

Kryptowährung – Ein Unterrichtsentwurf zum Verständnis der damit verbundenen Mathematik

Katharina Spehr

Kryptographische Hashfunktionen

- Bitcoin nutzt eine kryptographische Hashfunktion.
- eine Zeichenfolge von beliebiger Länge wird in eine Zeichenfolge fixer Länge umgewandelt; den Hashwert
- dieselben Eingangsdaten ergeben immer denselben Hashwert
- jede Veränderung der Eingangsdaten führt zu einem stark veränderten Hashwert

drei Eigenschaften.

Ausgehend vom Hashwert kann der ursprüngliche Dateninput nicht mit vertretbarem Aufwand bestimmt werden

Es ist nicht mit vertretbarem Aufwand möglich, einen zweiten Dateninput zu finden, der denselben Hashwert ergibt.

Es ist nicht mit vertretbarem Aufwand möglich, zwei verschiedene Dateninputs zu finden, die denselben Hashwert ergeben

SHA-0

National Institute of Standards and Technology (NIST) und National Security Agency (NSA) entwickelten eine Hash-Funktion als Bestandteil des Digital Signature Algorithms (DSA) für den Digital Signature Standard (DSS).

Die Funktion wurde 1993 veröffentlicht

Hashwert 160 bit

SHA-1

Der ursprüngliche SHA-0 wurde wegen eines „Konstruktionsfehlers“ schon 1995 korrigiert und spielte deswegen in der Praxis kaum eine Rolle.

Die Korrektur besteht nur in einem kleinen, aber essentiellen Detail: der Rotation

SHA-2

größere Hash-Werte mit 256 bzw. 512 Bit

SHA-224, SHA-256, SHA-384 und SHA-512

gleiches Konstruktionsprinzip aufgebaut wie SHA-1

SHA-3

zukunftsicherer als SHA-2

SHA-3 ist grundlegend anders als SHA-2 aufgebaut, nämlich als sogenannte *Sponge-Konstruktion*

SHA-3 ist die neueste, effizienteste und sicherste Hashfunktion der SHA-Reihe

SHA-3 soll SHA-2 nicht ersetzen, sondern ist eine Alternative.

Sollte SHA-2 irgendwann einmal gebrochen werden, kann man zu SHA-3 übergehen

Wofür steht SHA-256?

SHA ist die Abkürzung für *sicherer Hash-Algorithmus*,

Länge des Hashes ist genau 256 Bit

Hash = Ausgabewert, der scheinbar ohne bestimmte Regeln aus einem Eingabewert erzeugt wurde

asymmetrischen Verschlüsselung = Informationen können dabei immer nur in eine Richtung abgeleitet werden

So ist es etwa möglich, den öffentlichen Schlüssel (Public Key) vom privaten Schlüssel (Private Key) abzuleiten. Das gleiche Verfahren gelingt aber niemals umgekehrt. Man spricht daher von einer Einwegfunktion.

Diese Funktionsweise ist nicht auf SHA-256 begrenzt. Weitere Algorithmen, die anderen Blockchains dienen, fungieren nach dem gleichen Prinzip.

Der Sha-256-Algorithmus basiert auf der Merkle-Damgard-Konstruktionsmethode

SHA-256 wird in verschiedenen Teilen des Bitcoin-Netzwerks verwendet:

- Beim Mining als Proof-of-Work-Algorithmus
- bei der Erstellung von Bitcoin-Adressen, um die Sicherheit und den Datenschutz zu verbessern.

Kurze Ablaufbeschreibung

Auffüllen: Die Eingabedaten werden zunächst aufgefüllt, damit ihre Länge einem bestimmten Wert modulo 512 entspricht. Dies geschieht durch Hinzufügen eines 1-Bits, gefolgt von genügend 0-Bits, um die gewünschte Länge zu erreichen. Schließlich wird die ursprüngliche Länge der Daten als 64-Bit-Wert am Ende hinzugefügt.

Aufteilung in Blöcke: Die aufgefüllten Daten werden dann in Blöcke von 512 Bit aufgeteilt.

Festlegen der anfänglichen Hash-Werte: SHA-256 beginnt mit acht anfänglichen Hash-Werten, bei denen es sich um eine Reihe von 32-Bit-Werten handelt, die durch die ersten 32 Nachkommastellen der Quadratwurzeln der ersten acht Primzahlen bestimmt werden.

Die Hauptschleife: Jeder 512-Bit-Datenblock durchläuft 64 Hash-Runden, die eine Reihe von bitweisen Operationen und logischen Funktionen beinhalten. Zu diesen Operationen gehören AND, OR, XOR, Rechtsverschiebung und mehr.

Ausgabe: Nachdem alle Blöcke die Hauptschleife durchlaufen haben, werden die endgültigen Hash-Werte zu einem einzigen 256-Bit-Hash (32 Byte) zusammengefügt.

Link zum Originaltext

[Federal Information](#)

Kollisionsresistenz bei Hash-Funktionen

Schwache Kollisionsresistenz: es ist praktisch unmöglich, zu einer gegebenen Nachricht x eine weitere, davon verschiedene Nachricht x' zu finden, die zum selben Hash-Wert y führt

Starke Kollisionsresistenz: es ist praktisch unmöglich, zwei beliebige verschiedene Nachrichten x und x' zu finden, die denselben Hash-Wert y haben.

Beachte: Wenn man eine Hash-Funktion wie SHA-256 benutzt, um beliebigen Texten einen Hash-Wert zuzuordnen, so muss es Kollisionen (d.h. Texte mit gleichem Hash-Wert) geben, da es unendlich viele Texte, aber nur eine endliche Menge von Hash-Werten gibt.

Kollisionsresistenz bedeutet: Obwohl es (sogar unendlich) viele Kollisionen gibt, ist es uns praktisch unmöglich, solche Kollisionen zu erzeugen.

Angriffsszenarien

Urbildangriff (engl. preimage attack)

der Angreifer hat das Ziel, zu einem gegebenen Hashwert einer unbekannten Nachricht (Erster Urbildangriff) oder zu einer gegebenen Nachricht selbst (Zweiter Urbildangriff) eine weitere Nachricht zu konstruieren, die denselben Hashwert besitzt.

Kollisionsangriff (engl. collision attack)

der Angreifer hat das Ziel, zwei verschiedene Dokumente zu konstruieren, die beide denselben Hashwert besitzen.

Beachte, dass es sich hier um unterschiedliche Angriffsszenarien handelt. Das zeigt sich auch in der Praxis. Während Kollisionsangriffe bei SHA-1 möglich sind, sind Urbildangriffe bei SHA-2 derzeit noch nicht möglich.

Genaues Vorgehen

[SHA-256 | COMPLETE Step-By-Step Explanation \(W/ Example\)](#)

Hello world!

01001000 01100101 01101100 01101100

01101111 00100000 01110111 01101111

01110010 01101100 01100100 00100001

Sha 256 Hash

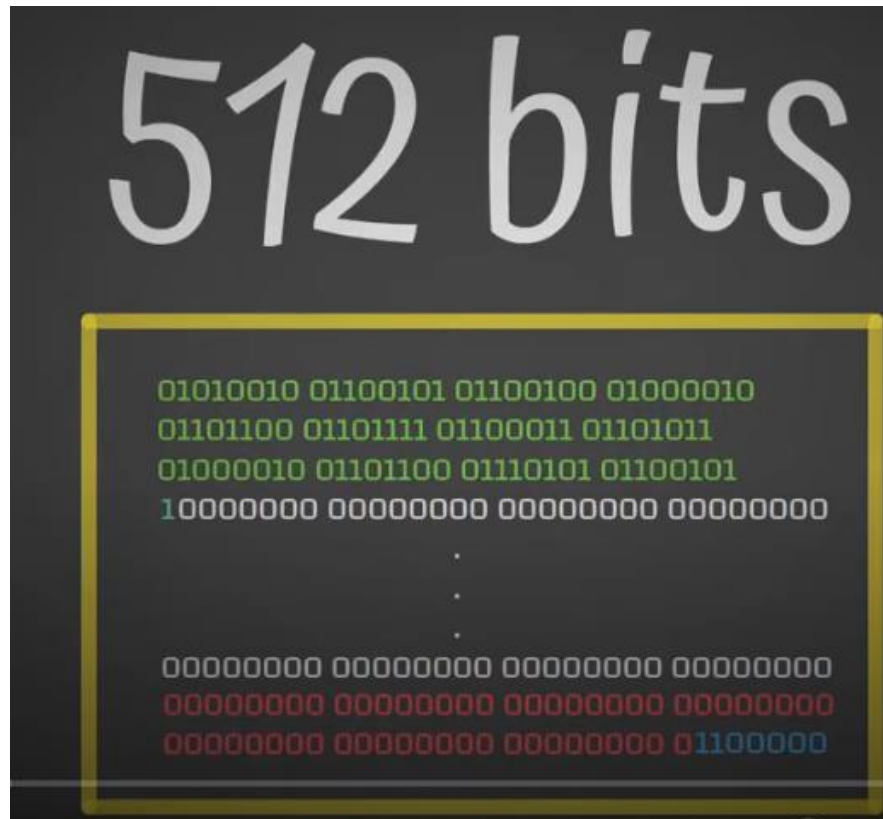
0ba904eae8773b70c75333db4de2f3ac45a8ad4ddba1b242f0b3cfc199391dd8

Hello world (ohne Ausrufezeichen)

1894a19c85ba153acbf743ac4e43fc004c891604b26f8c69e1e83ea2afc7c48f

Auffüllen

Die Eingabedaten werden zunächst aufgefüllt, damit ihre Länge einem bestimmten Wert modulo 512 entspricht. Dies geschieht durch Hinzufügen eines 1-Bits, gefolgt von genügend 0-Bits, um die gewünschte Länge zu erreichen. Schließlich wird die ursprüngliche Länge der Daten als 64-Bit-Wert am Ende hinzugefügt.



Text hier: Red Block Blue -> 96 Bits -> **1100000**

Aufteilung in Blöcke

Die aufgefüllten Daten werden dann in Blöcke von 512 Bit aufgeteilt.

N 512 Bit Blöcke im Beispiel ist $N = 1$

16 32 Bit Wörter

M_0 bis M_{15}

```
M0 = 01010010 01100101 01100100 01000010
M1 = 01101100 01101111 01100011 01101011
M2 = 01000010 01101100 01110101 01100101
M3 = 10000000 00000000 00000000 00000000
.
.
.
M13 = 00000000 00000000 00000000 00000000
M14 = 00000000 00000000 00000000 00000000
M15 = 00000000 00000000 00000000 01100000
```

Festlegen der anfänglichen Hash-Werte

SHA-256 beginnt mit acht anfänglichen Hash-Werten, bei denen es sich um eine Reihe von 32-Bit-Werten handelt, die durch die ersten 32 Bits der Nachkommastellen der Quadratwurzeln der ersten acht Primzahlen bestimmt werden.

$H_0^{(0)}$	=	6a09e667
$H_1^{(0)}$	=	bb67ae85
$H_2^{(0)}$	=	3c6ef372
$H_3^{(0)}$	=	a54ff53a
$H_4^{(0)}$	=	510e527f
$H_5^{(0)}$	=	9b05688c
$H_6^{(0)}$	=	1f83d9ab
$H_7^{(0)}$	=	5be0cd19

Wurzel 2, 32 Nachkommastellen in binäre Zahl umwandeln

$$\sqrt{2} = 1,41421\ 35623\ 73095\ 04880\ 16887\ 24209\ 69807\ 85696\ 71875\ 37694$$

~~1~~,0110 1010 0000 1001 1110 0110 0110 0111



Weitere Konstanten K

4.2.2 SHA-224 and SHA-256 Constants

SHA-224 and SHA-256 use the same sequence of sixty-four constant 32-bit words,

$K_0^{(256)}, K_1^{(256)}, \dots, K_{63}^{(256)}$. These words represent the first thirty-two bits of the fractional parts of the cube roots of the first sixty-four prime numbers. In hex, these constant words are (from left to right)

```
428a2f98 71374491 b5c0fbcf e9b5dba5 3956c25b 59f111f1 923f82a4 ab1c5ed5
d807aa98 12835b01 243185be 550c7dc3 72be5d74 80deb1fe 9bdc06a7 c19bf174
e49b69c1 efbe4786 0fc19dc6 240ca1cc 2de92c6f 4a7484aa 5cb0a9dc 76f988da
983e5152 a831c66d b00327c8 bf597fc7 c6e00bf3 d5a79147 06ca6351 14292967
27b70a85 2e1b2138 4d2c6dfc 53380d13 650a7354 766a0abb 81c2c92e 92722c85
a2bfe8a1 a81a664b c24b8b70 c76c51a3 d192e819 d6990624 f40e3585 106aa070
19a4c116 1e376c08 2748774c 34b0bcb5 391c0cb3 4ed8aa4a 5b9cca4f 682e6ff3
748f82ee 78a5636f 84c87814 8cc70208 90befffa a4506ceb bef9a3f7 c67178f2
```

K = erste 32 Bits der Nachkommastellen der dritten Wurzeln der ersten 64 Primzahlen in Hex

Verwendete Funktionen

4.1.2 SHA-224 and SHA-256 Functions

SHA-224 and SHA-256 both use six logical functions, where *each function operates on 32-bit words*, which are represented as x , y , and z . The result of each function is a new 32-bit word.

$$Ch(x, y, z) = (x \wedge y) \oplus (\neg x \wedge z) \quad (4.2)$$

$$Maj(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) \quad (4.3)$$

$$\sum_0^{\{256\}}(x) = ROTR^2(x) \oplus ROTR^{13}(x) \oplus ROTR^{22}(x) \quad (4.4)$$

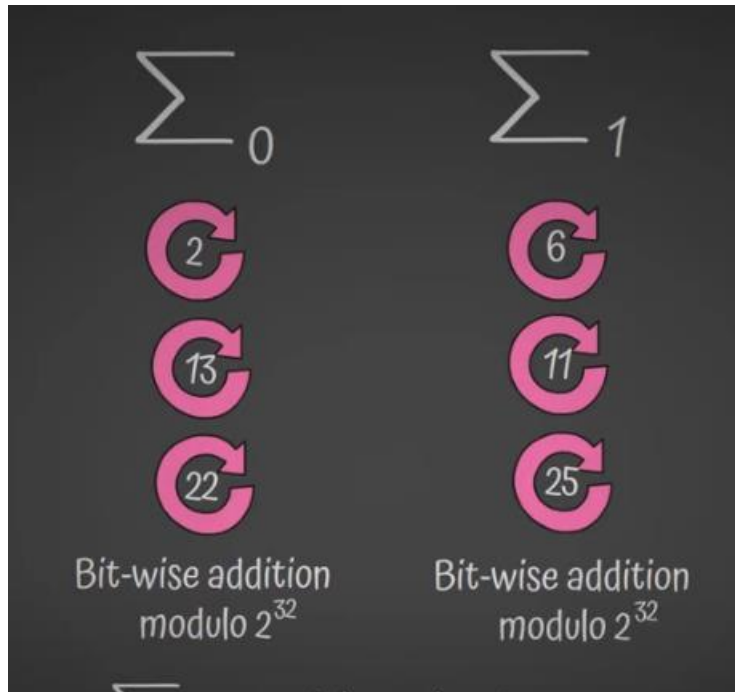
$$\sum_1^{\{256\}}(x) = ROTR^6(x) \oplus ROTR^{11}(x) \oplus ROTR^{25}(x) \quad (4.5)$$

$$\sigma_0^{\{256\}}(x) = ROTR^7(x) \oplus ROTR^{18}(x) \oplus SHR^3(x) \quad (4.6)$$

$$\sigma_1^{\{256\}}(x) = ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x) \quad (4.7)$$

Rotation

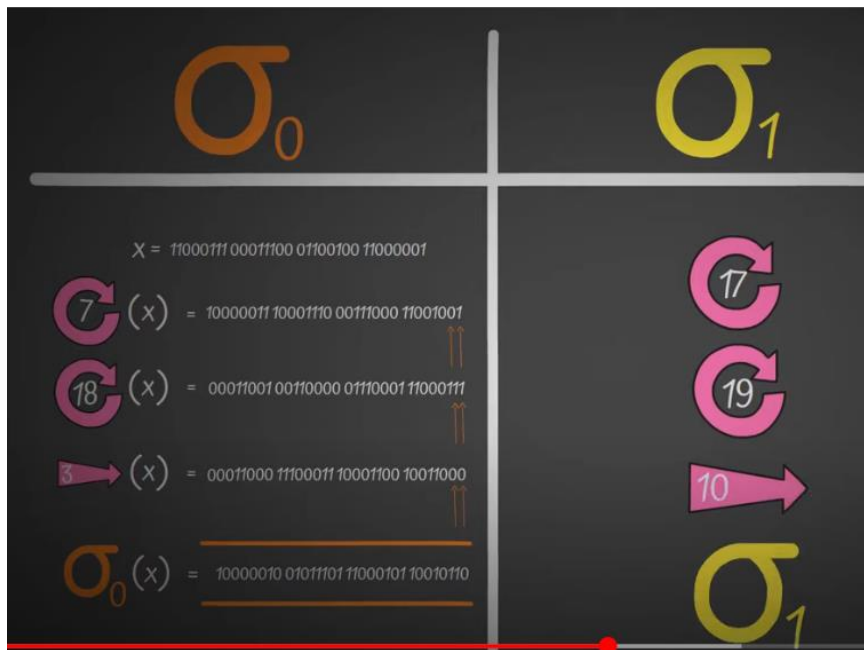
$$\Sigma_0^{\{256\}}(x) = \mathbf{ROTR}^2(x) \oplus \mathbf{ROTR}^{13}(x) \oplus \mathbf{ROTR}^{22}(x)$$



$\mathbf{ROTR}^2(x)$ Jedes Bit wird um 2 Stellen nach rechts versetzt, die Bits am Ende kommen nach vorne

Shift Right

$$\sigma_0^{\{256\}}(x) = ROTR^7(x) \oplus ROTR^{18}(x) \oplus SHR^3(x)$$



Unser Beispiel: Hell...

01001000 01100101 01101100 01101100

1101100 01001000 01100101 01101100 0

01 01101100 01101100 01001000 011001

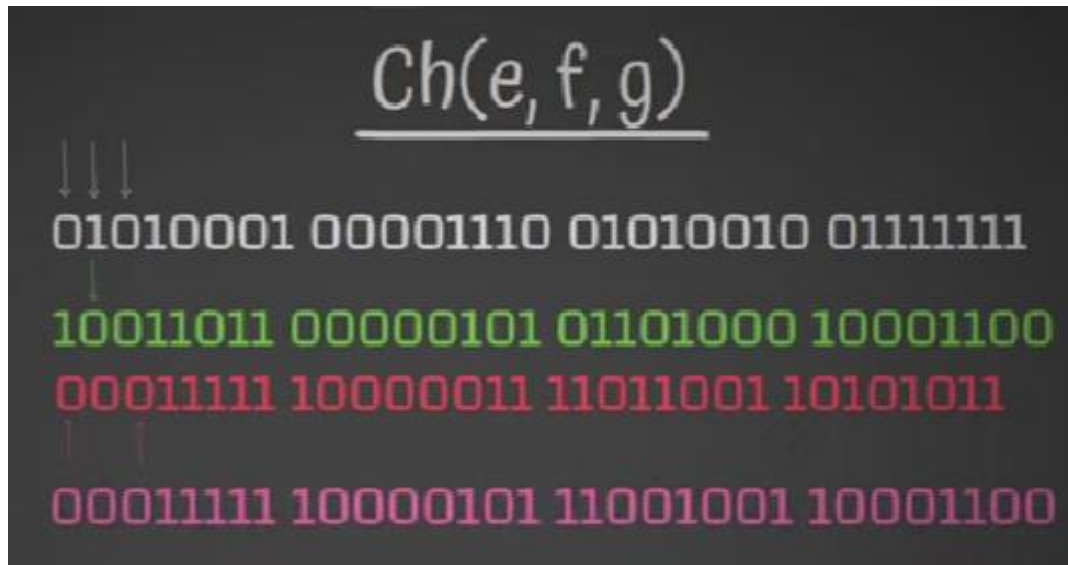
000 01001000 01100101 01101100 01101

$SHR^3(x)$ right Shift of 3
 alle Bits werden um 3 Stellen nach rechts verschoben,
 vorne werden 3 Nullen eingefügt

Choose

$$\text{Ch}(x,y,z) = (x \wedge y) \oplus (\neg x \wedge z)$$

^ ist 1, wenn beide 1 sind sonst 0



Majority

$$\text{Maj}(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$$

$\text{Maj}(a, b, c)$

01101010 00001001 11100110 01100111
10111011 01100111 10101110 10000101
00111100 01101110 11110011 01110010

↑ ↑ ↑
001

$T_1 = 01000101 11011101 01010001 10101010$

$T_2 = \sum_0(a) + \text{Maj}(a, b, c)$

Maj = Majority, man schaut welchen Wert die Mehrheit der Bits hat und notiert diesen Wert

Computation

Die Hauptschleife: Jeder 512-Bit-Datenblock durchläuft 64 Hash-Runden,

die eine Reihe von bitweisen Operationen und logischen Funktionen beinhalten.

Zu diesen Operationen gehören AND, OR, XOR, Rechtsverschiebung und mehr.

For $i=1$ to N :

{

1. Prepare the message schedule, $\{W_t\}$:

$$W_t = \begin{cases} M_t^{(i)} & 0 \leq t \leq 15 \\ \sigma_1^{(256)}(W_{t-2}) + W_{t-7} + \sigma_0^{(256)}(W_{t-15}) + W_{t-16} & 16 \leq t \leq 63 \end{cases}$$

$W_0, W_1, \dots, W_{62}, W_{63}$

$W_0 = M_0 = 01010010 \ 01100101 \ 01100100 \ 01000010$
 $W_1 = M_1 = 01101100 \ 01101111 \ 01100011 \ 01101011$
 $W_2 = M_2 = 01000010 \ 01101100 \ 01110101 \ 01100101$
 $W_3 = M_3 = 10000000 \ 00000000 \ 00000000 \ 00000000$
 $\vdots$
 $W_{13} = M_{13} = 00000000 \ 00000000 \ 00000000 \ 00000000$
 $W_{14} = M_{14} = 00000000 \ 00000000 \ 00000000 \ 00000000$
 $W_{15} = M_{15} = 00000000 \ 00000000 \ 00000000 \ 01100000$

$$W_t = \sigma_1(W_{t-2}) + W_{t-7} + \sigma_0(W_{t-15}) + W_t$$

$t = 16$

$$W_{16} = \sigma_1(W_{14}) + W_9 + \sigma_0(W_1) + W_0$$

$$W_{16} = \underbrace{\sigma_1(\text{00000000 00000000 00000000 00000000})}_{\text{00000000 00000000 00000000 00000000}} + \underbrace{\text{00000000 00000000 00000000 00000000}}_{\text{00000000 10001111 11101001 00000101}} + \underbrace{\sigma_0(\text{0101100 0110111 0100011 0110101})}_{\text{00000000 10001111 11101001 00000101}} + \underbrace{\text{01010010 01100101 01100100 01000010}}_{\text{00000000 10001111 11101001 00000101}}$$

2. Initialize the eight working variables, a , b , c , d , e , f , g , and h , with the $(i-1)^{\text{st}}$ hash value:

$a = H_0 = 6a09e667 = 01101010 \ 00001001 \ 11100110 \ 01100111$
 $b = H_1 = bb67ae85 = 10111011 \ 01100111 \ 10101110 \ 10000101$
 $c = H_2 = 3c6ef372 = 00111100 \ 01101110 \ 11110011 \ 01110010$
 $d = H_3 = a54ff53a = 10100101 \ 01001111 \ 11110101 \ 00111010$
 $e = H_4 = 510e527f = 01010001 \ 00001110 \ 01010010 \ 01111111$
 $f = H_5 = 9b05688c = 10011011 \ 00000101 \ 01101000 \ 10001100$
 $g = H_6 = 1f83d9ab = 00011111 \ 10000011 \ 11011001 \ 10101011$
 $h = H_7 = 5be0cd19 = 01011011 \ 11100000 \ 11001101 \ 00011001$

3. For $t=0$ to 63:

{

$$T_1 = h + \sum_1^{(256)} (e) + Ch(e, f, g) + K_t^{(256)} + W_t$$

$$T_2 = \sum_0^{(256)} (a) + Maj(a, b, c)$$

$$h = g$$

$$g = f$$

$$f = e$$

$$e = d + T_1$$

$$d = c$$

$$c = b$$

$$b = a$$

$$a = T_1 + T_2$$

}

$$T_1 = h + \sum_1 (e) + Ch(e, f, g) + K_0 + w_0$$

01011011 11100000 11001101 00011001

00110101 10000111 00100111 00101011

00011111 10000101 11001001 10001100

01000010 10001010 00101111 10011000

01010010 01100101 01100100 01000010

...0100

4. Compute the i^{th} intermediate hash value $H^{(i)}$:

$$H_0^{(i)} = a + H_0^{(i-1)}$$

$$H_1^{(i)} = b + H_1^{(i-1)}$$

$$H_2^{(i)} = c + H_2^{(i-1)}$$

$$H_3^{(i)} = d + H_3^{(i-1)}$$

$$H_4^{(i)} = e + H_4^{(i-1)}$$

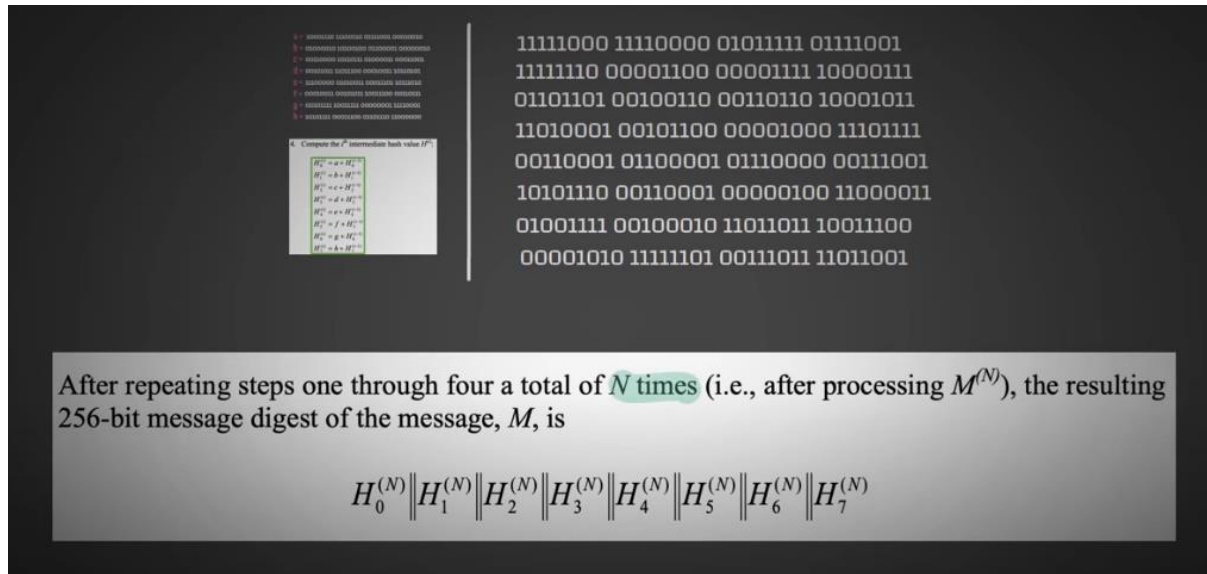
$$H_5^{(i)} = f + H_5^{(i-1)}$$

$$H_6^{(i)} = g + H_6^{(i-1)}$$

$$H_7^{(i)} = h + H_7^{(i-1)}$$

}

Ausgabe



1. Initialize variables: message, counter, hash, and digest.
2. Initialize variables: message, counter, hash, and digest.
3. Initialize variables: message, counter, hash, and digest.
4. Initialize variables: message, counter, hash, and digest.
5. Initialize variables: message, counter, hash, and digest.
6. Initialize variables: message, counter, hash, and digest.
7. Initialize variables: message, counter, hash, and digest.
8. Initialize variables: message, counter, hash, and digest.
9. Initialize variables: message, counter, hash, and digest.
10. Initialize variables: message, counter, hash, and digest.

4. Compute the H^i intermediate hash value H^i .

$H_0^{(N)} = a + H_0^{(N-1)}$
 $H_1^{(N)} = b + H_1^{(N-1)}$
 $H_2^{(N)} = c + H_2^{(N-1)}$
 $H_3^{(N)} = d + H_3^{(N-1)}$
 $H_4^{(N)} = e + H_4^{(N-1)}$
 $H_5^{(N)} = f + H_5^{(N-1)}$
 $H_6^{(N)} = g + H_6^{(N-1)}$
 $H_7^{(N)} = h + H_7^{(N-1)}$

11111000 11110000 01011111 01111001
11111110 00001100 00001111 10000111
01101101 00100110 00110110 10001011
11010001 00101100 00001000 11101111
00110001 01100001 01110000 00111001
10101110 00110001 00000100 11000011
01001111 00100010 11011011 10011100
00001010 11111101 00111011 11011001

After repeating steps one through four a total of N times (i.e., after processing $M^{(N)}$), the resulting 256-bit message digest of the message, M , is

$$H_0^{(N)} \| H_1^{(N)} \| H_2^{(N)} \| H_3^{(N)} \| H_4^{(N)} \| H_5^{(N)} \| H_6^{(N)} \| H_7^{(N)}$$

Nachdem alle Blöcke die Hauptschleife durchlaufen haben, werden die endgültigen Hash-Werte zu einem einzigen 256-Bit-Hash (32 Byte) zusammengefügt.