ElGamal-Verfahren, Public-Key-Verschlüsselung

Kryptographie mit MuPAD,

Prof. Dr. Dörte Haftendorn, Mathematik mit MuPAD 4.02, (es ex. Vorlesungs-Version)

(ex. in 2.5 Okt 99 , vom Nov.02 und in 3.11 Sept. 05) Feb.07

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

##

Verabredungsphase

Anton, Berta und Tobi wollen gemeinsam kommunizieren und eine gemeinsame Schlüsselliste vereinbaren. Tobi bereitet als Grundlage p und g vor.

```
p:=numlib::prevprime(floor(sqrt(5*10^150)));
```

$$223606797749978969640917366873127623544061835961152572427089724541052092557 9$$

```
//p:=113:g:=987:
r:= random((p div 2)..p-1): g:=r();
```

$$1125188049991322611767494049546974972203470820402313777879430824841788674992$$

Sie vereinbaren eine Verschlüsselungsfunktion

```
unassign(f,f_inv,kk,cc):      f:=(kk,cc)->(kk*cc):f(kk,cc);
```

$$cc \cdot kk$$

```
f_inv:=(kk,cc)->(cc div kk):f_inv(kk,cc);
```

$$cc \text{ div } kk$$

Er teilt dieses allen mit, jeder darf das wissen.
Jeder wählt sich eine beliebige Zahl t als geheimen Schlüssel und berechnet einen öffentlichenTeil seines Schlüssels:

```
r:= random((p div 2)..p-1): ta:=r();tAnton:=powermod(g,ta,p);
```

$$2047385427745329519482311892854523317525371989456698460765143373350196980407$$

$$2198106782170261952508385452733974228272525832517966680633537685245896848037$$

```
r:= random((p div 2)..p-1): tb:=r();tBerta:=powermod(g,tb,p);
```

$$2052016025056439191608935776425691938899349450977845212306350450652345311646$$

$$1781270606758837145672419927629150274911882304642030152001466861577557670 48$$

```
r:= random((p div 2)..p-1): tt:=r();tTobi:=powermod(g,tt,p);
```

$$1214145332087991122789359008099997507269022127504972142833439765247039604099$$

$$1665430760032805556447195403419979548784355216763938773781966237303598934947$$

```
schluessel:=matrix([tAnton,tBerta,tTobi]);//Öffentliche Schlüsselliste
```

$$\begin{pmatrix} 2198106782170261952508385452733974228272525832517966680633537685245896848037 \\ 1781270606758837145672419927629150274911882304642030152001466861577557670 48 \\ 1665430760032805556447195403419979548784355216763938773781966237303598934947 \end{pmatrix}$$

Vorbereitungsphase für eine Sendung

Anton will an Berta und Tobi etwas senden. Er wählt sich eine beliebige Zahl a und berechnet:

```
r:= random((p div 2)..p-1): a:=r();antonOffen:=powermod(g,a,p);
```

$$1545704247821516785553235759869102957995005419200827187207099968005422007510$$

$$1359403352784296067401588285547596442867248706789358298099197344688016397236$$

```
k_AB:=powermod(tBerta,a,p);k_AT:=powermod(tTobi,a,p);
```

$$1377090440809803598944721699744038082072463880357473769674641009396150853936$$

$$7961938821376854017847729422044327673362768107458029075309671688011547419 22$$

```
79619388213768540178477294220443276733627681074580290753096716880115474192 2
```

Dieses Antons Kommunikationsschlüssel für Berta und Tobi.
Nun will Anton an die anderen beiden einen Text senden, den nur die beiden lesen können.

```
txToZoo:=proc(w)
        local i,le,ur,wz;
        begin
          wz:=0:le:=length(w);ur:=numlib::toAscii(w);
           print(ur);
           (wz:=wz+(ur[i]-28)*100^(le-i)) $ i=1..le:
           return(wz);
         end_proc:
zooToTx:=proc(wz)
        local ur,le,i;
        begin
          ur:=[]:
          repeat  ur:=[wz mod 100].ur;
     wz:=floor(wz/100);
          until wz<=0 end_repeat;
          print(ur); le:=nops(ur);
          //erstmal aus der Zahl eine Liste machen
          //dann die Listenzahlen in ASCII
           for i from 1 to le do  ur[i]:=ur[i]+28  end_for:
           print(ur);
           return(numlib::fromAscii(ur));
      end_proc:
```

```
m:=txToZoo("Montag im Medley");
//m:=10;
[77, 111, 110, 116, 97, 103, 32, 105, 109, 32, 77, 101, 100, 108, 101, 121]
 4983828869750477810449737280739 3
```

Sendung

```
A_an_Berta:=f(k_AB,m);
A_an_Tobi:=f(k_AT,m);
```

```
 6863183095165310733451006884003124987165817905575375903317965554164094108186151684138260585925428260394884 8

 3968094055716505779614981844262796369791717621877051938515693443181465488584339862210313973205795952862934 6
```

Berta erhält antonOffen und A_an_Berta,
Tobi erhält antonOffen und A_an_Tobi.

Entschlüsselungsphase

Berta berechnet:

```
k_AB:=powermod(antonOffen,tb,p); vonAntonB:=f_inv(k_AB,A_an_Berta);
```

```
 1377090440809803598944721699744038082072463880357473769674641009396150853936

 4983828869750477810449737280739 3
```

Tobi berechnet:

```
k_AT:=powermod(antonOffen,tt,p);vonAntonT:=f_inv(k_AT,A_an_Tobi);
```

```
 79619388213768540178477294220443276733627681074580290753096716880115474192 2

 4983828869750477810449737280739 3
```

```
bool(vonAntonB=vonAntonT)
```

```
 TRUE
```

Beide haben also dasselbe erhalten.
Die Message-Zahl muss noch in Klartext verwandelt werden.

```
klar:=zooToTx(vonAntonB);
[49, 83, 82, 88, 69, 75, 4, 77, 81, 4, 49, 73, 72, 80, 73, 93]
[77, 111, 110, 116, 97, 103, 32, 105, 109, 32, 77, 101, 100, 108, 101, 121]
 "Montag im Medley"
```

##

Signatur mit dem ElGamal-Verfahren= DSS

= Digital Signature Standard

= Digital Signature Standard
Anton will seine Nachricht signieren, bekannt sind g, p und sein öffentlicher Schlüssel tAnton:
Er wählt ra teilerfremd zu p-1

```
repeat
r:= random((p div 2)..p-1): ra:=r():
  until gcd(p-1,ra)=1 end_repeat:ra;
```

152204874827439497882497692831649955338352261056227646715158927523402694484 7

```
kra:=powermod(g,ra,p);
```

165169299894188277910991412416397220193673866339989721708162452372502207473 7

```
matrix([igcdex(p-1,ra)]);
```

$$\begin{pmatrix} 1 \\ 28128635793993524098538819954357720973711226112590762613326762844105615979 6 \\ -4132426232800535797264350494090139323036383226797603045619182505594108149 21 \end{pmatrix}$$

```
ra_inv:=op(igcdex(p-1,ra),3);  // letztes Element euklid. Algorithmus
```

-4132426232800535797264350494090139323036383226797603045619182505594108149 21

```
if (ra_inv<0) then ra_inv:=ra_inv+p-1: end_if:ra_inv; //Korrektur bei negativem ra_inv
```

1822825354219736116682738619322262303136980036931765419708978994851110110657

```
//m:=77;
```

```
sa:=modp((m-ta*kra)*ra_inv,p-1);
```

1348199051406630123868860360848529951447543651850126021827387469835702448998

Anton sendet m und die digitale Unterschrift zu m also [m, sa,kra];

Prüfung der Unterschrift

Tobi will prüfen, ob die Nachricht wirklich von Anton kommt.
Er berechnet

```
test1:=powermod(g,m,p);
test2:=modp(powermod(tAnton,kra,p)*powermod(kra,sa,p),p);
```

2938000081347053381886914118958043112514813098260868710636674837149269215 3

2938000081347053381886914118958043112514813098260868710636674837149269215 3

```
if (test1=test2) then print("Das hat Anton geschrieben: m=".m)
else print("Vorsicht, das ist nicht von Anton Original!") end_if;
```

"Das hat Anton geschrieben: m=4983828869750477810449737280739 3"

Die von Anton gesendete Signatur ist mit der Nachrricht "verwoben". Sie läßt sich für keine andere Nachricht verwenden. Wenn die Nachricht verändert wurde, gelingt der Test ebenfalls nicht.

```
m:=m+7;
test1:=powermod(g,m,p);
test2:=modp(powermod(tAnton,kra,p)*powermod(kra,sa,p),p);
if (test1=test2) then print("Das hat Anton geschrieben: m=".m)
else print("Vorsicht, das ist nicht von Anton Original!") end_if;
```

4983828869750477810449737280740 0

1087048827930996556743982249696139521589031221084697226831998610495773719581

2938000081347053381886914118958043112514813098260868710636674837149269215 3

"Vorsicht, das ist nicht von Anton Original!"