekannten Bob die Nachricht „Hallo!“ zukommen lassen. Um sich zu vergewissern, dass die Nachricht auch nicht manipuliert wurde, erzeugt sie mit der Hash –Funktion MD-5 den Hash – Wert „F02a27e2f102fa7505d94fafb494b40875dec5b6“ und übermittelt diesen zusätzlich. Bob prüft die Nachricht mit derselben Hash – Funktion und kann feststellen, dass die beiden Hash – Werte sich gleichen. Daraus schlussfolgert er, dass die Nachricht nicht manipuliert wurde.

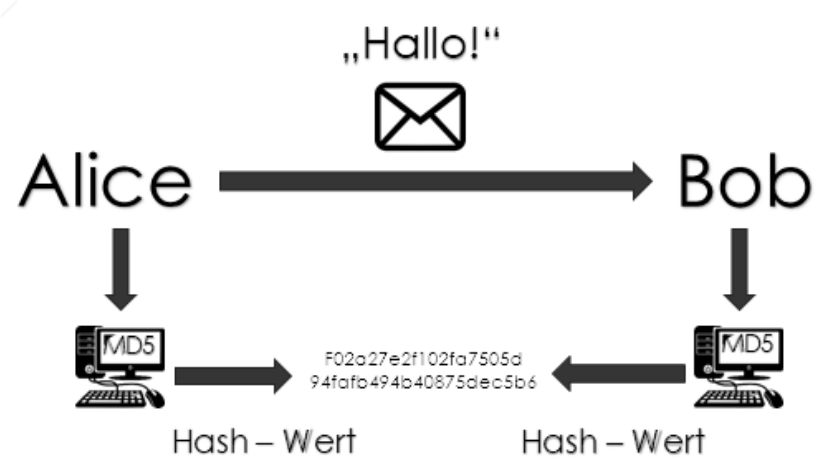


Abbildung 2 Digitaler Fingerabdruck

2. Kryptographische Hash – Funktionen und ihre Eigenschaften

a. Grundlegende Eigenschaften und Funktionsweise

Kryptographische Hash – Funktionen sind mathematische Einwegfunktionen, die einen bestimmten Wert X berechnen, den sogenannten „Hash –Wert“[2]. Hierbei spielt folgende Eigenschaft die wichtigsten Rolle in den Hash-Funktionen: Von X, also dem Hash-Wert, darf man nicht auf das Urbild M schließen können bzw. dieses leicht errechnen. Am Beispiel eines Tellers lässt sich das Funktionsprinzip einer Hash-Funktion am leichtesten erklären. Das Zerbrechen eines Tellers in hunderte Einzelteile ist leicht, doch diese Einzelteile wieder zu einem funktionierenden Teller zusammenzusetzen nahezu unmöglich. Mathematische ausgedrückt ist eine kryptographische Hash – Funktion ein Algorithmus, der zu einer beliebig langen Nachricht M einen in der Theorie nicht manipulierbaren Prüfwert X (digitaler Fingerabdruck“) mit einer festen Länge erzeugt (vgl. Abbildung 3). Der einzige Parameter, der in die Hash – Funktion eingeht, ist die Nachricht selbst, d.h. im Vergleich zu symmetrischen und asymmetrischen Verschlüsselungen, dass in die Berechnung von Hashwerten kein weiterer Schlüssel von außen eingeht. Es gibt somit kein Geheimnis, das sich Sender und Empfänger separat übermitteln müssen, um die verschlüsselte Nachricht zu entschlüsseln. Lediglich die Hash – Funktion dient als Verschlüsselungs- und Entschlüsselungsgerät. Zusätzlich ist der Hash – Wert immer einzigartig, das heißt, dass der Hash – Wert X_1 für Nachricht M_1 zu jedem Zeitpunkt gleich ist.[3]

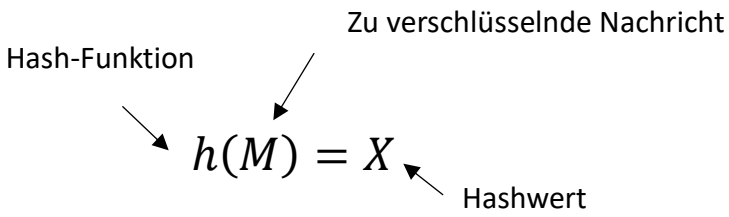


Abbildung 3 Mathematische Darstellung einer kryptographischen Hash - Funktion

Das Grundprinzip einer Hash – Funktion ist das Imitieren einer „echten“ Zufallsfunktion, in der Theorie sollen alle erzeugten Hash – Werte dem Zufall entspringen und gleichmäßig verteilt über den möglichen Wertebereich fallen. Zwei ähnliche Nachrichten sollen nach dem Zufallsprinzip vollkommen unterschiedliche Hash – Werte liefern und einem möglichen

[2] Eng. to hash = zerhacken
[3] Vgl. https://www.sans.edu/cyber-research/security-laboratory/article/hash-functions

Interessenten keine Angriffsmöglichkeit geben einen bestimmten Hash – Wert auf Basis einer ähnlichen Nachricht und dessen bekannten Hash – Wert zu entdecken.

Der Länge des Eingabestrings darf beliebig groß sein, da die Kernfunktion einer Hash – Funktion die sogenannte Kompressionsfunktion ist, die den Eingabestring auf eine vorbestimmte Größe erweitert, um diesen erfolgreich zu verarbeiten (siehe Abschnitt 4 „Konstruktionsprinzipien von Hash – Funktionen"). Der Ausgabestring hat eine feste Länge, die vom Programmierer der Hash – Funktion festgelegt wurde und nicht beeinflussbar ist. Diese Ausgabelänge ist charakteristisch und ausschlaggebend für die Hash – Funktion.[4]

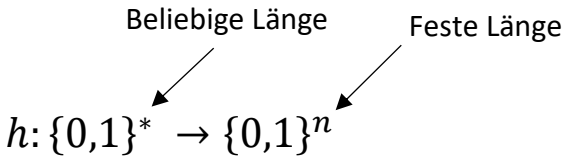


Abbildung 4 Eingabe- und Ausgabelänge für eine beliebige Hash - Funktion

In Abbildung 4 wird das Prinzip der „Kompressionsfunktion" deutlich, in dem eine beliebige Länge auf eine feste Länge komprimiert wird. Hierbei muss beachtet werden, dass „beliebig" immer bedeutet, dass der Eingabestring größer als der Ausgabestring ist. Der Ausgabewert einer Hash – Funktion ist der Hash – Wert selber. Dieser hat immer eine konstante Größe und ist charakteristisch für die Hash – Funktion. In den Anfängen gab es Hash – Funktionen mit einer Ausgabegröße von 64 Bit. Bedingt durch den technologischen Fortschritt musste man diese Größe viele Male nach oben hin korrigieren, da z.B. 64 Bit leicht zu knacken ist. Deshalb gibt es mittlerweile Hash – Funktionen, die einen Ausgabewert von 512 Bit haben, also das Achtfache von 64 Bits.

[4] Wätjen, Dietmar: Kryptographie – Grundlagen, Algorithmen, Protokolle, 1. Auflage, S. 87-100

b. Exkurs: Darstellung von Hash – Werten

Die Einheit Bit ist die kleinste Speichereinheit in der elektronischen Datenverarbeitung und kann genau zwei Zustände als Information darstellen. Es gibt entweder „AN“ oder „AUS“, „Wahr“ oder „Falsch, 1 oder 0. Das heißt ein Bit besteht entweder aus einer Eins oder einer Null. Wenn man nun mehrere Bits aneinander hängt, also mehrere Bits hintereinander schreibt (Binärcode), können folglich pro Bit zwei Zustände definiert werden. Das heißt für vier Bits kann man $2^4 = 16$ verschiedene Darstellungsweisen wählen. Computer rechnen ausschließlich mit Einsen und Nullen, sodass es für jede Zahl, Buchstaben, Ziffer etc. eine direkt Übersetzung in der Binärcode-Schreibweise gibt. In entsprechenden Tabellen wie der ASCII Tabelle, „American Standard Code for Information Interchange“, sind jedem Buchstaben, Zeichen oder Ziffer ein entsprechender Binärcode zugewiesen.

Hash – Werte werden basierend auf dem Binärcodesystem in einem Hexadezimalsystem ausgedrückt. Dieser besagt lediglich, dass die Zahlen 0 – 9 direkt verwendet werden und die folgenden Zahlen 10 -15 mit den Buchstaben A – F ersetzt werden, sodass es keine zweistelligen Zeichen im Hash – Wert gibt. In Abbildung 5 ist eine Übersicht zwischen dem Dezimalcode-, Binärcode- und Hexadezimalcodesystem zu sehen.

Dezimal	Binär	Hexadezimal
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

[5]

Abbildung 5 Darstellungsweise von Zahlen und Buchstaben in Dezimal, Binär und Hexadezimal

Darauf aufbauend werden somit Hash – Werte in einer Hexadezimalschreibweise dargestellt, die beispielsweise wie folgt aussieht. Der String „Hallo Welt!“ liefert mit der Hash – Funktion

[5] vgl. http://www.airnet.de/cr1-gfe/de/html/Math_learningObject9.xml

MD-5 diesen Hash – Wert: 9e107d9d372bb6826bd81d3542a419d6.[6] Eine Kombination aus Buchstaben von A-F und den Zahlen 0-9 ist deutlich erkennbar.

c. Kollision von Hash – Werten

Die Anzahl der unterschiedlichen Hash – Werte ist basierend auf den zwei Zuständen der Bits und der Ausgabelänge der Hash – Funktion auf 2^n begrenzt. Somit gibt es theoretisch für moderne Hash – Funktionen praktisch eine sehr große Zahl an Nachrichten für eine vorgegebenen Hash – Wert und anders herum. Zwei verschiedene Nachrichten M_1 und M_2, die denselben Hash – Wert liefern, bezeichnet man als Kollision und somit als Fehler in der Hash – Funktion. Daraus schlussfolgert man, dass eine gute Hash – Funktion kollisionsfrei sein sollte, was aber in der Theorie nahezu unmöglich ist. Es wird empfohlen, dass die Veränderung eines einzigen Bits in der Nachricht den Hash – Wert um mindestens der Hälfte verändern sollte, damit der Hash –Wert die Eingabe unkenntlich macht. Dieses Prinzip nennt sich „Lawineneffekt".[7] Für folgenden Eingabestring „Frank jagt im komplett verwahrlosten Taxi quer durch Bayern" liefert die SHA – 512 Hash – Funktion den Wert:

af9ed2de700433b803240a552b41b5a472a6ef3fe1431a722b2063c75e9f07451f67a28e37d09cde769424c96aea6f8971389db9e1993d6c565c3c71b855723c

Ändert man den Eingabestring zu „Frank jagt im komplett verwahrlosten Taxi quer durch Bayern" wird folgender Hash – Wert ausgegeben:

90b30ef9902ae4c4c691d2d78c2f8fa0aa785afbc5545286b310f68e91dd2299c84a2484f0419fc5eaa7de598940799e1091c4948926ae1c9488dddae180bb80

An diesem Beispiel wird deutlich, dass die Veränderung eines einzelnen Buchstabens den Hash – Wert vollkommen unkenntlich macht im Vergleich zu dem Hash Wert zuvor. Dieses Beispiel führt zu den drei grundlegenden Eigenschaften, die als Richtlinien zur Entwicklung und Prüfung von Hash – Funktionen aufgestellt wurden. Werden alle drei folgenden Eigenschaften eingehalten, gilt die Hash – Funktion als sicher:

[6] vgl. https://hashgenerator.de/

[7] Vgl. http://www-stud.rbi.informatik.uni-frankfurt.de/~razi/diplomarbeit/DHTML/node16.htm

1. Preimage Resistance: Es muss rechnerisch unmöglich sein zu einem gegebenen Hashwert h(m) eine beliebigen Nachricht m_1 zu finden (siehe Abbildung 6).

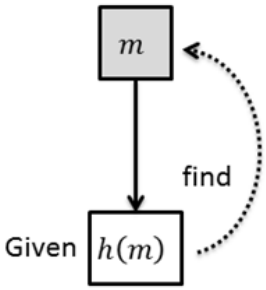


Abbildung 6 Preimage Resistance

2. Second Preimage Resistance: Es muss unmöglich sein zu einer vorgegebenen Nachricht m_1 und dem entsprechenden Hashwert $h(m_1)$ eine weitere Nachricht m_2 zu finden, die den gleichen Hashwert $h(m_1) = h(m_2)$ hat.

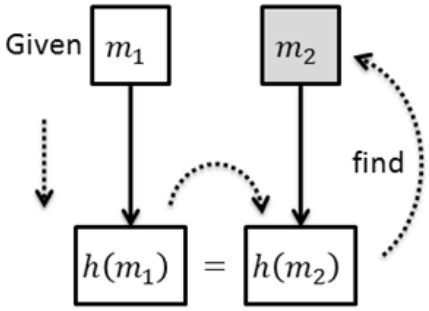


Abbildung 7 Second Preimage Resistance

3. Collision Resistance: Es muss unmöglich sein für zwei beliebige Nachrichten m_1 und m_2 $(m_1 \neq m_2)$ denselben Hashwert h(m) zu finden.

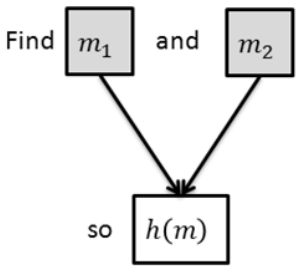


Abbildung 8 Collision Resistance

Diese Eigenschaften stellen somit die „perfekte“ kryptographische Hash – Funktion dar.[8]

Basierend auf diesen Eigenschaften wurde das Modell des „Random Oracle“ aufgestellt. Dieses gibt zu jeder Eingabe (n-bit lang) einen zufälligen Ausgabewert zurück, falls diese Eingabe zum ersten Mal abgefragt wird, ansonsten die gleiche Ausgabe wie bei der letzten Abfrage. Dies liefert somit die minimalen theoretischen Sicherheitsanforderungen einer Hash – Funktion. In der Realität sind alle Hash – Funktionen weniger sicher als ein Random Oracle Modell (ROM), die Theorie richtet sich aber nach dem ROM, um die Sicherheitsanforderungen so hoch wie möglich zu halten. Im Rahmen der drei Sicherheitseigenschaften, die zuvor vorgestellt wurden, lässt sich schließen, dass man, um ein Preimage oder Second Preimage zu bekommen, mindestens 2^n Berechnungsschritte machen muss. Für eine Kollision zweier Nachrichten, dass zwei Nachrichten den gleichen Hash – Wert liefern, liefert das „Geburtstagsparadoxon“ eine Anzahl von $2^{n/2}$ Berechnungsschritte.

Das heißt: Jeder Angriff auf eine Hash-Funktion, der ein Preimage oder Second Preimage in weniger als 2^n Schritte findet, bedeutet für die Hash-Funktion, dass sie weniger sicher als das Random Oracle ist.

d. Geburtstagsparadoxon

Das Geburtstagsparadoxon ist ein Beispiel dafür, dass bestimmte Wahrscheinlichkeiten intuitiv häufig falsch geschätzt werden. Es bezieht sich auf die Frage wie viele Personen sich in einem Raum befinden müssen, damit die Wahrscheinlichkeit, dass zwei beliebige Personen am selben Tag Geburtstag haben, bei 50 % liegt. Diese Statistik lässt sich auch auf die Problematik der Collision Resistance beziehen. Hierbei sind die Personen die Nachrichten und der Geburtstag ist der Hash – Wert. Somit stellt man sich die Frage wie viele Nachrichten gehasht werden müssen, damit man mindestens zwei Nachrichten mit demselben Hash-Wert mit einer Wahrscheinlichkeit von größer als 50 % findet. Die mathematische Herleitung ist für diesen Fall irrelevant. Relevant ist, dass ab einer Personenzahl von 23 die Wahrscheinlichkeit bei mehr als 50% liegt. Dies bedeutet für die Hash – Funktionen, dass im Falle der Collision Resistance die Anzahl der zu berechneten Nachrichten nun bei $2^{n/2}$ liegt und nicht bei 2^n.

[8] B. Denton, R. Adhami: Dept. of Electrical & Computer Engineering, The University of Alabama in Huntsville, Huntsville, AL, USA Modern Hash Function Construction

Damit hat sich die Anzahl der zu berechnenden Nachrichten um einen Schlag um die Hälfte halbiert. Diese Entdeckung war ausschlaggebend für die konstante Weiterentwicklung der Ausgabegröße der Hash – Funktionen. Von anfangs 64 Bits steigerte sich die Größe über 128 Bits bis hin zu 512 Bits bei der SHA – 512.[9]

3. Wie können Hash – Funktionen gebrochen werden?

a. Brute-Force-Attack

Der Brute-Force-Attack, übersetzt in Rohe-Gewalt-Angriff, behandelt das Ausprobieren und Raten aller möglichen Nachrichten bis möglicherweise eine Nachricht den gewünschten Hash – Wert liefert. Diese Methode ist anwendbar auf alle verschlüsselten Dateien, Nachrichten, Informationen und Passwörtern. Jedoch hat diese Methode einen offensichtlichen Nachteil: Die Rechenleistung der Computer ist zeitaufwendig und daraus schlussfolgernd irgendwann auch nicht mehr wirtschaftlich. Hierbei muss der Angreifer sich im Klaren sein wie viel Zeit und Kapazitäten er in den Versuch des Knackens setzen möchte. Grundsätzlich ist die Bitlänge des Hash-Wertes ausschlaggebend für die Entscheidungsfindung, da eine sehr große Bitlänge (bspw. 256Bit, 521Bit) rechnerisch mit der Brute-Force-Methode zu brechen ist.

Für die meisten privaten Anwender ist dieses Unterfangen schlichtweg nicht rentabel und viel zu zeitaufwendig, doch staatlich organisierte Unternehmen, die eventuell die genügend staatliche Förderungen genießen, besitzen mit Sicherheit über ausreichend große Kapazitäten einen Brute-Force-Attack für größere Bitlängen durchzuführen.

b. Dictionary Attack

Im Vergleich zur Brute-Force-Methode wird bei dem, übersetzt: Wörterbuchangriff, Dictionary Attack eine vorsortierte Liste von Strings verwendet, die dazu genutzt wird den gesuchten Hash – Wert zu aufzuspüren. Dabei werden nur Strings verwendet deren Erfolgschancen hoch genug sind und in einem sinnvollen Kontext stehen. Dies hat den Hintergrund, dass Benutzer grundsätzlich zu einfachen, kurzen und leicht merkbaren Passwörtern tendieren. Folglich werden keine zufällig kombinierten Buchstaben und Ziffern als Eingabestring geprüft, sondern nur eben solche, die für den Anwender Sinn ergeben. Jegliche Variationen der Passwörter, zum Beispiel willentlich eingebaute Fehler, die die Grammatik oder den Inhalt verzerren, würden ein Dictionary Attack verlangsamen oder

[9] Ertel, Wolfgang: Angewandte Kryptographie, 4. Auflage, S. 100ff.

möglicherweise fehlschlagen lassen. Wichtig hierbei ist die Überlegung welche möglichen Fehlervarianten bereits in dem „Dictionary“ des Angreifers aufgenommen wurden, da dies auch berücksichtigt werden muss.

c. Rainbow Tables

Die Rainbow Tables bilden den Mittelweg zwischen Zeitaufwand und notwendigem Speicherplatz. Die simpelste Methode wäre für jede Nachricht den Hash – Wert zu berechnen und dies solange durchzuführen bis man seinen Wert gefunden hat (vgl. Abbildung 9). Dies entspricht der Brute-Force-Methode, ist jedoch viel zu zeit- und rechenaufwendig. Die fortgeschrittene Variante wäre jeden errechneten Hash-Wert in einer Datenbank abzulegen und diese zu archivieren, so dass man Hashes nachschauen kann ohne diesen erneut berechnen zu müssen. Dies würde viel Zeit sparen, kostet aber für sehr große Bitlängen eine riesige Menge an Speicherkapazitäten, die nicht wirtschaftlich sein würden.

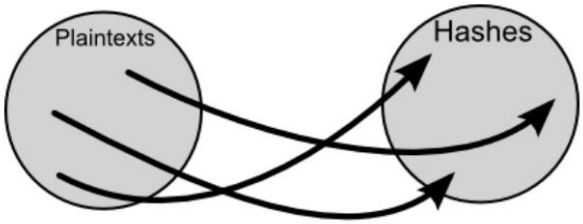


Abbildung 9 Umwandlung eines Klartext zu einem Hash –Wert

Rainbow Tables sind ein Kompromiss zwischen Zeit und Speicherplatz. Hierbei werden nach jeder Berechnung eines Hash – Wertes dieser durch eine Reduktionsfunktion wieder zu einem Klartext umgewandelt (vgl. Abbildung 10). Wichtig hierbei ist, dass es sich nicht um den zuvor gehashten Klartext handelt, da ansonsten die Eigenschaft einer Einwegfunktion hinüber wäre. Durch diese Reduktionsfunktion entstehen neue Klartexte, die ebenfalls durch die Hash – Funktion gehasht werden.

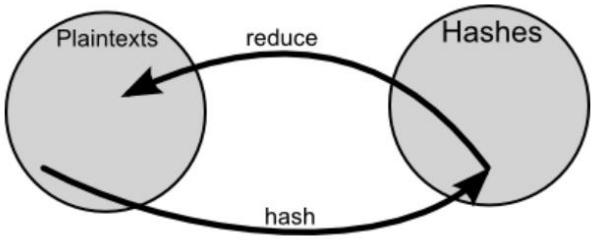


Abbildung 10 Reduktionsfunktion wandelt Hash-Wert in Klartext um

Nach diesem Prinzip entstehen Ketten von Hash – und Reduktionsfunktionen deren Sinn es ist jedem berechneten möglichen Passwort einem konkreten Hash-Wert zuzuordnen (vgl. Abbildung 11). Lediglich ein Start- und Endwert wird notiert und mit verschiedenen Startwerten wiederholt, um so viele Ketten zu erhalten wie möglich. Der Vorteil hierbei ist, dass man nur zwei Werte speichern muss, um Tausende von Hash – Werten durch einen einzigen Start- und Endwert zu repräsentieren. Dies ist speichereffektiv und verhältnismäßig schnell. Zusätzlich sind viele Rainbow Tables öffentlich zugänglich, sodass von anderen bereits Rechenarbeit geleistet wurde auf die man zugreifen kann. Der einzige Nachteil an dieser Methode ist die Möglichkeit sich im Kreis zu drehen, wenn die Reduktionfunktion einen bereits bekannten Klartext ausgibt.[10]

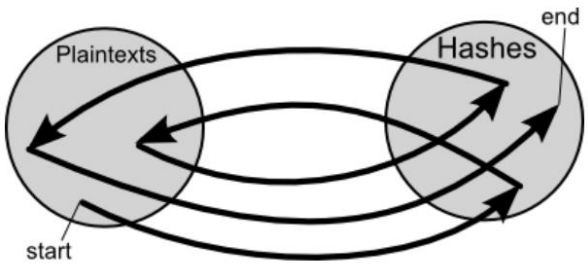


Abbildung 11 Kettenbildung mit einem Start - und Endwert

[10] http://kestas.kuliukas.com/RainbowTables/

4. Schutzmöglichkeiten vor Angriffen auf Hash – Funktionen

a. Salt & Pepper

Unter Salt & Pepper versteht man die Individualisierung seines Eingabewertes, damit bekannte Crackingmethoden wie der Dictionary Attack oder die Rainbow Tables nicht auf ihre gespeicherten Datenbanken zugreifen können. Der Salt ist ein zusätzlicher Datensatz, der als Input für die Hash – Funktion verwendet wird. Dieser wird für jedes Passwort zufällig generiert, auch wenn die Passwörter identisch sind. Somit wird der Hash Wert sowie das Salt in der Datenbank abgespeichert, sodass Angriffe überdurchschnittlich aufwendig werden. Für jeden Klartext muss jetzt der neue Hash (Passwort + Salt) berechnet werden. Dies verlangsamt die Arbeitsweise der Rainbow Tables um einiges und macht diese in vielen Fällen nutzlos. In Abbildung 12 wird die Funktionsweise eines Salts deutlich. Selbst wenn die Passwörter zweier Anwender gleich sind, verändert das Salt den entstehenden Hash –Wert und macht es dem Angreifer unmöglich Parallelen zwischen den Passwörtern zu ziehen.[11]

Username	Salt value	String to be hashed	Hashed value = SHA256 (Password + Salt value)
user1	E1F53135E559C253	password123+E1F53135E559C253	72AE25495A7981C40622D49F9A52E4F1565C90F048F59027BD9C8C8900D5C3D8
user2	84B03D034B409D4E	password123+84B03D034B409D4E	B4B6603ABC670967E99C7E7F1389E40CD16E78AD38EB1468EC2AA1E62B8BED3A

Abbildung 12 String und Salt ergeben einen neuen Hash - Wert

Zusätzlich gibt es den Pepper, der eine von extern zusätzlich eingefügte Zeichenfolge ist, die mit dem Passwort kombiniert wird. Der Pepper ist entweder für alle Passwörter gleich oder es gibt für jedes Passwort einen individuellen Pepper. Unabhängig davon wird der Pepper separat von den Passwörtern gespeichert, was der wesentliche Unterschied zu dem Salt ist.

[11] https://security.stackexchange.com/questions/51959/why-are-salted-hashes-more-secure-for-password-storage

b. Key Stretching – „Key Derivation Function“

Das Key Stretching stellt eine weitere Funktion dar, um die Sicherheit von Passwörtern zu erhöhen. Grundsätzlich unterscheidet man zwei Anwendungsfälle, in denen das Key Stretching von Vorteil sein kann:

1. „Zur Erhöhung der Entropie von Passwörtern, die als Schlüssel zu Verschlüsselung von Klartexten dienen sollen
2. Das Ausbremsen von Crackern, welche Passwort-Hashes mit Brute Force-Angriffen knacken wollen“[12]

Das Prinzip des Key Stretching besteht darin, dass das Passwort mehrfach gehasht wird, sodass eine Art Zyklus entsteht durch den der Prozess der Hash – Berechnung des Angreifers künstlich verlangsamt wird (vgl. Abbildung 13). In Zusammenarbeit mit dem „Salt“ wird somit das Cracken um einiges verlangsamt, da zum Beispiel übliche Rainbow Tables nicht mehr benutzt werden können. Für Anwender ist es irrelevant, wenn das Passwort in einer Sekunde statt einer Millisekunde überprüft wird. Für einen Cracker, der die Berechnung millionenfach ausführen muss, verlängert sich die Rechenzeit dagegen um viele Jahre und macht es somit für ihn unwirtschaftlich und zu aufwendig.

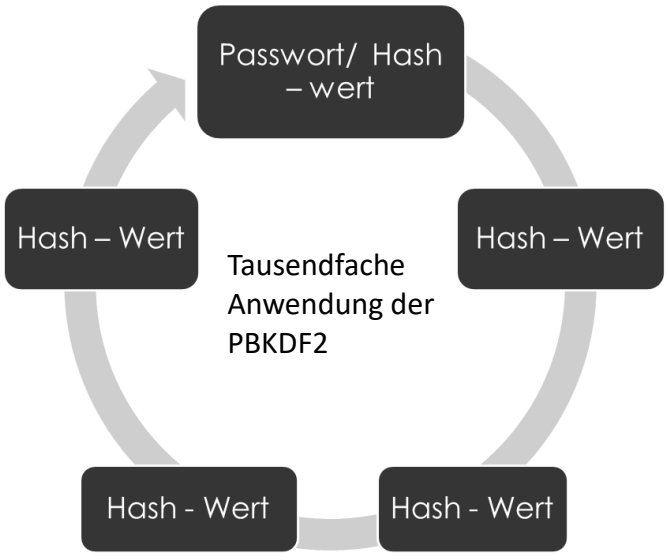


Abbildung 13 Zyklus des Key Stretching

In dem Bereich des Key Stretchings werden spezielle Hash – Funktionen verwendet, deren Ziel es ist das Hashing möglichst aufwendig zu gestalten. Hash – Funktionen wie PBKDF2 –

[12] http://www.it-administrator.de/lexikon/key_stretching.html

„Password Based Key Derivation Function 2“, „bcrypt“ und „scrypt“ benutzen dabei Algorithmen, die speziell auf das Verlangsamen dieses Prozesses ausgelegt sind. Vergleichsweise sind Hash – Funktionen wie MD-5 und SHA-1 mit dem Ziel konstruiert worden Daten möglichst effizient zu hashen, da sie etwa auch zur Verifizierung von großen Dateimengen verwendet werden. Hierbei soll der Memory Footprint möglichst gering gehalten und die Geschwindigkeit relativ hoch angesetzt werden. Dies liefert dementsprechend auch Angriffsspielraum für Brute Force Angriffe und Rainbow Tables.

5 . Konstruktionsprinzipien von Hash – Funktionen

a. Allgemeine iterative Konstruktionsweise

Wie bereits eingangs im Kapitel 2 „Hash – Funktionen und ihre Eigenschaften“ beschrieben, bildet eine Hash –Funktion Eingabestrings einer beliebigen Länge auf einen Ausgabestring mit fester und kürzerer Länge ab. In Abbildung 14 wird der iterative Prozess deutlich, in dem eine Nachricht in einer Kompressionsfunktion geleitet wird, welche die Nachricht verarbeitet und im Anschluss einen neuen Hash – Wert ausgibt. Innerhalb dieses Prozesses gibt es drei wesentliche Schritte, die vollzogen werden müssen.

h: Hash – Wert
g(h): Transformationsfunktion mit hat als Input

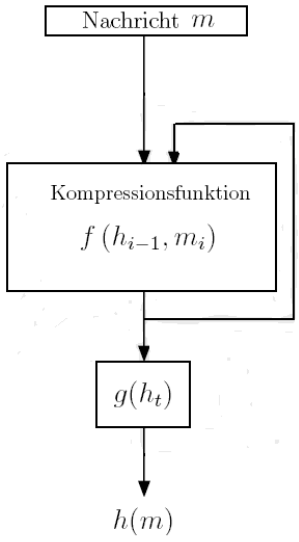


Abbildung 14 iterative Konstruktion einer Hash - Funktion

Im ersten Prozessschritt, der Vorbereitungsphase, wird die zu verarbeitende Nachricht (dargestellt in Binärcodeschreibweise) in eine bestimmte Anzahl an Blöcken aufgeteilt, die eine festgelegte Bit-Größe haben, welche durch die Hash – Funktion vorbestimmt ist. Ist die Nachrichtlänge kein Vielfaches der gewünschten Blockgröße muss die Nachricht verlängert werden. Dies geschieht mithilfe des „Paddings“ (siehe Kapitel 5b „Length Padding“).

Der zweite Prozessschritt befasst sich mit dem iterativen Verarbeiten der Nachrichtenblöcke unter Zuhilfenahme der Kompressionsfunktion, die eine eigenständige Funktion innerhalb der Hash – Funktion darstellt. Hierbei wird jeder Block wird mit dem Ergebnis der vorhergehenden Iteration als Input für die Kompressionsfunktion benutzt. Der daraus entstehende iterative Prozess liefert den Hash –Wert.

Im letzten, dritten Prozessschritt wird, wenn notwendig und die Hash – Funktion nicht die gewünschte Länge besitzt, eine Ausgabetransformation auf den Hash –Wert angewendet. [13]

b. Length Padding

Das „Length Padding" bezieht sich auf die Vorbereitungsphase der Konstruktion von Hash – Funktionen. Wie bereits erwähnt, können die Kompressionsfunktionen nur Blöcke mit definierter Größe aufnehmen. Dabei wird eine Nachricht, die nicht ein Vielfaches der notwendigen Blockgröße ist, mithilfe eines bestimmten Schemas auf ein Vielfaches erweitert. Betrachtet man bspw. die Hash – Funktion SHA-1, die einen Ausgabewert von 160 Bits und eine interne Blockgröße 512 Bits hat, so muss die eingehende Nachricht ein Vielfaches der 512 Bits sein. Die interne Blockgröße ist die Größe, in die die Nachricht aufgeteilt wird, damit die Kompressionsfunktion diese blockweise verarbeiten kann. Diese Blockgröße ist variabel und für jede Hash – Funktion unterschiedlich. Es gibt viele verschiedene Paddingschemata für Hash – Funktionen. Im Folgenden wird jedoch das von Ralph Merkle verwendete Schema (siehe Kap. 5c) beschrieben.

Das Padding-Prinzip funktioniert nach dem folgenden Schema:

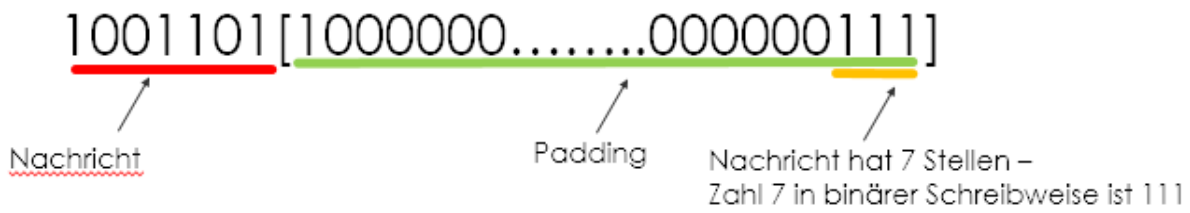


Abbildung 15 schematische Darstellung des Paddings

Die zu paddende Nachricht ist in Abbildung 15 mit einer roten Markierung dargestellt. Möchte man diese Nachricht nun auf eine Blocklänge von 512 Bits der SHA-1 erweitern, wird deutlich,

[13] Mohammed Meziani: Diplomarbeit, 13.08.2007, „Konstruktion von Hashfunktion" , S.19-21

dass „100110“ nicht 512 Bits sind. Deshalb muss die Nachricht erweitert werden. Man beginnt mit einer 1 und fügt solange 0 hinzu bis man den Punkt erreicht an dem zusätzlich noch die Länge der Nachricht in Binärcode angehängt wird (grüne und gelbe Markierung). [14]

c. Merkle – Damgård Konstruktion

Im Jahr 1979 wurde die Merkle – Damgård Konstruktion von Ralph Merkle in seiner Dissertation das erste Mal vorgestellt. Die iterative Konstruktionsweise einer Hash –Funktion, wie in Kapitel 5a beschrieben, ist Grundbestandteil der Merkle – Damgård Konstruktion.

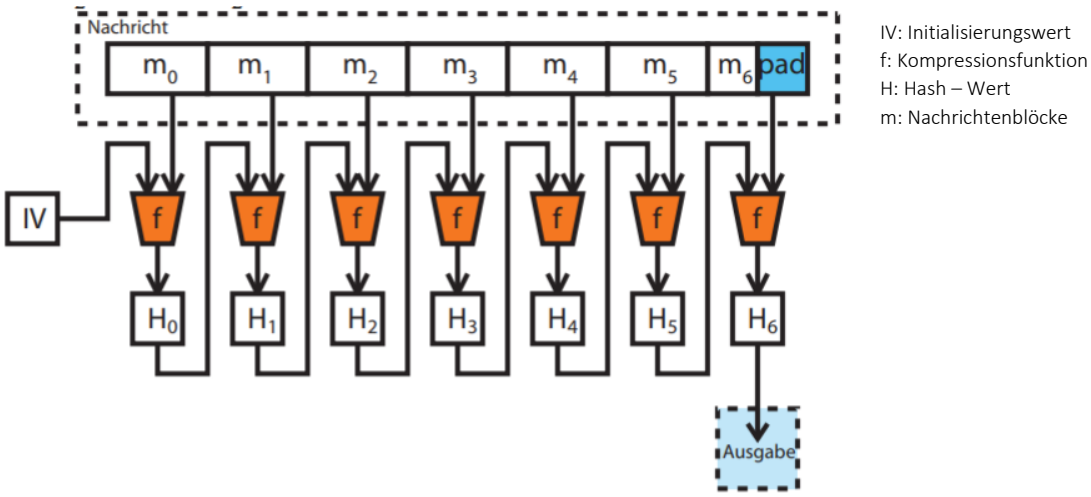


Abbildung 16 Merkle - Damgård Konstruktion

Eine Nachricht wird in Nachricht (beliebiger Größe) wird in eine bestimmte Anzahl gleichgroßer Nachrichtenblöcke $(m_0, \ldots m_{r-1})$ aufgeteilt, die eine bestimmte Bit Größe haben. Durch die beliebige Größe der Nachricht kann es passieren, dass die Kompressionsfunktionen die Nachricht nicht verarbeiten kann. Kompressionsfunktionen können nur einheitliche Eingangsgrößen verarbeiten. Dies wird durch die Padding-Funktion gelöst, sodass die Nachricht der vorgegebenen Länge entspricht. Die Kompressionsfunktion bricht die Nachricht in gleichgroße Blöcke auf und verarbeitet die Blöcke nacheinander. In jeder Runde wird das Ergebnis der Kompressionsfunktion der Vorrunde verwendet. Es entsteht die sogenannte „Chaining Value“. Das Ergebnis der Merkle – Damgård Konstruktion ist ein Hash – Wert.

[14] Mridul Nandi: Characterizing Padding Rules of MD Hash Function Preserving Collison Security, S. 3 - 4

d. Vertreter der Merkle – Damgård Konstruktion

Bekannte Vertreter der Merkle – Damgård Konstruktion ist die Modellreihe der Message Digest (MD) Hash – Funktionen. 1989 wurde die MD-2 Hash – Funktion von Ronald Rivest veröffentlicht, daraufhin wurden weitere Hash – Funktionen dieser Modellreihe veröffentlicht bis hin zur MD – 5 im Jahr 1992. Diese Hash – Funktion war jahrelang eine Standardverschlüsselung und hat eine Hash – Wert Größe von 128 Bit und eine interne Blockgröße von 512 Bit. Im Jahr 2008 wurde die MD-6 Hash – Funktion veröffentlicht, jedoch sind generell alle MD Funktionen als gebrochen angesehen, da mehrere Kollisionsangriffe erfolgreich durchgeführt wurden.

Im Jahr 1995 wurde die Hash – Funktion „Secure Hash Algorithm“ (SHA-1) von der NSA entwickelt und veröffentlicht. Kurz zuvor wurde die SHA-0 Funktion veröffentlicht, jedoch unmittelbar wieder eingezogen aufgrund von mangelnden Sicherheitseigenschaften. Die SHA-1 hat eine interne Blockgröße von 512 Bits und einen Output von 160 Bits. Bis 2017 war diese Funktion der Verschlüsselungsstandard von bspw. Microsoft, Google, Apple und Mozilla und offiziell bis 2010 der Verschlüsselungsstandard der US-Regierung. Danach wurde theoretisch auf SHA2 umgestellt, praktisch hat dies noch einige Jahre gedauert bis effektiv umgestellt wird.

Die SHA-2 Funktion wurde mit dem Hintergedanken entwickelt, dass es notwendiger wurde größere Hash – Werte zu entwickeln, da der technologische Fortschritt mehr Kapazität und Rechenleistung lieferte. Dadurch wurde es wirtschaftlicher für Cracker bzw. staatliche Institutionen mögliche Hash – Werte zu knacken. Die SHA – 2 Funktion hat entweder eine 512 Bit oder 1024 Bit interne Blockgröße. Mit der Entwicklung dieser Funktion wurde eine ganze Reihe an SHA – Algorithmen entwickelt, die wie folgt lauten: SHA–224, SHA–256, SHA–384, SHA–512. Dabei stellt der Zusatz die Hash – Wertgröße dar.[15] [16]

e. Length - Extension Attack

Um die Schwachstelle aller MD – Konstruktionen verstehen zu können, muss man sich die Kompressionsfunktion der Hash – Funktion genauer anschauen. Am Beispiel der SHA – 1 Funktion lässt sich dies in groben Zügen erklären:

Der Initialisierungswert besteht aus fünf 32 Bit Worten, die durch den Algorithmus mit den eingehenden 512 Bit Nachrichtenblöcken verrechnet werden. Dieser Prozess wird auf

[15] Mohammed Meziani: Diplomarbeit, 13.08.2007, „Konstruktion von Hashfunktion“ , S.21 -25
[16] Bruce Schneier: Angewandte Kryptographie, 1. Auflage, S. 498f.

festgelegte Weise wiederholt, sodass es mehrere „Runden“ gibt, die vom Algorithmus vollzogen werden. Der Output ist der Hash – Wert mit 160 Bits.

In dem Vorgang der internen „Runden“ kann der Length-Extension-Attack durchgeführt werden. Man kann nun die Nachricht so erweitern, indem man die momentane Länge der Datei schätzt und als Padding anhängt, sodass man wieder eine weitere „Runde“ erhält, die der Algorithmus dreht ohne dass es auffällt, dass es manipuliert wurde (vgl. Abbildung 17).

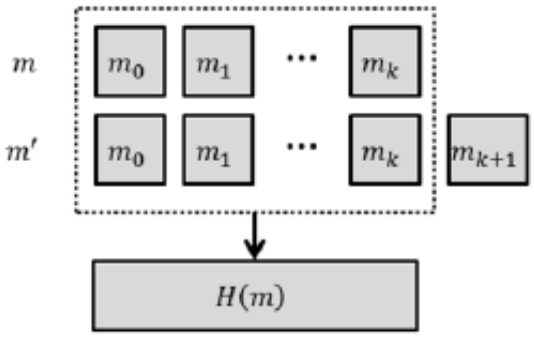


Abbildung 17 Length Extension Attack

Hierbei sind m_k die Nachrichtenblöcke, m und m‘ die beiden Nachrichten und H(m) stellt den Hash – Wert dar. Diese Eigenschaft ist die größte Schwachsteller aller Merkle – Damgård Konstruktionen, sodass man in jüngerer Zeit eine andere Konstruktionsmethode aufgebaut werden musste.[17]

f. Wide Pipe Konstruktion

Diese Konstruktionsvariante basiert auf der Merkle Damgård Konstruktion, aber hat eine Besonderheit. Der interne Zustand der Blöcke, also die Blockgröße, ist bei diesem Verfahren größer. Des Weiteren wird im letzten Schritt der Hash – Wert ausgegeben und zusätzlich durch eine Anwendung einer zweiten Kompressionsfunktion in den neuen Ausgabewert transformiert. Auf diese Weise wird eine Kompressionsfunktion doppelt angewendet. Die Familie der SHA – 2 Varianten sind durch die Wide Pipe Konstruktion entstanden. In Abbildung 18 wird die Funktionsweise deutlich, in dem durch C’ und C‘‘ zwei Kompressionsfunktionen Anwendung finden. H[0] stellt den Initialisierungswert dar und H[L-1] die jeweiligen Hash – Werte der Kompressionsfunktion. [18]

[17] B. Denton, R. Adhami: Dept. of Electrical & Computer Engineering, The University of Alabama in Huntsville, Huntsville, AL, USA Modern Hash Function Construction

[18] Stefan Lucks: Design Principles for Iterated Hash Functions, Sept. 2004, S.8ff.

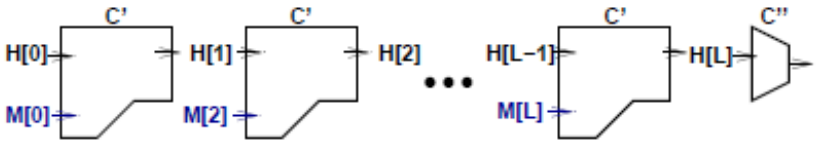


Abbildung 18 Wide Pipe Konstruktion

g. Double Pipe Konstruktion

Diese Variante stellt die Fortsetzung der Wide Pipe Konstruktion dar. Hierbei werden zwei Kompressionsfunktionen parallel auf die eingehenden Nachrichtenblöcke angewendet. Ein Nachrichtenblock wird mit dem Hash-Wert der anderen Kompressionsfunktion verbunden und als Eingabe für die andere Kompressionsfunktion verwendet (vgl. Abbildung 19).[19]

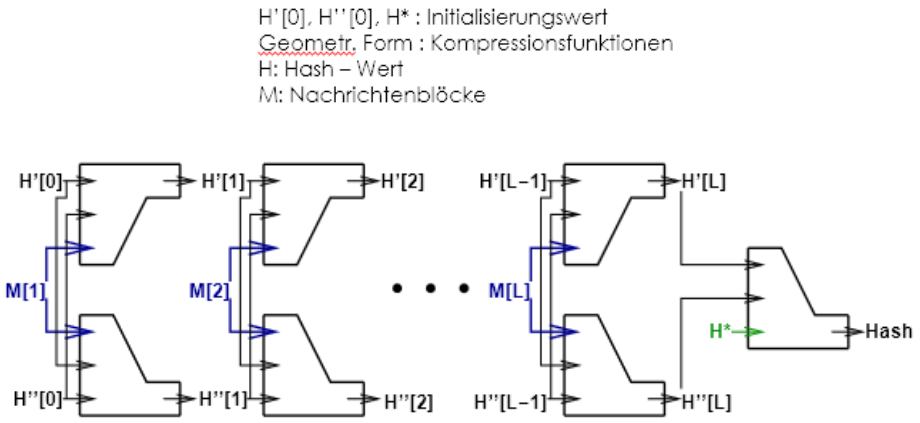


Abbildung 19 Double Pipe Konstruktion

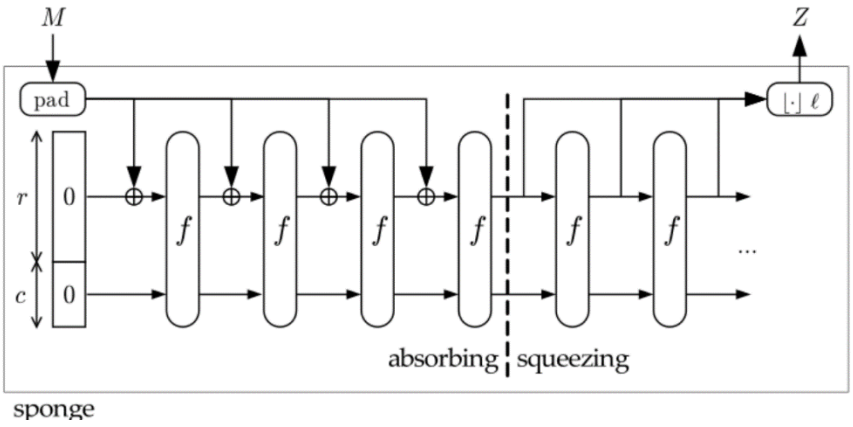
h. Sponge Konstruktion

Da die Konstruktionsweise auf Basis der Merkle – Damgård Konstruktion immer auffälligere Schwachstellen zeigte (Length - Extension Attack), musste eine völlig neue Art und Weise zur Erzeugung von Hash – Werten erfunden werden. Im Jahr 2004 gab es mehrere erfolgreiche Angriff gegen damals weit verbreitete Hash –Funktionen wie MD – 5 und SHA -1. Dadurch reduzierte sich der Rechenaufwand für bestimmte Kollisionsangriffe wesentlich. Auch wenn es zu diesem Zeitpunkt keine nachweislich erfolgreichen Angriffe auf die Hash – Funktionen der SHA -2 Familie gab, war man sich bewusst, dass man keine standardisierte und als sicher eingestufte zertifizierte Nachfolger hatte. Aus diesem Grund hat das National Institute of Standard and Technology (NIST) zu einem internationalen Wettbewerb aufgerufen, in dem jeder seine selbst entwickelten kryptographischen Hash – Funktionen vorstellen und von der

[19] Stefan Lucks: Design Principles for Iterated Hash Functions, Sept. 2004, S.11f.

Öffentlichkeit bewerten lassen konnte. Während dieses Wettbewerbs entstanden viele neue Hash – Funktionen mit vollkommen neuen Konstruktionsstrukturen, die jedoch als gebrochen wieder verworfen wurden. Im Dezember 2010 waren nur noch fünf Hash – Funktionen übrig von denen sich die Sponge – Konstruktion, auch „Keccak“ genannt, als Sieger herausstellte. Im Jahr 2015 wurde Keccak als SHA-3 Standard eingeführt.

Die Funktionsweise der Sponge Konstruktion unterscheidet sich wesentlich im Vergleich zu den iterativen Konstruktionsmethoden. Grundsätzlich ist die Größe der Nachrichtenblöcke für die Kompressionsfunktion variabel bestimmbar. Im Vergleich, die iterative Methode setzte eine vorher festgelegte Blockgröße voraus ohne die die Kompressionsfunktion nicht funktionieren würde. Des Weiteren ist der Output, also die Hash – Wert Größe, ebenfalls vom Anwender bestimmbar.



M: Nachricht/Eingabestring
r: Blockgröße
c: Kapazität
f: Permutationsfunktion / Transformationsfunktion
Z: Ausgabewert / Hash – Wert

Abbildung 20 Sponge Konstruktion

Die Sponge Konstruktion verfolgt zwei wesentliche Prozessschritte. Zuallererst wird der Eingabestring M durch ein bestimmtes Paddingschema auf eine entsprechende Blocklänge gebracht. Wie der Name „Sponge“ bereits andeutet, gibt es im nächsten Schritt eine „Absorbing“ Phase des Schwammes, in dem der erste Block mit dem Initialisierungswert über den XOR-Operator verrechnet und durch eine Permutationsfunktion verschachtelt wird. Das erste verschachtelte Ergebnis wird mit dem nächsten Nachrichtenblock verrechnet, sodass ein

fortlaufender Prozess entsteht. Sind alle Nachrichtenblöcke verarbeitet, wechselt die Sponge Konstruktion in den zweiten Prozessschritt über. In der „Squeezing“ Phase wird die vom Anwender festgelegte Anzahl an Output Blöcken ausgegeben und mit einer weiteren Funktion verschachtelt. Das Ergebnis ist der Hash – Wert.

Wichtig hierbei ist, dass, wie in Abbildung 20 dargestellt, der interne Prozess der Sponge Funktion in die Bereiche r und c aufgeteilt ist. Dabei stellt c die Kapazität dar, die über den gesamten Verschachtelungsprozess nicht berührt wird und das Sicherheitsniveau der Sponge Konstruktion darstellt.[20]

[20] Guido Bertoni, Joan Daemen, Michael Peeters, Gilles Van Assche: Cryptographic Sponge Function, Jan. 2011, S.12f.