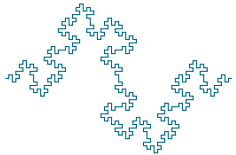
Secure Hash Algorithm

SHA-256

Chi Trung Nguyen
T-Systems



20. Juni 2012

INHALT

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

1. 2^{64} bits = 2 147 483 648 gigabytes
2. auch wenn 2^{128} bits an daten möglich sind, werden 2^{64} in der realität nicht überschritten

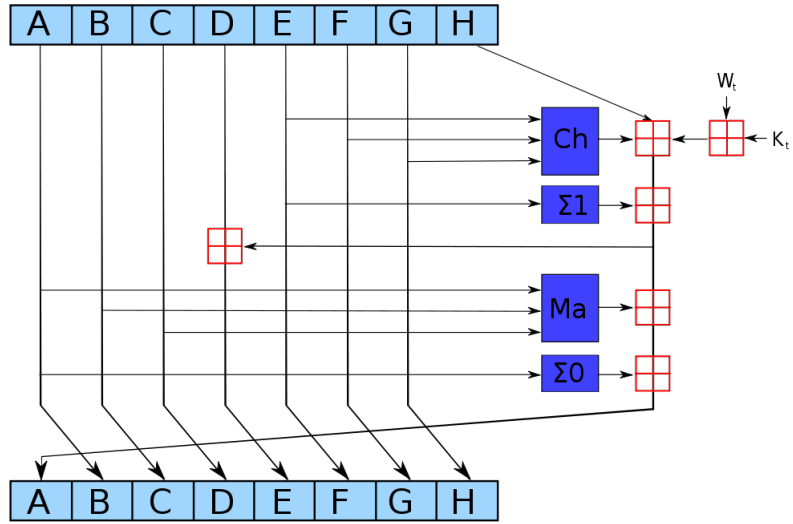
Tabelle : Secure Hash Algorithmus Eigenschaften

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,[...],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,[...], 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 >= 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 >= 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 hinzufgen

PREPROCESSING

- bit 1 zur *message* hinzufügen
- anzahl von k bits 0 hinzufügen, wobei k die kleinst mögliche Zahl >= 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 hinzufgen

- *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 {

$s0$:= ($w[i-15]$ rightrotate 7) xor ($w[i-15]$ rightrotate 18) xor ($w[i-15]$ rightshift 3)

$s1$:= ($w[i-2]$ rightrotate 17) xor ($w[i-2]$ rightrotate 19) xor ($w[i-2]$ rightshift 10)

$w[i] := w[i-16] + s0 + w[i-7] + s1$

}

HASHZUWEISUNG

$$
\begin{aligned}
a &:= h0 \\
b &:= h1 \\
c &:= h2 \\
d &:= h3 \\
e &:= h4 \\
f &:= h5 \\
g &:= h6 \\
h &:= h7
\end{aligned}
$$

HAUPTSCHLEIFE

```
for i = 0 to 63 {
    S0 := (a rightrotate 2) xor (a rightrotate
    13) xor (a rightrotate 22)
```

HAUPTSCHLEIFE

```
for i = 0 to 63 {
    S0 := (a rightrotate 2) xor (a rightrotate 13) xor (a rightrotate 22)
    maj := (a and b) xor (a and c) xor (b and c)
```

HAUPTSCHLEIFE

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

HAUPTSCHLEIFE

```
for i = 0 to 63 {
    S0 := (a rightrotate 2) xor (a rightrotate
    13) xor (a rightrotate 22)
    maj := (a and b) xor (a and c) xor (b and c)
    t2 := S0 + maj
    S1 := (e rightrotate 6) xor (e rightrotate
    11) xor (e rightrotate 25)
```

HAUPTSCHLEIFE

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

HAUPTSCHLEIFE

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

HAUPTSCHLEIFE

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

$h0 := h0 + a$
$h1 := h1 + b$
$h2 := h2 + c$
$h3 := h3 + d$
$h4 := h4 + e$
$h5 := h5 + f$
$h6 := h6 + g$
$h7 := h7 + h$
}
} *//Ende der foreach-Schleife*

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
  - htpasswd(Apache): sha1, md5
  - MySQL: sha1

VERWENDUNGSZWECK

- Digitale Zertifikate und Signaturen
- Passwortverschlüsselung
  - pam_unix: sha2, md5
  - htpasswd(Apache): sha1, md5
  - MySQL: sha1
- Prüfsummen bei Downloads

1. allgemein wird in starke und schwache hashes unterschieden
2. wieder auf folie gucken
3. sha2 fragil, kleine änderung im algo → grosse auswirkung auf sicherheit
4. siehe quelle 14

SICHERHEITSLÜCKEN & ANGRIFFSVEKTOREN

- Preimage-Angriff:
  - Wie schwer ist es, zu einem vorgegebenen Hash-Wert eine Nachricht zu erzeugen, die denselben Hash-Wert ergibt?

1. allgemein wird in starke und schwache hashes unterschieden
2. wieder auf folie gucken
3. sha2 fragil, kleine änderung im algo → grosse auswirkung auf sicherheit
4. siehe quelle 14

SICHERHEITSLÜCKEN & ANGRIFFSVEKTOREN

- Preimage-Angriff:
  - Wie schwer ist es, zu einem vorgegebenen Hash-Wert eine Nachricht zu erzeugen, die denselben Hash-Wert ergibt?
  - Kollisionsangriff: Wie schwer ist es, zwei verschiedenen Nachrichten mit gleicher Prfsumme 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