Kryptographie, Hash-Funktion

Mathematik mit MuPAD 4, Prof. Dr. Dörte Haftendorn 1.12.02 (in MuPAD 2) Update Juni 07

http://haftendorn.uni-lueneburg.de **www.mathematik-verstehen.de**

##

Eine Hash-Funktion ist ein Einweg-Funktion, die die Nachricht komprimiert.
Sie wird im Zusammenhang mit der digitalen Signatur gebraucht.
Foderungen an eine Hashfunktion:
Sie soll möglichst **kollisionsresistent** sein.
Schwache Kollisionsresistenz liegt vor, wenn das Verändern einer Nachricht so gut wie immer zu einem
veränderten Hashwert führt.
Starke Kollisionsresistenz liegt vor, wenn das Konstruieren einer Nachricht mit gleichem Hashwert
mindestens so schwierig ist, wie den diskreten Logarithmus modolo p zu bestimmen.

##
#########

Die eigenen Funktionen, die unbedingt abgeschickt werden müsssen sind rot.

Zum Verstehen unten erst das Von - Hand- Beispiel durchführen

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

```
delete a,x,p,q:
h:=(x,y)->modp(powermod(a,x,p)*powermod(b,y,p),p):
```

Zunächst muss ein Paar von Primzahlen p und q bestimmt werden,
die p=2 q+1 erfüllen.
Suche nach passenden p und q

```
suhash:=proc(start)
          local saat,p,q;
          begin
            repeat
              saat:=random(start div 2..start);
              q:=numlib::prevprime(saat());
              p:=2*q+1;
            until isprime(p) end_repeat;
            return(p,q);
           end_proc :
```

```
pequ:=suhash(123451629872389712) //10^18
```

188890043276902823, 94445021638451411

```
p:=pequ[1]; q:=pequ[2];
```

188890043276902823

94445021638451411

Eine Primitivwurzel a von p ist zu bestimmen.

d. h. <a>=zstern(p). Bei der oben erzeugten Konstellation hat zstern(p) die Ordnung p-1=2q.
Damit kommt als Ordnung der Elemente von zstern(p) nur 2 oder q oder p-1 infrage
Wenn also modulo p weder a^2=1 noch a^q=1 gilt, dann erzeugt a zstern(p), ist also Primitivwurzel.
Der Sinn ist, dass die hash-Funktion, die ja a^x enthält, alle Werte aus zstern(p) annehmen kann.


```
pw:=proc(p,q)
      local aproc,a;
      begin
        repeat
```

```
        repeat
          aproc:=random(2..p-1);a:=aproc();
        until  (modp(a*a,p) >1) and (powermod(a,q,p)>1) end_repeat;
        return(a)
      end_proc:
```

```
a:=pw(p,q)
```

95450097444728963

Von diesem a weiß man also, dass seine Potenzen ganz zstern(p) ausschöpfen.
b ist dann die zu signierende Message irgendwie falsch, b beliebige Zahl (auch Pw)

```
bproc:=random(p): //b ist beliebige Zahl aus zstern(p)
b:=pw(p,q);
```

303074860567748890

Damit ist die Hashfunktion fertig vorbereitet.

Anwendungsphase

Nachrricht in zwei Teilen

```
h(1234567895252512 32,8976868686867898)
```

147968553472982798

```
h(1234567895252512 31,8976868686867898)
```

20611803739367713

Schon die Änderung nur einer Ziffer ergibt einen völlig anderen Hashwert.
Somit liegt "**schwache Kollisionsrestistenz**" vor
Die Nachricht wird in zwei Teile geteilt, die Hashfunktion halbiert die Länge.
Daher ist es naheliegend, sie mehrfach anzuwenden.
Für diese Hashfunktion kann man "**starke Kollisionsresistenz**" beweisen.
Diese Hashfunktion verkürz

Von Hand- Hashfunktion

```
q:=5:   p:=2*q+1; //p prim?, sonst Neuwahl von q
```

11

```
a:=7: powermod(a,2,p); powermod(a,q,p);
```

5

10

Wenn hier eine 1 steht, Neuwahl von a < p. Nun erzeugt a wirklich zstern(p), Probe dazu:

```
zstern(p)
```

zstern(11)

```
powermod(a,k,p) $ k=1..p-1
```

7, 5, 2, 3, 10, 4, 6, 9, 8, 1

Wahl von b aus zstern(p) beliebig

Wahl von b aus zstern(p) beliebig

```
b:=2: //beliebig<p  // auch pw
powermod(b,k,p) $ k=1..p-1
```

2, 4, 8, 5, 10, 9, 7, 3, 6, 1

Nachricht 45, aufgeteilt in zwei Blöcke

```
x:=4: y:=5:
ax:=powermod(a,x,p);by:=powermod(b,y,p); modp(ax*by,p);
```

3

10

8

Die letzte Zahl ist der Hashwert der Nachricht.

Überlegungen zur Kollisionsfreiheit

Will man den 1. Teil der Nachricht manipulieren, also auch 8 als Hashwert erhalten, so sieht man an der folgenden Zeile, dass es keinen anderen Wert für x gibt, der auch 8 erzeugt.
Das liegt daran dass a Primitivwurzel ist.

```
modp(powermod(a,k,p)*by,p) $ k=1..p-1;
```

4, 6, 9, 8, 1, 7, 5, 2, 3, 10

```
modp(powermod(2,k,p),p) $ k=1..p-1;  //ord(b) ablesbar
```

2, 4, 8, 5, 10, 9, 7, 3, 6, 1

Ändert man den ersten und den zweiten Teil der Nachrricht, so kann man 8 erzeugen, in jeder Zeile
kommt 8 genau einmal vor. Z.B. x=4, y=7. Eine solche Kombination ist aber bei großem p nicht zu finden.

```
array(1..p-1,1..p-1,
  [[h(k,j) $ k=1..p-1] $ j=1..p-1]);
```

$$\begin{pmatrix} 3 & 10 & 4 & 6 & 9 & 8 & 1 & 7 & 5 & 2 \\ 6 & 9 & 8 & 1 & 7 & 5 & 2 & 3 & 10 & 4 \\ 1 & 7 & 5 & 2 & 3 & 10 & 4 & 6 & 9 & 8 \\ 2 & 3 & 10 & 4 & 6 & 9 & 8 & 1 & 7 & 5 \\ 4 & 6 & 9 & 8 & 1 & 7 & 5 & 2 & 3 & 10 \\ 8 & 1 & 7 & 5 & 2 & 3 & 10 & 4 & 6 & 9 \\ 5 & 2 & 3 & 10 & 4 & 6 & 9 & 8 & 1 & 7 \\ 10 & 4 & 6 & 9 & 8 & 1 & 7 & 5 & 2 & 3 \\ 9 & 8 & 1 & 7 & 5 & 2 & 3 & 10 & 4 & 6 \\ 7 & 5 & 2 & 3 & 10 & 4 & 6 & 9 & 8 & 1 \end{pmatrix}$$