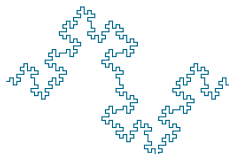


# Secure Hash Algorithm

## SHA-256

Chi Trung Nguyen  
*T-Systems*



19. Juni 2012

# INHALT

## EINFÜHRUNG

Was ist ein Hash?

## GESCHICHTE

SHA

SHA-0

SHA-1

SHA-2

Übersicht

## IMPLEMENTIERUNG

Algorithmus

Pseudocode

Pseudocode2

Pseudocode3

Pseudocode4

Pseudocode5

## ANWENDUNG

Vergleichszweck

# WAS IST EIN HASH?

- ▶ deutsch: „zerhacken“, „verstreuen“

# WAS IST EIN HASH?

- ▶ deutsch: „zerhacken“, „verstreuen“
- ▶ Hashfunktion oder Streuwertfunktion erstellt aus beliebiger großer Quellmenge eine immer gleich große Zielmenge
  - ▶  $f(x) = f(x')$

# WAS IST EIN HASH?

- ▶ deutsch: „zerhacken“, „verstreuen“
- ▶ Hashfunktion oder Streuwertfunktion erstellt aus beliebiger großer Quellmenge eine immer gleich große Zielmenge
  - ▶  $f(x) = f(x')$
- ▶ Einwegfunktion

# SHA ALLGEMEIN

- ▶ 1993 vom **National Institute of Standards(NIST)** als ein **U.S. Federal Information Processing Standard (FIPS)** veröffentlicht

# SHA ALLGEMEIN

- ▶ 1993 vom **National Institute of Standards(NIST)** als ein **U.S. Federal Information Processing Standard (FIPS)** veröffentlicht
- ▶ Gruppe von kryptologischer Hashfunktionen
  - ▶ SHA-0
  - ▶ SHA-1
  - ▶ SHA-2
  - ▶ SHA-3

# SHA-0

- ▶ 1993 veröffentlicht



# SHA-0

- ▶ 1993 veröffentlicht
- ▶ Bestandteil des Digital Signature Algorithms (DSA) für Digital Signature Standard (DSS)

# SHA-1

- 1995 veröffentlicht

# SHA-1

- ▶ 1995 veröffentlicht
- ▶ aufgrund Designfehler in SHA-0

# SHA-2

- ▶ 2002 veröffentlicht

# SHA-2

- ▶ 2002 veröffentlicht
- ▶ existiert in mehreren Bit Variante

Tabelle : Secure Hash Algorithmus Eigenschaften

Algorithmus	Message Größe(bits)	Block Größe(bits)	Word Größe(bits)	Message Digest Größe(bits)
SHA-1	$< 2^{64}$	512	32	160
SHA-224	$< 2^{64}$	512	32	224
SHA-256	$< 2^{64}$	512	32	256
SHA-384	$< 2^{128}$	1024	64	384
SHA-512	$< 2^{128}$	1024	64	512

# FUNKTIONEN

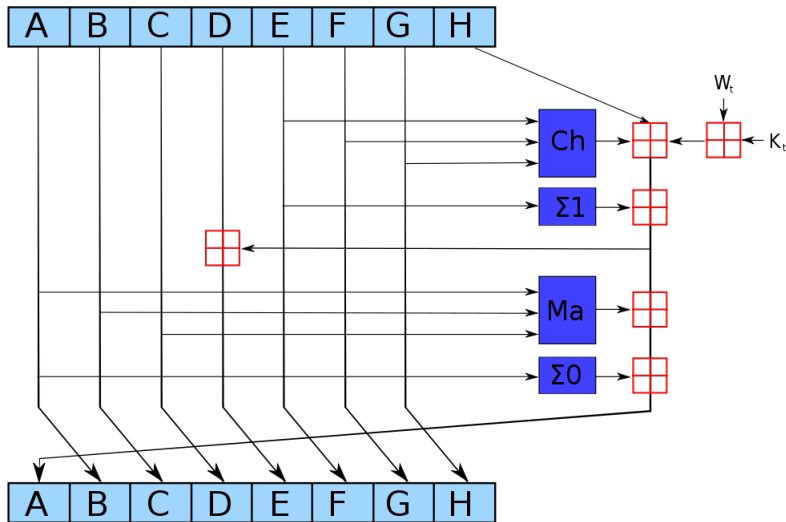
$$Ch(E, F, G) = (E \wedge F) \oplus (\neg E \wedge G)$$

$$Maj(A, B, C) = (A \wedge B) \oplus (A \wedge C) \oplus (B \wedge C)$$

$$\Sigma_0 = (A \ggg 2) \oplus (A \ggg 13) \oplus (A \ggg 22)$$

$$\Sigma_1 = (A \ggg 6) \oplus (A \ggg 11) \oplus (A \ggg 25)$$

# DARSTELLUNG DES ALGORITHMUS





# PSEUDOCODE

- Initialize variables (first 32 bits of the fractional parts of the square roots of the first 8 primes 2..19):

`h[0..7] := 0x6a09e667, [...], 0x5be0cd19`

# PSEUDOCODE

- ▶ Initialize variables (first 32 bits of the fractional parts of the square roots of the first 8 primes 2..19):

$h[0..7] := 0x6a09e667, [\dots], 0x5be0cd19$

- ▶ Initialize table of round constants (first 32 bits of the fractional parts of the cube roots of the first 64 primes 2..311):

$k[0..63] := 0x428a2f98, [\dots], 0xc67178f2$

# PREPROCESSING

- ▶ bit 1 zur *message* hinzufügen

# PREPROCESSING

- ▶ bit 1 zur *message* hinzufügen
- ▶ anzahl von  $k$  bits 0 hinzufügen, wobei  $k$  die kleinst mögliche Zahl  $\geq 0$ , so dass die Länge der *message* (in bits) Modulo 512 minus 64 bits ist

# PREPROCESSING

- ▶ bit 1 zur *message* hinzufügen
- ▶ anzahl von  $k$  bits 0 hinzufügen, wobei  $k$  die kleinst mögliche Zahl  $\geq 0$ , so dass die Länge der *message* (in bits) Modulo 512 minus 64 bits ist
- ▶ Länge der *message* (vor dem Preprocessing), in bits, als 64-bit big-endian integer hinzufügen

# PREPROCESSING

- ▶ bit 1 zur *message* hinzufügen
  - ▶ anzahl von  $k$  bits 0 hinzufügen, wobei  $k$  die kleinst mögliche Zahl  $\geq 0$ , so dass die Länge der *message* (in bits) Modulo 512 minus 64 bits ist
  - ▶ Länge der *message* (vor dem Preprocessing), in bits, als 64-bit big-endian integer hinzufügen
- 
- ▶ *message* in 512-bit chunks teilen
  - ▶ foreach chunk{  
  teile chunk in sechzehn 32-bit big-endian  
  Worte  $w[0..15]$

# ERWEITERUNG DER WORTE

```
for  $i = 16$  to  $63$  {  
     $s0 := (w[i - 15] \text{ rightrotate } 7) \text{ xor } (w[i - 15] \text{ rightrotate } 18)$   
     $\text{ xor } (w[i - 15] \text{ rightshift } 3)$   
  
     $s1 := (w[i - 2] \text{ rightrotate } 17) \text{ xor } (w[i - 2] \text{ rightrotate } 19) \text{ xor}$   
     $(w[i - 2] \text{ rightshift } 10)$   
  
     $w[i] := w[i - 16] + s0 + w[i - 7] + s1$   
}
```

# HASHZUWEISUNG

$a := h0$

$b := h1$

$c := h2$

$d := h3$

$e := h4$

$f := h5$

$g := h6$

$h := h7$



# HAUPTSCHLEIFE

```

for  $i = 0$  to 63 {
     $S0 := (a \text{ rightrotate } 2) \text{ xor } (a \text{ rightrotate } 13) \text{ xor } (a \text{ rightrotate } 22)$ 
     $maj := (a \text{ and } b) \text{ xor } (a \text{ and } c) \text{ xor } (b \text{ and } c)$ 
     $t2 := S0 + maj$ 
     $S1 := (e \text{ rightrotate } 6) \text{ xor } (e \text{ rightrotate } 11) \text{ xor } (e \text{ rightrotate } 25)$ 
     $ch := (e \text{ and } f) \text{ xor } ((\text{not } e) \text{ and } g)$ 
     $t1 := h + S1 + ch + k[i] + w[i]$ 
     $h := g \quad g := f \quad f := e \quad e := d + t1 \quad d := c \quad c := b \quad b := a \quad a := t1 + t2$ 
}

```

# VERWENDUNGSZWECK

- ▶ Digitale Zertifikate und Signaturen

# VERWENDUNGSZWECK

- ▶ Digitale Zertifikate und Signaturen
- ▶ Passwortverschlüsselung
  - ▶ pam\_unix: sha2, md5
  - ▶ httpasswd(Apache): sha1, md5
  - ▶ MySQL: sha1

# VERWENDUNGSZWECK

- ▶ Digitale Zertifikate und Signaturen
- ▶ Passwortverschlüsselung
  - ▶ pam\_unix: sha2, md5
  - ▶ httpasswd(Apache): sha1, md5
  - ▶ MySQL: sha1
- ▶ Prüfsummen bei Downloads

# SICHERHEITSLÜCKEN & ANGRIFFSVEKTOREN

# SHA-3