Poročilo 2. domače naloge pri TKK 2025/26

Anton Luka Šijanec, 63230317

2. maj 2026

Povzetek

Rešitev 2. domače naloge pri predmetu Teorija kodiranja in kriptografija v 3. letniku IŠRM 2025/26.

Kazalo

1 Izračun šifriranih podatkov

S spodaj pripetim programom sem izračunal $m_i^3 \% n$ in dobil rezultat

$$64466811878321639095616630852269512425534540964724043 4.$$

2 Padding z zaporednim nonceom

Na roke[1] razpišem m_i^3 na člene. Upoštevam namig in razpišem $d_i \coloneqq c_i - c_{i-1}$ in pokrajšam, kolikor se da. Še vedno dobim člene, večje od n, čeprav se je številka s tem občutno zmanjšala. Sporočil m_{i-2} in m_{i-3} še nisem uporabil, zato razpišem še $e_i \coloneqq d_i - d_{i-1}$ in res dobim še manjšo številko, natančneje

$$e_i = 3u\left(A\right) \cdot 2^{120} \cdot 13^2 \cdot 2 + 3 \cdot 2i \cdot 13^3 - 2 \cdot 3 \cdot 13^3 + 3 \cdot 2 \cdot 13^2 \cdot 17.$$

Razpišem še $e_i - e_{i-1}$ in dobim

$$e_i - e_{i-1} = c_i - 3 \cdot c_{i-1} + 3 \cdot c_{i-2} - c_{i-3} = 3 \cdot 2 \cdot 13^3,$$

kar potrdi moje izračune, kajti $c_i - 3 \cdot c_{i-1} + 3 \cdot c_{i-2} - c_{i-3} \equiv 3 \cdot 2 \cdot 13^3 \mod n$ res velja, ko preverim s programom.

Sedaj je treba iz e_i izluščiti $u\left(A\right)$. e_i je žal še vedno večji od n, toda tega se lahko rešimo, kajti $\frac{e_i}{13^2}$ je že manjše od n. Multiplikativni inverz 13^2 v $\mathbb{Z}_n$ znamo izračunati hitro, označimo ga z $\left(13^2\right)^{-1}$. Velja torej

$$e_i \cdot \left(13^2\right)^{-1} \equiv 3u\left(A\right) \cdot 2^{120} \cdot 2 + 3 \cdot 2 \cdot i - 2 \cdot 3 \cdot 13 + 3 \cdot 2 \cdot 13^2 \cdot 17,$$

Torej

$$u\left(A\right) = \left\lfloor \frac{e_i \cdot \left(13^2\right)^{-1}}{2^{120} \cdot 2 \cdot 3} \right\rfloor.$$

In res, s spodaj pripetim programom dobimo $u\left(A\right) = 9876543210123456$.

[1] slike izpeljave so na koncu dokumenta

3 Padding z naključnim, a majhnim nonceom

Ker je r majhen, to diši po bruteforcu. Poskusimo vse $r-$je. Pretvarjajmo se, da r poznamo, torej poznamo $s \cdot 2^{100} + r \cdot 2^{40} =: B$. Velja $c_B = (B+k)^3 = B^3 + 3B^2k + 3Bk^2 + k^3$, kjer je k iskani skrivni ključ. Imamo polinom tretje stopnje v k. Iskani k je ničla polinoma

$$k^3 + 3Bk^2 + 3B^2k^1 + \left(B^3 - c_B\right) k^0$$

in je tokrat za razliko od prejšnjih dveh nalog priročno kratek (le 40−bitov) napram 180−bitnemu n. Uporabimo lahko Coppersmithovo metodo za iskanje malih ničel polinoma v $\mathbb{Z}_n$, saj je $k < n^{1/e} = \sqrt[3]{n}$, kot svetuje primer v sage-mathovi dokumentaciji in kot opisujejo 5. vaje pri predmetu KIRV na FRI.

Poskusimo torej vse možne $r-$je in za vsakega uporabimo Coppersmithovo metodo (small_roots v knjižnici sagemath) na polinomu $(B+k)^3$. Kandidate za ključ shranimo v tabelo in nato ponovimo isto še na drugem prestreženem kriptogramu, da potrdimo najdeno rešitev. Presek (in hkrati unija) iskanja na obeh kriptogramih je

$$\{867530912345\},$$

to je torej iskani ključ. Kot bonus dobimo s to metodo tudi vrednosti r; pri prvem kriptogramu je ta vrednost 845678 in pri drugem 623456.

Program za izračun je tekel devet minut in 17 sekund na mojem prenosniku z dvanajstimi procesorskimi jedri.

Dalo bi se najti še hitreje; empirično preizkušeno bi s Coppersmithovo metodo lahko iskali še do 4 bite $r-$ja, kar bi pohitrilo iskanje 16−krat; uganiti bi morali le 16 bitov $r-$ja[2]. Če ga prilagodimo (spremenljivko ugani_bite_r nastavimo na 4), program po le 23 sekundah izpljune edinega kandidata

$$\{867530912345\}$$

in pravilni vrednosti r.

Če s Coppersmithovo metodo iščemo 5 bitov r, pa nam metoda najde pravilno ničlo 867530912345 le v prvem primeru (s_1, c_1), ne pa tudi v drugem; takrat ne najde nobene. Resda program takrat teče samo deset sekund in sicer najde pravilen odgovor, vendar ga ne potrdi še vdrugo.

4 Priloga

Program za prvo in drugo nalogo:

```
#!/usr/bin/python3
print("----------zaÄetek 1. naloge----------")
uB = 34100010166843172
n = 135304092231989671072994844074211352614066206912423757 1
e = 3
i = 1909
m = lambda i : uB*2**120+13*i+17
c = lambda i : pow(m(i), e, n)
print(f"1. {c(i)}")
print("----------konec 1. naloge----------")
print("----------zaÄetek 2. naloge----------")
print(f"    {hex(n)=}")
print(f"{hex(m(i)**e)=}")
print(f"{hex((13*i+17)**3)=}")
print(f"{hex(3*uB*2**120*(13*i+17)**2)=}")
print(f"{hex(3*uB**2*2**240*(13*i+17))=}")
print(f"{hex(uB**3*2**360)=}")
print(f"{hex(c(i))=}")
print(3*uB**2*2**240*13)
print(n)
print("----------")
```

[2]To ne bi bilo možno, če bi bil koeficient pred $r-$jem večji, denimo 2^{50} namesto 2^{40}.

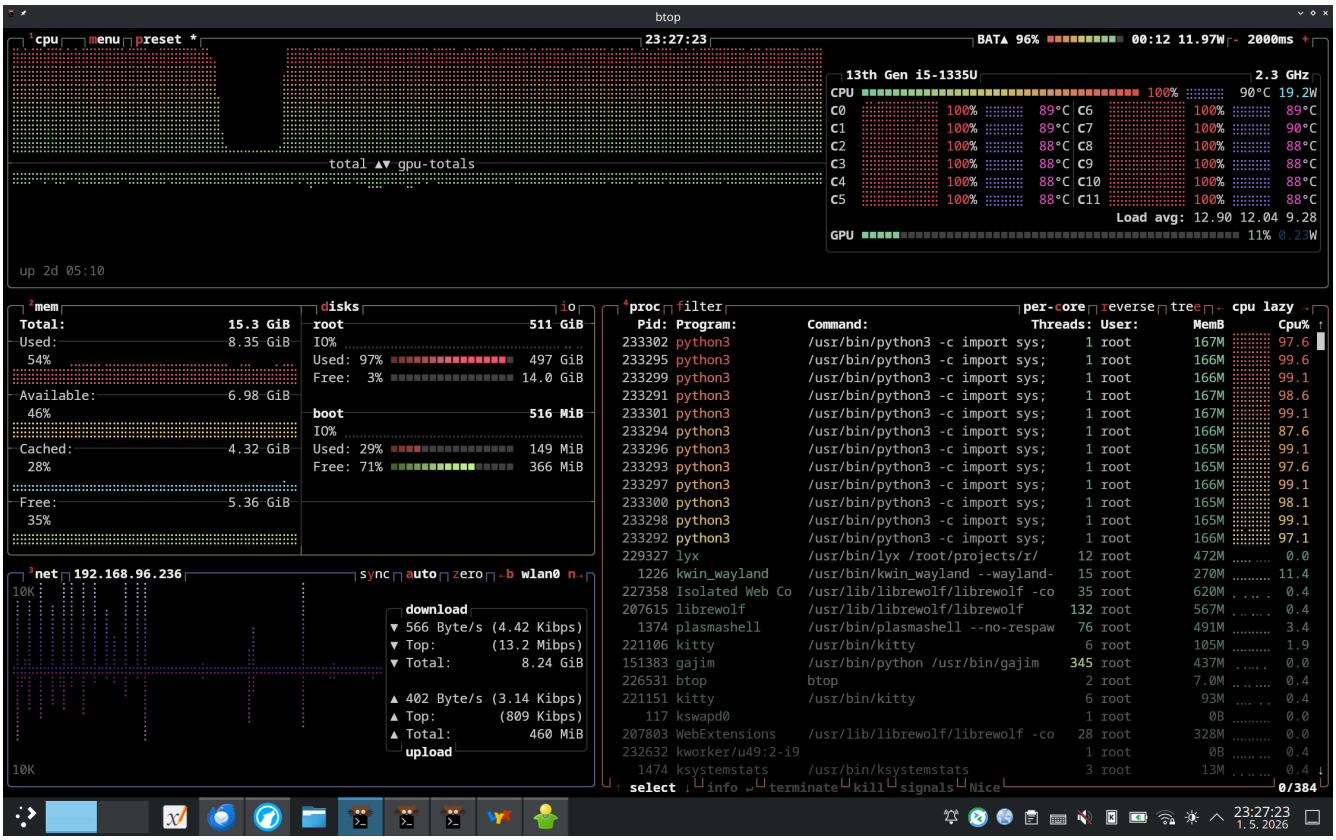
```
print((3*uB*2**120*2+3*2*i*13-2*3*13+3*2*17))
print(n)
d = lambda i : c(i)-c(i-1)
ee = lambda i : d(i)-d(i-1)
print(f"{(ee(i)-ee(i-1))=}")
print(f"{(3*2*13**3)=}")
assert ee(i)-ee(i-1) == 3*2*13**3
print(True)
inverz_13__2 = pow(13**2, -1, n)
print(f"{inverz_13__2=}")
zlato = (ee(i)*inverz_13__2)%n
print(f"{hex(zlato)=}")
print(f"{hex(3*uB*2**120*2+3*2*i*13-2*3*13+3*2*17)=}")
assert (zlato >> 120)//3//2 == uB
print(True) # uspeĹĄno smo naĹĄli iz c(i)-jev spet uB
# print sedaj uporabimo podatke iz naloge namesto c(i)
c = lambda i : [491900335802125197979949495516790900284557545229938837,
                233058220177869423059510212747217854347711320372298942,
                110882002846072119715603331816287294065382504194781193 4,
                413103916010887098809621930279529106921574571708015853][i]
d = lambda i : c(i)-c(i-1)
ee = lambda i : d(i)-d(i-1)
zlato = (ee(3)*inverz_13__2)%n
uA = (zlato >> 120)//3//2
print(uA)
print("----------konec 2. naloge----------")
print("----------zaÄetek 3. naloge----------")
import math
from sympy.ntheory import factorint
s1 = 927461238746123987461239
s2 = 618273645918273645918273
c1 = 13428393004501700296112741245334633027395740325761875 46
c2 = 761018752843619041400692865320021546042962812592957443
print(f"{math.log(n, 2)=}")
print(f"{math.log(s1, 2)=}")
print(f"{math.log(s2, 2)=}")
print(f"{factorint(s1)=}")
print(f"{factorint(s2)=}")
# skupni faktor je torej 3
d1 = (c1 - s1**3*2**300)%n
d2 = (c2 - s2**3*2**300)%n
print(f"{math.gcd(c1, c2)=}")
print(f"{math.gcd(d1, d2)=}")
print(f"{s1-s2=}")
```

Program za tretjo nalogo:

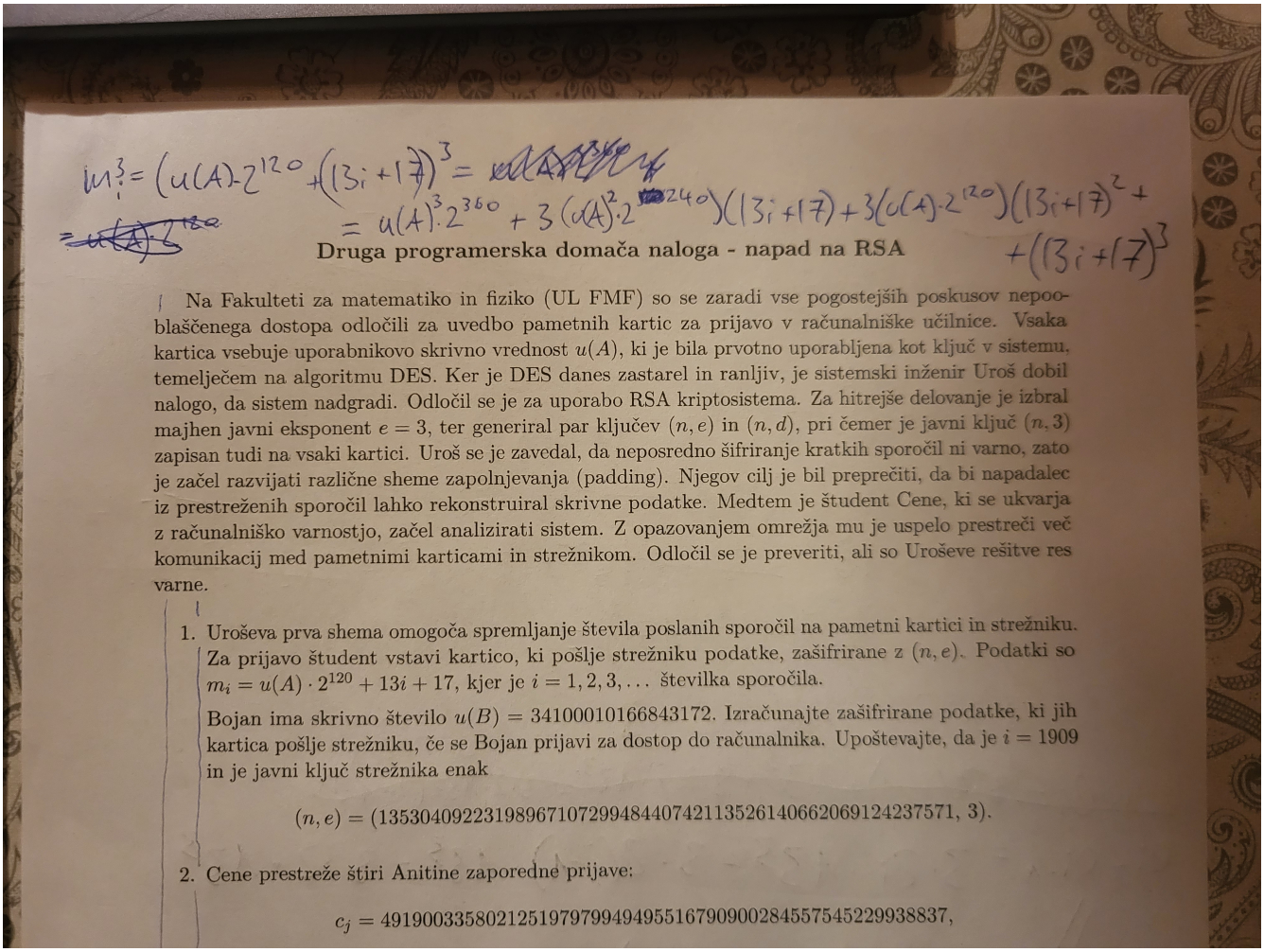
```
#!/usr/bin/python3
ugani_bite_r = 0
n = 135304092231989671072994844074211352614066206912423757 1
e = 3
s1 = 927461238746123987461239
s2 = 618273645918273645918273
c1 = 13428393004501700296112741245334633027395740325761875 46
c2 = 761018752843619041400692865320021546042962812592957443
# pretvarjajmo se, da poznamo k in r
# k = 34100010166843172
# realk = k
```

```
# r = 1
# B = s1*2**100 + r*2**40
# c1 = ((B+k)**e)%n
# print(c1)
# print(((B+k)**e-c1)%n)
# https://doc.sagemath.org/html/en/reference/polynomial_rings/sage/rings/polynomial/polynomia
from sage.all import Zmod, PolynomialRing
import multiprocessing as mp
import sys
workers = 1
if len(sys.argv) == 2:
        workers = int(sys.argv[1])
def crack (arg):
        (worker, cc, ss) = arg
        ZmodN = Zmod(n)
        P = PolynomialRing(ZmodN, implementation='NTL', names=('k',)); 
        (k,) = P._first_ngens(1) # da k postane spremenljivka/ime
        oldpct = -1
        chunksize = 2**(20-ugani_bite_r)//workers
        myworkstart = chunksize*worker
        myworkend = chunksize*(worker+1)
        if worker == workers-1:
                myworkend == 2**(20-ugani_bite_r)
        candidates = []
        for r in range(myworkstart, myworkend):
                curpct = int(100*(r-myworkstart)/chunksize)
                if worker == 0 and curpct != oldpct:
                        oldpct = curpct
                        print("\r%d%%" % oldpct, file=sys.stderr, end="")
                B = ss*2**100 + r*2**(40+ugani_bite_r)
                f = (B+k)**e-cc
                for sr in f.small_roots(2**(40+ugani_bite_r)):
                        if sr > 2**60: # small_roots obÄasno vrne tudi veliko
                                continue # ĹĄtevilo, pove dokumentacija
                        sr = int(sr)
                        rr = (r<<ugani_bite_r) + (sr>>40)
                        kk = sr & 0xffffffffff
                        candidates.append((kk, rr))
        if worker == 0:
                print("\n", file=sys.stderr)
        return candidates
if __name__ == "__main__":
        c = c1
        s = s1
        pool = mp.Pool(workers)
        responses = pool.map(crack, list(zip(range(workers),
                                        (c1 for _ in range(workers)),
                                        (s1 for _ in range(workers)))))
        candidates1 = []
        for response in responses:
                candidates1 += response
        print("candidates for first message: " + str(candidates1))
        c = c2
        s = s2
        responses = pool.map(crack, list(zip(range(workers),
                                        (c2 for _ in range(workers)),
```

```
                                        (s2 for _ in range(workers)))))
candidates2 = []
for response in responses:
        candidates2 += response
print("candidates for second message: " + str(candidates2))
print("intersection: " + str(list(set(list(zip(*candidates1))[0])
                                  & set(list(zip(*candidates2))[0]))))
```



Slika 1: Obremenjenost računalnika med iskanjem r (brez optimizacije; iskanje vseh 20 bitov z grobo silo).



$m_i^3 = (u(A)\cdot 2^{120} + (13i+17))^3 =$

$= u(A)^3 \cdot 2^{360} + 3(u(A)^2 \cdot 2^{240})(13i+17) + 3(u(A)\cdot 2^{120})(13i+17)^2 +$

$+ (13i+17)^3$

Druga programerska domača naloga - napad na RSA

Na Fakulteti za matematiko in fiziko (UL FMF) so se zaradi vse pogostejših poskusov nepooblaščenega dostopa odločili za uvedbo pametnih kartic za prijavo v računalniške učilnice. Vsaka kartica vsebuje uporabnikovo skrivno vrednost $u(A)$, ki je bila prvotno uporabljena kot ključ v sistemu, temelječem na algoritmu DES. Ker je DES danes zastarel in ranljiv, je sistemski inženir Uroš dobil nalogo, da sistem nadgradi. Odločil se je za uporabo RSA kriptosistema. Za hitrejše delovanje je izbral majhen javni eksponent $e = 3$, ter generiral par ključev (n, e) in (n, d), pri čemer je javni ključ $(n, 3)$ zapisan tudi na vsaki kartici. Uroš se je zavedal, da neposredno šifriranje kratkih sporočil ni varno, zato je začel razvijati različne sheme zapolnjevanja (padding). Njegov cilj je bil preprečiti, da bi napadalec iz prestreženih sporočil lahko rekonstruiral skrivne podatke. Medtem je študent Cene, ki se ukvarja z računalniško varnostjo, začel analizirati sistem. Z opazovanjem omrežja mu je uspelo prestreči več komunikacij med pametnimi karticami in strežnikom. Odločil se je preveriti, ali so Uroševe rešitve res varne.

1. Uroševa prva shema omogoča spremljanje števila poslanih sporočil na pametni kartici in strežniku. Za prijavo študent vstavi kartico, ki pošlje strežniku podatke, zašifrirane z (n, e). Podatki so $m_i = u(A) \cdot 2^{120} + 13i + 17$, kjer je $i = 1, 2, 3, \ldots$ številka sporočila.

   Bojan ima skrivno število $u(B) = 34100010166843172$. Izračunajte zašifrirane podatke, ki jih kartica pošlje strežniku, če se Bojan prijavi za dostop do računalnika. Upoštevajte, da je $i = 1909$ in je javni ključ strežnika enak

   $$(n, e) = (135304092231989671072994844074211352614066206912423757 1, 3).$$

2. Cene prestreže štiri Anitine zaporedne prijave:

   $$c_j = 491900335802125197979949495516790900284557545229938837,$$

Slika 2: Izpeljava pri drugi nalogi 1/3

$= u(A)^3 \cdot 2^{360} + 3\,u(A)^2 \cdot 2^{240} \cdot 13i + 3 \cdot 17 \cdot u(A)^2 \cdot 2^{240} +$ ~~$3u(A)2^{120}13i$~~

$+ 3u(A) \cdot 2^{120}(13^2 i^2 + 13i \cdot 17 + 17^2) + 13^3 i^3 + 3 \cdot 13^2 i^2 \cdot 17 + 3 \cdot 13 i \cdot 17^2 + 17$

$= u(A)^3 \cdot 2^{360} + 3\,u(A)^2 \cdot 2^{240} \cdot 13i + 3 \cdot 17 u(A)^2 \cdot 2^{240} + 3 \cdot u(A) \cdot 2^{120} \cdot 13^2 \cdot i^2 +$

$+ 3u(A) \cdot 2^{120} \cdot 13i \cdot 17 + 3u(A) \cdot 2^{120} \cdot 17^2 + 13^3 i^3 + 3 \cdot 13^2 i^2 \cdot 17 + 3 \cdot 13 i \cdot 17^2$

$+ 17^3$

→ izpostavi i, poglej, za koliko se spremeni c ob spremembi i za 1.

$m_i = u(A)^3 \cdot 2^{360} + 3u(A)^2 \cdot 2^{240} \cdot 13i + 3 \cdot 17 u(A)^2 2^{240} +$

$+ 3u(A) 2^{120} \cdot 13^2 \cdot i^2 + 3u(A) \cdot 2^{120} \cdot 13i \cdot 17 +$

$+ 3 \cdot u(A) \cdot 2^{120} \cdot 17^2 + 13^3 i^3 + 3 \cdot 13^2 i^2 \cdot 17 + 3 \cdot 13 i \cdot 17^2 +$

$+ 17^3.$

$m_i - m_{i-1} = $ ~~$3u(A)^2 \cdot 2^{240} 13i + 3u(A) \cdot 2^{120} 13^2 \cdot i^2 +$~~

$\|$ ~~$3u(A) 2^{120} \cdot 13i \cdot 17 + 13^3 3 + 3 \cdot 13^2 \cdot 2 \cdot 17 +$~~

d_i ~~$+ 3 \cdot 13i \cdot 17^2$~~ $-$ (

$3u(A)^2 \cdot 2^{240} \cdot 13(i-1) + 3u(A) \cdot 2^{120} \cdot 13^2 \cdot (i-1)^2 +$

$+ 3u(A) \cdot 2^{120} \cdot 13(i-1) \cdot 17 + 13^3 (i-1)^3 +$

$+ 3 \cdot 13^2 (i-1)^2 \cdot 17 + 3 \cdot 13 (i-1) \cdot 17^2 =$

~~$3u(A)^2 \cdot 2^{240} \cdot 13i$~~ $- 3u(A)^2 \cdot 2^{240} \cdot 13 +$ ~~$3u(A) \cdot 2^{120} \cdot 13^2 \cdot i^2$~~ $-$

$- 3u(A) \cdot 2^{120} \cdot 13^2 \cdot 2i + 3u(A) \cdot 2^{120} \cdot 13^2 +$

$+$ ~~$17 \cdot 3u(A) 2^{120} \cdot 13i$~~ $- 3u(A) \cdot 2^{120} \cdot 13 \cdot 17 +$ ~~$13^3 3$~~ $-$

$- 3i^2 \cdot 13^3 + 3i \cdot 13^3 - 1 \cdot 13^3 +$ ~~$3 \cdot 13^2 \cdot 17 \cdot i^2$~~ $- 3 \cdot 2 \cdot 13^2 \cdot 17 \cdot i -$

$+ 3 \cdot 13^2 17 +$ ~~$3 \cdot 13 \cdot 17^2 i$~~ $- 3 \cdot 13 \cdot 17^2$) $=$

$= 3u(A)^2 \cdot 2^{240} \cdot 13 + 3u(A) \cdot 2^{120} \cdot 13^2 \cdot 2i - 3u(A) \cdot 2^{120} \cdot (13^2$

$+ 3u(A) 2^{120} \cdot 13 \cdot 17 + 3i^2 13^3 - 3i 13^3 + 13^3 + 3 \cdot 2 \cdot 13^2 \cdot 17 \cdot$

$- 3 \cdot 13^2 17 + 3 \cdot 13 \cdot 17^2$

$d_i - d_{i-1} = (u_i - u_{i-1}) - (u_{i-1} - u_{i-2}) =$

$= 3u(A) \cdot 2^{120} \cdot 13^2 \cdot 2i + 3 \cdot 2 \cdot 13^3 - 3 \cdot 13^3 + 3 \cdot 2 \cdot 13^2 \cdot 17 -$

$- (3u(A) \cdot 2^{120} \cdot 13^2 \cdot 2(i-1) + 3(i-1)^2 \cdot 13^3 - 3(i-1) 13^3 +$

$+ 3 \cdot 2 \cdot 13^2 \cdot 17 \cdot (i-1) =$

$= 3u(A) 2^{120} \cdot 13^2 \cdot 2i - 3u(A) \cdot 2^{120} \cdot 13^2 \cdot 2 + 3 \cdot 2^2 \cdot 13^3 - 3 \cdot 2i \cdot 13^3 +$

$+ 3 \cdot 13^3 - 3 \cdot 2^2 \cdot 13^3 + 3 \cdot 13^3 + 3 \cdot 2 \cdot 13^2 \cdot 17 -$

$- 3 \cdot 2 \cdot 13^2 \cdot 17) =$

$= + 3u(A) \cdot 2^{120} \cdot 13^2 \cdot 2 + 3 \cdot 2i \cdot 13^3 - 2 \cdot 3 \cdot 13^3 + 3 \cdot 2 \cdot 13^2 \cdot 17$

$e_{i-1} - e_i = 3 \cdot 2(i-1) \cdot 13^3 - 3 \cdot 2i \cdot 13^3 = 3 \cdot 2 \cdot 13^3 (i-1-i)$

$e_i - e_{i-1} = 3 \cdot 2i \cdot 13^3 - 3 \cdot 2(i-1) \cdot 13^3 = 3 \cdot 2 \cdot 13^3$

$(d_i - d_{i-1}) - (d_{i-1} - d_{i-2}) = ((u_i - u_{i-1}) - (u_{i-1} - u_{i-2})) -$

$- ((u_{i-1} - u_{i-2}) - (u_{i-2} - u_{i-3})) =$

$= (u_i - u_{i-1}) - (u_{i-1} - u_{i-2}) - (u_{i-1} - u_{i-2}) + (u_{i-2} - u_{i-3})$

$= u_i - u_{i-1} - u_{i-1} + u_{i-2} - u_{i-1} + u_{i-2} + u_{i-2} - u_{i-3} =$

$= u_i - 3 \cdot u_{i-1} + 3u_{i-2} - u_{i-3}$

$e_i / 13^2 = 3u(A) \cdot 2^{120} \cdot 2 + 3 \cdot 2i \cdot 13 - 2 \cdot 3 \cdot 13 + 3 \cdot 2 \cdot 17$

$e_i + 2 \cdot 3 \cdot 13^3 - 3 \cdot 2 \cdot 13^2 \cdot 17 = 3u(A) \cdot 2^{120} \cdot 13^2 \cdot 2 + 3 \cdot 2i \cdot$

$(13^2, -1, n)$