Univerzitet Crne Gore

Prirodno-matematički fakultet

Računarstvo i informacione tehnologije

Projekat iz *Kriptografije*

Algoritam RSA – kriptografija javnog ključa

Profesor: Vladimir Božović

Autori: Pavle Milić 9/15

Petar Bjelica 16/15

Luka Čelebić 8/15

Uputstvo za korisnike

Algoritam je implementiran iz tri dijela/modula. Pomoću jednog od njih mogu se izgenerisati javni i privatni ključevi potrebni za algoritam, dok druga dva vrše enkripciju/dekripciju koristeći ključeve koji im se predaju kao argumenti programa. Sva tri modula je moguće pokretati nezavisno jedan od drugog putem komandnog interfejsa, a takođe je obezbijeđen grafički interfejs koji objedinjuje sva tri modula i pruža *user-friendly* iskustvo.

rsa_genkey

Ovo je modul koji se koristi za generisanje javnog i privatnog ključa potrebnog za rad RSA algoritma. Program se može pokretati nezavisno od drugih modula putem komandne linije.

Ulazni parametri

Nema.

Primjer korišćenja: ***python rsa_genkey.py***

Izlaz programa

*Štampaju se javni i privatni ključevi na standardni izlaz. Javni ključ je u obliku (**e**, **N**), gdje je **e** eksponent kriptovane poruke, **N** vrijednost modula, a **m** tekst za enkripciju u formuli:*

$$C = m^e \ (mod\ N)$$

*Privatni ključ je u obliku (**d**, **N**), gdje je **d** prost broj za koji važi:*

$$e * d \equiv 1 \ (mod\ N)$$

rsa_enc

Modul za enkripciju ***X*** za dati javni ključ $(\boldsymbol{e}, \boldsymbol{N})$ *Napomena: Moguće je umjesto vrijednosti javnog ključa algoritmu predati vrijednosti privatnog ključa, ukoliko se algoritam koristi u svrhe digitalnog potpisivanja.*

Ulazni parametri

Ulazni parametri programa se razdvajaju blankom i po sledećem redosledu:

X *– tekst koji se kriptuje*

e *– eksponent*

N *– moduo*

Preporučuje se da ulaz bude string koji sadrži samo ASCII karaktere, jer program ne podržava Unicode, i prilikom enkripcije se gube takvi karakteri.

Primjer korišćenja: ***python rsa_enc.py X e N***

Izlaz programa

Štampa se $\boldsymbol{X'}$ *odnosno ulaz* ***X*** *u enkriptovanom obliku. Računa se po formuli*

$$X' = m^e \ (mod\ N)$$

za svaki karakter $\boldsymbol{m}$ stringa $\boldsymbol{X}$.

rsa_dec

Modul za dekripciju ***X*** za dati privatni ključ $(\boldsymbol{d}, \boldsymbol{N})$.

Ulazni parametri

Ulazni parametri programa se razdvajaju blankom i po sledećem redosledu:

$\boldsymbol{X'}$ *– kriptovani tekst*

d *– eksponent*

N *– moduo*

Primjer korišćenja: ***python rsa_dec.py X e N***

Izlaz programa

Štampa se ***X*** *odnosno ulaz* $\boldsymbol{X'}$ *u originalnom obliku. Računa se po formuli*

$$X = m^d \ (mod\ N)$$

za svaki karakter $\boldsymbol{m}$ stringa $\boldsymbol{X'}$.

Postoji i grafički interfejs koji objedinjuje sve ove module i startuje se takođe preko komandne linije u direktorijumu projekta koristeći komandu:

python main.py

Postupak rada grupe

Sa algoritmom koji smo implementirali u ovom projektu upoznali smo se u okviru kursa iz *Kriptografije* tokom ljetnjeg semestra 2018. godine. Kada smo započeli projekat nastojali smo da prvo steknemo dovoljnu teorijsku osnovu za njegovu implementaciju. Za ovo smo se pored znanja koje smo stekli na kursu koristili dodatnim izvorima sa interneta.

Takođe smo se upoznali sa njegovim implementacionim problemima i neophodnim optimizacijama koji će nas očekivati tokom izrade projekta. Ono što nas je najviše interesovalo je bila praktična upotreba ovog algoritma. Saznali smo o njegovoj primjeni u implementaciji bezbjednosnih protokola kao što je SSL. Zatim smo krenuli sa pisanjem modula za generisanje javnog i privatnog ključa.

Generisanje ključeva

Ovdje smo koristili *Fermatov test za proste brojeve* prilikom generisanja p i q. Ovaj algoritam smo odabrali zbog njegove jednostavnije implementacije u odnosu na *Miler-Rabinov*. Bitna stvar kod *Fermatovog testa* je to da i u slučaju njegovog pozitivnog ishoda ovaj algoritam ne garantuje da je testirani broj prost. Zato se kaže da je broj koji zadovoljava *Fermatov test* "vjerovatno prost". Tako, da bismo zasigurno utvrdili da je neki broj prost test je potrebno uraditi više puta, svaki put sa drugim nasumičnim brojem kao bazom za test. Na ovaj način svakim novim pozitivnim testom povećavamo vjerovatnoću da je broj prost.

Posle toga bilo je potrebno naći eksponent koji je dio javnog ključa. Ovdje je bilo potrebno naći veliki broj za koga važi da je $p, q < e < \varphi(N)$. Takođe bilo je potrebno provjeriti da li je ovaj eksponent uzajamno prost sa $\varphi(N)$. Za ovu provjeru koristili smo se Pythonovom funkcijom *gcd(a, b)* iz modula *fractions*.

Posle eksponenta javnog ključa na red dolazi računanje eksponenta privatnog ključa d za koji treba da važi:

$$e * d \equiv 1 \ (mod \ N)$$

Kao optimizaciju ovdje je dovoljno pronaći koeficijent od e koji dobijamo u rezultatu obrnutog euklidovog postupka za utvrđivanje *NZD* za brojeve e i N:

$$NZD(e, N) = e(a) + N(b)$$

gdje je a koeficijent koji tražimo. Ovaj koeficijent zadovoljava tvrđenje odozgo s toga je jasno da je $d = a$. Na ovaj način smo uštedjeli na procesorskom vremenu koje bi inače potrošili kada bismo računali eksponent d koristeći se *brute-force* metodom.

Doprinos članova grupe

Luka Čelebić: Prikupljanje materijala za učenje o RSA algoritmu, kao i izgradnja grafičkog interfejsa.

Petar Bjelica: Pomoć pri implementaciji modula za generisanje ključeva, kao i modula za enkripciju/dekripciju

Pavle Milić: Implementacija modula za generisanje ključeva, enkripciju i dekripciju