Mathematisches Institut
der Universität München
Pascal Reisert

Wintersemester 2015/2016
Übungsblatt 6
02. Dezember 2015

Übungen zur Algorithmischen Zahlentheorie
Lösung

Aufgabe 21

a) Da $p \nmid a$ folgt $\gcd(a,p) = 1$ und natürlich auch $\gcd(2,p) = 1$. Sei $p = b(2a)^{-1}$ und $q = c - ap^2$. Dann gilt

$$ax^2 + bx + c = a(x+p)^2 + q.$$

Die Gleichung $ax^2 + bx + c = 0$ hat also eine Lösung, wenn $-q/a$ ein Quadrat modulo p ist. Das heißt, für das Legendre-Symbol muss gelten $\left(\frac{-q/a}{p}\right) = 1$. Anders ausgeschrieben bedeutet dies:

$$\left(\frac{b^2(2)^{-2}a^{-2} - ca^{-1}}{p}\right) = \left(\frac{b^2-4ac}{p}\right)\left(\frac{(4a^2)^{-1}}{p}\right) = \left(\frac{b^2-4ac}{p}\right),$$

wobei wir die Multiplikativität des Legendre-Symbols verwendet haben. Angenommen $\left(\frac{b^2-4ac}{p}\right) = -1$, dann hat die Gleichung keine, d.h.

$$1 + \left(\frac{b^2-4ac}{p}\right) = 0$$

Lösungen. Weiter wissen wir, dass im nullteilerfreien Ring $\mathbb{F}_p$ aus $u^2 = v^2 \Rightarrow (u-v)(u+v) = 0$ bereits $u = \pm v$ folgt. In Charakteristik $\neq 2$ gilt weiter $u \neq -u$ für $u \neq 0$. Damit finden wir falls $\left(\frac{b^2-4ac}{p}\right) = 1$ genau zwei y mit $y^2 = b^2 - 4ac \bmod p$ und entsprechend auch genau

$$2 = 1 + \left(\frac{b^2-4ac}{p}\right)$$

Lösungen von $ax^2 + bx + c = 0$:

$$(2a)^{-1}(-b \pm y).$$

Der Fall $\left(\frac{b^2-4ac}{p}\right) = 0$ gibt entsprechend eine Lösung $(2a)^{-1}(-b)$.

b) Es ist $13^2 - 4 \cdot 7 \cdot 71 \bmod p = 24 \bmod p$, $24 = 2 \cdot 2 \cdot 2 \cdot 3$ und somit gibt es eine Lösung, wenn

$$\left(\frac{2}{97}\right)^3\left(\frac{3}{97}\right) = \left(\frac{2}{97}\right)\left(\frac{3}{97}\right) = 1.$$

Da 3 offensichtlich quadratischer Rest ist ($10^2 \bmod 97 = 3 \bmod 97$) und ebenso 2 mod $97 = 14^2 - 2 \cdot 97 \bmod 97$ ist auch 24 quadratischer Rest. Wir erhalten $14^3 \cdot 10 \bmod 97 = 86 \bmod 97$ und $86^2 \bmod 97 = 24 \bmod 97$. Mit $7^{-1} \bmod 97 = 14 \bmod 97$ hat die Gleichung somit die Lösungen

$$(2a)^{-1}(-b \pm y) = (49 \cdot 14)(-13 \pm 86) \bmod 97 = 26 \vee 83 \bmod 97.$$

Aufgabe 22

a) Wir zeigen die Aussage allgemein für $q \nmid p-1$ und $a \equiv x^q \bmod p$. Man beachte zunächst, dass es auch für $a = 0$ eine offensichtliche Lösung gibt.
Sei aber $\gcd(a,p) = 1$ und b eine primitive Einheitswurzel mit $b^k \equiv a \bmod p$. Wegen $\gcd(q, p-1) = 1$ gibt es c, d mit $k = kc(p-1) + kdq$ und somit $a \equiv b^k \equiv b^{kc(p-1)+kdq} \equiv (b^{kd})^q$. Somit haben wir eine Lösung gefunden. Da es aber für jede Zahl $0 \leqslant a \leqslant p-1$ eine Lösung geben muss, ist diese bereits eindeutig.

b) Angenommen $a^{(p-1)/q} \equiv 1 \bmod p$, dann gibt es wie zuvor eine Einheitswurzel b und eine ganze Zahl k mit $b^k \equiv a \bmod p$. Es folgt $b^{k(p-1)/q} \equiv 1 \bmod p$ und somit $p-1 \mid k(p-1)/q \Rightarrow q \mid k$. Es folgt $(b^{k/q})^q = b^k \equiv a \bmod p$ ist eine Lösung. Umgekehrt erkennt man mit $x^q \equiv a \bmod p$ sofort, dass $a^{(p-1)/q} \equiv (x^q)^{(p-1)/q} \bmod p \equiv x^{p-1} \bmod p \equiv 1 \bmod p$. Angenommen es gibt $x \neq y$ mit $x^3 \equiv y^3$ und $\gcd(x,p) = \gcd(y,p) = 1$. Dann gilt $(xy^{-1})^3 \equiv 1 \bmod p$. Da es aber genau eine Untergruppe U der Ordnung 3 gibt folgt bereits $xy^{-1} \in U$ und somit gibt es genau drei verschiedene Lösungen.

Aufgabe 23

a) *Behauptung.* $x^2 \equiv a \bmod 2^k$ ist genau dann lösbar, wenn $a \equiv 1 \bmod 8$. In diesem Fall gibt es genau 4 Lösungen.

Wir zeigen die Aussage mittels Induktion über k.
$k = 3$. Wir erhalten wegen

$$1^2 \equiv 3^2 \equiv 5^2 \equiv 7^2 \equiv 1$$

und die einzigen 4 Lösungen für $a = 1$.

$k \to k+1$. Angenommen $x^2 \equiv a \bmod 2^{k+1}$, dann gilt $x^2 \equiv a \bmod 2^k$ und somit nach Induktionsvoraussetzung $a \equiv 1 \bmod 8$. Sei jetzt umgekehrt $a \equiv 1 \bmod 8$, dann gibt es ein x mit $x^2 = a \bmod 2^k \Leftrightarrow \exists r \in \mathbb{Z} : x^2 = a + 2^k r$. Ist r gerade so sind wir fertig. Ist r ungerade betrachte man $x' = (x + 2^{k-1})$ mit

$$(x + 2^{k-1})^2 = x^2 + 2^k x + 2^{2(k-1)} \equiv x^2 + 2^k x \bmod 2^{k+1}, \quad k \geqslant 3.$$

Schließlich beachte man, dass x ungerade sein muss, also $(x')^2 \equiv a \bmod 2^{k+1}$. Die Anzahl lässt sich aus diesem Vorgehen ebenfalls ablesen, denn genau eine der beiden Lösung $x^2 \equiv (x + 2^{k-1})^2 \equiv a \bmod 2^k$ ist auch Lösung von $x^2 \equiv a \bmod 2^{k+1}$. Zu jeder solchen Lösung y ist aber auch $y + 2^k$ eine Lösung der Gleichung modulo 2^{k+1}. Insgesamt bleibt die Anzahl der Lösungen also immer bei 4. Insbesondere sind dies alle Lösungen, da sich die Anzahl der a mit $a \equiv 1 \bmod 8$ mit jedem Schritt verdoppelt.

b) siehe a).

c) Man erhält die Lösungen $111, 401, 623, 913$ wie folgt:

$$\begin{gathered} 33 \bmod 8 = 1 \text{ und } x \in \{1,3,5,7\}; \\ 33 = 1 \bmod 16 \text{ und } x \in \{1,7,1+8,7+8\}; \\ 33 = 1 \bmod 32 \text{ und } x \in \{1,15,17,31\}; \\ 33 = 33 \bmod 64 \text{ und } x \in \{15,17,47,49\}; \\ 33 = 33 \bmod 128 \text{ und } x \in \{17,47,81,111\}; \end{gathered}$$

$$33 = 33 \bmod 256 \text{ und } x \in \{17, 111, 145, 239\};$$
$$33 = 33 \bmod 512 \text{ und } x \in \{111, 145, 367, 401\};$$
$$33 = 33 \bmod 1024 \text{ und } x \in \{111, 401, 623, 913\}.$$

Aufgabe 24

a) Es genügt Injektivität zu zeigen. Allerdings folgt aus $F_K(x) = F_K(y) \Rightarrow (x-y)(2(x+y)+K) \equiv 0 \bmod 2^r$ und $2(x+y)+K$ ungerade, also teilerfremd zu 2^r. Somit ist $(x-y) \equiv 0 \bmod 2^r$.

b) Um den verschlüsselten Text auflösen zu können müssen wir ein Umkehrfunktion zu F_K definieren. Man beachte

$$F_K^r(2^{r-1}+x) = 2^{r-1}K + 2x^2 + Kx = 2^{r-1} + 2x^2 + Kx = F_K^r(x) + 2^{r-1}.$$

Es genügt also das Inverse der ersten 2^{r-1} Zahlen zu bestimmen. Sei $\operatorname{div} m$ die Funktion die einer ganzen Zahl r die größte ganze Zahl s zuordnet mit $sm \leqslant r$. Sei G_K^{r-1} die Umkehrfunktion zu F_K^{r-1}, dann erhalten wir G_K^r wie folgt

$$G_K^r(y) := G_K^{r-1}(y \bmod 2^{r-1}) + \begin{cases} 0 & \text{wenn } (F_K^r(G_K^{r-1}(y \bmod 2^{r-1})) - y) \operatorname{div} 2^{r-1} \text{ gerade,} \\ 2^{r-1} & \text{sonst.} \end{cases}$$

Um dies nachzuprüfen beachte man, dass $F_K^r(x) \bmod 2^{r-1} = F_K^{r-1}(x)$ und $z \bmod 2^{r-1} = z \bmod 2^r$ genau dann, wenn $z \operatorname{div} 2^{r-1}$ gerade ist, $z \in \mathbb{Z}$. D.h. aus $(F_K^r(G_K^{r-1}(y \bmod 2^{r-1})) - y) \operatorname{div} 2^{r-1}$ gerade folgt

$$F_K^r(G_K^{r-1}(y \bmod 2^{r-1})) - y \bmod 2^{r-1} = F_K^r(G_K^{r-1}(y \bmod 2^{r-1})) - y \bmod 2^r$$

wobei die linke Seite nach Definition 0 ist. Es bleibt der Fall $(F_K^r(G_K^{r-1}(y \bmod 2^{r-1})) - y) \operatorname{div} 2^{r-1}$ ungerade. Hier gilt

$$\begin{aligned} 0 &= F_K^r(G_K^{r-1}(y \bmod 2^{r-1})) - y \bmod 2^{r-1} = F_K^r(G_K^{r-1}(y \bmod 2^{r-1})) - y - 2^{r-1} \bmod 2^r \\ &= F_K^r(G_K^{r-1}(y \bmod 2^{r-1}) + 2^{r-1}) - y \bmod 2^r. \end{aligned}$$

Nun lässt sich die Umkehrfunktion rekursiv aus dem Fall $r = 1$ mit $G_K^1(0) = 0$ und $G_K^1(1) = 1$ berechnen. Um den passenden Schlüssel zu finden, kann man den entschlüsselten Text für alle $k \in \{1, \ldots, 2^{16}-1\}$ in einen möglichen Klartext entkodieren lassen und das Ergebnis auf Konsistenz überprüfen. Um nicht alle 2^{16} Ergebnisse manuell durchsehen zu müssen, bietet es sich an den entschlüsselten Text auf die Existenz von Textzeichen zu überprüfen. In ASCII ist bei Textzeichen das achte Bit nicht gesetzt. Eine Lösung sollte also nur sehr wenige Zeichen `bit_xor( ,0x8)`$\neq$`0` besitzen. Wir erhalten:

```
Federal Information Processing Standards Publication 81 (FIPS #81) of
December 2, 1980, specified the ECB, CBC, CFB and OFB modes of operation
for DES
```

Der folgende Code von Prof. Forster liefert die `aribas` die gesuchte Lösung:

```
(*****************************************************************)
(*
```

```
** Vorlesung Algorithmische Zahlentheorie und Kryptographie
** von Otto Forster im WS 2015/16 am Math. Inst. der LMU Muenchen
**
** Aribas Code zur Loesung der Aufgabe 20b)
** File author: Otto Forster,  Email:  forster AT math.lmu.de
** Date of last change: 2015-11-26
**
** Nach dem Laden dieser Datei in Aribas wird die
** Loesung berechnet mit dem Befehl
**
** ==> go20().
*)
(*****************************************************************)
(*
** cipher text was generated by ecrypting an English Ascii plaintext
** by 16-bit block cipher  x2x(16,x,K) in CBC mode
** with initial vector  AFFE
*)
CC := $AFFE_9350_1105_CD29_6BDC_BAE2_DC81_26BC_A10C_88C4_DE3C_06E6_13FC_6E67_
D757_23C7_ED11_83C9_6716_FEBF_3277_AAAC_191B_4F91_AC1F_1838_91B7_3B09_A232_
2286_F54A_9E6A_F729_BE84_16DD_F0F6_7417_B19D_DCE3_23D4_ED4B_0B18_D261_8265_
2615_AF7C_627D_15C0_7C9C_588F_AC5E_F321_AE02_6C88_CCA9_0682_C291_CB20_B300_
6AB0_86E7_F308_3BCE_4E1D_C207_3BC1_140A_F40C_C4EA_AE4B_7279_D7FC_119E_F66D_
CE75_9501;
(*-------------------------------------------------*)
(*
** function  x -->  x*(2*x + K) mod 2**n
*)
function x2x(n,x,K: integer): integer;
begin
    return x*(2*x+K) mod 2**n;
end;
(*-------------------------------------------------*)
(*
** Inverse function of x2x
** K must be odd
*)
function x2xinv(n,y,K: integer): integer;
var
    x,delta,k: integer;
begin
    if even(K) then inc(K); end;
    x := y mod 2;
    for k := 1 to n-1 do
        delta := (2*x**2 + K*x - y) div 2**k;
        delta := delta mod 2;
        x := x + delta*2**k;
    end;
    return x;
```

```
end;
(*-----------------------------------------------------------*)
(*
** encoding of a block of length 2 bytes
** unsing the functon x2x(n,x,K) with n=16
*)
function encod(bb: byte_string[2]; K: integer): byte_string;
var
    x,y: integer;
    cc: byte_string[2];
begin
    x := bb[0] + 2**8*bb[1];
    y := x2x(16,x,K);
    cc[0] := y mod 2**8;
    cc[1] := y div 2**8;
    return cc;
end;
(*-----------------------------------------------------------*)
(*
** inverse function of encod
*)
function decod(cc: byte_string[2]; K: integer): byte_string;
var
    x,y: integer;
    bb: byte_string[2];
begin
    y := cc[0] + 2**8*cc[1];
    x := x2xinv(16,y,K);
    bb[0] := x mod 2**8;
    bb[1] := x div 2**8;
    return bb;
end;
(*-----------------------------------------------------------*)
(*
** encrypts a plaintext  bb  (length even)
** in CBC mode with initial vector  iv
*)
function enccbc(iv,bb: byte_string; K: integer): byte_string;
var
    k, len: integer;
    tt, zz, CC: byte_string;
begin
    len := length(bb);
    if odd(len) then
dec(len);
    end;
    CC := alloc(byte_string,len+2);
    tt := iv;
    CC[0..1] := tt;
```

```
    for k := 0 to (len div 2)-1 do
        zz := mem_xor(tt,bb[2*k..2*k+1]);
        tt := encod(zz,K);
        CC[2*k+2..2*k+3] := tt;
    end;
    return CC;
end;
(*-----------------------------------------------------------*)
(*
** inverse function of enccbc
*)
function deccbc(CC: byte_string; K: integer): byte_string;
var
    k, len: integer;
    BB, tt, xx, yy: byte_string;
begin
    len := length(CC);
    BB := alloc(byte_string,len-2);
    tt := CC[0..1];
    for k := 1 to (len div 2)-1 do
        yy := CC[2*k..2*k+1];
        xx := decod(yy,K);
        BB[2*k-2..2*k-1] := mem_xor(xx,tt);
        tt := yy;
    end;
    return BB;
end;
(*-----------------------------------------------------------*)
(*
** counts the number of bytes in  bb  whose highest
** bit (value=128) is set.
** (printable characters are usually represented
** by bytes with value < 128)
*)
function highbitcount(bb: byte_string): integer;
var
    k,len,count,x: integer;
begin
    count := 0;
    len := length(bb);
    for k := 0 to len-1 do
        x := bb[k];
        if bit_and(x,0x80) then inc(count); end;
    end;
    return count;
end;
(*-----------------------------------------------------------*)
function go20()
external
```

```
    CC: byte_string;
var
    K: integer;
    bb: byte_string;
begin
    for K := 1 to 2**16 by 2 do
        if bit_and(K,0xFF) = 1 then
            write('.');
        end;
        bb := deccbc(CC,K);
        if highbitcount(bb) <= 5 then
            writeln();
            writeln("K = ",K);
            break;
        end;
    end;
    return(string(bb));
end;
```