RSA-Verfahren und RSA-Signatur

Mathematik in wxMaxima www.mathematik-verstehen.de Haftendorn Dez 2010

0.1 Handlinghilfen

0.2 Inhalt

1 Schüsselerzeugung

Anton will sich ein Schlüsselpaar für das RSA-Verfahren herstellen

1.1 Grundlage n und phi

n wird grundsätzlich als Produkt aus zwei Primzahlen

```
(%i1) p:next_prime(floor(sqrt(random(1.0)*10^120)));
      10^60;
(%o1) 955933888724997456550138541878313750680772931707277120897041
(%o2) 1000000000000000000000000000000000000000000000000000000000000000
```

```
(%i5) q:next_prime(floor(sqrt(random(1.0)*10^120)));
      10^60;
(%o5) 694543393065990918287965977690161272389083391170992551231517
(%o6) 1000000000000000000000000000000000000000000000000000000000000000
```

```
(%i29) n:p*q;  phi:(p-1)*(q-1);
(%o29) 663937566621827132528252166075[60 digits]032226281142225126140311241197
(%o30) 663937566621827132528252166075[60 digits]557203211285902247870639112640
```

1.2 Schlüssel e und geheimes d

e muss teilerfremd zu phi gewählt werden, damit es in Z*(phi) ein Inverses d hat.

```
(%i33) /* $ unterdrückt die Ausgabe und # steht für ungleich */
       e:2$  while  gcd(e,phi)#1  do e:random(phi)$ e;
(%o35) 28710871824285197099467497723 1[60 digits]05380880823792918329753425241 7
```

```
(%i36) d:gcdex(e,phi)[1]; if d<0 then d:d+phi$ d;
(%o36) -28811803657030531445514577034 1[60 digits]05594518081798500603520916876 7
(%o38) 37581953005152181807310639573 3[60 digits]50125803046791724183542994387 3
```

```
(%i39) mod(e*d,phi);
(%o39) 1
```

1.3 Antons öffentliches Schlüsselpaar

```
(%i17) e;n;
(%o17) 20818676757219178433328379272 4[60 digits]17744209706485658484747161584 7
(%o18) 66393756662182713252825216607 5[60 digits]03222628114222512614031124119 7
```

Dies stellt Anton öffentlich zur Verfügung, z.B. aus seiner Internetseite.

2 Hilfsfunktionen

3 Anwendungphase, Verschlüsselung

Berta will Anton einen Text senden, den nur Anton lesen kann.
Sie liest im Internet (z.B.) Antons öffentliches Schlüsselpaar (e,n).

3.1 Nachricht als Zahl erzeugen

```
(%i42) m:txToZoo("Montag Treffen um 8 im Medley");
[77,111,110,116,97,103,32,84,114,101,102,102,101,110,32,117,109,32,56,32,105,109,
32,77,101,100,108,101,121]
[49,83,82,88,69,75,4,56,86,73,74,74,73,82,4,89,81,4,28,4,77,81,4,49,73,72,80,73,93
]
(%o42) 49838288697504568673747473820489810428047781044973728073 3
```

```
(%i43) /* soll 1 sein */ gcd(m,n);
(%o43) 1
```

3.2 Verschlüsseln

Nun potenziert sie m mit e modulo n

```
(%i44) c:power_mod(m,e,n);
(%o44) 22277807484954931652859259870 5[60 digits]03302015195223208595004613463 8
```

Berta sendet c an Anton

4 Empfang und Verschlüsseln

Anton empfängt c.

4.1 Entschlüsseln der Nachricht

```
(%i45) mm:power_mod(c,d,n);
(%o45) 49838288697504568673747473820489810428047781044973728073 93
```

4.2 Zurückverwandeln in Text

```
(%i46) zooToTx(mm);
[49,83,82,88,69,75,4,56,86,73,74,74,73,82,4,89,81,4,28,4,77,81,4,49,73,72,80,73,93
]
[77,111,110,116,97,103,32,84,114,101,102,102,101,110,32,117,109,32,56,32,105,109,
32,77,101,100,108,101,121]
(%o46) Montag Treffen um 8 im Medley
```

5 Digitale Signatur

Anton will einen offen lesbaren Text signieren. Es soll sich jeder überzeugen können, dass der Text nicht verändert ist. Auch bei der Signatur mit RSA wird modulo n potenziert, allerdings erst mit d und dann mit c.

5.1 Erzeugung der Signatur

Antond Text:

```
(%i47) M:txToZoo("Vertraut Emil nicht");
[86,101,114,116,114,97,117,116,32,69,109,105,108,32,110,105,99,104,116]
[58,73,86,88,86,69,89,88,4,41,81,77,80,4,82,77,71,76,88]
(%o47) 587386888669898804418177800482777176 88
```

Signatur

```
(%i48) sig:power_mod(M,d,n);
(%o48) 4707955686389403540388445829 68[60 digits]346466804686132287799052295704
```

Normalerweise wird die Zahl M erst noch einer "Hash-Funktion" übergeben, die weniger Datenplatz braucht und es wird h(M) signiert.

5.2 Prüfung der Signatur

Berta will sich überzeugen, dass der auf Antons Site lesbare Text von niemandem manipuliert wurde. Sie verwendet dazu das öffenliche RSA-Schlüsselpaar (e,n).

```
(%i49) MM:power_mod(sig,e,n);
(%o49) 587386888669898804418177800482777176 88
```

Falls keine Hashfunktion werwendet wurde, kann sie diese Zahl in Text umwandeln und lesen. Es muss derselbe Text sein, den sie bei Anton gelesen hatte.

```
(%i50) zooToTx(MM);
[58,73,86,88,86,69,89,88,4,41,81,77,80,4,82,77,71,76,88]
[86,101,114,116,114,97,117,116,32,69,109,105,108,32,110,105,99,104,116]
(%o50) Vertraut Emil nicht
```

Hat Anton einen Hashwert signiert, nimmt Berta anstelle des Textes auch den Hashwert h(M) und vergleicht mit MM.
Anton hat in diesem Fall das Hashverfahren angegeben.

5.3 Veränderng des Textes

Mister X macht aus Antons Text etwas anderes

```
(%i51) MX:txToZoo("Vertraut Emil");
[86,101,114,116,114,97,117,116,32,69,109,105,108]
[58,73,86,88,86,69,89,88,4,41,81,77,80]
(%o51) 587386888669898804418177 80
```

Wenn er die Signatur so lässt, merkt Berta sofort, dass das ein anderer Text ist. Aber die kleinste Veränderung der Signatur ist auch merkbar:

```
(%i52) sigX:sig+1;
(%o52) 4707955686389403540388445829 68[60 digits]3464668046861322877990522957 05
```

Berta nimmt erstmal diese Signatur als richtig an und rechnet

```
(%i53) MMX:power_mod(sigX,e,n);
(%o53) 4761605386592558603397888841 38[60 digits]3814670167697748605364402920 07
```

```
(%i54) zooToTx(MMX);
[47,61,60,53,86,59,25,58,60,33,97,88,88,41,38,98,58,25,65,86,4,88,22,68,84,42,24,
48,34,52,60,43,1,39,98,34,63,91,23,59,83,63,87,50,78,38,14,67,1,67,69,77,48,60,53,
64,40,29,20,7]
[75,89,88,81,114,87,53,86,88,61,125,116,116,69,66,126,86,53,93,114,32,116,50,96,
112,70,52,76,62,80,88,71,29,67,126,62,91,119,51,87,111,91,115,78,106,66,42,95,29,
95,97,105,76,88,81,92,68,57,48,35]
(%o54) KYXQrW5VX=}ttEB~V5]r t2`pF4L>PXG?C~>[w3Wo[sNjB*_?_aiLXQlD90#
```

Das ergibt nichts Sinnvolles!
Bei Hash-Verwendung würden h(M) und MMX nicht übereinstimmen.