Univerzitet u Beogradu,
Elektrotehnički fakultet

Složenost algoritama i odabrane metode optimizacije

Seminarski rad

Asimetrični algoritmi za enkripciju podataka

Autori
David Đukić, 2018/0057
Sofija Đurić, 2018/0085
Ljubomir Kaluđerović, 2018/0319
Danilo Tonić, 2018/0400

Mart 2020.

Sadržaj

1 Uvod **1**

1.1 Pojam steganografije i kriptografije 1

2 Asimetrični algoritmi u praksi **1**

2.1 Nedostaci simetričnog algoritma 2

2.2 Koncept rada asimetričnog algoritma 3

2.3 Upotreba asimetričnog algoritma 3

3 RSA algoritam **4**

3.1 Koncept rada RSA algoritma 5

3.2 Konvertovanje poruke u broj 5

3.3 Odabir početnih prostih brojeva 6

3.4 Ojlerova funkcija i Ojlerova teorema 7

3.5 Generisanje eksponenti 9

3.6 Enkripcija i dekripcija 9

3.7 Primer rada RSA algoritma 10

4 Matematička podloga RSA algoritma **11**

4.1 Euklidov algoritam 11

4.2 Ojlerova funkcija 12

4.3 Ojlerova teorema 13

4.4 Prošireni Euklidov algoritam 14

5 Implementacija **15**

5.1 Funkcija za proveravanje da li je broj prost 15

5.2 Funkcija za proveravanje da li su brojevi uzajamno prosti 16

5.3 Prošireni Euklidov algoritam . . . 17
5.4 Miller-Rabinov algoritam . . . 17
5.5 Generisanje dva prosta broja nasumičnim odabirom . . . 19
5.6 Funkcija za generisanje 'e' dela ključa . . . 20
5.7 Funkcija za generisanje 'd' dela ključa . . . 20
5.8 Funkcija za generisanje ključeva . . . 21
5.9 Radix27 algoritam . . . 22
5.10 Radix27-inverse algoritam . . . 22
5.11 RSA konverzija . . . 23
5.12 Enkriptor . . . 24
5.13 Dekriptor . . . 24

6 Sigurnost RSA sistema **25**

6.1 Erastostenovo sito . . . 26

7 Zanimljivosti **26**

1 Uvod

Tema ovog rada jeste metod asimetrične enkripcije, odnosno šifrovanja, i dekripcije, odnosno dešifrovanja. Saznaćemo osnovne koncepte RSA algoritma, zašto je danas zastupljeniji od simetrične enkripcije i dekripcije, a zatim ćemo ga detaljno proučiti analizirajući delove koda napisanog u programskom jeziku C++.

Pažnju ćemo posvetiti i matematičko-teoretskom alatu koji je neophodan da bi se shvatio, a potom i dokazao mehanizam funkcionisanja RSA algoritama. Najvažnije matematičke teoreme na kojima se zasniva RSA algoritam jesu Euklidov algoritam i Ojlerova teorema.

1.1 Pojam steganografije i kriptografije

Steganografija predstavlja disciplinu koja se sastoji pre svega iz metoda skrivanja poruka. Termin steganografija nastao je od grčkih reči : *steganos* što znači prikriveno ili zaštićeno i *graphein* tj. pisanje. Steganografija, kao takva, predstavlja neki vid preteče kriptografije, međutim, ove dve naučne discipline nisu direktno povezane, iako imaju isti cilj, a to je da zaštite i prikriju informaciju. Razlika je u tome što se kod steganografije teži da se poruka sakrije, ili još bolje uklopi u neki sadržaj tako da se uopšte i ne posumnja da se šalje poruka, odnosno ona uopšte ne menja informaciju, nego je "kamuflira" i samim tim ne privlači pažnju na nju. Cilj kriptografije jeste da promeni informaciju (šifrira) do te mere da je ona nerazumljiva trećoj strani. Reč kriptografija je grčkog porekla, sastoji se od reči *kryptos* što znači skriveno ili tajna, i reči *graphein* tj. pisanje.

Postoje četiri osnovna postulata kriptografije :

(a) *Poverljivost (privatnost ili tajnost) poruke* - samo autorizovane osobe mogu pristupiti datoj informaciji koja se prenosi u poruci

(b) *Integritet poruke* - ispitivanje originalnosti poruke, tj. da li je pre prijema i dekripcije poruke došlo do neautorizovanih promena na samom šifratu

(c) *Autentifikacija pošiljaoca* - sposobnost primaoca poruke da iz iste utvrdi identitet pošiljaoca

(d) *Neporicanje pošiljaoca* - sprečavanje korisnika da poriče izvršavanje, odnosno neizvršavanje određenih akcija u prošlosti

2 Asimetrični algoritmi u praksi

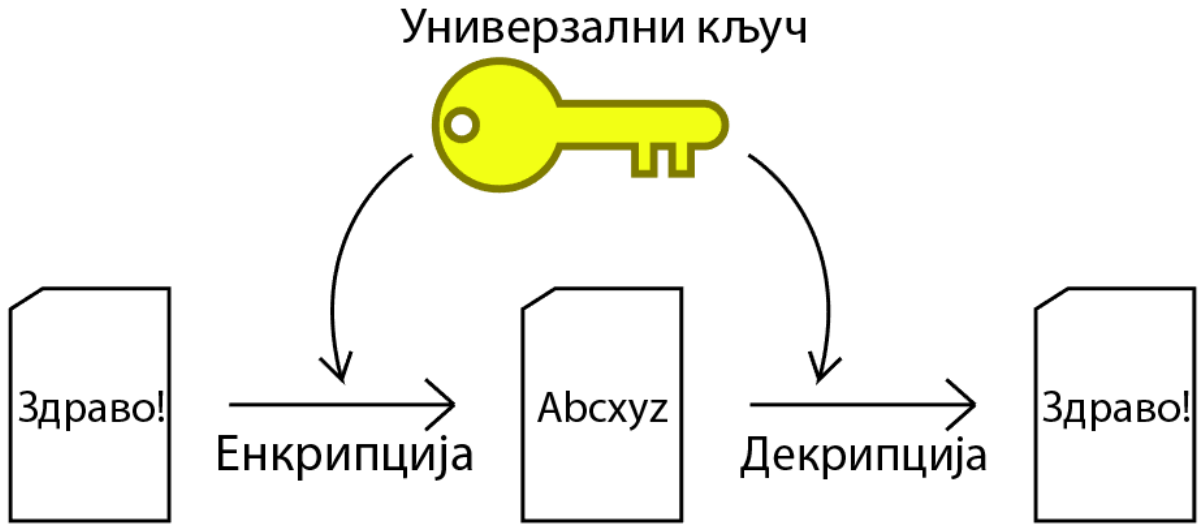
Nekada, pre nego što su računari ušli u široku upotrebu, tj. pre nego su se dovoljno razvili, većina kriptografskih metoda šifriranja se bazirala na tajnosti šifre. Tako bazirani algoritmi su se pokazali dosta nepouzdani, te su se morale pronaći neke druge metode šifrovanja. Današnje metode

šifrovanja zasnivaju se na upotrebi ključa. Ključ je najvažniji deo u pravilnoj enkripciji i dekripciji poruka. Upravo, u zavisnosti o načinu korištenja ključa, razvile su se dve klase algoritama. Jedna je simetrična, a druga asimetrična klasa. Osnovna razlika je u tome da simetrični algoritmi koriste isti ključ za enkripciju i dekripciju neke poruke (ili se ključ za dekripciju može lako proizvesti iz originalnog ključa za enkripciju), dok asimetrični algoritmi koriste različite ključeve za enkripciju i dekripciju iste.

Da bismo lakše razumeli princip rada algoritama za enkripciju uzećemo za primer dve osobe. Osobu A nazvaćemo Ana, a osobu B nazvaćemo Bojan. Ana želi Bojanu da pošalje poruku tako da je samo on može rastumačiti i pročitati. Ovom problemu možemo pristupiti na dva načina: simetričnom ili asimetričnom enkripcijom.

2.1 Nedostaci simetričnog algoritma

Koncept simetričnog algoritma sastoji se iz ideje da Ana koristi univerzalni tajni ključ da šifruje poruku, pri čemu dobija tekst koji deluje besmisleno, a zatim taj tekst šalje Bojanu koji ga istim univerzalnim ključem dešifruje i dobija originalnu poruku (Slika 1).

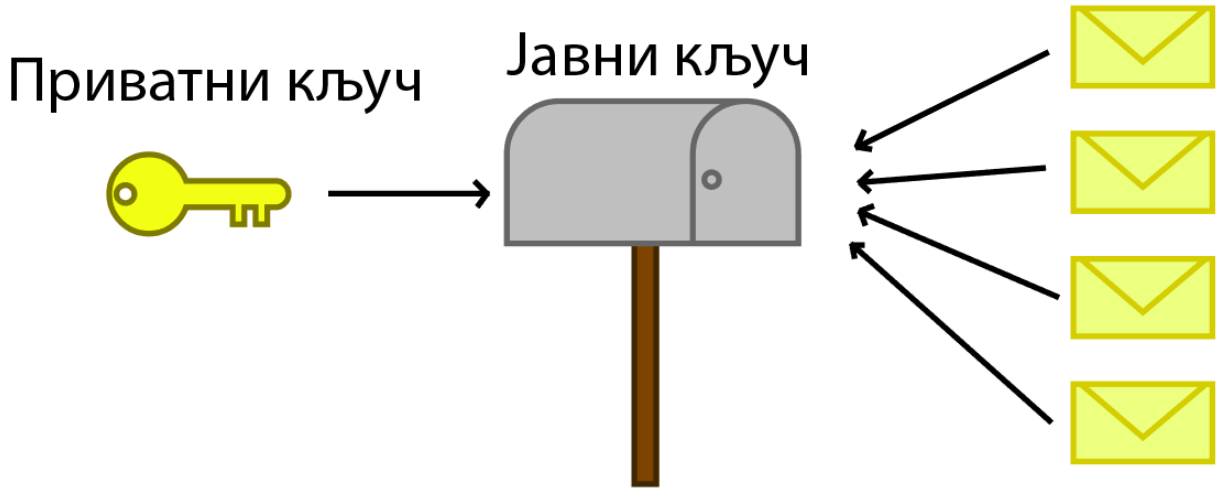


Slika 1: Prikaz simetrične enkripcije/dekripcije

Problem sa ovim konceptom jeste što bi Ana i Bojan morali oboje da poseduju isti ključ, a razmenu ključa opet bi morali da obave tajno i pouzdano. Poslati ključ na isti način kao i poruku ne bi bilo bezbedno jer ukoliko bi neko pored Bojana otkrio taj ključ mogao bi da dešifruje svaku poruku razmenjenu između Ane i Bojana. Drugi problem jeste što ukoliko bi Ana želela da komunicira sa još nekim pored Bojana, za svaku konverzaciju morala bi da poseduje poseban ključ i da vodi evidenciju o ključevima. Baš ovo su problemi koje asimetrična enkripcija ima za cilj da spreči.

2.2 Koncept rada asimetričnog algoritma

Tvorci asimetrične kriptografije su Whitefield Diffie i Martin Hellman koji su 1976. godine opisali ideju kriptografije koja se temelji na dva ključa, privatnom (ili često zvanim tajnim) i javnom ključu. Ideja za asimetričnu enkripciju sastoji se u tome da korisnik poseduje par ključeva – jedan za enkripciju, drugi za dekripciju, pri čemu ključ kojim se dešifruje poruka mora biti iz istog para kao ključ kojim je pre toga šifrovana. U tom slučaju jedan ključ je javni i vlasnik ga deli sa drugima dok drugi čuva za sebe i koristi ga on isključivo. Uglavnom se za javni ključ bira enkripcioni, a za privatni dekripcioni, pri čemu vlasnik ključeva omogućava da mu drugi korisnici šalju poruke koje samo on može dekriptovati. Možemo to zamisliti na primeru poštanskog sandučeta (slika 3): javni ključ predstavlja adresu sandučeta koju ko god poseduje može da pošalje poruku vlasniku dok privatni ključ prestavlja ključ za sanduče koji poseduje samo vlasnik koji može da otvori sanduče i pročita poruke u njemu.



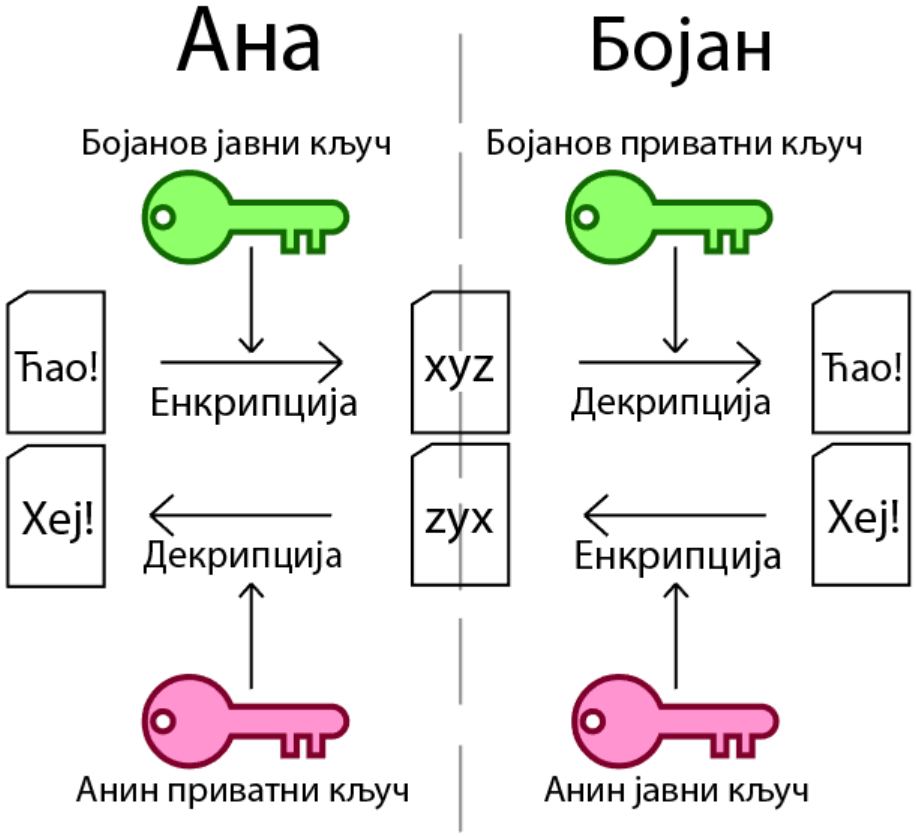
Slika 2: Prikaz asimetrične enkripcije/dekripcije

U današnje vreme međutim nije strano ni da se za pristupačnost ključeva bira obrnut slučaj: da privatni ključ bude enkripcioni, a javni dekripcioni. Pri ovakvom odabiru svaki korisnik ima dekripcioni ključ, ali poruku može dekriptovati samo ukoliko ju je poslala osoba koja poseduje privatni, odnosno enkripcioni ključ. Tako možemo biti sigurni da je poruka koja nam je stigla upravo od osobe od koje je očekujemo, pa se ovakav odabir ključeva danas masovno koristi pri upotrebi digitalnih potpisa.

2.3 Upotreba asimetričnog algoritma

U našem primeru fokusiraćemo se na prvi slučaj odabira ključeva, pa i Ana i Bojan moraju imati svoj par ključeva koji se popularno generiše RSA sistemom o kome ćemo govoriti

malo kasnije. Kada svako ima svoj par, razmenjuju svoje javne ključeve i sada Ana ima Bojanov javni ključ i može da šifruje poruku tako da samo on može da je pročita uz pomoć svog privatnog ključa. Takođe Bojan uz pomoć Aninog javnog ključa može da odgovori šifrujući poruku tako da samo ona može da je pročita (Slika 3). Ovo omogućava sasvim bezbednu komunikaciju, jer niko ne može da dešifruje poruke između Ane i Bojana ukoliko ne nabavi jedan od njihovih privatnih ključeva, a čak i ukoliko nabavi jedan, recimo Anin, moći će samo da dešifruje poruke upućene njoj dok će poruke upućene Bojanu i dalje biti bezbedne.



Slika 3: Šema asimetrične enkripcije/dekripcije

Ipak, asimetrična enkripcija, iako zastupljenija i bezbednija, kao i simetrična ima svoje prednosti i mane, pa radi maksimalne efikasnosti danas se koristi kombinacija ove dve enkripcije. Ključevi za simetričnu enkripciju razmenjuju se asimetričnom enkripcijom, a dalja razmena podataka vrši se tim ključevima zbog znatno veće brzine algoritama simetrične enkripcije.

3 RSA algoritam

RSA algoritam predstavlja danas jedan od najrasprostranjenijih, a samim time i najpoznatijih metoda kriptografije sa javnim ključem. Upotrebljava se veoma široko, a samo neki od

primera su : slanje šifrovanih poruka između korisnika, autentifikacija digitalnih potpisa, pristup obezbeđenim arhivama ili bazama podataka kao u sistemu sa platnim ili kreditnim karticama itd.

Akronim RSA predstavlja početna slova prezimena individualaca koji su prvi javno objavili ovaj algoritam 1977. godine na MIT univerzitetu : Rivest, Shamir i Adleman.

3.1 Koncept rada RSA algoritma

Korisnik RSA sistema pruža nam dve javne informacije, poznate kao N i e. Ovi brojevi nazivaju se modulusom kongruencije i javnom eksponentom enkripcije respektivno. Poruku koju želimo poslati pre svega transformišemo u neki broj M koji treba da zadovoljava uslov $0 \leq M \leq N-1$. Enkripcija se vrši uz pomoć prethodno datih brojeva N i e pri čemu dobijamo broj C koji predstavlja enkriptovanu poruku, a zatim broj C šaljemo datom korisniku. Korisnik vrši dekripciju uz pomoć para N i d, gde je N kao i malopre modulus kongruencije, a d je samo njemu poznata eksponenta dekripcije, pri čemu dobija broj M koji konvertuje nazad u originalnu poruku.

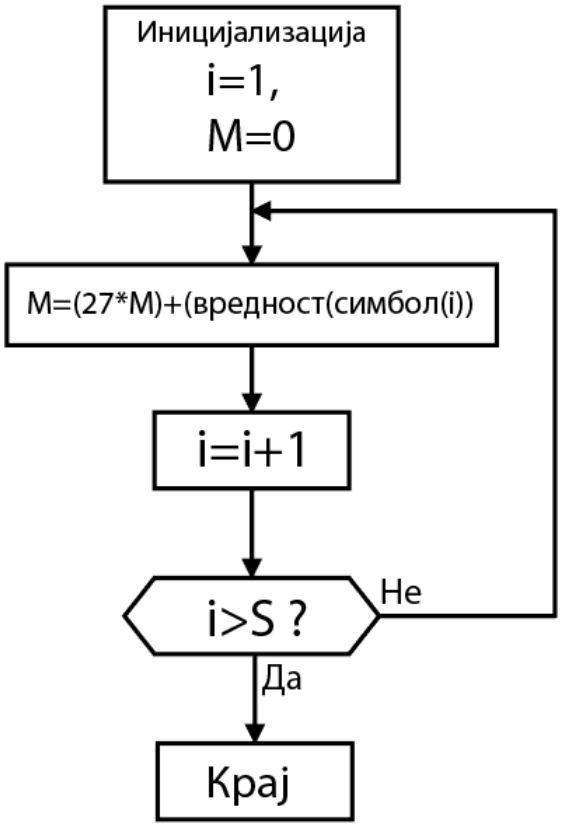
3.2 Konvertovanje poruke u broj

Kao što je pomenuto bitno je poruku koju šaljemo prethodno pretvoriti u broj između 0 i $N-1$. Ovo konvertovanje možemo izvršiti na mnogo načina sve dok korisnik može lako izvršiti inverzno konvertovanje. Na primer, ograničimo tekst poruke na simbole $A, B, C, \ldots, Y, Z$ i simbol „space“ koji ćemo koristiti za razmak. Tada poruku možemo šifrovati uz pomoć tabele za konverziju simbola u brojeve (Slika 4), pri čemu bismo jednostavno umesto simbola postavili odgovarajuću brojevnu vrednost. Moramo paziti da ispred vrednosti simbola od A do J koji imaju jednocifrene brojevne vrednosti dodamo 0 kako bi se izbegla dvosmislenost koja može da se pojavi – recimo da konvertujemo uzastopne simbole BB kao 11 umesto 0101, javila bi se dvosmislenost jer ta vrednost može predstavljati i simbol L. Pri ovoj konverziji jednostavna poruka THE prevela bi se u broj 190704.

Симбол	Вредност	Симбол	Вредност	Симбол	Вредност
A	0	J	9	S	18
B	1	K	10	T	19
C	2	L	11	U	20
D	3	M	12	V	21
E	4	N	13	W	22
F	5	O	14	X	23
G	6	P	15	Y	24
H	7	Q	16	Z	25
I	8	R	17	space	26

Slika 4: Tabela konvertovanja simbola u brojeve

Mogli bismo proces konverzije učiniti malo efikasnijim koristeći takozvanu „Radix-27“ konverziju. Ova konverzija daje nam specifičnu sumu vrednosti simbola koja se množi sa 27 pre dodavanja svake sledeće vrednosti simbola. Algoritam ove konverzije prikazan je na slici 5 koristeći za vrednosti simbola tabelu sa slike 4. Pri ovoj konverziji jednostavna poruka THE prevela bi se u broj $((19*27)+7)*27+4 = 14044$ što je dosta kraće i efikasnije od predhodno dobijenog broja.



Slika 5: Algoritam „Radix-27“ konverzije (i - redni broj simbola u poruci, M - rezultni broj konverzije, S - ukupan broj simbola u poruci)

Dekodiranje ovako konvertovane poruke jako je jednostavno: poslednji simbol je ostatak pri deljenju broja M sa 27, zatim se sam broj M celobrojno deli sa 27, sledeći, odnosno pretposlednji simbol je ostatak pri deljenju dobijenog broja sa 27 i ovaj ciklus se ponavlja sve dok ne dobijemo i prvi simbol poruke.

3.3 Odabir početnih prostih brojeva

Da bismo generisali sve bitne komponente za rad RSA sistema, prvi korak bio bi nalaženje dva velika prosta broja (svaki veličine makar 155 decimalnih cifara). Pri traženju ovih brojeva koristićemo dve bitne činjenice i jednu bitnu pretpostavku:

Činjenica 1. Lako je naći proste brojeve p i q određenog reda veličine.

Ovo proizlazi iz dve činjenice: prosti brojevi bilo koje veličine su dosta česti i lako je proveriti da li je broj prost. Stoga možemo jednostavno birati nasumične brojeve neke veličine i proveravati da li su prosti. Teorema o prostim brojevima tvrdi da je, za dovoljno veliko $n \in N$, verovatnoća da je slučajno odabran broj n baš prost broj jednaka $\frac{1}{\log n}$, tako da ne bi trebalo biti previše problema oko odabira velikog prostog broja.

Provera da li je broj prost nije oduvek bila jednostavna, jer su traženi svi faktori broja i proveravano je da li su to samo jedinica i sam taj broj, ali sedamdesetih godina otkrivene su metode koje su taj proces znatno pojednostavile ispitujući određena svojstva koja imaju prosti brojevi, dok ih nemaju složeni, a ne same faktore broja. Bez ovih metoda većina modernih kriptografskih sistema ne bi bila efikasna zbog zavisnosti od generisanja prostih brojeva. Kako smo već pomenuli p i q moraju biti veliki brojevi. Pod „velikim“ u kriptografskom kontekstu misli se na $512 - bitne$ brojeve (brojeve sa 155 dekadnih cifara) ili veće.

Činjenica 2. Lako je naći broj N kao proizvod brojeva p i q.

Koliko god veliki bili, dva broja uvek se mogu pomnožiti klasičnim školskim postupkom množenjem cifre po cifre, tako da je kompleksnost procesa uvek mala.

Pretpostavka 1. Faktorisanje broja N i dobijanje brojeva p i q iz njega je jako težak proces.

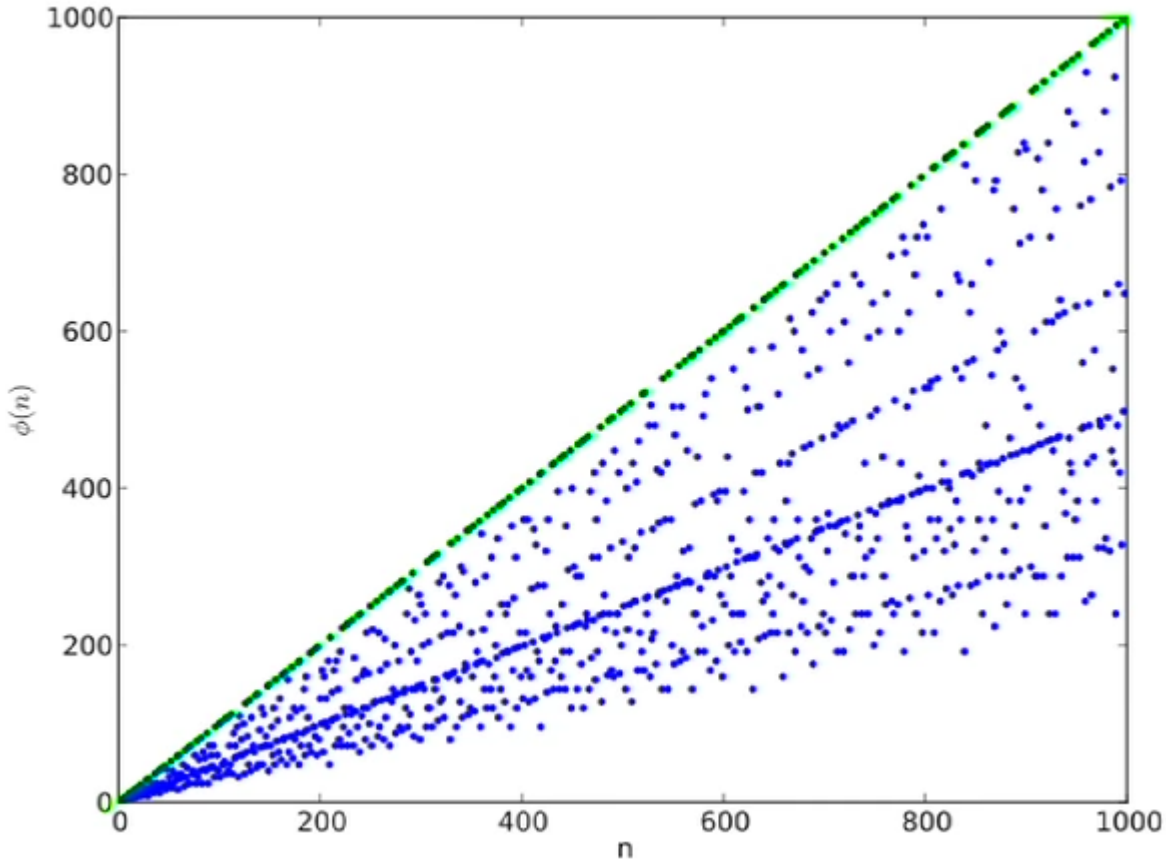
Uprkos viševekovnom izučavanju pronalaženje prostih činioca velikih brojeva i dalje je jako zahtevan proces. Najbrže metode za obavljanje ovog procesa imale bi kompleksnost reda $\sqrt{N}$. Svakako, to je u slučaju velikih brojeva i dalje jako skup proces. Uzmimo za primer $1024 - bitni$ broj. Mašini koja košta 10 miliona američkih dolara trebalo bi oko godinu dana da nađe dva prosta faktora tog broja. Za nalaženje faktora $2048 - bitnog$ broja trebalo bi par milijardi puta više vremena. Iz ovog razloga bezbedno je javno objaviti N u paru sa enkripcionom eksponentom e jer su šanse da će neko iz njega naći brojeve p i q jako male. Ovo nam je jako bitno jer je bitno da brojevi p i q ostanu tajni iz razloga što će se dalje koristiti za generisanje privatne dekripcione (i javne enkripcione) eksponente.

3.4 Ojlerova funkcija i Ojlerova teorema

Takođe jako bitna za generisanje eksponenti kako u RSA tako i u drugim modernim kriptografskim sistemima jeste čuvena Ojlerova ϕ funkcija. Ova funkcija za određenu promenljivu predstavlja broj uzajamno prostih brojeva manjih od nje same.

Definicija 3.1. Neka je $n \in N$, $n > 1$. **Ojlerova funkcija** predstavlja ukupan broj prirodnih brojeva koji nisu veći od n, a koji su uzajamno prosti sa n, tj. broj elemenata svedenog sistema ostataka po modulu n i označava se sa $\phi(n)$.

Na slici 6 možemo da vidimo grafički prikaz ove funkcije za sve cele brojeve od 0 do 1000, a iz njega je očigledno da se ova funkcija zbog svoje nedoslednosti jako teško može računati osim u



Slika 6: Grafički prikaz Ojlerove funkcije

jednom trivijalnom slučaju. Taj slučaj predstavljaju vrednosti na liniji na vrhu (obojene zelenom bojom), odnosno vrednosti ϕ funkcije za proste brojeve koji su zbog svoje prirode uzajamno prosti svim brojevima osim sebi samima. Iz tog razloga za svaki prosti broj p vrednost ove funkcije je upravo:

$$\phi(p) = p - 1 \tag{3.4.1}$$

Takođe, vrlo važno svojstvo Ojlerove funkcije jeste njena multiplikativnost (teorema 2.1.):

Teorema 3.1. Ojlerova funkcija ϕ je multiplikativna, odnosno važi:

$$\phi(A \cdot B) = \phi(A) \cdot \phi(B) \tag{3.4.2}$$

Dokaz ove teoreme se nalazi u sekciji 4.2 .

Takođe izuzetno značajna u generisanju eksponenti jeste Ojlerova teorema (Dokaz sledi u poglavlju 4.3)), poznata i kao Ojler-Fermaova teorema, koja predstavlja uopštenje Fermaove male teoreme i glasi:

Teorema 3.2. Ojlerova teorema. Ako je $NZD(a, m) = 1$ onda je:

$$a^{\phi(m)} \equiv 1 \pmod{m} \tag{3.4.3}$$

3.5 Generisanje eksponenti

Kada smo izabrali proste brojeve p i q, odredili njihov proizvod $N = pq$, i izračunali za njega Ojlerovu funkciju $\phi(N) = (p-1)(q-1)$ konačno možemo generisati javni i privatni ključ za RSA sistem. Prvo što je neophodno jeste izabrati enkripcionu eksponentu e, za šta postoji uslov da e bude pozitivan broj iz skupa $\{1, 2, ..., \phi(N) - 1\}$ i da bude uzajamno prost sa $\phi(N)$. Nije neophodno ali je preporučljivo da e bude jako velik broj. Sledeći korak jeste biranje dekripcione eksponente d takve da važi:

$$e \cdot d \equiv 1 \pmod{\phi(N)} \Rightarrow e \cdot d = k \cdot \phi(N) + 1, k \in Z \tag{3.5.1}$$

U poglavlju Matematička podloga RSA algoritma videćemo da je uslov da e bude uzajamno prost sa $\phi(N)$ neophodan da bi se osiguralo postojanje njemu inverznog broja d po modulu $\phi(N)$. Nalaženje modularnih inverza najčešće i najlakše se vrši primenom Proširenog Euklidovog algoritma ili sličnih metoda. Takođe, na osnovu osobine multiplikativnosti Ojlerove funkcije vidimo da je lako izračunati d samo ako su nam poznati prosti faktori broja N (p i q), kako bismo izračunali $\phi(N)$, a po pretpostavci 1 (navedena u sekciji 3.3.) te faktore je jako teško izračunati ukoliko je dat samo njihov proizvod. Samim tim osigurane su bezbednost i privatnost same dekripcione eksponente d.

3.6 Enkripcija i dekripcija

Kada konačno imamo generisane eksponente, možemo sastaviti ključeve u vidu uređenih parova: par (e, N), kao što je već rečeno, predstavljaće javni, enkripcioni ključ koji će korisnik deliti svakome ko želi da mu pošalje poruku, dok će par (d, N), kao privatni, dekripcioni ključ držati za sebe. Kada neko želi da pošalje korisniku poruku, enkriptovaće je tako što će je prvo podići na e-ti stepen, a zatim izvršiti kongruenciju po modulu N. Odnosno, ukoliko je poruka konvertovana u broj M, enkripcija se vrši na sledeći način:

$$C = M^e \pmod{N} \tag{3.6.1}$$

Gde je C broj koji predstavlja enkriptovanu poruku.
Korisnik će poruku dekriptovati koristeći svoj privatni ključ, tako što će broj C podići na d-ti stepen, a zatim izvršiti kongruenciju po modulu N. Dakle, originalna poruka M dobija se na sledeći način:

$$M = C^d \pmod{N} \tag{3.6.2}$$

Lako možemo dokazati da se na određeni način dobija upravo M jer:

$$\begin{aligned} C^d \pmod{N} = (M^e)^d \pmod{N} = M^{e \cdot d} \pmod{N} = M^{k \cdot \phi(N)+1} \pmod{N} = M \cdot M^{k \cdot \phi(N)} \pmod{N} \\ = M \cdot 1^k \pmod{N} = M \pmod{N} = M \end{aligned}$$

U nastavku pokazaćemo na primeru kako radi sve što smo do sada pomenuli i pokazati delove koda vezane za rad sistema RSA.

3.7 Primer rada RSA algoritma

Podsetimo se Ane i Bojana. Njihov cilj je da Bojan što bezbednije pošalje Ani tajnu poruku. Da bi se ovo ostvarilo Ana prvo mora generisati svoje ključeve za enkripciju i dekripciju. Prvo što Ana mora uraditi jeste da izabere dva prosta broja p i q. Radi jednostavnosti primera zanemarićemo to da ti brojevi moraju da budu jako veliki, pa Ana bira sledeće brojeve:

$$p = 2$$

$$q = 7$$

Zatim Ana računa njihov proizvod i Ojlerovu funkciju njihovog proizvoda:

$$N = 2 \cdot 7 = 14$$

$$\phi(N) = (2-1)(7-1) = 6$$

Zatim bira enkripcionu eksponentu uzajamno prostu sa $\phi(N)$:

$$e = 5$$

I konačno bira dekripcionu eksponentu koristeći se takvu da važi:

$$5 \cdot d \pmod 6 = 1$$

Koristeći se Proširenim Euklidovim algoritmom (koji ćemo razjasniti u poglavlju Matematička podloga RSA algoritma) da nađe skup ovakvih brojeva, Ana za d bira broj 11. Sada kada Ana ima javni ključ $(5, 14)$ i privatni ključ $(11, 14)$, šalje javni ključ Bojanu. Bojan želi da pošalje Ani jako jednostavnu poruku, recimo latinično slovo B. Bojan pre svega konvertuje svoju poruku u broj M, služivši se tabelom sa Slike 4 ($M = 2$), a zatim vrši enkripciju na sledeći način:

$$C = 2^5 (mod 14) = 32 (mod 14) = 4$$

Pri čemu dobijeni broj C šalje Ani koja ga dekriptuje na sledeći način:

$$M = 4^{11} \pmod{14} = 4194304 \pmod{14} = 2$$

Ana pritom dobija originalan broj M koji konvertuje nazad u poruku B koristeći se takođe tabelom sa Slike 4.

4 Matematička podloga RSA algoritma

Definicija 4.1. Deljivost. Za date brojeve $n, m \in Z \setminus \{0\}$ kažemo da n deli m, odnosno da je m deljivo sa n ukoliko postoji ceo broj b takav da važi $m = b \cdot n$. To označavamo sa $n|m$. Ako $n|m$ onda još kažemo da je n delilac od m, odnosno da je m sadržalac od n.

Definicija 4.2. Najveći zajednički delilac. Za date $n, m \in Z \setminus \{0\}$ zajednički delilac je prirodan broj d takav da važi $d|n$ i $d|m$. Ako je barem jedan od brojeva n i m različit od nule, onda postoji samo konačno mnogo zajedničkih delilaca brojeva n i m. Najveći među njima zove se najveći zajednički delilac brojeva n i m i označava se sa $NZD(n, m)$.

Definicija 4.3. Uzajamno prosti brojevi. Celi brojevi n i m su uzajamno (relativno) prosti ukoliko je $NZD(n, m) = 1$.

Teorema 4.1. Teorema o deljenju sa ostatkom Za proizvoljan prirodan broj b i celi broj a postoje jedinstveni celi brojevi q i r takvi da je:

$$a = q \cdot b + r, 0 \leqslant r < b. \tag{4.0.1}$$

Broj q se zove količnik, a r se zove ostatak pri deljenju a sa b.

Teorema 4.2. Neka su a,b,q i r celi brojevi takvi da je $b > 0, 0 \leqslant r < b, a = b \cdot q + r$. Tada je $NZD(a, b) = NZD(b, r)$.

Lema 4.3. Ako je $a = b \cdot q$ i $b \geqslant 0$, onda je $NZD(a, b) = b$.

4.1 Euklidov algoritam

Sledeća teorema, tj. Euklidov algoritam za nalaženje najvećeg zajedničkog delioca dva prirodna broja je najstariji algoritam koji je i danas u upotrebi. On se zasniva na prethodno navedenim teoremama i lemi.

Teorema 4.4. Euklidov algoritam Neka su $a, b > 0$ prirodni brojevi. Pretpostavimo da je uzastopnom primenom teoreme o deljenju sa ostatkom dobijen niz jednakosti :

$$\begin{aligned} a &= b \cdot q_1 + r_1, 0 \leqslant r_1 < b \\ b &= r_1 \cdot q_2 + r_2, 0 \leqslant r_2 < r_1 \\ r_1 &= r_2 \cdot q_3 + r_3, 0 \leqslant r_3 < r_2 \\ &\dots \\ r_{j-2} &= r_{j-1} \cdot q_j + r_j, 0 \leqslant r_j < r_{j-1} \\ r_{j-1} &= r_j \cdot q_{j+1} \end{aligned} \tag{4.1.1}$$

tada je $NZD(a, b)$ jednak r_j , tj. poslednjem ostatku različitom od nule.

Dokaz: Pre svega, prokomentarišimo zašto je ovakav niz jednakosti konačan, tj. zašto se u Euklidovom algoritmu može formulisati ovakav konačan niz jednačina. Naime, niz ostataka r_i, $i = \{1, ..., j\}$, $j \in N$ jeste konačan jer je strogo opadajući niz prirodnih brojeva te ograničen odozdo nulom (ne može biti negativan broj). Zato je svaki par prirodnih brojeva a i b moguće predstaviti jednačinama iznad u konačno mnogo koraka.

Na osnovu Teoreme 2.2. za niz jednakosti navedenih u Euklidovom algoritmu važi sledeće :

$$NZD(a, b) = NZD(b, r_1) = ... = NZD(r_{j-2}, r_{j-1}) = NZD(r_{j-1}, r_j) \tag{4.1.2}$$

Pošto $r_j | r_{j-1}$ iz Leme 2.3. dobijamo i $NZD(a, b) = r_j$, što je i trebalo pokazati. □

4.2 Ojlerova funkcija

Takođe jako bitna za generisanje eksponenti kako u RSA tako i u drugim modernim kriptografskim sistemima jeste čuvena Ojlerova ϕ funkcija. Ova funkcija za određenu promenljivu predstavlja broj uzajamno prostih brojeva manjih od nje same.

Definicija 4.4. Neka je $n \in N$, $n > 1$. **Ojlerova funkcija** predstavlja ukupan broj prirodnih brojeva koji nisu veći od n, a koji su uzajamno prosti sa n, tj. broj elemenata svedenog sistema ostataka po modulu n i označava se sa $phi(n)$.

Još jedno bitno svojstvo ove funkcije jeste to da je **multiplikativna**, odnosno važi:

$$\phi(m \cdot n) = \phi(m) \cdot \phi(n) \tag{4.2.1}$$

Dokaz multiplikativnosti: Neka su m i n uzajamno prosti brojevi. Znamo da je broj uzajamno prost sa $m \cdot n$ samo ako je uzajamno prost i sa m i sa n. Da bismo izračunali koliko od 1 do $m \times n$ ima uzajamno prostih sa m i sa n, napisaćemo brojeve u obliku tabele:

1	*2*	.	.	.	*k*	.	.	.	*m*
m + 1	*m + 2*	.	.	.	*m + k*	.	.	.	2m
2m + 1	*2m + 2*	.	.	.	*2m + k*	.	.	.	3m
.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.
(n - 1)m + 1	*(n - 1)m + 2*	.	.	.	*(n - 1)m + k*	.	.	.	*(n - 1)m + m*

U k-toj koloni svi brojevi imaju oblik $a \cdot m + k$, dakle svi su kongruentni po modulu m, tj. svi su uzajamno prosti sa m. U prvoj vrsti tabele ima $\phi(m)$ brojeva uzajamno prostih sa m,

što znači da u tabeli ima $\phi(m)$ kolona koje sadrže samo sumu brojeva uzajamno prostih sa m, dok ostali brojevi u tabeli nisu uzajamno prosti sa m.

Sada ćemo da utvrdimo koliko u svakoj koloni ima brojeva uzajamno prostih sa n. Posmatrajmo brojeve u k-toj koloni. Među njima ne postoje dva kongruentna broja po modulu n jer ako postoje, za $0 < s < t < n$ važi $s \cdot m + k = t \cdot m + k \pmod{n}$, odnosno $s \cdot m = t \cdot m \pmod{n}$, a kako smo rekli da su m i n uzajamno prosti, onda je $s = t \pmod{n}$. Brojevi s i t pripadaju potpunom sistemu ostataka $0, 1, 2, \ldots, n-1$, pa iz poslednje kongruencije sledi da je $s = t$.

Iz ovog razmatranja sledi da n brojeva koji pripadaju istoj koloni tabele obrazuju potpun sistem ostataka po modulu n, a to znači da među njima ima $\phi(n)$ brojeva uzajamno prostih sa n. □

Teorema 4.5.

$$\varphi(p^n) = p^{n-1}(p-1) = p^n(1 - \frac{1}{p}) \tag{4.2.2}$$

, gde su p i n prirodni brojevi, pri čemu je p prost broj.

Dokaz : Svaki od prirodnih brojeva od 1 do p^n ili je uzajamno prost ili je deljiv sa brojem p. Broj onih koji su deljivi sa p jednak je p^{n-1}. Dakle, broj onih koji su uzajamno prosti broju p, a samim tim i broju p^n, jednak je:

$$p^n - p^{n-1} = p^n(1 - \frac{1}{p}).$$

□

Primer :

$$\varphi(36) = \varphi(2^2 3^2) = 36(1 - \frac{1}{2})(1 - \frac{1}{3}) = 36 \times \frac{1}{2} \times \frac{2}{3} = 12.$$

4.3 Ojlerova teorema

Lema 4.6. Neka je $NZD(a, m) = 1$ i neka je $\{\alpha_1, \alpha_2, ..., \alpha_k\}$ proizvoljan potpun (sveden) sistem ostataka po modulu m. Tada je i $\{a\alpha_1, a\alpha_2, ..., a\alpha_k\}$ isto tako potpun (sveden) sistem ostataka po modulu m.

Teorema 4.7. Ojlerova teorema. Ako je $NZD(a, m) = 1$ onda je:

$$a^{\phi(m)} \equiv 1 \pmod{m} \tag{4.3.1}$$

Dokaz: Neka je $\{\alpha_1, \alpha_2, ..., \alpha_{\phi(m)}\}$ svedeni sistem ostataka po modulu m. Onda je prema Lemi 4.6. i $\{a\alpha_1, a\alpha_2, ..., a\alpha_{\phi(m)}\}$ svedeni sistem ostataka po modulu m. Zato, za svaki broj α_i postoji samo jedan broj α_j takav da važi :

$$a \cdot \alpha_j \equiv \alpha_i \pmod{m}. \tag{4.3.2}$$

Kako ima $\phi(m)$ takvih brojeva ima toliko i kongruencija :

$$\begin{aligned} \alpha_1 &\equiv a\alpha_1 \pmod{m} \\ \alpha_2 &\equiv a\alpha_2 \pmod{m} \\ &... \\ \alpha_{\phi(m)} &\equiv a\alpha_{\phi(m)} \pmod{m} \end{aligned} \tag{4.3.3}$$

Pomnoživši sve ove kongruencije, dobijamo sledeću kongruenciju :

$$a^{\phi(m)}\alpha_1\alpha_2...\alpha_{\phi(m)} \equiv \alpha_1\alpha_2...\alpha_{\phi(m)} \pmod{m} \tag{4.3.4}$$

a kako je ispunjen uslov leme 4.6. da je $(\alpha_i, m) = 1$, $i = 1, 2, ..., \phi(m)$ možemo izvršiti skraćivanje, pa konačno dobijamo :

$$a \cdot \phi(m) \equiv 1 \pmod{m} \tag{4.3.5}$$

□

4.4 Prošireni Euklidov algoritam

Kao što smo već videli, Euklidovim algoritmom dolazimo do NZD od dva broja. Međutim, uz pomoć proširenog Euklidovog algoritma možemo i više od toga, predstaviti taj NZD kao linearnu kombinaciju brojeva čiji je on NZD, tj. on nam omogućava da nađemo i x i y takve da za dva broja a i b za koje smo tražili NZD važi $ax + by = NZD(a, b)$.

Teorema 4.8. Prošireni Euklidov algoritam. Neka su $a, b > 0$ celi brojevi, a Euklidovim algoritmom dobijen je $NZD(a, b) = r$. Tada postoje brojevi $x, y \in Z$ koji zadovoljavaju jednačinu:

$$r = x \cdot a + y \cdot b \tag{4.4.1}$$

Dokaz:. Posmatrajmo skup celih brojeva koji su oblika $xa + yb, x, y \in Z$. Primetimo da u tom skupu imamo i pozitivne i negativne cele brojeve, a i nulu (možemo izabrati takvo $x, y \in Z$ da je $xa + yb = 0$). Izaberimo najmanji pozitivan element iz takvog skupa (mora da postoji jer skup sadrži nulu pa i negativne brojeve). Neka je to broj $c = x_0a + y_0b$, za neke konkretne cele brojeve x_0, y_0. Dokažimo sada da $c|a$.

Pretpostavimo suprotno, da c ne deli a, a onda postoje celi brojevi q i r, $0 < r < c$ takvi da važi $a = cq + r$. Onda je :

$$r = a - cq = a - q(x_0a + y_0b) = (1 - x_0q)a - y_0qb \tag{4.4.2}$$

Broj r je po definiciji pozitivan i manji od c, a i oblika $xa+yb$ (za konkretne $x = 1-x_0q$ i $y = y_0 \cdot q$), što je u suprotnosti sa pretpostavkom da je c najmanji takav pozitivan broj. Iz toga sledi da ipak $c|a$. Analogno se pokazuje da i $c|b$, tj. da je c zajednički delilac brojeva a i b, a onda i $c|d$. Iz činjenica da $d|a$, $d|b$ i da je $c = x_0 \cdot a + y_0 \cdot b$ sledi da $d|c$. Time je pokazano da je zapravo $d = c$. □

Napomena: Kako je *NZD(a,b)* uglavnom manji i od a i od b, barem jedan od x ili y će najčešće biti negativan.

Kada tražimo dekripcionu eksponentu d, ona mora zadovoljavati uslov da je $e \cdot d \equiv 1 \pmod{\phi(n)}$. Ovakve brojeve e i d zovemo modularnim inverzima po modulu $\phi(n)$. Jedna metoda za izračunavanje broja d ukoliko nam je poznat broj e je tzv. naivna metoda. Osnov ove metode je u tome da se nađe svaki broj d od 1 do $\phi(n) - 1$ i da se izabere onaj koji zadovoljava gore dati uslov. Naravno, kako je $\phi(n)$ proizvod jako velikih brojeva, pa samim tim i sam jako veliki broj, ova metoda bila bi jako spora. Zbog toga se najčešće pri nalaženju modularnih inverza koristi brža metoda proširenog Euklidovog algoritma.

Setićemo se da je pri biranju enkripcione eksponente e bilo bitno da bude uzajamno prosta sa $\phi(n)$ da bi se osiguralo postojanje dekripcione eksponente d. Ako su e i $\phi(n)$ uzajamno prosti $NZD(e, \phi(n)) = 1$ pa u proširenom Euklidovom algoritmu, za $x = d$, $ed + \phi(n)y = 1$, što možemo zapisati kao $ed = (-y)\phi(n) + 1$, odnosno $ed \equiv 1 \pmod{\phi(n)}$, pa vidimo da je uslov da su ova dva broja uzajamno prosta krucijalan za uslov traženja modularnog inverza.

Algoritam za rešavanje ove jednačine i nalaženje x i y najlakše je implementirati rekurzivno, pri čemu bi izlazak iz rekurzije bio uslov da je prvi član 0. Tada je $x = 0, y = 1$, a NZD je drugi član. Ukoliko algoritam za ulazne parametre ima a i b,a za izlazne x i y, rekurzivni korak izvršavao bi se na sledeći način:

$$\begin{aligned} a_1 &\equiv a \pmod{b} \\ b_1 &= a \\ x &= y_1 - \frac{a}{b}x_1 \\ y &= x_1. \end{aligned}$$

5 Implementacija

U narednim sekcijama predložićemo funkcije koje implementiraju RSA kriptografski sistem, zajedno sa pomoćnim funkcijama.

5.1 Funkcija za proveravanje da li je broj prost

Osnovni način za proveru da li je broj prost jeste da se proveri da li neki broj do korena datog broja deli dati broj. Njena složenost je zavisna od broja za koji se provera vrši. Ukoliko je broj prost ova funkcija će izvršiti $\sqrt{n}$ iteracija, dok će u drugim slučajevima izvršiti onoliko iteracija koliki je najmanji faktor broja n.

Funkcija izgleda ovako:

```
bool prost(long long int n)
{
    if (n>=2)
    {
        for (int i = 2; i < int(sqrt(n)); i++)
        {
            if (n % i == 0)
            {
                return false;
            }
        }

        return true;
    }
    else return false;
}
```

5.2 Funkcija za proveravanje da li su brojevi uzajamno prosti

Jedan od načina da se proveri da li su brojevi uzajamno prosti se oslanja na to da ostatak pri deljenju ta dva broja mora biti deljiv njihovim najvećim zajedničkim deliocem. Izlaz se utvrđuje proverom da li je NZD veći ili jednak jedinici.

Funkcija izgleda ovako:

```
bool uzajamno_prosti(long long int a, long long int b)
{
    while(b != 0)
    {
        unsigned long long int pom = a%b;
        a = b;
        b = pom;
    }

    return !(a>1);
}
```

5.3 Proširenі Euklidov algoritam

Proširenі Euklidov algoritam služi da se, pored utvrđivanja najvećeg zajedničkog delioca, isti predstavi kao linearna kombinacija dva broja za koje se definiše. Proširenі Euklidov algoritam je primer algoritma pogodnog za rekurzivno definisanje.

Funkcija izgleda ovako:

```
int prosireni_nzd(long long a, long long b, int* x, int* y)
{
    if (a == 0)
    {
        *x = 0;
        *y = 1;
        return b;
    }

    int x1, x2;
    int nzd = prosireni_nzd(b%a, a, &x1, &x2);
    *x = y1 - (b/a) * x1;
    *y = x1;

    return nzd;
}
```

5.4 Miller-Rabinov algoritam

Najčešće korišćen algoritam za proveru da li je broj prost u RSA sistemima je Miller-Rabinov algoritam. Ovaj algoritam je efikasniji od klasičnog algoritma provere, ali daje rezultat sa određenom preciznošću. Postoje dva moguća ishoda provere: broj je složen ili broj je verovatno prost. U praksi verovatnoća da će složen broj proći test kao verovatno prost je vrlo mala i u mnogim situacijama je to prihvatljivo. Preciznost algoritma se podešava zadavanjem broja iteracija i taj parametar se prosleđuje funkciji kao argument.

Greška ovog algoritma se meri verovatnoćom da će složen broj biti deklarisan kao verovatno prost. Ukoliko je n složen broj, Miller-Rabinov test će broj n deklarisati kao verovatno prost sa verovatnoćom najviše 4^{-k}, gde je k broj iteracija. U prosečnom slučaju, za dovoljno velike vrednosti broja n, ova verovatnoća je znatno manja, ali u kriptografiji nije moguće osloniti se na to.

Pomoćna funkcija koja računa modul proizvoda dva broja bez obzira na to da li je taj proizvod u opsegu long long:

```
long long mulmod(long long a, long long b, long long mod)
{
        long long x = 0, y = a % mod;
        while (b > 0)
        {
                if (b % 2 == 1)
                {
                        x = (x + y) % mod;
                }
                y = (y * 2) % mod;
                b /= 2;
        }
        return x % mod;
}
```

Pomoćna funkcija koja računa modul eksponenta bez obzira na to da li je vrednost eksponenta u opsegu long long:

```
long long modulo(long long base, long long exponent, long long mod)
{
        long long x = 1;
        long long y = base;
        while (exponent > 0)
        {
                if (exponent % 2 == 1)
                        x = (x * y) % mod;
                y = (y * y) % mod;
                exponent = exponent / 2;
        }
        return x % mod;
}
```

Miller-Rabinova funkcija:

```
bool Miller(long long p, int iteration)
{
        if (p < 2)
        {
                return false;
        }
        if (p != 2 && p % 2 == 0)
        {
                return false;
        }
        long long s = p - 1;
        while (s % 2 == 0)
        {
                s /= 2;
        }
        for (int i = 0; i < iteration; i++)
        {
                long long a = rand() % (p - 1) + 1, temp = s;
                long long mod = modulo(a, temp, p);
                while (temp != p - 1 && mod != 1 && mod != p - 1)
                {
                        mod = mulmod(mod, mod, p);
                        temp *= 2;
                }
                if (mod != p - 1 && temp % 2 == 0)
                {
                        return false;
                }
        }
        return true;
}
```

5.5 Generisanje dva prosta broja nasumičnim odabirom

Generisanje brojeva koji čine i određuju ključ vršićemo korišćenjem rand() funkcije. Kako je potrebno da ova dva broja budu što veća onda je moguće postaviti dodatni uslov za minimalnu vrednost u nasumično odabiranje ovih brojeva, ali mi predlažemo verziju koja sadrži takav uslov.

Funkcija izgleda ovako:

```
void generisi_pq(long long int *N, long long int *fi)
{
    srand(time(NULL));

    long long int p, q;

    do
    {
        p = rand();
    } while (!prost(p));

    do
    {
        q = rand();
    } while (!prost(q) || p == q);

    *N = p*q;
    *fi = (p-1)*(q-1);
}
```

5.6 Funkcija za generisanje 'e' dela ključa

Kao što je opisano u teorijskom uvodu, 'e' deo ključa je broj manji od vrednosti Ojlerove funkcije za broj M i broj koji je sa njime uzajamno prost.

Funkcija izgleda ovako:

```
void generisi_e(long long int fi, int *e)
{
    do
    {
        *e = rand();
    } while (*e>=fi || !uzajamno_prosti(*e, fi));
}
```

5.7 Funkcija za generisanje 'd' dela ključa

Deo privatnog ključa 'd' je broj čiji je proizvod sa brojem e po modulu Ojlerove funkcije broja M jednak jedan. Njega tražimo oslanjajući se na prošireni Euklidov algoritam.

Funkcija izgleda ovako:

```
void generisi_d(int e, long long int fi, int *d)
{
    int x, y;
    prosireni_nzd(e, fi, &x, &y);

    if (x<0) x += fi;

    *d = x;
}
```

5.8 Funkcija za generisanje ključeva

Generisanje 'M' dela ključa se svodi na generisanje dva prosta broja. Pozivom funkcija *generisi_pq* i potom *generisi_e* i *generisi_d* u potpunosti formiramo par ključeva koji se potom smeštaju u odgovarajuće fajlove.

Funkcija izgleda ovako:

```
void generisi_kljuceve()
{
    long long int M, fi;

    generisi_pq(&M, &fi);

    int e, d;

    generisi_e(fi, &e);
    generisi_d(e, fi, &d);

    fstream javni_kljuc_fajl("javni_kljuc.txt", ios::out);
    javni_kljuc_fajl <<M <<e;

    fstream privatni_kljuc_fajl("privatni_kljuc.txt", ios::out);
    privatni_kljuc_fajl <<M <<d;

    javni_kljuc_fajl.close();
    privatni_kljuc_fajl.close();
}
```

5.9 Radix27 algoritam

Radix27 algoritam služi za konvertovanje stringa u jedinstveni prirodan broj odnosno za njegovo kodiranje. U našem slučaju za azbuku su uzeti znak razmaka i velika slova alfabeta, što ukupno čini 27 karaktera azbuke. Prema njihovoj poziciji u alfabetu dodeljen im je jedinstven broj u opsegu (0,25) i za razmak je uzet kodni broj 26. Ukoliko string čiji su karakteri kodirani na ovaj način posmatramo kao broj u sistemu sa osnovom 27 onda se Radix27 algoritam može posmatrati kao postupak za generisanje dekadnog oblika tog broja. Pomenuti algoritam je pogodan za primenu u RSA sistemima jer se njime omogućuje primena ključeva na celu poruku umesto na karaktere poruke pojedinačno, što doprinosi sigurnosti u prenosu poruke.

Funkcija izgleda ovako:

```
long long int radix27(string tekst)
{
    char C;
    int I;
    long long int poruka = 0;

    while (tekst != "")
    {
        C = tekst.at(0);
        tekst.erase(0, 1);

        if ('A' <= C <= 'Z') I = C - 'A';
        if (C == ' ') I = 26;

        poruka = 27*poruka + I;
    }

    return poruka;
}
```

5.10 Radix27-inverse algoritam

Radix27 algoritam ima svoj inverzni oblik koji služi za konvertovanje dekadnog broja u string. Ako bismo posmatrali analizu u opisu Radix27 algoritma, njegov inverz predstavlja prebacivanje dekadnog broja u oblik sa osnovom 27.

Funkcija izgleda ovako

```
string radix27inverse(long long int broj)
{
    char C;
    int I;
    string poruka = "";

    while (broj > 0)
    {
        I = broj%27;
        broj /= 27;

        C = 'A' + I;
        if (I == 26) C = '_';

        poruka = C + poruka;
    }

    return poruka;
}
```

5.11 RSA konverzija

RSA konverzija ne predstavlja ništa drugo do običnog pronalaženja modula eksponenta nekog broja. Kako bi izbegli overflow predlažemo izbegavanje operatora ^.

Funkcija izgleda ovako:

```
long long int rsa(long long int a, long long int M, int e)
{
    long long int b=1;

    for(int i=0; i<e; i++)
    {
        b=(b*a)%M;
    }

    return b;
}
```

5.12 Enkriptor

Enkriptor je funkcija koja ima ulogu u razmeni podataka. Ona dohvata javni ključ i poruku, vrši konverziju i kodiranu poruku u obliku stringa smešta u poseban fajl.

Funkcija izgleda ovako:

```
void enkriptor()
{
    long long int M;
    int e;

    fstream javni_kljuc_fajl("javni_kljuc.txt", ios::in);
    javni_kljuc_fajl >>M >>e;

    string poruka;
    fstream poruka_fajl("poruka.txt", ios::in);
    poruka_fajl >>poruka;

    fstream poruka_kodirano_fajl("poruka_kodirano.txt", ios::out);
    string poruka_kodirano;
    poruka_kodirano = radix27inverse(rsa(radix27(poruka), M, e));
    poruka_kodirano_fajl <<poruka_kodirano;

    javni_kljuc_fajl.close();
    poruka_kodirano_fajl.close();
    poruka_fajl.close();
}
```

5.13 Dekriptor

Dekriptor je funkcija koja dohvata privatni ključ i kodiranu poruku, vrši povratnu konverziju i prijemnu poruku smešta u poseban fajl.

Funkcija izgleda ovako:

```
void dekriptor()
{
    long long int M;
    int d;

    fstream privatni_kljuc_fajl("privatni_kljuc.txt", ios::in);
    privatni_kljuc_fajl >>M >>d;

    string poruka_kodirano;
    fstream poruka_kodirano_fajl("poruka_kodirano.txt", ios::in);
    poruka_kodirano_fajl >>poruka_kodirano;

    fstream poruka_prijem_fajl("poruka_prijem.txt", ios::out);
    string poruka_prijem;
    poruka_prijem = radix27inverse(rsa(radix27(poruka_kodirano), M, d));
    poruka_prijem_fajl <<poruka_prijem;

    privatni_kljuc_fajl.close();
    poruka_kodirano_fajl.close();
    poruka_prijem_fajl.close();
}
```

6 Sigurnost RSA sistema

Da bi ispitali sigurnost RSA sistema moramo da utvrdimo koliko je teško dešifrovati poruku bez direktnog pristupa privatnom ključu. Nepoznati ili privatni deo privatnog ključa je broj d. Problem se svodi na otkrivanje broja d na osnovu podataka iz javnog ključa. Jedina povezanost broja d sa elementima javnog ključa je činjenica da je važi sledeći uslov:

$$ed \equiv 1 \pmod{\phi(N)} \tag{6.0.1}$$

Kako nam je broj e poznat iz javnog ključa, moramo da dođemo do vrednosti $\phi(N)$. Jedini trag je činjenica da je $N = p \cdot q$ i u tom trenutku problem svodimo na faktorisanje broja N. Trenutno ne postoji način za utvrđivanje prostih faktora nekog broja bez isprobavanja potencijalnih prostih faktora ili, jednostavnije, ne postoji efikasan način za takav poduhvat. Potencijalne faktore možemo tražiti nasumičnim odabirom u opsegu $(2, \sqrt{N})$. Kako su raspored prostih brojeva i njihov broj u određenom opsegu stvari koje se za sada mogu samo procenjivati, smanjivanje skupa u kome tražimo faktor zahteva dodatni algoritam.

6.1 Erastostenovo sito

Eratostenovo sito je algoritam koji služi za pronalaženje svih prostih brojeva u rasponu od 1 do N. Algoritam funkcioniše tako što popunjava niz od N članova nulama i jedinicama u zavisnosti da li je indeks određenog člana prost ili složen broj. Niz je inicijalno popunjen nulama i kreće se od broja 2. Kada se naiđe na indeks u koji je upisana nula prelazi se na postupak markiranja složenih brojeva deljivih tim brojem. Markiranje podrazumeva upisivanje jedinice u odgovarajući složen broj tako da kada se algoritmom naiđe na njega neće doći do nepotrebnog markiranja njegovih sadržaoca. Mana ovog algoritma se ogleda u tome što je potrebno izdvojiti niz veličine najvećeg broja u opsegu koji pretražujemo, što je za velike brojeve nemoguće.

Funkcija izgleda ovako:

```
void sito(long long int n, vector<int>& prosti)
{
    vector<int> brojevi(n+1, 0);

    for (long long int i=2; i<=n; i++)
    {
        if (brojevi[i] == 0)
        {
            prosti.push_back(i);
            for (long long int j=2*i; j<=n; j+=i) brojevi[j]++;
        }
    }
}
```

7 Zanimljivosti

1977. godine, kada su patentirali RSA algoritam, njegovi tvorci Rivest, Samir i Adelman, sa MIT-a, zadali su zadatak za koji su mislili da će biti potrebno milijarde godina pre nego što ga neko reši. U suštini, njihov zadatak zasnivao se na faktorizaciji sledećeg broja :

$$N = 114381625757888867669235779976146612010218296721242362562561842935706935245733897830597123563958705058989075147599290026879543547$$

koji je zajedno sa eksponentom enkripcije $e = 9007$ činio javni ključ. Ponudili su 100 dolara bilo kome ko dekriptuje njihovu poruku. Takođe, dodali su i digitalni potpis upotrebljavajući privatni ključ istog algoritma, tj. eksponent dekripcije d koji zadovoljava uslov $ed = k\phi(N)+1, k \in Z$:

$$d = 167178611503808442460152713891683982454369010323583112178350384469290626554487922371144905095786086556624965779748400040570203737$$

Deo njihove šifre koji sadrži digitalni potpis imao je sledeću formu :

0609181920001915122205180023091419001
5140500082114041805040004151212011819

kada bi se dešifrovao on bi značio sledeće : *Prva osoba koja ovo reši osvaja stotinu dolara* što je zapravo značilo da je poruka zaista došla sa MIT-a. Mnogo pre nego što su to ljudi sa MIT-a očekivali, 17 godina kasnije, holandski matematičar Arjen K. Lenstra sa svojim timom u 8 meseci rešio je problem, metodom sita (multiple polynomial quadratic sieve). Uz svoj tim i metodu sita, Lenstra je takođe trebao stotine saradnika širom sveta te njihove računare, a čitava kooperacija odvijala se preko interneta. Rezultat dvodnevnog računanja bila su dva prosta broja, 64-cifreni i 65-cifreni p i q. Pomoću njih Lenstra je dekriptovao poruku sa MIT-a, a ona je glasila:

The magic words are squeamish ossifrage. (Magične reči su gadljivi bradati lešinar.)

Rivest, Samir i Adleman su, nakon što je ta, sada već u kriptografiji poznata rečenica otkrivena rekli da je ona u suštini besmislena i nema neku posebnu poruku, a takva je jer su mislili da je niko (bar za njihovo vreme) neće otkriti. Što im se činilo nemogućim rešeno je za samo sedamnaest godina, a to je samo dokaz da je kriptologija još uvek eksperimentalna nauka, tj. koliko brzo i efikasno se kriptoanaliza, a samim time i kriptografija, tačnije čitava kriptologija, razvija i napreduje.

Neka novija istraživanja pokazuju da će u budućnosti važnu ulogu u kriptografiji imati kvantni računari, koji će zasigurno podići kriptografske standarde i biti pretnja za sve one koji žele da osiguraju svoje podatke na duži vremenski period. IT stručnjaci smatraju da je praktično nemoguće za klasične računare da faktorišu brojeve duže od 2048bita, što danas čini osnovu najkorišćenije forme za RSA enkripciju. Međutim, istraživanja iz 2019. godine pokazala su da je kvantnom računaru dovoljno 20 miliona kjubita kako bi mogao da sprovede takav račun. To bi značilo da bi takvoj mašini bilo potrebno samo 8 sati da sprovede račun kojim bi slomila 2048bitnu RSA enkripciju. Istina je da danas, još uvek, mašina takvog kapaciteta i performansi postoji samo u planovima za budućnost, ali to ne sprečava kriptografske stručnjake da već rade na novim i naprednijim algoritmima za enkripciju koje čak ni kvantni računari ne bi mogli da slome.

Literatura

[1] J. E. Hershey, *Cryptography Demystified*. The McGraw-Hill Telecommunications, 2003.

[2] M. Ouwehand, *The (simple) mathematics of RSA*. 2001.

[3] B. Kaliski, "The mathematics of the rsa public-key cryptosystem."

[4] D. Divjaković, "Osnove kriptologije, otp i rsa algoritam," master rad, Univerzitet u Novom Sadu, Prirodno - matematički fakultet, Departman za matematiku i informatiku, 2016.