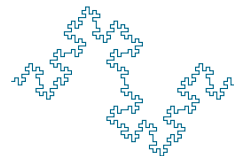


Secure Hash Algorithm

SHA-256

Chi Trung Nguyen
T-Systems



21. Juni 2012

1. um erinnerung etwas aufzufrischen: Was ist ein hash
2. geschichte gegliedert in einzelne mitglieder
3. implementierung sha256 und anwendungen von hashalgorithmen bzw sha2
4. schlussendlich kleiner ausblick auf sha3 der im sommer 2012 vorgestellt werden soll

AGENDA

EINFÜHRUNG

Was ist ein Hash?

1. um erinnerung etwas aufzufrischen: Was ist ein hash
2. geschichte gegliedert in einzelne mitglieder
3. implementierung sha256 und anwendungen von hashalgorithmen bzw sha2
4. schlussendlich kleiner ausblick auf sha3 der im sommer 2012 vorgestellt werden soll

EINFÜHRUNG	GESCHICHTE	IMPLEMENTIERUNG	ANWENDUNG	AUSBLICK
○	○○○○	○○○○○○○○	○○○○	○○
<h1>AGENDA</h1>				
<h2>EINFÜHRUNG</h2>				
<h3>Was ist ein Hash?</h3>				
<h2>GESCHICHTE</h2>				
<h3>SHA Allgemein</h3>				
<h3>SHA-0</h3>				
<h3>SHA-1</h3>				
<h3>SHA-2</h3>				

1. um erinnerung etwas aufzufrischen: Was ist ein hash
2. geschichte gegliedert in einzelne mitglieder
3. implementierung sha256 und anwendungen von hashalgorithmen bzw sha2
4. schlussendlich kleiner ausblick auf sha3 der im sommer 2012 vorgestellt werden soll

AGENDA

EINFÜHRUNG

Was ist ein Hash?

GESCHICHTE

SHA Allgemein

SHA-0

SHA-1

SHA-2

IMPLEMENTIERUNG

Algorithmus

Pseudocode

1. um erinnerung etwas aufzufrischen: Was ist ein hash
2. geschichte gegliedert in einzelne mitglieder
3. implementierung sha256 und anwendungen von hashalgorithmen bzw sha2
4. schlussendlich kleiner ausblick auf sha3 der im sommer 2012 vorgestellt werden soll

AGENDA

EINFÜHRUNG

Was ist ein Hash?

GESCHICHTE

SHA Allgemein

SHA-0

SHA-1

SHA-2

IMPLEMENTIERUNG

Algorithmus

Pseudocode

ANWENDUNG

Verwendungszweck

Schwachstellen/ Angriffsvektoren

1. um erinnerung etwas aufzufrischen: Was ist ein hash
2. geschichte gegliedert in einzelne mitglieder
3. implementierung sha256 und anwendungen von hashalgorithmen bzw sha2
4. schlussendlich kleiner ausblick auf sha3 der im sommer 2012 vorgestellt werden soll

AGENDA

EINFÜHRUNG

Was ist ein Hash?

GESCHICHTE

SHA Allgemein

SHA-0

SHA-1

SHA-2

IMPLEMENTIERUNG

Algorithmus

Pseudocode

ANWENDUNG

Verwendungszweck

Schwachstellen/ Angriffsvektoren

AUSBLICK

SHA-3

1. bsp automarken, mercedes = 1, bmw = 2
2. krypto hash, hash unterschied: hash beliebig abgebildet (z.B. Perl),
krypthash = berechnet mit algorithmus

WAS IST EIN HASH?

- deutsch: „zerhacken“, „verstreuen“

1. bsp automarken, mercedes = 1, bmw = 2
2. krypto hash, hash unterschied: hash beliebig abgebildet (z.B. Perl),
krypthash = berechnet mit algorithmus

WAS IST EIN HASH?

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

1. bsp automarken, mercedes = 1, bmw = 2
2. krypto hash, hash unterschied: hash beliebig abgebildet (z.B. Perl),
krypthash = berechnet mit algorithmus

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

1. sha-0 wurde...
2. sha steht für secure hash algorithm und ist eine gruppe von hash algorithmn

SHA ALLGEMEIN

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

1. sha-0 wurde...
2. sha steht für secure hash algorithm und ist eine gruppe von hash algorithmn

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

1. ursprünglich als
2. 1991 empfohlen

SHA-0

- ▶ 1993 veröffentlicht

1. ursprünglich als
2. 1991 empfohlen

SHA-0

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

1. alles optional, nicht vorlesen!
2. 2005 von Xiaoyun Wang, Yiqun Lisa Yin und Hongbo Yu an der Shandong University in China gebrochen
3. Ihnen war es gelungen, den Aufwand zur Kollisionsberechnung von 2^{80} auf 2^{69} zu verringern
4. es wurde ein rechtsshift durch ein linksshift ersetzt
5. 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. alles optional, nicht vorlesen!
2. 2005 von Xiaoyun Wang, Yiqun Lisa Yin und Hongbo Yu an der Shandong University in China gebrochen
3. Ihnen war es gelungen, den Aufwand zur Kollisionsberechnung von 2^{80} auf 2^{69} zu verringern
4. es wurde ein rechtsshift durch ein linksshift ersetzt
5. 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

1. die hier gelistet sind(nächste folie)

SHA-2

► 2002 veröffentlicht

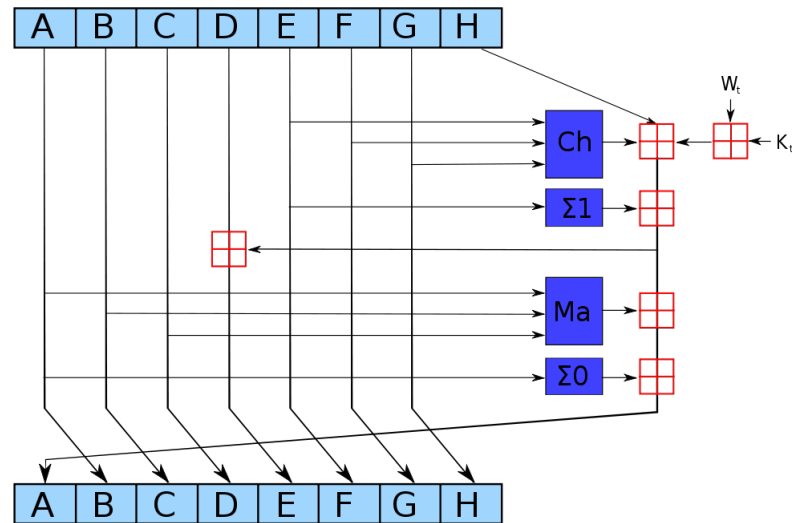
1. die hier gelistet sind(nächste folie)

SHA-2

- ▶ 2002 veröffentlicht
- ▶ existiert in mehreren Bit Variante

1. SHA2 teilt erst in messages, dann in 64 byte chunks, wendet mathematische funktionen auf jeden chunk, und fügt diese dann dem hash hinzu.
2. der algorithmus tut dies für jeden chunk und fügt es dann dem hash wert hinzu.
3. am ende bekommt man einen hash wert mit einer genauen länge

DARSTELLUNG DES ALGORITHMUS



1. rotate!!

FUNKTIONEN

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

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

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

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

PSEUDOCODE

- Initialisiere Variablen (die ersten 32 Bits der Nachkommastellen der Quadratwurzeln von den ersten 8 Primzahlen 2..19):

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

PSEUDOCODE

- Initialisiere Variablen (die ersten 32 Bits der Nachkommastellen der Quadratwurzeln von den ersten 8 Primzahlen 2..19):

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

- Initialisiere Variablen der Runden Konstanten (die ersten 32 Bits der Nachkommastellen der Kubikwurzel von den ersten 64 Primzahlen 2..311):

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

1. bsp nachricht „abc“
2. representation in ascii: 97 = a, 98=b,99=c
3. a= 0110 0001
4. b= 0110 0010
5. c= 0110 0011
6. da $8 \times 3 = 24$ ergibt sich $448 - (24 + 1) = 423$ zero Bits
7. am ende nachricht hinzufügen

PREPROCESSING

- bit 1 zur *message* hinzufügen

1. bsp nachricht „abc“
2. representation in ascii: 97 = a, 98=b,99=c
3. a= 0110 0001
4. b= 0110 0010
5. c= 0110 0011
6. da $8 \times 3 = 24$ ergibt sich $448 - (24 + 1) = 423$ zero Bits
7. am ende nachricht 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

1. bsp nachricht „abc“
2. representation in ascii: 97 = a, 98=b,99=c
3. a= 0110 0001
4. b= 0110 0010
5. c= 0110 0011
6. da $8 \times 3 = 24$ ergibt sich $448 - (24 + 1) = 423$ zero Bits
7. am ende nachricht hinzufugen

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 hinzufugen

1. bsp nachricht „abc“
2. representation in ascii: 97 = a, 98=b,99=c
3. a= 0110 0001
4. b= 0110 0010
5. c= 0110 0011
6. da $8 \times 3 = 24$ ergibt sich $448 - (24 + 1) = 423$ zero Bits
7. am ende nachricht 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
 - ▶ 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 $w[0..15]$

423
64
01100001 01100010 01100011 1 00...00
00...011000

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 {  
     $S0 := (a \text{ rightrotate } 2) \text{ xor } (a \text{ rightrotate } 13) \text{ xor } (a \text{ rightrotate } 22)$   
     $maj := (a \text{ and } b) \text{ xor } (a \text{ and } c) \text{ xor } (b \text{ and } c)$   
     $t2 := S0 + maj$   
     $S1 := (e \text{ rightrotate } 6) \text{ xor } (e \text{ rightrotate } 11) \text{ xor } (e \text{ rightrotate } 25)$ 
```


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

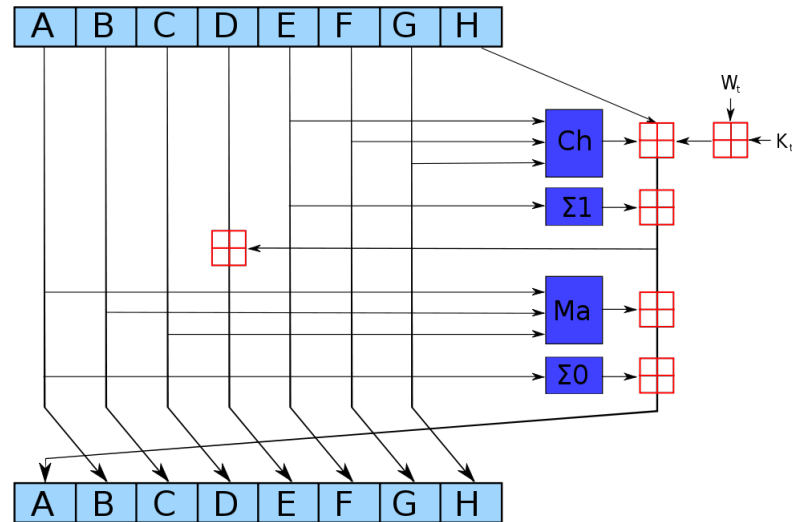
```
h0 := h0 + a
h1 := h1 + b
h2 := h2 + c
h3 := h3 + d
h4 := h4 + e
h5 := h5 + f
h6 := h6 + g
h7 := h7 + h
}
} //Ende der foreach-Schleife
```

AUSGABE

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

1. SHA2 teilt erst in messages, dann in 64 byte chunks, wendet mathematische funktionen auf jeden chunk, und fügt diese dann dem hash hinzu.
2. der algorithmus tut dies für jeden chunk und fügt es dann dem hash wert hinzu.
3. am ende bekommt man einen hash wert mit einer genauen länge

DARSTELLUNG DES ALGORITHMUS



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

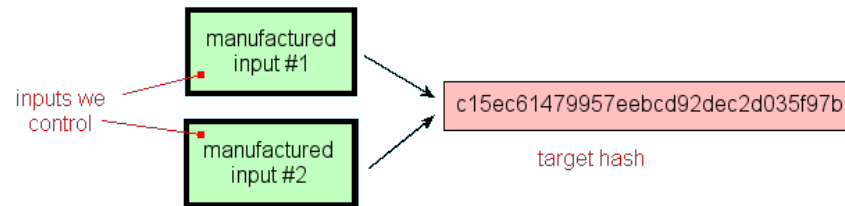
SCHWACHSTELLEN / ANGRIFFSVEKTOREN

- ▶ Resistenzen:
 - ▶ Kollisionsresistenz
 - ▶ Preimage Resistenz
 - ▶ Second Preimage Resistenz

1. kollisionsfreiheit unmöglich, es muss nur schwer sein welche zu finden: da es unendlich viele eingaben, aber nur endlich viele hashes gibt
2. schwache kr: vorgegebene eingabe, gesucht ist eine eingabe, die zum gleichen hash führt
3. schwache kr: gesucht sind zwei beliebige eingaben, die zum gleichen hash führt

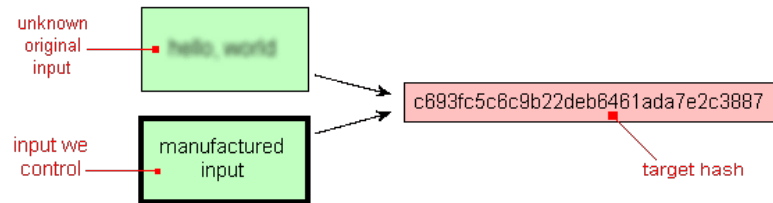
KOLLISIONSRESISTENZ

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



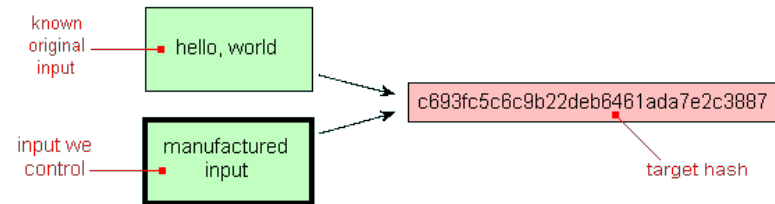
PREIMAGE RESISTENZ

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?



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“
3. bisher langsamer, eventuell unnötig da sha2 ungebrochen + hoher migrationsaufwand

SHA-3

- 2007 rief NIST zu einem Wettbewerb auf

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“
3. bisher langsamer, eventuell unnötig da sha2 ungebrochen + hoher migrationsaufwand

SHA-3

- ▶ 2007 rief NIST zu einem Wettbewerb auf
- ▶ 191 Einreichungen, 5 Finalisten

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“
3. bisher langsamer, eventuell unnötig da sha2 ungebrochen + hoher migrationsaufwand

SHA-3

- ▶ 2007 rief NIST zu einem Wettbewerb auf
- ▶ 191 Einreichungen, 5 Finalisten
- ▶ bisher langsamer als SHA2

Fragen?