RSA-Verfahren, Public-Key-Verschlüsselung

Kryptographie mit MuPAD, Buch-Version
Prof. Dr. Dörte Haftendorn, Mathematik mit MuPAD 4.02,
(ex. in 2.5 vom Sept 99 Nov.02 und in 3.11 Sept. 05) Feb.07, Juni 07
http://haftendorn.uni-lueneburg.de **www.mathematik-verstehen.de**

www.mathematik-sehen-und-verstehen.de
Buch: Mathematik sehen und verstehen Seite 36ff (mit Angriff Seite 38)
##
----------eigene Zahlentheorie
Ergänzungen----------
Im Dateimenu bei "Eigenschaften" sehen die beien Prozeduren zur Umwandlung von Text in Zahl und Zahl in Text, ebenso zstern(n), daher können sie hier ausgeführt werden.
----------eigene Zahlentheorie
Ergänzungen----------

Vorbereitungsphase

Anton bereitet seine Schlüssel vor.

```
saat:=random(100001..999999): saat():
```

```
p:=numlib::prevprime(floor(sqrt(saat()*10^50)));
10^27
//Exponent 50 z.B.auch 300
```

$$9879888663340290901053659039$$

$$1000000000000000000000000000$$

```
q:=numlib::prevprime(floor(sqrt(27*10^50)));
```

$$5196152422706631880582331 3$$

```
//p:=19;q:=23; //für kleine Beispiele
```

```
n:=p*q;
```

$$51337407414087439481814738015798697468396311957937620 7$$

```
ph:=(p-1)*(q-1);
```

$$51337407414087439481814737022613678711660589971989385 6$$

Wahl einer beliebigen, aber zu ph teilerfremden Zahl e

```
repeat
r:= random(2..ph): e:=r():
  until gcd(ph,e)=1 end_repeat:e;
```

$$14211102285635178709199347646723264762117225576112405 5$$

Bestimmung des modularen Inversen zu dieser Zahl $d \underset{\varphi}{\equiv} e^{-1}$

```
igcdex(ph,e);
d:=op(igcdex(ph,e),3); // letztes Element euklid. Algorithmus
```

$$1,\ -18543081669218052151458707763529171253790716040530324,\ 6698662209528915845103492069026$$

$$6698662209528915845103492069026480671183399573245987 9$$

```
if (d<0) then d:=d+ph: end_if:d; //Korrektur bei negativem d
```

$$6698662209528915845103492069026480671183399573245987 9$$

```
6698662209528915845103492069026480671183399573245987 9
```

```
d*e mod ph
```

```
1
```

d ist Antons geheimer Schlüssel.
Der Öffentlichkeit gibt er e und n bekannt.

```
e;n;
```

```
1421110228563517870919934764672326476211722557611240 55

5133740741408743948181473801579869746839631195793762 07
```

##

Anwendungsphase Berta schreibt an Anton

Berta will Anton einen Text senden, den nur Anton lesen kann.

----------eigene Zahlentheorie
Ergänzungen----------
Im Dateimenu bei "Eigenschaften" sehen die beien Prozeduren zur Umwandlung von Text in Zahl und Zahl in Text, daher können sie hier ausgeführt werden.
----------eigene Zahlentheorie
Ergänzungen----------

```
m:=txToZoo("Dienstag im Medley");
// ASCII, ASCII-28->zweistellig
[68, 105, 101, 110, 115, 116, 97, 103, 32, 105, 109, 32, 77, 101, 100, 108

   101, 121]
```

```
407773828788697504778104497372807393
```

```
gcd(m,n); // soll 1 sein, Probe
```

```
1
```

$$m^{e} \bmod n$$

```
c:=powermod(m,e,n);
```

```
422092815980147018484557424023146462288454662211412762
```

##

Anwendungsphase Anton liest, was Berta schrieb

Anton empfängt diesen verschlüsselten Text $c^{d} \bmod n$ **und wandelt ihn in Klartext um.**

```
m2:=powermod(c,d,n);
```

```
407773828788697504778104497372807393
```

```
407773828788697504778104497372807393
```

```
zooToTx(m2)//Zweierpakete +28->ASCII
[40, 77, 73, 82, 87, 88, 69, 75, 4, 77, 81, 4, 49, 73, 72, 80, 73, 93]
[68, 105, 101, 110, 115, 116, 97, 103, 32, 105, 109, 32, 77, 101, 100, 108

   101, 121]
```

"Dienstag im Medley"

##

Signatur Erstellung

Anton signiert einen der Öffentlichkeit präsentierten Text

```
M:=txToZoo("Vertraut Emil nicht");
[86, 101, 114, 116, 114, 97, 117, 116, 32, 69, 109, 105, 108, 32, 110, 105

   104, 116]
```

587386888669898804418177800482777176 88

$$sig \equiv_n M^d$$

```
sig:=powermod(M,d,n);
```

180283533172088836555685852507364223143184624941812 16

Anton stellt den Text und sig öffentlich aus.

Eigentlich wendet er auf M vorher noch eine Hashfunktion an,
die öffentlich bekannt ist und M auf 128 Bit reduziert. Hier ist h(M)=M.

##

Signatur Verifizierung

Berta will prüfen, ob das wirklich Antons unveränderter Text ist.

```
MB:=M: //Berta berechnet ebenfalls aus dem erhaltenen MB das h(M),
      //hier ist h(MB)=MB. Und sie bildet ein Mtest aus der Signatur.
```

```
Mtest:=powermod(sig,e,n);
```

587386888669898804418177800482777176 88

$$sig^e \equiv_n M$$

```
if (modp(MB,n)=modp(Mtest,n)) then print("Anton hat wirklich unterschrieben")
else print("Vorsicht, das ist nicht Antons Original!") end_if;
"Anton hat wirklich unterschrieben"
```

Wenn aber nun MisterX verändert hat

Wenn aber nun MisterX verändert hat

```
//MB:=txToZoo("Vertraut Emil ");
//MB:=M+1234567890
MB:=M+1;
Mtest:=powermod(sig,e,n);
```

587386888669898804418177800482777176 89

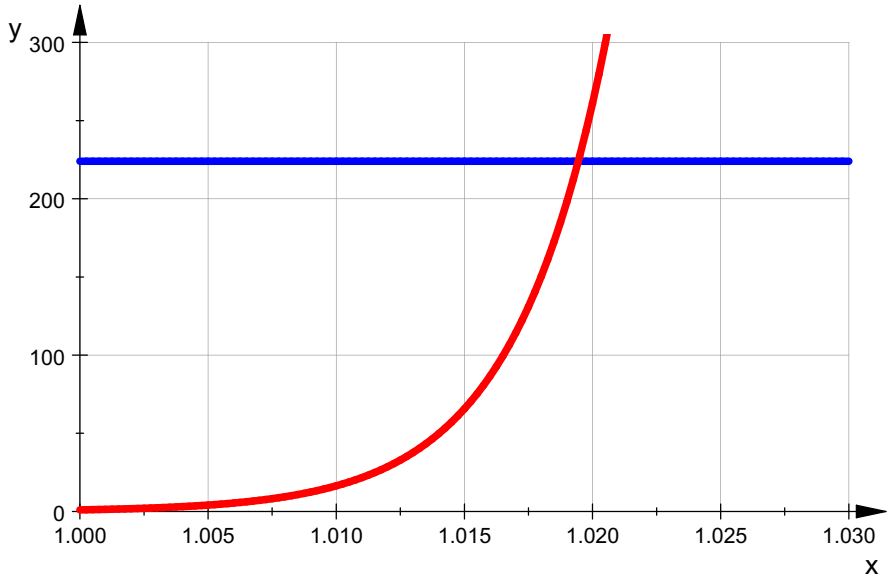
587386888669898804418177800482777176 88

```
if (modp(MB,n)=modp(Mtest,n)) then print("Anton hat wirklich unterschrieben")
else print("Vorsicht, das ist nicht Antons Original!") end_if;
```

"Vorsicht, das ist nicht Antons Original!"

########################## Angriff auf das RSA-Verfahren#########
Stetige Potenzfunktion und Umkehrfrage:
Für wWelches x wird der Wert 224 erreicht?

```
plotfunc2d(224,x^(281), x=1..1.03,
ViewingBoxYRange=0..300,
LegendVisible=FALSE, LineWidth=1, GridVisible=TRUE)
```



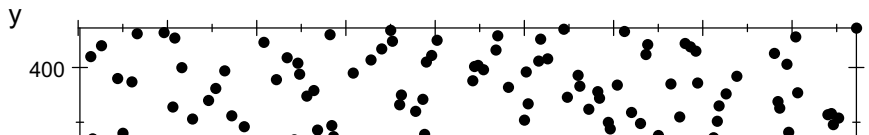
Diskrete Potenzfunktion und Umkehrfrage:
Für welches x wird der Wert 224 erreicht?

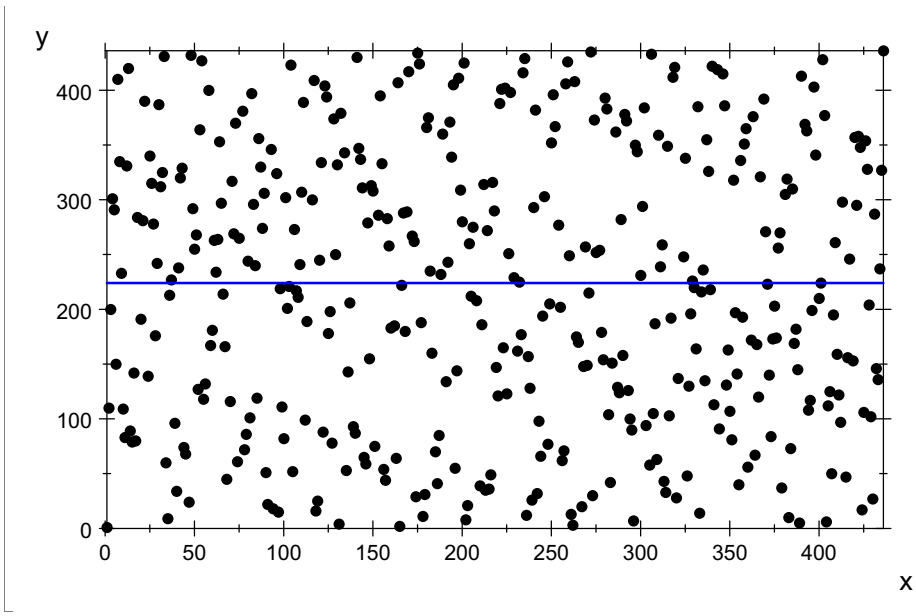
```
li:=zstern(437);
pli:=[[k,powermod(k,281,437)] $ k in li];
```

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 20, 21, 22, 24, 25, 26, 27, 28, 29, 30, 31, 32,

[[1, 1], [2, 110], [3, 200], [4, 301], [5, 291], [6, 150], [7, 410], [8, 335], [9, 233], [10, 109], [11, 83], [12

```
plot(plot::Listplot(pli, LinesVisible=FALSE),plot::Polygon2d([[1,224], [436,224]]) );
```


y
400
300
200
100
0
0
50
100
150
200
250
300
350
400
x