

EI_Gamal-Verfahren

EI-Gamal-Verfahren und DSS-Signatur
 www.mathematik-verstehen.de/Haftendorn 2012 **Signatur im 2. Teil**

1 Schlüsselerzeugung

Anton, Berta und Tobi wollen gemeinsam kommunizieren. Dazu müssen sie sich auf eine Verschlüsselungsfunktion einigen und zu zwei Zahlen p prim und g eine gemeinsame öffentliche Schlüsselliste erstellen.

1.1. p wird als Primzahl gewählt: (Mehr als 14 Stellen ging nicht.)
 $p := \text{kry'nextprime}(\text{floor}(\text{randint}(1000, 10^{14}))) - 93515877911021$
 $g := \text{randint}(1000, p-2) + 10100786731165$

1.2. Verschlüsselungsfunktion

Es muss irgendeine invertierbare Funktion gewählt werden. Sie soll die Nachricht und den (geheimen) Kommunikationsschlüssel miteinander verrechnen. Der Einfachheit halber wird hier das gewöhnliche Produkt genommen.
 $f(k,m) := k \cdot m$ Fertigkeit $\text{finv}(k,c) := c/k$ Fertigkeit Die Division findet "normal" statt, nicht modulo p wie fast alle nachfolgenden Rechnungen. Darum kann die Nachricht auch viel länger sein als die Schlüssel.

1.1

1.3 Öffentliche Schlüsselliste

Die drei wählen jeder für sich geheime Zahlen t_a, t_b, t_t und bilden den öffentlichen Teil ihres Schlüssels:

Anton: $t_a := \text{randint}(10, p-2)$
 $\text{anton} := \text{kry'pmod}(g, t_a, p)$
 Berta: $t_b := \text{randint}(10, p-2) + 585718616759$
 $\text{berta} := \text{kry'pmod}(g, t_b, p) + 50089582435202$
 Tobi: $t_t := \text{randint}(10, p-2) + 51338924569939$
 $\text{tobi} := \text{kry'pmod}(g, t_t, p) + 24289576046835$

Die öffentliche Schlüsselliste ist nun

Anton $\text{anton} + 12472395282269$
 Berta $\text{berta} + 50089582435202$
 Tobi $\text{tobi} + 24289576046835$

1.2

3. Sendung

Jeder der drei kann sich nun entschließen, den beiden anderen etwas zu senden. Hier will Anton an Berta und Tobi eine geheime Nachricht senden.

3.1. Vorbereitung

Anton wählt ein a und bildet eine Zahl antonoffen , die der zusammen mit der verschlüsselten Nachricht schicken wird.
 $a := \text{randint}(10, p-2) + 80010344218109$
 $\text{antonoffen} := \text{kry'pmod}(g, a, p) + 28709408957110$

Er berechnet für die Kommunikation mit Berta einen Schlüssel
 $k_{ab} := \text{kry'pmod}(\text{berta}, a, p) + 56923304170682$

Er berechnet für die Kommunikation mit Tobi einen Schlüssel
 $k_{at} := \text{kry'pmod}(\text{tobi}, a, p) + 30217639169933$

3.2 Verwendung der Nachricht **Text-to-Zahl-Funktion sind am Ende!**
 $m := \text{txtozo}("Nach dem Krieg um Acht.")$
 $+ 506971760472738104478677375048981043771768818$

1.3

$a_{an_berta} := f(k_{ab}, m)$
 $+ 28858507727335808854631419518653189877296788555787717393876$
 $a_{an_tobi} := f(k_{at}, m)$
 $+ 153194897273109015550947713776349156172737004245048692549194$

Achtung: Berta und Tobi erhalten für dieselbe Nachricht verschiedene Kryptogramme. Beide erhalten aber dieselbe Zahl $\text{antonoffen} + 28709408957110$ dazu.

3.3. Entschlüsselung der empfangenen Daten

Berta errechnet aus antonoffen mit ihrem geheimen t_b auch einen Kommunikationsschlüssel
 $k_{b_ab} := \text{kry'pmod}(\text{antonoffen}, t_b, p) + 56923304170682$

Tobi errechnet aus antonoffen mit seinem geheimen t_t auch einen Kommunikationsschlüssel
 $k_{t_ab} := \text{kry'pmod}(\text{antonoffen}, t_t, p) + 30217639169933$

Diese Schlüssel stimmen mit denen, die Anton geheim erzeugt hatte überein:
 $k_{ab} + 56923304170682$ $k_{at} + 30217639169933$

1.4

Diese Übereinstimmung muss man beweisen!!! (Siehe unten)

Aber wegen dieser Übereinstimmung können Berta und Tobi nun mit Erfolg die inverse Verschlüsselungsfunktion anwenden.

Berta rechnet $b_{\text{von_anton}} := \text{finv}(k_{b_ab}, a_{an_berta})$
 $+ 506971760472738104478677375048981043771768818$
 und $\text{klar}_b := \text{zootox}(b_{\text{von_anton}})$ Nach dem Krieg um Acht.

Tobi rechnet $t_{\text{von_anton}} := \text{finv}(k_{t_ab}, a_{an_tobi})$
 $+ 506971760472738104478677375048981043771768818$
 und $\text{klar}_t := \text{zootox}(t_{\text{von_anton}})$ Nach dem Krieg um Acht.

Text aus "Woyzeck" von Georg Büchner, siehe Wikipedia

Beweis: $k_{b_ab} := \text{antonoffen} \cdot t_b = (g^a)^{t_b} = (g^{t_b})^a = (\text{berta})^a = k_{ab}$ und natürlich auch
 $k_{t_ab} := \text{antonoffen} \cdot t_t = (g^a)^{t_t} = (g^{t_t})^a = (\text{tobi})^a = k_{at}$ q.e.d.
 Im Beispiel $k_{b_ab} = k_{ab} + \text{true}$ $k_{t_ab} = k_{at} + \text{true}$

1.5

Hilfsfunktionen 1

$\text{ctest} := \text{txtozo}("Der Affe rennt!")$
 $+ 407386043774747304867382828805$

Grundidee:
Text to Zahl 00 (zweistellig):
 Von dem Text werden von links immer ein Buchstabe weggenommen, mit ord wird seine ASCII-Nummer bestimmt. Von der werden 28 abgezogen, damit das Ergebnis zweistellig ist. Aus der Liste li dieser zweistelligen Zahlen wird dann eine Zahl gebildet. Dazu werden die immer das bisherige Ergebnis mit 100 multipliziert und dann eine neue Zahl der Liste genommen.

txtozo 11/11

```

Define LibPub txtozo (tex)=
Func
Local i,li,intli,n,z,d,te
li:={}:d:=dim(tex):te:=tex
For i,1,d
li:=augment(li,{ord(left(te,i))-28})
te:=right(te,d-i)
EndFor
z:=0
For i,1,d
z:=100*z+li[i]
EndFor
Return z
EndFunc
    
```

1.6

Hilfsfunktionen 2

$\text{zootox}(\text{ctest})$ Der Affe rennt!
 $\text{state} := \text{txtozo}("Krypto ist gut")$
 $+ 4786938488830477878804758988$
 $\text{zootox}(\text{state})$ Krypto ist gut

Grundidee:
 Die letzten beiden Ziffern werden genommen und um 28 vermehrt in eine Liste geschrieben. Die Zahl wird hinten um zwei Stellen gekürzt.
 Dies wird wiederholt, wobei die Liste nach links verlängert wird.
 Mit char werden die Buchstaben aus den ASCII-Nummern wieder hergestellt und dann von links nach rechts aneinandergelängt.

zootox 0/11

```

Define LibPub zootox (z)=
Func
Local i,zz,li,te,teli
li:={}:zz:=z:te:= ""
While zz>=1
li:=augment(li,{mod(zz,100)+28},li)
zz:=floor(zz/100)
EndWhile
teli:=char(li)
For i,1,dim(li)
te:=te&teli[i]
EndFor
Return te
EndFunc
    
```

1.7

El_Gamal-Signatur

```

Signatur mit dem El-Gamal-Verfahren, DSS-Signatur
www.mathematik-verstehen.de/Haftendorn2012

Im 1. Teil ist das El-Gamal-Verfahren kommentiert dargestellt. Dieses ist der 2.
Teil Signatur mit dem El-Gamal-Verfahren, DSS Digital Signation Standard. Hier
agiert nur Anton, er will einen Text signieren. Er verwendet p prim und g aus dem
El-Gamalverfahren. Die Verifizierer, also Bertla und Tobl brauchen nur p,g, den
El-Gamal-Schlüssel tanton, den Klartext und die zur Signatur von Anton erzeugten drei Zahlen.
Sie brauchen nicht ihre eigenen El-Gamal-Schlüssel.

1. Schlüsselerzeugung 1.1. p wird als Primzahl und g werden gewählt: (Mehr als 14
Stellen ging nicht.)  $p := \text{kry}(\text{nextprime}(\text{floor}(\text{randint}(100000, 10^{14})))) + 97708424649959$ 
 $g := \text{randint}(1000, p-2) + 27193117001079$ 
1.2. Antons geheime Zahl  $ta := \text{randint}(10, p-2) + 26890755181010$ 
und sein El-Gamal-Schlüssel  $\text{tanton} := \text{kry}(\text{pmod}(g, ta, p)) + 66659475980037$ 

2. Nachricht und ihre Signatur Text-to-Zahl-Funktion sind am Ende!
2.1 klar := "Vertraut Emil" + "Vertraut Emil"
 $m := \text{txtozo}(\text{klar}) + 58738688866989880441817780$ 
    
```

2.1

```

2.2 Erzeugung der Signatur
Anton wählt zufällig ein  $ra := p-3$  also sinnvoll aus  $Z(p-1)^*$ :
Anton:  $ra := \text{randint}(10, p-3) + 11899922277605$  und bestimmt dazu die Inverse in
dieser Gruppe, die für die Exponenten von Potenzen in  $Z(p)$  relevant ist.
 $ea := \text{kry}(\text{ggte}(ra, p-1)) + [1 \ 40250718941437 \ -4902140513848]$  [Achtung! oben ra
neu Wählen, wenn die erste Zahl nicht 1 ist.
 $ri := \text{mod}(ea[1,2], p-1) + 40250718941437$  Probe  $\text{mod}(ra \cdot ri, p-1) + 1$  Hier muss 1
stehen. Denn: Eulerscher Satz: Für s aus  $Z(p)^*$  gilt  $s^{p-1} = 1$  modulo p
 $\text{kry}(\text{pmod}(g, ra \cdot ri, p)) - g + 0$ 
 $kra := \text{kry}(\text{pmod}(g, ra, p)) + 14185746229194$  probe
 $\text{kry}(\text{pmod}(g, ri \cdot ra, p)) + 27193117001079$  g + 27193117001079
 $sa := \text{mod}((m - ta \cdot kra) \cdot ri, p-1) + 53502626673406$ 
----- Anton gibt bekannt -----
klar + Vertraut Emil m + 58738688866989880441817780
sa + 53502626673406 kra + 14185746229194
    
```

2.2

```

3. Verifizierung
Jeder, der p und g und tanton kennt, kann nun den Klartext verifizieren.
 $\text{test1} := \text{kry}(\text{pmod}(g, m, p)) + 83210723030884$ 
 $\text{test2} := \text{mod}(\text{kry}(\text{pmod}(\text{tanton}, kra, p)) \cdot \text{kry}(\text{pmod}(kra, sa, p)), p) + 83210723030884$ 

Beweis:  $\text{test2} := \text{tanton} \cdot kra \cdot kra^{sa} \cdot (g \cdot ta)^{kra} \cdot (g \cdot ra)^{sa} \cdot ta \cdot kra \cdot g \cdot ra \cdot sa =$ 
 $g^{ta \cdot kra + ra \cdot (m - ta \cdot kra) \cdot ri} \cdot g^{ta \cdot kra + m - ta \cdot kra} \cdot g^m = \text{test2}$ 
wegen  $g^{ra \cdot ri} = g^1 = g$  q.e.d.
Anmerkung: Der Klartext kann ruhig länger sein, als  $m > p$  ist unschädlich, denn
ma muss ja nicht wiederhergestellt werden. Sowohl bei test1 als auch bei test2
wirkt m nur modulo  $p-1$ .
    
```

2.3

Hilfsfunktionen 1 $\text{ctst} := \text{txtozo}(\text{"Der Affe rennt!"})$ Grundidee: Text to Zahl 00 (zweistellig): Von dem Text werden von links immer ein Buchstabe weggenommen, mit ord wird seine ASCII-Nummer bestimmt. Von der werden 28 abgezogen, damit das Ergebnis zweistellig ist. Aus der Liste li dieser zweistelligen Zahlen wird dann eine Zahl gebildet. Dazu werden die immer das bisherige Ergebnis mit 100 multipliziert und dann eine neue Zahl der Liste genommen. Die andere Übersetzungsrichtung ist im oberen Teil.	btozo01 11/11 Define LibPub txtozo01 (<i>tex</i>)= Func Local <i>i, li, intli, n, z, d, te</i> $li := \{ \dots \}; d := \text{dim}(tex); te := tex$ For <i>i, 1, d</i> $te := \text{augment}(li, \{ \text{ord}(\text{left}(te, 1)) - 28 \})$ $te := \text{right}(te, d-i)$ EndFor $z := 0$ For <i>i, 1, d</i> $z := 100 \cdot z + li[i]$ EndFor Return z EndFunc
---	---

2.4