MyLib Krypto-Befehle

kry.tns Bibliotheks-Datei für Kryptografie Haftendorn Okt 2011

Vorhandene Befehle: mod(a,m) ist a modulo m, also mod(54,7) ▸ 5 (Eingebauter Befehl)

Powermod ist unten programmiert: **pmod**(a,k,m) ist a^k *modulo m* also **pmod**(2,5,7) ▸ 4 .

Das ist dasselbe wie mod(2^5,7) ▸ 4, nur pmod ist für sehr große Zahlen möglich.

mod(12345^{6789},7) ▸ mod(∞,7) ⚠ geht nicht, aber **pmod**(12345,6789,7) ▸ 1 klappt.

ordo(a,m) berechnet die Ordnung von a in Z*(m), **ordo**(5,13) ▸ 4 sagt vorher: mod(5^4,13) ▸ 1

maltafel(6) ▸ $\begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 4 & 0 & 2 & 4 \\ 3 & 0 & 3 & 0 & 3 \\ 4 & 2 & 0 & 4 & 2 \\ 5 & 4 & 3 & 2 & 1 \end{bmatrix}$ **maltafel**(7) ▸ $\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 4 & 6 & 1 & 3 & 5 \\ 3 & 6 & 2 & 5 & 1 & 4 \\ 4 & 1 & 5 & 2 & 6 & 3 \\ 5 & 3 & 1 & 6 & 4 & 2 \\ 6 & 5 & 4 & 3 & 2 & 1 \end{bmatrix}$ zeigt Multiplikationstafeln

zstern(m) zeigt die Menge der zu m teilerfremden Zahlen. **zstern**(10) ▸ {1,3,7,9}

malstern(m) gibt die Maltafel von Z*(m) **malstern**(10) ▸ $\begin{bmatrix} 1 & 3 & 7 & 9 \\ 3 & 9 & 1 & 7 \\ 7 & 1 & 9 & 3 \\ 9 & 7 & 3 & 1 \end{bmatrix}$

eulerphi(m) gibt die Anzahl der Elemente von z*(m) an, also die Zahl der zu m teilerfremden
eulerphi(10) ▸ 4

1.1

Weitere Befehle:
eingebaut ist: isPrime(71) ▸ true isPrime(120000000000031) ▸ true
und factor(91) ▸ 7·13 factor(71) ▸ 71 factor(120000000000031·1−2) ▸ 1433849·83690821
In dieser Datei programmierte Befehle
nextprime(120000000000000) ▸ 120000000000031 die Angebe der nächst größeren Primzahl
teiler(m) gibt alle Teiler von m an **teiler**(24) ▸ {1,2,3,4,6,8,12,24}
Das sind −außer der 1− gerade die Zahlen, die in **zstern**(24) ▸ {1,5,7,11,13,17,19,23} fehlen.
Für das Verstehen von Kryptografie sind vor allem die Potenzen in Z*(m) wichtig

potstern(18) ▸ $\begin{bmatrix} 1 & 1 & 5 & 7 & 11 & 13 & 17 \\ 2 & 1 & 7 & 13 & 13 & 7 & 1 \\ 3 & 1 & 17 & 1 & 17 & 1 & 17 \\ 4 & 1 & 13 & 7 & 7 & 13 & 1 \\ 5 & 1 & 11 & 13 & 5 & 7 & 17 \\ 6 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$ Die erste Spalte gibt den Exponenten k an. mod(11^3,18) ▸ 17

Die erste Zeile (ab Platz 2) ist die Basis a , innen steht dann a^k *modulo m*
Die Ordnung von a ist die Zeilennummer k der als erstes von oben nach unten auftauchenden 1.
Der Eulersche Satz besagt: in den Potenztafeln von z*(m) steht in der letzen Zeile überall 1.
Stelle diese Datei in das Verzeichnis MyLib auf den Computer oder dem Handheld. Mache "Bibliotheken aktualisieren". Klappe "Bibliotheken" auf, wähle kry und greife die benötigten Befehle z.B. **kryp\pmod**()

1.2

Erweiterter Euklidischer Algorithmus und Bestimmung des Inversen modulo m
Eingebaut ist der größte gemeinsame Teiler von a und b, deutsch ggt, hier gcd(a,b) also gcd(45,18) ▸ 9
greatest common divisor
und das kleinste gemeinsame Vielfache, deutsch kgv, hier lcm(a,b), hier lcm(12,18) ▸ 36
Hier programmierte Befehle:
ggte(a,b) ergibt die Zahlen [g,a,b] der Vielfach−Summen−Darstellung $g=s\cdot a+t\cdot b$ g ist dabei der größte gemeinsame Teiler. **ggte**(16,21) ▸ [1 4 −3] ist also zu deuten als 1=4·16+−3·21 ▸ true
Dieses nützt für die Suche nach dem multiplikativ Inversen modulo b (oder a). Obige Gleichung modulo 21 betrachtet ergibt 1=4·16 modulo 21, also ist 4 das Inverse von 16 in Z*(21) Probe mod(4·16,21) ▸ 1
Man kann die Gleichung auch modulo 16 nutzen: 1=−3·21 modulo 16. Zu negativen Zahlen addiert einmal die Modul−Zahl m, also 1=13·21 modulo 16, Probe mod(13·21,16) ▸ 1

Das Verfahren, das in ggte programmiert ist, heißt: erweiterter eulidischer Algorithmus. Er arbeitet auch für die riesigen Zahlen der Kryptografie effektiv.
Die Zahlen s und t heißen auch "Bézout Koeffizienten", ihre Existenz sichert das "**Lemma von Bézout**"
Siehe Wikipedia. Man kann es aber einfach (schulisch sinnvoll) begründen durch Rückwärtsarbeiten mit dem Euklidischen Algorithmus.

1.3

pmod 1/11

```
Define LibPub pmod(a,k,m)=
Func
© (a,k,m) → a^k mod m, Powermod
Local xx,kk,pot
kk:=k
xx:=1
pot:=a:Loop:
If mod(kk,2)=1 Then:
xx:=mod(xx·pot,m):
If kk=1 Then:
Return xx:        Exit:
  EndIf:     kk:=kk−1:
  EndIf:    kk:=kk/2:     pot:=mod(pot·pot,m): EndLoop
EndFunc
```

pmod(2,7,50)	28
pmod(2,10,5)	4

2/2

1.4

ordo 5/11

```
Define LibPub ordo(a,m)=
Func
©(a,m) → Ordnung von a in Z·m
Local ordn,pot
ordn:=1: pot:=mod (a,m):
If gcd(a,m)>1 Then
    Return "Ordnung ist nur bei teilerfremden sinnvoll.  "
EndIf:
While not (pot=1)
  pot:=mod(pot·a,m):
  ordn:=ordn+1:
EndWhile
Return ordn
EndFunc
```

ordo(5,13) ▸ 4 gibt die Ordnung vom 5 modulo 13 an.
Da heißt das gilt: 5^4 *modulo* 13 = 1 Probe mod(5^4,13) ▸ 1
Für die Kryptographie sind Elemente mit zu kleiner Ordnung unbrauchbar.
$m-1$ hat immer die Ordnung 2 Proben mod(16^2,17) ▸ 1
expand($(m-1)^2$) ▸ $m^2-2\cdot m+1$ modulo m ist dies 1.

1.5

"maltafel" erfolgreich gespeichert

```
Define LibPub maltafel (m)=
Func
© (m)→ Mal−Tafel Zm
Local i,j,aa:
aa:=newMat(m−1,m−1):
For i,1,m−1
For j,1,m−1
  aa[i,j]:=mod(i·j,m):
EndFor
EndFor
Return aa
EndFunc
```

maltafel(6) ▸ $\begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 4 & 0 & 2 & 4 \\ 3 & 0 & 3 & 0 & 3 \\ 4 & 2 & 0 & 4 & 2 \\ 5 & 4 & 3 & 2 & 1 \end{bmatrix}$

1.6

zstern 9/9

```
Define LibPub zstern (m)=
Func
© (m) → Z·m, die teilerfremden
Local i,s
s:={1}
For i,2,m−1
  If gcd(m,i)=1 Then
    s:=augment(s,{i})
  EndIf
  EndFor
Return s
EndFunc
```

zstern(20) ▸ {1,3,7,9,11,13,17,19}
zstern(48) ▸ {1,5,7,11,13,17,19,23,25,29,31,35,37,41,43,47}
zstern(17) ▸ {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16}
erzeugt die Menge Z*(m) , die Menge der zu m teilerfremden Zahlen von 1 bis m−1.
Z*(m) ist Gruppe bezüglich der Multiplikation. Es gibt also zu jedem Element ein Inverse.
Das erzeugt man mit ggte (siehe dort).

1.7

malstern 11/11

```
Define LibPub malstern (m)=
Func
© (m)→ Mal−Tafel Z·m
Local i,j,aa,zs,k:
zs:=zstern (m):
k:=eulerphi (m):
aa:=newMat(k,k)
For i,1,k
For j,1,k
  aa[i,j]:=mod(zs[i]·zs[j],m)
EndFor
EndFor
Return aa
EndFunc
```

malstern(12) ▸ $\begin{bmatrix} 1 & 5 & 7 & 11 \\ 5 & 1 & 11 & 7 \\ 7 & 11 & 1 & 5 \\ 11 & 7 & 5 & 1 \end{bmatrix}$ ist die Gruppentafel der Gruppe z*(m)

1.8

eulerphi 9/9

```
Define LibPub eulerphi(m)=
Func
© (m) → Anzahl in Zstern (m), der teilerfremden
Local i,s
s:=1
For i,2,m-1
  If gcd(m,i)=1 Then
    s:=s+1
  EndIf
EndFor
Return s
EndFunc
```

eulerphi(20) ▸ 8 ist die Anzahl der Elemente in Z*(m), also die Anzahl der Teilerfemden. (pos. <m)
eulerphi(19) ▸ 18 Es gilt eulerphi(p)=p−1
und eulerphi(p·q)=(p−1)·(q−1) für Primzahlen p und q eulerphi(5·7) ▸ 24 (5−1)·(7−1) ▸ 24

1.9

nextprime 11/16

```
Define LibPub nextprime(a)=
Func
© (a) →nächste Primzahl größer a
Local i,aa:
aa:=a:
If mod(aa,2)=0 Then
  i:=1:
 Else
  i:=0:
 EndIf:
 displ·i:
  Loop
   If isPrime(aa+i) Then
     Return aa+i:
     Exit:
    EndIf:
    i:=i+2:
  EndLoop
EndFunc
```

nextprime(100) ▸ 101 nextprime(randInt(100,1000)) ▸ 661 für beliebige 3-stellige Primzahl

1.10

teiler 9/9

```
Define LibPub teiler(m)=
Func
©(m) →{alle Teiler von m }
Local i,t
t:={}
For i,1,floor(m)
  If mod(m,i)=0 Then
    t:=augment(t,{i})
  EndIf:
EndFor:
Return t
EndFunc
```

teiler(72) ▸ {1,2,3,4,6,8,9,12,18,24,36,72}
teiler(31) ▸ {1,31} gibt die Menge der Teiler von M aus.
a heißt Teiler von b, wenn es ein k aus N mit a*k=b gibt.

1.11

potstern 8/14

```
Define LibPub potstern(m)=
Func
© (m)→ Potenztafel, Exponenten 1.Spalte
Local i,j,aa,zs,k
zs:=zstern (m)
k:=eulerphi (m)
aa:=newMat(k,k+1)
For i,1,k
For j,2,k+1
  aa[i,j]:=pmod(zs[j−1],i,m):
EndFor
EndFor
For i,1,k
aa[i,1]:=i:
EndFor
Return aa
EndFunc
```

$$\text{potstern}(20) ▸ \begin{bmatrix} 1 & 1 & 3 & 7 & 9 & 11 & 13 & 17 & 19 \\ 2 & 1 & 9 & 9 & 1 & 1 & 9 & 9 & 1 \\ 3 & 1 & 7 & 3 & 9 & 11 & 17 & 13 & 19 \\ 4 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 5 & 1 & 3 & 7 & 9 & 11 & 13 & 17 & 19 \\ 6 & 1 & 9 & 9 & 1 & 1 & 9 & 9 & 1 \\ 7 & 1 & 7 & 3 & 9 & 11 & 17 & 13 & 19 \\ 8 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Für das Verstehen von Kryptografie sind vor allem die Potenzen in Z*(m) wichtig
Die erste Spalte gibt den Exponenten k an.
$\text{mod}(11^3,18) ▸ 17$
Die erste Zeile (ab Platz 2) ist die Basis a , innen steht dann a^k *modulo m*
Die Ordnung von a ist die Zeilennummer k der als erstes von oben nach unten auftauchenden 1.

$$\text{potstern}(10) ▸ \begin{bmatrix} 1 & 1 & 3 & 7 & 9 \\ 2 & 1 & 9 & 9 & 1 \\ 3 & 1 & 7 & 3 & 9 \\ 4 & 1 & 1 & 1 & 1 \end{bmatrix}$$

1.12

ggte 3/16

```
Define LibPub ggte(a,b)=
Func
©(a,b) → [ggt  s  t] mit ggt=sa+tb
Local r0,r1,r2,q0,q1,s0,s1,s2,t0,t1,t2
r0:=a:r1:=b: s0:=0:s1:=1:t0:=1:t1:=−iPart(a/b):
Loop
  r2:=mod(r0,r1)
  If r2=0 Then
    Return [r1  s0  t0]
    Exit
   EndIf
   q0:=iPart(r0/r1)
   q1:=iPart(r1/r2)
   s2:=s0−q1·s1
   t2:=t0−q1·t1
   r0:=r1: r1:=r2: q0:=q1:
   s0:=s1:s1:=s2:t0:=t1:t1:=t2
 EndLoop
EndFunc
```

Erweiterter Euklidischer Algorithmus

ggte(12345,6789) ▸ [3 −903 1642]

ggte(a,b) ergibt die Zahlen [g,a,b] der Vielfach-Summen-Darstellung $g=s\cdot a+t\cdot b$ g ist dabei der größte gemeinsame Teiler.
ggte(16,21) ▸ [1 4 −3] ist also zu deuten als $1=4\cdot 16+-3\cdot 21$
Dieses nützt für die Suche nach dem multiplikativ Inversen modulo b (oder a).
Obige Gleichung modulo 21 betrachtet ergibt $1=4\cdot 16$ modulo 21, also ist 4 das Inverse von 16 in Z*(21) Probe mod(4·16,21) ▸ 1
Man kann die Gleichung auch modulo 16 nutzen: $1=-3\cdot 21$ modulo 16. Zu negativen Zahlen addiert einmal die Modul-Zahl m, also $1=13\cdot 21$ modulo 16. Probe mod(13·21,16) ▸ 1

1.13