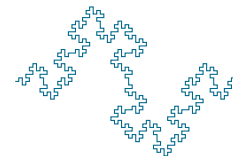


# Secure Hash Algorithm

## SHA-256

Chi Trung Nguyen  
*T-Systems*



20. Juni 2012

# INHALT

## EINFÜHRUNG

Was ist ein Hash?

## GESCHICHTE

SHA Allgemein

SHA-0

SHA-1

SHA-2

## IMPLEMENTIERUNG

Algorithmus

Pseudocode

## ANWENDUNG

Verwendungszweck

Schwachstellen

## AUSBLICK

SHA-3

# 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)

1. 2005 von Xiaoyun Wang, Yiqun Lisa Yin und Hongbo Yu an der Shandong University in China gebrochen
2. Ihnen war es gelungen, den Aufwand zur Kollisionsberechnung von  $2^{80}$  auf  $2^{69}$  zu verringern
3. es wurde ein rechtsshift durch ein linksshift ersetzt
4. August 2005, wurde von Xiaoyun Wang, Andrew Yao und Frances Yao auf der Konferenz CRYPTO 2005 ein weiterer, effizienterer Kollisionsangriff auf SHA-1 vorgestellt, welcher den Berechnungsaufwand auf  $2^{63}$  reduziert

# SHA-1

► 1995 veröffentlicht

1. 2005 von Xiaoyun Wang, Yiqun Lisa Yin und Hongbo Yu an der Shandong University in China gebrochen
2. Ihnen war es gelungen, den Aufwand zur Kollisionsberechnung von  $2^{80}$  auf  $2^{69}$  zu verringern
3. es wurde ein rechtsshift durch ein linksshift ersetzt
4. August 2005, wurde von Xiaoyun Wang, Andrew Yao und Frances Yao auf der Konferenz CRYPTO 2005 ein weiterer, effizienterer Kollisionsangriff auf SHA-1 vorgestellt, welcher den Berechnungsaufwand auf  $2^{63}$  reduziert

# SHA-1

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

# SHA-2

- 2002 veröffentlicht

# SHA-2

- ▶ 2002 veröffentlicht
- ▶ existiert in mehreren Bit Variante



# FUNKTIONEN

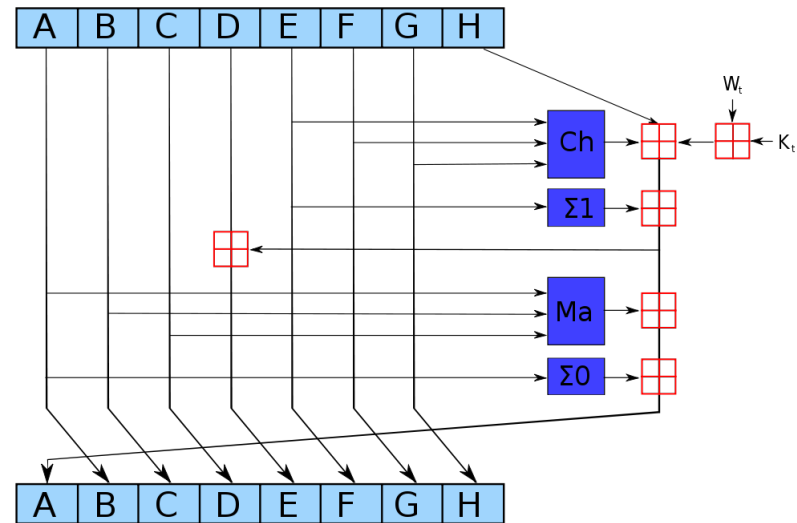
$$Ch(E, F, G) = (E \wedge F) \oplus (\neg E \wedge G)$$

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

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

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

## DARSTELLUNG DES ALGORITHMUS





# PSEUDOCODE

- Initialize variables (first 32 bits of the fractional parts of the square roots of the first 8 primes 2..19):  
 $h[0..7] := 0x6a09e667, [\dots], 0x5be0cd19$

## 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  $w[0..15]$

was ist ein rechtsrotate? was ist padding?

## 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 {  
     $S_0 := (a \text{ rightrotate } 2) \text{ xor } (a \text{ rightrotate } 13) \text{ xor } (a \text{ rightrotate } 22)$ 
```

## HAUPTSCHLEIFE

```
for  $i = 0$  to 63 {  
     $S_0 := (a \text{ rightrotate } 2) \text{ xor } (a \text{ rightrotate } 13) \text{ xor } (a \text{ rightrotate } 22)$   
     $maj := (a \text{ and } b) \text{ xor } (a \text{ and } c) \text{ xor } (b \text{ and } c)$ 
```

# HAUPTSCHLEIFE

```
for  $i = 0$  to 63 {  
     $S0 := (a \text{ rightrotate } 2) \text{ xor } (a \text{ rightrotate } 13) \text{ xor } (a \text{ rightrotate } 22)$   
     $maj := (a \text{ and } b) \text{ xor } (a \text{ and } c) \text{ xor } (b \text{ and } c)$   
     $t2 := S0 + maj$ 
```

## HAUPTSCHLEIFE

```
for  $i = 0$  to 63 {  
     $S_0 := (a \text{ rightrotate } 2) \text{ xor } (a \text{ rightrotate } 13) \text{ xor } (a \text{ rightrotate } 22)$   
     $maj := (a \text{ and } b) \text{ xor } (a \text{ and } c) \text{ xor } (b \text{ and } c)$   
     $t_2 := S_0 + maj$   
     $S_1 := (e \text{ rightrotate } 6) \text{ xor } (e \text{ rightrotate } 11) \text{ xor } (e \text{ rightrotate } 25)$ 
```

## HAUPTSCHLEIFE

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

## HAUPTSCHLEIFE

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

## HAUPTSCHLEIFE

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

# HAUPTSCHLEIFE

```

$$\begin{aligned} h0 &:= h0 + a \\ h1 &:= h1 + b \\ h2 &:= h2 + c \\ h3 &:= h3 + d \\ h4 &:= h4 + e \\ h5 &:= h5 + f \\ h6 &:= h6 + g \\ h7 &:= h7 + h \\ &\} \\ &\} // \text{Ende der foreach-Schleife} \end{aligned}$$

```



# AUSGABE

```
digest = hash = h0 append h1 append h2 append h3  
append h4 append h5 append h6 append h7
```

# 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

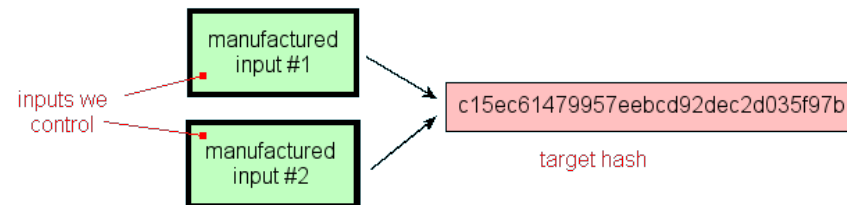
1. bis zum heutigen tage noch keine sicherheitslücken,daher eher schwachstellen als sicherheitslücken
2. allgemein wird in starke und schwache hashefunktionen unterschieden
3. wieder auf folie gucken
4. sha2 fragil, kleine änderung im algo → grosse auswirkung auf sicherheit
5. siehe quelle 14

## SCHWACHSTELLEN

- ▶ Resistenzen:
  - ▶ Kollisionsresistenz
  - ▶ Preimage Resistenz
    - ▶ Wie schwer ist es, zu einem vorgegebenen Hash-Wert eine Nachricht zu erzeugen, die denselben Hash-Wert ergibt?
  - ▶ Second Preimage Resistenz
    - ▶ Wie schwer ist es, zu einer vorgegebene Nachricht einen Hash-Wert eine Nachricht zu finden, die denselben Hash-Wert ergeben?

# KOLLISIONSRESISTENZ

Wie schwer ist es, zwei verschiedenen Nachrichten mit gleicher Prüfsumme zu finden?



1. sha3 finalisten allesamt nicht von attacken gegen prinzipielles „merkle damgard“ verfahren betroffen
2. merkle damgard: „Aus den Nachrichtenblöcken wird durch wiederholte Anwendung der Kompressionsfunktion der Hashwert erzeugt“

## SHA-3