Domaći zadatak iz Osnova telekomunikacija

Dešifrovanje poruka šifrovanih RSA algoritmom

Prvi zadatak

U ovom zadatku je potrebno pronaći privatni ključ iz datog javnog ključa $(n\colon 3233, e\colon 101)$. Imamo da je privatni ključ dat sa (n, d). Takođe za e i d važi sledeća relacija:

$$ed \equiv 1 \left(mod\ \varphi(n)\right),$$

gde je sa φ označena *Euler*-ova funkcija. To znači da moramo prvo izračunati $\varphi(n)$, a zatim pronaći multiplikativni inverz elementa e, što se može uraditi proširenim *Euclid*-ovim algoritmom. Izračunavanje $\varphi(n)$ je teško u opštem slučaju, ali postaje lako ukoliko znamo proste faktore broja n jer je funkcija φ multiplikativna, odnosno za uzajamno proste brojeve p i q ($gcd(p, q) = 1$) važi: $\varphi(pq) = \varphi(p)\varphi(q)$. Primetimo da za svaki prost broj p, važi $\varphi(p) = p - 1$, jer $(\forall k)(k > 0 \wedge k < p) \gcd(k, p) = 1$. To znači da se problem nalaženja $\varphi(n)$ svodi na pronalaženje prostih faktora p i q takvih da važi $pq = n$. Tada je:

$$\varphi(n) = \varphi(pq) = \varphi(p)\varphi(q) = (p - 1)(q - 1).$$

Za ovaj problem do današnjeg dana nije pronađen efikasan algoritam i upravo tu leži (računarska) sigurnost RSA algoritma. S obzirom da znamo da se n sastoji od 2 prosta faktora, proveravanjem redom brojeva od 2 do $\sqrt{n}$ ćemo sigurno pronaći manji faktor, nazovimo ga p, a zatim q dobijamo sa n/p. Nakon toga je $\varphi(n) = (p - 1)(q - 1)$ i d pronalazimo koristeći prošireni *Euclid*-ov algoritam koji za ulaz e i $\varphi(n)$ vraća cele brojeve x i y, koji uvek postoje i za koje važi $xe + y\,\varphi(n) = \gcd\left(e, \varphi(n)\right) = 1$. Iz ovoga vidimo da je $d = x\ mod\ \varphi(n)$ kao i da e treba biti tako izabrano da je $\gcd\left(e, \varphi(n)\right) = 1$. Bitno je napomenuti da je *Euclid*-ov algoritam veoma efikasan. Za ulaz (a, b) vremenska složenost je $O(\log(\min(a, b)))$, odnosno meri se brojem cifara manjeg od ova dva broja, a prostorna složenost je $O(1)$. Za $n = 3233$ ovakvo rešenje, napisano u *python*-u (koji je dosta sporiji od C/C++) pronalazi privatni ključ u proseku za $3 \cdot 10^{-5}$ sekundi. Za veće vrednosti n, bolje rešenje je prvo generisati sve proste brojeve koji su manji ili jednaki $\sqrt{n}$, na primer Eratostenovim sitom, a zatim ići redom po generisanim prostim brojevima i proveravati da li neki od njih deli n, što je i implementirano. Na *Slici 1.* je prikazan rezultat programa koji rešava ovaj zadatak. Program se pokreće sa `python main.py`.

```
C:\Users\Dusko Sretenovic\Desktop\src>python main.py
Task #1
Private key is: (n: 3233, d: 1421)
Private key calculated in 2.9399999999887e-05 seconds.
```

Slika 1. Rešenje prvog zadatka

Kao što vidimo, privatni ključ je (3233, 1421).

Drugi zadatak

U ovom zadatku moramo dešifrovati 3 kriptograma, a poznati su nam javni ključevi. Potrebno je iz svakog datog javnog ključa (n, e) pronaći privatni ključ (n, d), a zatim koristeći privatni ključ za svaki broj c u kriptogramu, izračunati $ASCII(c^d mod\ n + 54)$. Rezultat dešifrovanja je dat na *Slici 2*.

```
Task #2
Ciphertext: [30, 11, 20, 7, 11, 32, 16, 23, 2, 6, 9, 20, 25, 6, 25]
Plaintext: NAPRAVOMSTEPUTU

Ciphertext: [16, 8, 14, 3, 8, 6, 4, 26, 3, 21, 4, 26, 4, 32, 11, 6, 4]
Plaintext: USPESNODEKODOVANO

Ciphertext: [21, 19, 13, 5, 24, 31, 8, 19, 11, 25, 13, 26, 11]
Plaintext: KRIPTOGRAFIJA
```

Slika 2. Rešenje drugog zadatka

Dakle, poruke su: na pravom ste putu, uspešno dekodovano i kriptografija.

Program

Celokupni program se sastoji od 6 modula (fajlova): `main.py`, `rsa.py`, `keys.py`, `prime.py`, `sieve.py` i `discrete.py`. U modulu `main.py` se nalazi program koji rešava dva zadatka koja su zadata u okviru domaćeg. Unutar `rsa.py` se nalaze funkcije za enkripciju i dekripciju poruka pomoću RSA algoritma, uz pomoć javnog i privatnog ključa, kao i funkcije za pretvaranje karaktera u poruci u brojeve, onako kako je specificirano zadatkom. U okviru modula `keys.py` su napravljene klase za javne i privatne ključeve. Unutar klase `PrivateKey` se nalazi metoda `calculate_private_key` koja kao argument prima javni ključ, računa privatni ključ i vraća ga pozivaocu. Unutar modula `prime.py` se nalaze funkcije koje proveravaju da li je zadati broj prost, kao i funkcije koje faktorišu broj n koji je oblika $p \cdot q$. Implementirana je funkcija koja vrši determinističku proveru da li je broj prost, a pored nje su implementirana i dva nedeterministička algoritma (*Fermat's primality test* i *Miller-Rabin primality test*). Uz trenutne parametre, verovatnoća greške u ovim testovima je manja od 10^{-15} za *Fermat*-ov test, odnosno manja od 10^{-30} za *Miller-Rabin*-ov test. Kod je prilično dokumentovan, pa je ove parametre lako pronaći i menjati. Za faktorizaciju broja n se najpre gleda da li je manji od 10^{12}. Ako jeste, onda je i proveravanje svakog neparnog broja do $\sqrt{n}$ relativno brzo (svega nekoliko sekundi), dok se za veće n prvo generišu svi prosti brojevi koji su manji ili jednaki $\sqrt{n}$ (Eratostenovim ili Atkinovim sitom – modul `sieve.py`) i upišu se u fajl, a zatim se redom proverava da li neki od njih deli n. Ako fajl sa prostim brojevima već postoji, onda se on odmah čita. Iz tog razloga je samo prvo izvršavanje programa duže, jer se tada i generiše fajl i radi faktorizacija, dok svako sledeće pokretanje programa samo čita iz fajla. Za $n > 10^{18}$ se fajl sa prostim brojevima manjim od $\sqrt{n}$ neće generisati jer bi zauzeo previše prostora na korisnikovom računaru. Primera radi, fajl sa prostim brojevima manjim od 10^9 zauzima oko 512MB. U modulu `discrete.py` se nalaze funkcije za brzo modularno stepenovanje (*fast modular exponentiation*), pronalaženje najvećeg zajedničkog delioca (*gcd)* dva broja (*Euclid*-ov algoritam) kao i za pronalaženje multiplikativnog inverza.

Duško Sretenović 0042/2017, Računarska tehnika i informatika.