Ludwig-Maximilans-
Universität München
Dr. Pascal Reisert

Wintersemester 2018/2019
Übungsblatt 8
5. Dezember 2018

Übungen zur Kryptographie
Lösung

Aufgabe 30

Wir folgen der Anleitung: Angenommen $N = k \cdot 2^n + 1$ ist prim für $3 \nmid k \leq 2^n$, dann ist $3^{\frac{N-1}{2}} \bmod N = \left(\frac{3}{N}\right) = -1$, da $N = 1 \bmod 4$ und $N = 0 \bmod 3$ oder $N = 2 \bmod 3$. $N = 0 \bmod 3$ gilt für die Primzahl N sicher nicht, also $N = 5 \bmod 12$.
Sei nun $N = pq^r$ mit einer Primzahl q und $\gcd(p, q) = 1$. Dann gilt nach dem Chinesischen Restsatz $\mathbb{Z}/N \simeq \mathbb{Z}/q^r \times \mathbb{Z}/p$. Dem Hinweis weiter folgend betrachten wir $(3^k)^{2^n} = 1 \bmod q^r$, da $q^r \mid N$. Damit folgt aber bereits $\text{ord}(3^k) | 2^n \Rightarrow \text{ord}(3^k) = 2^j$, $j \leq n$ und wegen $(3^k)^{2^{n-1}} = -1 \bmod q^r$ bereits $j = n$. Weiter teilt $\text{ord}(3^k)$ auch $\varphi(q) = q^{r-1}(q-1)$ und da q ungerade ist wegen N ungerade, ergibt sich $2^n \mid q - 1$, also $q = 2^l s + 1$ für $l \geq n$ und $s \in \mathbb{Z}$.[1] Somit teilt $2^l s + 1$ bereits $2^n k + 1$ oder $2^n k + 1 \bmod 2^l s + 1 = 0 \Rightarrow 0 = 2^l ks + 2^{l-n} s = 2^{n-l} s - k \bmod 2^l + 1 \Rightarrow k = 2^{l-n} s$ da $k, 2^{l-n} s < 2^l s + 1$. Es ist $k = 2^{l-n} s = 1$ und $q = N$.

Aufgabe 32

Mit `aribas` können wir die $(p-1)$-Faktorisierung wie folgt umsetzen

```
function ppexpo(B0,B1: integer): integer;
var
    x, m0, m1, i: integer;
begin
    x := 1;
    m0 := max(2,isqrt(B0)+1); m1 := isqrt(B1);
    for i := m0 to m1 do
        x := x*i;
    end;
    if odd(B0) then inc(B0) end;
    for i := B0+1 to B1 by 2 do
        if prime32test(i) > 0 then x := x*i end;
    end;
    return x;
end;

function p1_factorize(N: integer; bound := 32000): integer;
const
    anz0 = 128;
var
    base, d, B0, B1, ex, count: integer;
```

[1] Da es immer ein $q \leq \sqrt{N}$ gibt, findet man bereits den Widerspruch $2^n < q - 1 \leq \sqrt{N} - 1 < 2^n$.

```
begin
    base := 2 + random(N-2);
    if (d := gcd(base,N)) > 1 then return d end;
    write("working ");
    count := 0;
    for B0 := 0 to bound-1 by anz0 do
        B1 := min(B0+anz0, bound);
        ex := ppexpo(B0,B1);
        base := base ** ex mod N;
        if base = 1 then return 0 end;
        d := gcd(base-1,N);
        if d > 1 then
            writeln();
            writeln("factor found with bound ",B1);
            return d;
        end;
        inc(count);
        if bit_and(count,7) = 0 then write('.'); end;
    end;
    return 0;
end.

N = 12_79811_59831_64126_95784_33445_38403_78005_59935_60685_09015_
35408_61990_30519_73522_83881_56101_22317_38979_86917

p = 476_11899_38062_30257_47027_69554_72216_01662_07604_23639
q = 2688_00786_14071_57570_79641_07001_07801_97885_44394_44003
```

Wir kennen bereits die Zahl

```
z_{1256}
= 0x06C4_12BB_B7C0_E5B8_6DD8_B4B2_4DF5_F97D_9A07_294A_9103_
F374_CB38_75A9_7AB2_23C1_288A_0064_9767_7463
= 56452_72599_80720_81846_96687_34303_43477_27707_48308_40343_
81721_13239_44328_55024_86872_40710_95215_05574_88227
```

Sei a gegeben und gesucht sei ein x mit $x^2 = a \bmod N$. Angenommen wir finden eine Lösung x_p mit $x_p^2 = a \bmod p$ und eine Lösung x_q mit $x_q^2 = a \bmod q$. Dann können wir eine Lösung x wie folgt konstruieren. Sei $bp + cq = 1$, dann ist $x = x_p \cdot c \cdot q + x_q \cdot b \cdot p$ eine Lösung von $x^2 = a \bmod N$:

$$\begin{aligned} x^2 \bmod N &= x_p^2 c^2 q^2 + x_q^2 b^2 p^2 \bmod N = ac^2q^2 + ab^2p^2 \bmod N \\ &= a(c^2q^2 + b^2p^2) \bmod N = a \bmod N. \end{aligned}$$

Für die Primzahlen p, q finden wir aber wie in Aufgabe 31 Lösungen der Gleichung $x^2 = a \bmod p$ reps. $x^2 = a \bmod q$. Sei $p = 3 \bmod 4$ und $\left(\frac{a}{p}\right) = 1$, dann gilt für $x = a^{(p+1)/4}$

$$x^2 = a^{(p+1)/2} = aa^{(p-1)/2} = a \bmod p.$$

Als `aribas`-Funktion kann dies z.B. wie folgt realisiert werden:

```
function rroot(y,p:integer): integer;
begin
if p mod 4= 3 then y:=y**((p+1) div 4) mod p;
return y;
else writeln("invalid p");
halt;
end;
end.
```

b und c mit $bp + cq = 1$ lässt sich in `aribas` mittels der Funktion `gcdx` berechnen:

```
gcdx(p,q,b,c)
```

```
b = 2111_61585_97154_12962_13248_97035_01833_66245_40827_26282,
c = -374_02435_94029_77741_05845_35777_24203_54742_56857_41399.
```

Nun können wir z_0 berechnen mittels

```
function calcz0(y:integer):integer;
external
p,q,b,c,N;
var
k:integer;
begin
for k:=1 to 1256 do;
y:=(rroot(y,p)*c*q+rroot(y,q)*b*p) mod N;
end;
return y;
end.
```

und

```
z_0 = 8_28921_93698_63653_93427_20737_24730_16747_96963_87365_04220_21936_17410_
18996_81416_31482_87801_43859_92653_62669
```

Der verschlüsselte Text

```
CC := 4FEB_C7FD_42DD_544E_BB70_CD8F_2F39_77C1_4145_F5E7_0DF9_9180_C1FA_FD74_38D9_
0FD7_217D_8D5C_09BE_4C5A_BD22_E7E4_ECAC_27CA_F543_79A0_F7A6_AC9D_245A_A0A0_
9793_2B0F_C09C_4AFE_328B_E398_CB8D_15CC_7981_BC85_1541_BECB_EA67_BD15_BB51_
6C84_57D2_6B38_52AF_8F10_D0BA_C3C1_1644_7A66_6402_CD84_D423_89BD_6D75_6510_
49F4_2401_63BA_0A7A_158C_9C5B_C198_562E_304A_AEF5_7800_D66B_B9AF_E40B_0358_
C9C4_3FC3_9DAB_E3
```

kann nun mittels

```
function go32(y:integer):byte_string;
external
N,CC;
var
k:integer;
bb: byte_string[157];
begin
```

```
bb := alloc(byte_string,157,0);
for k:=0 to 1256 do;
if y mod 2=1 then mem_bset(bb,k) end;
y:=y**2 mod N;
end;
return string(mem_xor(bb,CC));
end.
```

zu

```
The  x^2 mod N  pseudo-random generator was proposed by Leonore Blum, Michael\
 Blum and Michael Shub at the CRYPTO '82 Conference in Santa Barbara, Californ\
ia
```

aufgelöst werden.