Vaja 2 – Kriptografski sistem RSA

1. Splošna predstavitev problema

Beseda kriptografija je skovanka, ki izhaja iz dveh grških besed, kryptos in gráph. Prva pomeni skrito ali skrivno, druga pa pisavo. Kljub grškemu izvoru besed, pa v stari Grčiji ni bilo prevelike potrebe po šifriranju, saj je bilo zaradi velike nepismenosti, že samo branje dovolj zahtevna naloga, da je večino sporočil ostalo tajnih. Kljub temu pa naj bi kriptografijo prvi uporabljali Špartanci, čeprav ne v smislu, kot ga danes razumemo pod tem pojmom. Prvi, ki je dokazano šifriral svoja sporočila je bil rimski poveljnik in kasnejši cesar Gaj Julij Cezar, ki je za komunikacijo s svojimi generali uporabljal zamenjavo črk in cifer s črkami, ki so bile za vnaprej določeno razdaljo nižje v abecedi. Prejemnik je lahko sporočilo prebral le, če je poznal to razdaljo. Ta način kodiranja je dobil ime Cezarjeva šifra, podatek, ki ga potrebujemo, da lahko sporočilo beremo, pa ključ. Tak način kodiranja se je nato ohranil skozi stoletja. Tako pošiljatelj kot sprejemnik sta imela enak skrivni ključ, ki pa sta si ga morala na nek varen način izmenjati, po navadi preko kurirja ali osebno. Z nastankom računalnika in računalniških mrež, je postal tak način kodiranja neučinkovit, zato se je razvil sistem z asimetričnim ključem, kjer ključ za enkripcijo ni več enak ključu za dekripcijo. Vsak uporabnik ima tako par ključev, javni in zasebni ključ. Medtem, ko je javni ključ prosto dostopen, pa ima zasebni ključ zgolj uporabnik sam in z njim dekodira sporočila, ki so bila namenjena njemu. Eden prvih tovrstnih algoritmov, ki je bil sposoben tako generiranja elektronskih podpisov, kot tudi kodiranja sporočil, je algoritem RSA. Algoritem je dobil svoje ime po priimkih avtorjev, ki so ga leta 1978 prvi javno predstavili (Ron Rivest, Adi Shamir in Leonard Adleman) in je zelo razširjen predvsem med aplikacijami za digitalno poslovanje.
Kot smo že omenili, za šifriranje in branje sporočil potrebujemo dva ključa, javnega in zasebnega, ki ju je potrebno zgenerirati tako, da lahko sporočilo zakodirano z javnim ključem, preberemo pa samo s privatnim ključem. V nadaljevanju bomo pogledali, kako ju generiramo in uporabimo.

2. Pomoč pri implementaciji

Za uspešno šifriranje in branje sporočil moramo rešiti tri stvari. Najprej je potrebno generirati oba ključa, nato pa le ta uporabiti za kodiranje in dekodiranje sporočil. Vse to pa je mogoče le, če imamo ustrezne ključe, zato bomo najprej pogledati, kako jih dobimo.

Algoritem za generiranje ključev

Pri generiranju ustreznih ključev je izrednega pomena generator praštevil, saj jih rabimo v več korakih algoritma. Ta je naslednji:

1. Izberimo dve približno enako veliki, vendar različni praštevili p in q ($p \neq q$).

2. Izračunamo produkt $n = p\,q$ ter Eulerjevo funkcijo tega produkta, $\varphi(n)$. Ker je n produkt praštevil, funkcijo $\varphi(n)$ izračunamo kot naslednji produkt: $\varphi(n) = (p-1)(q-1)$.

3. Izberimo naključno manjše liho število e, tako da velja $1 < e < \varphi(n)$ in $\gcd(e, \varphi(n)) = 1$.

4. Izračunajmo skriti eksponent d kot multiplikativni inverz števila e po modulu $\varphi(n)$, kar pomeni, da je potrebno izpolniti enačbo $e\,d \equiv 1 \pmod{\varphi(n)}$.

5. Javni ključ P predstavlja naslednji par $P = (e, n)$, skrivni ključ S pa par $S = (d, n)$.

Praštevila v prvem koraku izračunamo s pomočjo generatorja praštevil, ki je bil predmet prve vaje. Ker morata biti števili približno enako veliki, to dosežemo tako, da jih omejimo s številom decimalnih mest.

Generator naključnih števil bomo uporabili tudi v tretjem koraku. Potrebno je preveriti le to, če največji skupni delitelj števila e in $\varphi(n)$ je večji kot 1. Če to velja, potem je potrebno generirati novo praštevilo e, sicer pa se lahko premaknemo na korak 4.

Izračun multiplikativnega inverza je poseben primer modulske linearne enačbe oblike, $a\,x \equiv b \pmod{n}$. Ta tip enačb rešujemo s pomočjo posebne funkcije `MODULAR_LINEAR_EQUATION_SOLVER(a, b, n)`, ki je predstavljena v izpisu 1.

```
function MODULAR_LINEAR_EQUATION_SOLVER(a, b, n)
 begin
      EXTENDED_EUCLID(a, n, d, x, y);

      if d deljivo z b then
        begin
         x0 := x*(b/d) mod n;
         for i := 0 to d-1 do
           print(x0 + i*(n/d) mod n);
        end
      else
        print(Rešitev ne obstaja);
end
```

Izpis 1: Iskanje rešitve modulske linearne enačbe

Iz izpisa 1 lahko vidimo, da podobno kot pri sistemu linearnih enačb, tudi tukaj ni nujno, da rešitev obstaja. Toda, če upoštevamo, da v kolikor računamo multiplikativni inverz, v našem primeru števil e in $\varphi(n)$, je vrednost spremenljivke b enaka ena. Ker pa sta številu e in $\varphi(n)$ tuji, nam tudi funkcija `EXTENDED_EUCLID` vrne kot največji skupni delitelj, d, vrednost ena. Zaradi tega je pogoj v pogojnem **if** stavku izpolnjen in rešitev obstaja, še več, rešitev je natanko ena.

Da lahko funkcijo `MODULAR_LINEAR_EQUATION_SOLVER` implementiramo, potrebujemo še funkcijo `EXTENDED_EUCLID`. Ta nam poišče največji skupni delitelj števil a in

b, to je vrednost d, ter vednosti x in y, ki izpolnita naslednjo enačbo: $d = a\,x + b\,y$. Psevdokod funkcije `EXTENDED_EUCLID` je prikazan v izpisu 2.

```
function EXTENDED_EUCLID(a, b, d, x, y)
 begin
      if b = 0 then
        begin
          d := a;
          x := 1;
          y := 0;
        end
      else
        begin
          EXTENDED_EUCLID(b, a mod b, n_d, n_x, n_y);
          d := n_d;
          x := n_y;
          y := n_x - (a/b)*n_y;
        end
end
```

Izpis 2: Pseudokod funkcije `EXTENDED_EUCLID`

Funkcija `EXTENDED_EUCLID` lahko vrne v parametru x tudi negativno vrednost, zaradi tega je potrebno opozoriti, da je ostanek pri deljenju, ki ga uporabljamo v funkcijah, vezan na naravna števila, ki pa negativnih vrednosti ne poznajo, medtem ko je funkcija, ki jo imamo vgrajena v programskih jezikih vezana na cela števila. To pomeni, da v kolikor računamo $a \bmod b$ in je a negativno število, je potrebno kot rezultat vrniti vsoto $a + b$. Ker je v našem primeru skriti eksponent d naravno število, je potrebno to upoštevati pri izračunu.

Kodiranje sporočil

Kodiranje sporočil poteka po naslednjem algoritmu:

1. Pridobiti je potrebno javni ključ naslovnika sporočila $P = (e, n)$.
2. Sporočilo predstavimo kot naravno število M.
3. Izračunajmo tajnopis C po naslednji enačbi: $C = M^e \pmod n$.
4. Tajnopis C pošljimo naslovniku.

Ker gre v primeru izračuna tajnopisa C za izračun potence velikih števil, je to težko izračunati zaradi velikosti tipov števil, ki jih imamo na voljo. Zaradi tega za to uporabimo algoritem `MODULAR_EXPONENTATION`, ki je prikazan v izpisu 3.

```
function MODULAR_EXPONENTATION(a, b, n)
 begin

    d := 1;
    Razbij b v dvojiško predstavitev [b_j, bj_-1,...,b_0]

    for i:=j downto 0 do
      begin
        d := d*d mod n;
        if b_i = 1 then
          d := d*a mod n;
      end
    return d;
end
```

Izpis 3: Pseudokod funkcije `MODULAR_EXPONENTATION`

Isto metodo uporabimo tudi ob dekodiranju sporočila.

Dekodiranje sporočil

Dekodiranje sporočila poteka po naslednjem postopku:

1. Naslovnik uporabi svoj skrivni ključ $S=(d,n)$, da izračuna naravno število M z enačbo $M=C^d(\mathrm{mod}\, n)$.

2. Naslovnik vrednost M pretvori v tekstovno sporočilo.

3. Zahteve naloge

Implementirati je potrebno aplikacijo, ki bo pretvorila poljubno dolgo sporočilo v zaporedje znakov in ga zakodirala znak po znak. Kodirano sporočilo naj se shrani v tekstovno datoteko *sporocilo.txt* med posameznimi kodiranimi znaki naj bo presledek. Prvi znak v zakodiranem sporočilu je število znakov v sporočilu, ki služi za preverjanje pravilnosti sporočila. V posebno datoteko *kljuci.txt* pa naj se shranita tudi oba ključa. Program naj omogoča branje obeh datotek in izpis kodiranega sporočila. Uporabnik naj ima možnost določiti število mest praštevil p in q, ki sta osnova za generiranje ključa, prav tako pa naj program omogoča kodiranje in dekodiranje številskega sporočila, ki pa mora biti manjše od produkta pq.