

UNIVERZITET U SARAJEVU
ELEKTROTEHNIČKI FAKULTET SARAJEVO

**Implementacija RSA kriptografskog algoritma sa
automatskim generiranjem ključeva**

- ZAVRŠNI RAD -

Mentor:
doc. dr. Željko Jurić

Kandidat:
Nedim Šrndić, 14572

Sarajevo, septembar 2008.

Postavka zadatka

Cilj ove teme je sastaviti program koji u potpunosti implementira RSA kriptografski algoritam sa automatskim generiranjem ključeva za enkripciju i dekripciju. Za tu svrhu, kandidat treba prvo implementirati efikasan tip podataka koji omogućava rad sa cijelim brojevima proizvoljne dužine, s obzirom da se u praktičnim primjenama RSA algoritma barata sa brojevima dužine nekoliko desetina pa i stotina cifara. Radi postizanja zadovoljavajuće brzine rada, množenje je potrebno implementirati koristeći neki sofisticiraniji algoritam u odnosu na naivni (školski) algoritam, recimo neke algoritme tipa divide-and-conquer ili eventualno čak superbrze algoritme znanovane na brzom Fourierovoj transformaciji. Dalje, za potrebe generiranja ključeva za enkripciju i dekripciju potrebno je implementirati neke naprednije algoritme iz elementarne teorije brojeva, poput efikasnih probablističkih algoritama za testiranje prostosti brojeva (npr. Solovay-Strassenov test ili Miller-Rabinov test) i efikasnih algoritama za nalaženje najvećeg zajedničkog djelioca i inverznog elementa u modularnoj aritmetici (recimo, prošireni Euclidov algoritam). Konačno, sama RSA enkripcija i dekripcija zahtijeva efikasnu implementaciju stepenovanja u modularnoj aritmetici (recimo, square-and-multiply algoritam). Razumije se da svi opisani algoritmi trebaju biti izvedeni nad brojevima proizvoljne dužine, koristeći prethodno razvijeni tip podataka za rad sa brojevima proizvoljne dužine.

Mentor:

doc. dr. Željko Jurić

Sadržaj

1. Uvod	1
2. Kriptografija	3
2.1. Uvod	3
2.2. Razvoj	3
2.3. Osnovni pojmovi	4
2.4. Kriptografija sa javnim ključem	5
2.4.1. Enkripcija	6
2.4.2. Digitalni potpis	6
2.5. Zaključak	6
3. RSA kriptosistem	7
3.1. Uvod	7
3.2. Generisanje ključeva	7
3.3. Enkripcija	8
3.4. Dekripcija	8
3.5. Digitalni potpis	8
3.6. Jako jednostavan primjer upotrebe	9
3.6.1. Generisanje ključeva	9
3.6.2. Enkripcija	10
3.6.3. Dekripcija	10
3.7. Sigurnost	10
3.8. Brzina	11
3.9. Zaključak	11
4. Karakteristični algoritmi potrebni za implementaciju RSA kriptosistema .	12
4.1. Uvod	12
4.2. Stepenovanje u modularnoj aritmetici	12
4.2.1. Naivni pristup	12
4.2.2. Stepenovanje uzastopnim kvadriranjem (repeated squaring)	13
4.3. Generisanje velikih prostih brojeva	13
4.3.1. Naivni pristup	14
4.3.2. Miller-Rabinov probabilistički test prostosti	14
4.4. Računanje inverznog elementa za množenje u modularnoj aritmetici	16
4.4.1. Prošireni Euklidov algoritam	17
4.5. Zaključak	18
5. Rad sa velikim cijelim brojevima	20
5.1. Uvod	20
5.2. Memorisanje	20
5.2.1. Binarno	21
5.2.2. Decimalno	22
5.3. Sabiranje	22
5.4. Oduzimanje	23
5.5. Množenje	24
5.5.1. Školski algoritam (long multiplication)	25
5.5.2. Podijeli pa vladaj (divide and conquer) množenje	25
5.6. Zaključak	27
6. Implementacioni detalji	28
6.1. Uvod	28
6.2. Korišteni alati	28

6.3. Izvorni kôd	28
6.3.1. <i>RSA</i>	29
6.3.2. <i>PrimeGenerator</i>	29
6.3.3. <i>KeyPair</i>	30
6.3.4. <i>Key</i>	30
6.3.5. <i>BigInt</i>	31
6.4. Zaključak	33
7. Zaključak	34
Literatura	35

1. Uvod

Već hiljadama godina, ljudski rod ima potrebu za komunikacijom. Jedan dio ljudske komunikacije, privatna komunikacija, uvijek je bio namijenjen ograničenoj grupi ljudi, bilo da se radi o važnim državnim, industrijskim i vojnim tajnama ili svakodnevnim aktivnostima kao što su povjerljivi razgovori ili bankovne transakcije. Postoje arheološki dokazi koji ukazuju na to da su tehnike i alati **povjerljive komunikacije** bili poznati još i prije više hiljada godina, a njihova važnost i dostupnost se sa vremenom povećavala. Jedan od mehanizama kojim se omogućava povjerljivost komunikacije, ali i još neke pogodnosti, jeste **RSA kriptosistem**. Primjena RSA obuhvata sigurnu komunikaciju, identifikaciju, autentifikaciju, elektronsku trgovinu, certifikaciju ili pristup sa daljine. U svakodnevnom životu koristimo kriptografiju kada obavljamo bankovne transakcije, bilo putem Interneta ili bankomata, kada gledamo kodirane kanale satelitske televizije, razgovaramo preko GSM mobilne telefonije, koristimo zaštićen pristup bežičnoj računarskoj mreži itd.

U ovom diplomskom radu opisana je programska **implementacija RSA kriptografskog algoritma za šifrovanje sa automatskim generiranjem ključeva**. Od otkrića ovog algoritma, 1978. godine, pa do danas, napravljeno je mnoštvo različitih programskih implementacija istog, od kojih su neke slobodno dostupne za preuzimanje sa Interneta. Zato doprinos ovog diplomskog rada nije u tome da bude jedina, već potpuna i lako razumljiva implementacija RSA algoritma.

Potpunost se ogleda u činjenici da su sve strukture podataka i za RSA karakteristični algoritmi samostalno isprogramirani, što nije slučaj kod mnogih dostupnih primjera, gdje se npr. koriste prethodno razvijene strukture podataka za rad sa velikim cijelim brojevima. **Razumljivost** se ogleda u jednostavnosti i čitljivosti programskog koda, što nije odlika implementacija korištenih u industriji, jer se zahtjev za velikom optimizacijom efikasnosti negativno odražava na te attribute.

Pristup korišten pri izradi ovog diplomskog rada jeste upotreba programskog jezika C++ [10] definiranog ANSI ISO/IEC 14882:1998 standardom.

Diplomski rad se sastoji iz sedam poglavlja. Svako poglavlje izuzev prvog i posljednjeg, osim osnovne teme sadrži i kratki uvod i zaključak. U ovom poglavlju data su uvodna razmatranja, te su ukratko opisani **sadržaj i struktura rada**.

U drugom poglavlju je ukratko predstavljena **kriptografija** kao nauka i praksa skrivanja informacija. Dat je pregled važnijih otkrića u historiji kriptografije, uvedeni su osnovni pojmovi kriptografije, kao i objašnjenje kriptografije sa javnim i tajnim ključem.

Treće poglavlje je posvećeno teoretskim razmatranjima **RSA algoritma šifrovanja**. Predstavljen je nastanak algoritma i dat njegov opis i način rada. Opisani su postupci šifrovanja, dešifrovanja i digitalnog potpisa koristeći RSA, kao i jednostavan primjer istih. Dat je i kratak osvrt na sigurnost RSA i neke probleme praktične implementacije.

Četvrto poglavlje posvećeno je objašnjenju osnovnih matematskih problema koji se javljaju pri **praktičnoj implementaciji RSA**, kao što su generisanje jako velikih prostih brojeva, stepenovanje i računanje inverznog elementa za množenje u modularnoj aritmetici. Nakon opisa problema, opisani su i algoritmi koji se koriste za rješavanje tih problema, sa osvrtom na računarsku efikasnost.

U petom poglavlju predstavljeni su aspekti **rada sa velikim cijelim brojevima** na računar. To je nezaobilazna tema pri implementaciji RSA, jer se sigurnost RSA zasniva na upotrebi ključeva za šifrovanje čija je dužina (tipično između 1024 i 2048 bita) mnogostruko veća od dužine riječi kod standardnog PC-a (maksimalno 64 bita). Biti će riječi o memorisanju cijelih brojeva i aritmetičkim operacijama nad tim brojevima. Kao i u četvrtom poglavlju, osim opisa predstavljene su i osnovne informacije o računarskoj efikasnosti pojedinačnih algoritama.

Šesto poglavlje opisuje **implementacione detalje** programa pod nazivom **rsa**, koji je nastao kao rezultat ovog diplomskog rada. Cjelokupan programski kôd je priložen na pratećem CD-u uz ovaj rad, te objavljen na web lokaciji <http://code.google.com/p/rsa/>.

Posljednje, sedmo poglavlje, sadrži **generalni zaključak**. Zaključak se sastoji od osvrta na urađeno, kao i nekoliko smjernica o tome šta bi se još moglo uraditi na ovu temu.

2. Kriptografija

2.1. Uvod

Artikulirana komunikacija je jedna od osnovnih osobina po kojima se čovjek razlikuje od ostatka živog i neživog svijeta. **Komunikacija** se javila vrlo rano u historiji ljudskih bića, razvijala se i razvija se i danas neprevaziđenom brzinom. Međutim, uvijek je postojao problem **privatnosti u komunikaciji**. Neke poruke su važne sa aspekta ljudske intime, druge pak sa aspekta ekonomije, sigurnosti ili nauke, i potrebno je obezbijediti da iste budu dostavljene samo onima kojih se tiču, u neizmijenjenom stanju i sa garancijom identiteta pošiljaoca. To su samo neke od prednosti koje je donio razvoj kriptografije.

U ovom poglavlju, najprije će biti predstavljen razvoj kriptografije od njenih početaka do danas, zatim će biti uvedeni osnovni pojmovi koje je potrebno poznavati prije nego se posvetimo RSA kriptosistemu, te će biti opisana kriptografija sa javnim ključem kao dio kriptografije kojem pripada RSA. U okviru kriptografije sa javnim ključem biće opisani postupci enkripcije i digitalnog potpisa.

2.2. Razvoj

Historija kriptografije počinje prije više hiljada godina. Smatra se da se prva poznata upotreba kriptografije desila u Egiptu za vrijeme Starog carstva (prije otprilike 4500 godina), kada su nestandardni hijeroglifi uklesivani u monumente. Ne smatra se da je to bio ozbiljan pokušaj tajne komunikacije, već vrsta misterije, intrige ili zabave za pismene čitaoce. Nešto mlađe glinene ploče iz Mezopotamije su jasno pokazivale namjeru zaštite informacija – to su bili šifrovani recepti, vjerovatno komercijalno vrijedni. Još kasnije, počevši oko 600. ili 500. g. p. n. e., jevrejski učenjaci koristili su jednostavne jednoalfabetske algoritme šifrovanja sa zamjenom znakova.

Najznačajniji napredak u kriptanalizi do Drugog svjetskog rata pripisuje se arapskom matematičaru al-Kindi-u, koji je oko 800. g. n. e., vjerovatno religiozno motiviran tekstualnom analizom Kur'an-a, izumio tehniku **frekventne analize** za razbijanje jednoalfabetskih algoritama šifrovanja sa zamjenom znakova. U osnovi, svi su algoritmi šifrovanja bili ranjivi na ovu tehniku do izuma višealfabetskih algoritama, za čiji izum se smatra zaslužnim Alberti (oko 1456. g.), mada postoje indikacije da su isti bili poznati na Bliskom istoku i do 500 godina ranije.

Za vrijeme Drugog svjetskog rata, zaraćene strane su imale različite ručne, mehaničke ili elektromehaničke mašine za šifrovanje. Njemačka vojska se značajno oslanjala na upotrebu elektromehaničke mašine pod nazivom **Enigma**. Matematičar Marijan Rejewsky je u decembru 1932. u poljskom Birou za kriptografiju uspio rekonstruisati njemačku vojnu Enigma mašinu, koristeći matematička znanja i ograničenu dokumentaciju koju je dostavio kapetan Gustave Bertrand iz francuske vojne obavještajne službe. Ovaj događaj se smatra najvećim napretkom u kriptanalizi u 1000 godina ili više. Rejewsky je sa kolegama nastavio sa radom na praćenju evolucije Enigma mašine i usavršavanjem sposobnosti dešifrovanja poruka šifrovanih ovom mašinom. Pred sami rat upoznali su francuske i britanske kolege sa svojim radom, te su 1. septembra 1939., netom nakon izbijanja rata evakuirani iz Poljske da bi kasnije radili nedaleko od Pariza, u saradnji sa britanskim kriptolozima iz Bletchley parka do kraja rata. Razbijanje šifre

Enigma mašine je imalo neprocjenjiv značaj za pobjedu Savezničkih snaga u Drugom svjetskom ratu. Saveznici su uspjeli da probiju i sve mašine japanske vojske u određenoj mjeri, dok ne postoje podaci o razbijanju američke mašine SIGABA ili britanske TypeX za vrijeme rata.

Era moderne kriptografije zapravo počinje radom **Claude Shannon-a**. Njegovi radovi o teoriji informacija i teoriji komunikacije objavljeni nakon Drugog svjetskog rata su postavili čvrste teoretske temelje za razvoj kriptografije i kriptanalize. Tada je kriptografija manje-više postala domen rada isključivo tajnih vladinih organizacija i jako malo rada je javno objavljivano do sredine sedamdesetih, kada se mnogo šta promijenilo.

Prvi značajan događaj desio se 17. marta 1975. godine, kada je objavljen nacrt standarda pod nazivom **Data Encryption Standard (DES)** od strane IBM-a u Sjedinjenim američkim državama. To predstavlja prvi javni pokušaj razvijanja infrastrukture za sigurnu elektronsku komunikaciju za poslovne organizacije kao što su banke. 2001. godine DES je zbog svojih nedostataka zamijenjen novim standardom, pod nazivom **Advanced Encryption Standard (AES)**.

Drugi događaj, koji se desio 1976. godine, bio je objavljivanje rada “New Directions in Cryptography” od autora Whitfield Diffie-a i Martin Hellman-a. Ovaj rad uveo je koncept **kriptografije sa javnim ključem** kroz upotrebu asimetričnih ključeva i algoritama za rad sa istim. 1978. godine javno je opisan **RSA kriptografski algoritam** [1], a autori su Ron Rivest, Adi Shamir i Leonard Adleman. Naziv RSA potiče od prvih slova njihovih prezimena. Po prvi put u historiji, javnosti je bila dostupna visokokvalitetna kriptografija, koju nije mogao razbiti niko, uključujući vlade. Međutim, 1997. godine britanska obavještajna organizacija GCHQ je javnosti objavila da je kriptografiju sa javnim ključem izumio James H. Ellis i da su, tokom 70-ih godina, Diffie-Hellman i RSA algoritmi najprije razvijeni od strane Malcolm J. Williamson-a i Clifford Cocks-a, respektivno.

Dok se moderni kriptosistemi poput AES-a smatraju sigurnim, dešava se da i danas bivaju usvajani neadekvatni sigurnosni dizajni, te su poznati važni uspješni **kriptoanalitički napadi** na široko rasprostranjene kriptosisteme posljednjih godina. Među najznačajnijim su DES, prva enkripcijska šema WEP za bežični mrežni saobraćaj (Wi-Fi), Content Scrambling System koji se koristi za enkripciju i kontrolu upotrebe DVD-a ili A5/1 i A5/2 sistemi šifrovanja za GSM mobilnu telefoniju. Svi ovi sistemi koriste tajne ključeve. Do danas, ni jedna matematska ideja koja leži u pozadini kriptografije sa javnim ključem nije dokazano “sigurna”, tako da bi budući napredak mogao da učini sisteme koji se oslanjaju na istu nesigurnim. Iako tek rijetki iz struke predviđaju takav razvoj događaja, preporučena **veličina sigurnosnih ključeva se postepeno povećava**, kako sve veća računarska snaga potrebna za razbijanje istih postaje jeftinija i dostupnija [D1].

2.3. Osnovni pojmovi

Za većinu ljudi, kriptografija je nauka o očuvanju privatnosti komunikacije. I zaista, kako smo vidjeli, zaštita povjerljivosti komunikacije je glavni način primjene kriptografije većim dijelom njene historije. Međutim, kako je polje kriptografije napredovalo, granica između onoga što jeste i onoga što nije kriptografija postepeno je bivala tanja.

Danas se **kriptografija** može opisati kao nauka o tehnikama i primjenama koje zavise od postojanja teških (matematskih) problema. **Kriptoanaliza** je nauka o načinima kompromitovanja kriptografskih mehanizama. **Kriptologija** je nauka koja obuhvata kriptografiju i kriptoanalizu [4].

Enkripcija/šifrovanje je transformacija podataka u oblik koji skoro nemoguće ili nemoguće čitati/razumjeti bez posjedovanja odgovarajućeg znanja (ključa). Svrha enkripcije jeste da osigura privatnost sakrivanjem informacije od bilo koga kome ista nije namijenjena, čak i onome ko ima

pristup šifrovanom podatku. **Dekripcija/dešifrovanje** je postupak obrnut šifrovanju; to je transformacija šifrovanih podataka u smislen oblik. Šifrovanje i dešifrovanje su matematske funkcije, a upotreba računara u oba procesa je neizostavna [5].

Šifrovanje i dešifrovanje generalno zahtijevaju upotrebu neke tajne informacije, koju nazivamo **ključem**. Memorijski prostor koji zauzima ključ naziva se **dužina ključa**. **Upravljanje ključevima** se bavi sigurnim generisanjem, distribucijom i čuvanjem ključeva.

Kriptografski algoritam je matematska funkcija koja se koristi za enkripciju i dekripciju (generalno, to su dvije različite međusobno povezane funkcije). **Kriptosistem** čine algoritam sa svim mogućim porukama, običnim i šifriranim, i svojim ključevima.

Moderna kriptografija ne predstavlja puko šifrovanje i dešifrovanje. **Autentifikacija** je jednako važan dio naših života kao i privatnost. To je bilo koji proces kojim je moguće dokazati i verificirati određenu informaciju. Nekada je potrebno verificirati porijeklo dokumenta, identitet pošiljaoca, vrijeme i datum kada je dokument poslan/potpisan, identitet računara ili korisnika i sl. **Digitalni potpis** je kriptografski način na koji se gore navedeno može verificirati. Digitalni potpis jednog dokumenta je informacija zasnovana kako na dokumentu, tako i na privatnom ključu onoga ko je potpisao dokument.

Koristimo autentifikaciju u svakodnevnom životu, npr. kada potpisujemo neki dokument, i, kako postepeno prelazimo u svijet elektronske komunikacije, potrebne su nam elektronske tehnike autentifikacije. Kriptografija nam pruža mehanizme za te tehnike. Digitalni potpis povezuje dokument sa osobom koja posjeduje odgovarajući ključ, dok digitalni vremenski pečat (timestamp) veže dokument sa njegovim nastankom u dato vrijeme. Ovi mehanizmi se mogu koristiti za kontrolu pristupa dijeljenom hard disku, ustanovama visoke sigurnosti ili kodiranim kanalima satelitske televizije.

2.4. Kriptografija sa javnim ključem

Algoritme (mehanizme) šifrovanja kod kojih se ključ za enkripciju može izračunati iz ključa za dekripciju (to su često identični ključevi), nazivamo **simetričnim mehanizmima šifrovanja (kriptografijom)** ili **mehanizmima šifrovanja (kriptografijom) sa tajnim ključem**. Primjeri uključuju AES, Blowfish, DES, Triple DES, Serpent, Twofish i mnoštvo drugih.

Problem kod simetričnih algoritama predstavlja **razmjena ključa** između stranaka koje žele da uspostave komunikaciju, jer ova razmjena zahtijeva siguran način prenosa. Također, problem predstavlja i **upravljanje ključevima**, jer je za komunikaciju sa svakom strankom potreban različit ključ. Tako u mreži od n korisnika za ostvarivanje tajne komunikacije upotrebom simetričnog kriptosistema, svaka dva korisnika moraju razmijeniti ključeve, što predstavlja

$$\binom{n}{2} = \frac{n \cdot (n-1)}{2}$$

ključeva.

Drugu vrstu algoritama, kod kojih se koriste različiti ključevi za enkripciju i za dekripciju i jedan se iz drugog ne može izračunati (bar u razumnom vremenu) nazivamo **asimetričnim mehanizmima šifrovanja (kriptografijom)** ili **mehanizmima šifrovanja (kriptografijom) sa javnim ključem**. RSA je jedan primjer kriptosistema sa javnim ključem, dok su drugi značajniji primjeri ElGamal, Pallier ili Cramer-Shoup.

Kriptosistemi sa javnim ključem se prvenstveno koriste u dvije svrhe: šifrovanje i digitalni potpis. U ovom sistemu, svaka osoba dobija po jedan par ključeva, od kojih se jedan zove **javni ključ**, a drugi **tajni ključ**. Javni ključ se objavljuje, dok se tajni čuva. Ovo rješava problem razmjene ključa kod asimetričnih kriptosistema, jer sva komunikacija zahtijeva samo javne ključeve, a tajni ključevi se nikad ne šalju i ne dijele. Pri upotrebi ovog sistema nije više potrebno vjerovati sigurnosti nekog načina komunikacije. Jedini zahtjev jeste da javni ključevi budu vezani za odgovarajućeg korisnika na siguran (autentificiran) način. Ovim je također riješen i problem upravljanja ključevima sa aspekta broja ključeva u zavisnosti od broja učesnika u komunikaciji. Tako je kod ovog kriptosistema potrebno imati $2n$ ključeva, gdje je n broj korisnika.

2.4.1. Enkripcija

Neka imamo dvije osobe, Alice i Bob. Kada Alice želi da pošalje tajnu poruku Bobu, ona pronađe Bobov javni ključ u nekom direktoriju ključeva, iskoristi ga da šifruje poruku i pošalje istu. Bob tada upotrijebi svoj privatni ključ da dešifruje poruku i pročita je. Niko ko sluša ne može dešifrovati poruku. Svako može poslati šifrovanu poruku Bobu, ali **samo Bob je može pročitati** (jer samo on zna svoj privatni ključ).

2.4.2. Digitalni potpis

Pretpostavimo da Alice želi da pošalje potpisanu poruku Bobu. Prvi korak je da se primijeni hash funkcija nad porukom, kreirajući hash kôd poruke. Ovaj kôd se smatra znatno kraćim od originalne poruke. Ustvari, zadatak hash funkcije jeste da poruku proizvoljne dužine skрати na fiksnu dužinu. Za kreiranje digitalnog potpisa Alice mora šifrovati hash kôd poruke vlastitim privatnim ključem. Alice sada šalje Bobu šifrovani hash kôd poruke kao i samu poruku, koju može i ne mora šifrovati Bobovim javnim ključem. Kako bi Bob autentificirao potpis, on mora primijeniti istu hash funkciju nad primljenim (i, ako je potrebno, dešifrovanim) dokumentom i uporediti taj kôd sa kôdom koji dobije kada šifrovani hash kôd primljen od Alice dešifruje njenim javnim ključem. Ako su oba kôda ista, verifikacija je uspješna. To znači da je **poruku poslala Alice**, i da **poruka nije izmijenjena**.

2.5. Zaključak

Iako je stara hiljadama godina, **kriptografija** postaje jako značajna tek u posljednjih stotinjak godina, kroz Prvi i posebno Drugi svjetski rat. Međutim, njena civilna upotreba u značajnijoj mjeri datira tek od 70-ih godina 20. vijeka, sa pojavom **kriptografije sa javnim ključem**. Od tada pa do danas neprekidno se radi, s jedne strane na stvaranju novih i unapređenju postojećih kriptosistema, a sa druge strane na razbijanju istih.

Teško bi bilo ostvariti **zaštitu privatnosti** i **autentifikaciju** bez dvije ključne prednosti moderne kriptografije: enkripcije i digitalnog potpisa. Ova dva jednostavna koncepta modernom čovjeku umnogome olakšavaju život, omogućavajući mu obavljanje sigurnih bankovnih transakcije sa daljine, očuvanje privatnosti komunikacije, bilo putem Interneta ili mobilne telefonije, zaštitu pristupa ustanovama visoke sigurnosti itd.

3. RSA kriptosistem

3.1. Uvod

Od svih poznatih kriptografskih algoritama sa javnim ključem, **RSA** je doživio najširu upotrebu i najveći broj implementacija, za različite uređaje. To je posljedica relativne jednostavnosti i razumljivosti algoritma. Međutim, uprkos širokoj rasprostranjenosti, sigurnost RSA algoritma do danas nije matematski dokazana niti opovrgnuta.

U ovom poglavlju biti će riječi o generisanju RSA ključeva, RSA enkripciji i dekripciji, kao i o digitalnom potpisu¹. Zatim ćemo, nakon predstavljanja jednostavnog primjera, obratiti pažnju na brzinu i osnove sigurnosti RSA.

3.2. Generisanje ključeva

Kako bi dva subjekta komunicirala koristeći bilo koji kriptografski algoritam sa javnim ključem, pa tako i RSA, potrebna su im po dva ključa, jedan javni i jedan privatni. U RSA kriptosistemu, ovi **ključevi** se **generišu** na sljedeći način [2]:

1. Izabrati slučajnim izborom dva velika prosta broja p i q , takva da je $p \neq q$. Prosti brojevi p i q mogu imati dužinu od po , naprimjer, 512 bita.
2. Izračunati n po jednačini $n = p \cdot q$.
3. Izabrati mali neparan cijeli broj e uzajamno prost sa vrijednošću Eulerove funkcije $\varphi(n)$, koja za n iznosi $(p - 1)(q - 1)$, i vrijedi $1 < e < \varphi(n)$.
4. Izračunati d kao inverzni element za operaciju množenja broju e , po modulu $\varphi(n)$.
5. Objaviti uređeni par $P = (e, n)$ kao javni RSA ključ.
6. Sačuvati u tajnosti uređeni par $S = (d, n)$ kao privatni RSA ključ.

Kao što možemo primijetiti, ključevi se sastoje od uređenih parova. Drugi element para, koji je zajednički i za javni i za tajni ključ naziva se **modul**, dok se prvi element, po kome se ključevi razlikuju, naziva **eksponent**.

U praksi, u trećem koraku algoritma čest izbor za e su brojevi 3, 17 i 65537 (tj. $2^{16} + 1$) [D3]. To su Fermatovi prosti brojevi, koji se ponekad još nazivaju i F_0 , F_2 i F_4 , respektivno. Ovi brojevi su čest izbor jer se njihovom upotrebom ubrzava proces stepenovanja po modulu, koji se, kako ćemo kasnije pokazati, koristi pri enkripciji i dekripciji. Međutim, pokazano je kako je sigurnost RSA kriptosistema ugrožena ako se koristi mali javni eksponent e , posebno za $e = 3$ [4]. Članak [12] američkog instituta za standardizaciju NIST ne dozvoljava javne eksponente manje od 65537, iako ne navodi razlog za ovu restrikciju.

U nastavku ćemo opisati procese enkripcije, dekripcije i digitalnog potpisa. Kao pretpostavku uzmimo da postoje dva subjekta, nazovimo ih Alice i Bob, koji žele da ostvare

¹ Algoritme koji se koriste za navedene postupke detaljnije ćemo objasniti u narednom poglavlju.

komunikaciju. Alice ima svoj javni ($P_A = (e_A, n_A)$) i tajni ($S_A = (d_A, n_A)$) ključ, kao i Bob ($P_B = (e_B, n_B)$, $S_B = (d_B, n_B)$). Poruku koju Alice šalje označimo sa M , a šifrovanu poruku označimo sa C .

3.3. Enkripcija

RSA koristi relativno jednostavan algoritam za **šifrovanje**, ali koji zahtijeva značajne računarske resurse za izračunavanje kada se koriste dovoljno dugi ključevi. Proces enkripcije sastoji se u sljedećem:

1. Alice želi da pošalje poruku M Bobu. Ona najprije pronađe Bobov javni ključ P_B i pretvori poruku M u cijeli broj m , takav da je $m < n_B$. Ako je $m \geq n_B$, tada se poruka ne pretvara u jedan veliki, nego u više manjih cijelih brojeva m_i , $i = 1..k$, gdje je k količina tih brojeva. Postupak pretvorbe poruke u broj(eve) je unaprijed dogovoren i reverzibilan.
2. Sada se dobiveni broj m šifrjuje prema izrazu 3.3.1, kako bi se dobila šifrovana poruka c .

$$c = m^{e_B} \bmod n_B$$

Izraz 3.3.1: RSA
enkripcija

Ova matematska operacija naziva se stepenovanje po modulu. Alice sada može da pošalje šifrovanu poruku c Bobu [2].

3.4. Dekripcija

Proces **dešifrovanja** je analogan procesu šifrovanja i proceduralno i po pitanju brzine izvršavanja. Kada Bob primi šifrovanu poruku c od Alice, sljedećim postupkom može da je dešifrjuje:

1. Bob može da iz šifrovane poruke c izračuna cijeli broj m na sljedeći način:

$$m = c^{d_B} \bmod n_B$$

Izraz 3.4.1: RSA
dekripcija

2. Sada kada je izračunao cijeli broj m , dovoljno je da ga pretvori u poruku M procesom suprotnim od onog koji je Alice koristila za pretvorbu poruke M u cijeli broj m . Time je poruka dešifrovana [2].

3.5. Digitalni potpis

RSA digitalni potpis se kreira na sljedeći način:

1. Alice kreira digitalno potpisanu poruku C_M tako što stepenuje originalnu poruku M na stepen d_A po modulu n_A , ili kraće:

$$C_M = M^{d_A} \bmod n_A .$$

Izraz 3.5.1

Možemo primijetiti da je ovo identično postupku šifrovanja poruke M Alice-inim tajnim ključem S_A . C_M nazivamo digitalno potpisanom porukom M. Alice sada može poslati ovu poruku Bobu.

2. Da bi dešifrovao digitalno potpisanu poruku C_M , Bob treba da je stepenuje na stepen e_A po modulu n_A , ili kraće:

$$M = C_M^{e_A} \bmod n_A .$$

Izraz 3.5.2

Vidimo da je ovaj postupak zapravo dešifrovanje digitalno potpisane poruke C_M Alice-inim javnim ključem.

Na ovaj način Bob ostvaruje dvije značajne **prednosti**:

1. siguran je da je poruka **neizmijenjena**;
2. siguran je da je **pošiljalac** poruke Alice;

tj. Bob može biti siguran u **autentičnost** poruke [3].

Ipak, svako ko ima pristup Alice-inom javnom ključu (koji je javno objavljen) može da dešifruje i pročita digitalno potpisane poruke koje Alice šalje. Zato, ako je potrebno sačuvati povjerljivost ovih poruka, tada Alice kreiranu digitalno potpisanu poruku C_M prije slanja još treba šifrovati Bobovim javnim ključem.

3.6. *Jako jednostavan primjer upotrebe*

U nastavku je predstavljen jako **jednostavan primjer upotrebe**, adaptiran iz [1], gdje su brojevi sa kojima se radi mali i nedovoljni za praktičnu upotrebu. Zato primjer ima čisto edukativnu svrhu. Predstavljanje primjera iz prakse (sa ključem od npr. 1024 bita) bi značajno umanjilo čitljivost i razumljivost.

U ovom primjeru ćemo generisati jedan par RSA ključeva za Boba, koji se sastoji iz javnog P_B i tajnog S_B ključa. Zatim će Alice šifrovati i poslati Bobu poruku “920”, koju će Bob dešifrovati i pročitati. Izabrana je numerička poruka a ne znakovna, kako bi primjer bio jasniji.

3.6.1. **Generisanje ključeva**

Neka smo pronašli proste brojeve $p = 47$, $q = 59$. Sada možemo izračunati modul javnog i tajnog ključa n kao $n = p \cdot q = 47 \cdot 59 = 2773$. Vrijednost Eulerove funkcije ϕ za broj $n = 2773$ iznosi $\phi(n) = (47 - 1)(59 - 1) = 2668$. Broj e , takav da je $1 < e < \phi(n)$ i da je e uzajamno prost sa $\phi(n)$ iznosi $e = 17$. Broj d , izračunat prema gore navedenoj formuli, iznosi $d = 157$. Sada imamo sve potrebne brojeve za kreiranje ključeva.

Javni ključ P_B je uređeni par (17, 2773), dok je **tajni ključ** S_B jednak (157, 2773). Bob objavljuje svoj javni ključ.

3.6.2. Enkripcija

Alice je napisala poruku $M = 920$ i pronašla Bobov javni ključ P_B . Sada ona pristupa **enkripciji** poruke M prema formuli:

$$C = M^{e_B} \bmod n_B = 920^{17} \bmod 2773 = 948 \bmod 2773 .$$

Izraz 3.6.1

Dakle, šifrovana poruka C iznosi 948. Alice šalje C Bobu.

3.6.3. Dekripcija

Bob je primio šifrovanu poruku $C = 948$ od Alice i pristupa **dekripciji**. To se vrši prema formuli:

$$M = C^{d_B} \bmod n_B = 948^{157} \bmod 2773 = 920 \bmod 2773 .$$

Izraz 3.6.2

Vidimo da je Bob primio upravo onu poruku koju mu je Alice poslala, tj. 920.

3.7. Sigurnost

Sigurnost RSA kriptosistema se uveliko oslanja na težinu (u matematskom smislu) problema **faktorisanja velikih cijelih brojeva**. Ako bi neko bio u stanju da faktoriše modul n javnog ključa, tada bi on mogao da izračuna tajni ključ iz javnog, koristeći poznavanje faktora p i q na isti način kao i onaj ko je kreirao ključeve. Dakle ako je faktorisanje velikih cijelih brojeva lako, tada je probijanje RSA kriptosistema lako. Nije dokazano da je probijanje RSA teško ako je faktorisanje velikih cijelih brojeva teško. Međutim, ni nakon tri decenije istraživanja nije pronađen lakši metod za razbijanje RSA kriptosistema sa javnim ključem od faktorisanja modula n . Faktorisanje velikih cijelih brojeva je iznenađujuće težak matematski problem. Slučajno izabirući i množeći dva 1024-bitna broja, moguće je kreirati javni ključ koji ne može biti “razbijen” u razumnom vremenskom periodu sa trenutno javno raspoloživom tehnologijom. U odsustvu značajnijeg napretka u dizajnu algoritama teorije brojeva, i kada se implementira sa pažnjom i prema preporučenim standardima, RSA kriptosistem je sposoban pružiti visok stepen sigurnosti u praksi [2].

Od 2005. godine, najveći broj faktorisan algoritmom za faktorisanje opće upotrebe je veličine 663 bita [D9]. Da bi se ostvarila sigurnost RSA kriptosistema, trenutno se preporučuje rad sa cijelim brojevima dužine **2048 bita** ili više [D4].

3.8. Brzina

Iako rješava dva osnovna nedostatka simetričnih kriptosistema, asimetrični kriptosistemi uvode jedan novi nedostatak, **smanjenu brzinu enkripcije/dekripcije**. RSA nije izuzetak.

Zbog efikasnosti, RSA se često koristi u jednom “hibridnom” obliku kriptosistema u sprezi sa brzim simetričnim algoritmima. U ovom scenariju, kada Alice želi da privatno komunicira sa Bobom, ona treba da slučajno izabere jedan ključ K za brzi simetrični algoritam šifrovanja i šifruje poruku M koristeći K , dobivši C . Tako je C otprilike veličine M , ali je K prilično kratak. Tada ona šifruje K koristeći Bobov javni RSA ključ. Budući da je K kratak, dešifrovanje poruka šifrovanih uz pomoć K je puno brže nego dešifrovanje istih poruka šifrovanih koristeći javni RSA ključ. Ona tada pošalje C i prethodno šifrovani ključ K Bobu, koji to dekriptuje da bi dobio K , koji koristi da dekriptuje C , i tako dobija M [2].

3.9. Zaključak

RSA kriptografski algoritam, objavljen 1976., koristi se kod kriptografije sa javnim ključem kako u svrhe **enkripcije/dekripcije**, tako i u svrhe **digitalnog potpisa**. Pri upotrebi RSA svaki korisnik ima **dva ključa**, javni, koji se objavljuje, i tajni, koji se čuva u tajnosti. Ovi ključevi trebaju biti jedinstveni za sve subjekte.

RSA enkripcija i dekripcija su matematske operacije nad velikim brojevima. Osoba A može povjerljivo poslati poruku osobi B tako što je enkriptuje javnim ključem osobe B , a osoba B istu dekriptuje svojim privatnim ključem.

Digitalni potpis poruke M od osobe A omogućava onome ko ga verificira da bude siguran da je osoba A autor, da je poruka **autentična** i neizmijenjena.

Sigurnost RSA se zasniva na teškoći faktorisanja velikih cijelih brojeva, i povećava se sa upotrebom dužih ključeva. Trenutno je preporučena upotreba ključeva dužine 2048 ili više bita.

Brzina RSA pri enkripciji/dekripciji ili digitalnom potpisivanju je značajno manja nego brzina enkripcije/dekripcije kod simetričnih algoritama, ali taj se nedostatak može ukloniti **spregom obje vrste algoritama**.

4. Karakteristični algoritmi potrebni za implementaciju RSA kriptosistema

4.1. Uvod

RSA kriptosistem se, po pitanju zahtjevnosti implementacije, smatra jednim od jednostavnijih kriptosistema sa javnim ključem [5]. Međutim, iako pojedinačni koraci algoritma, opisani u odjeljcima 3.2, 3.3 i 3.4, na prvi pogled djeluju jednostavno jer su lako čitljivi i razumljivi, za njihovu implementaciju potrebno je najprije razumjeti **nekoliko karakterističnih matematskih problema**, a zatim pronaći i odabrati odgovarajuće **algoritme za njihovo rješavanje**.

Neki od tih problema su generisanje velikih prostih brojeva, stepenovanje i računanje inverznog elementa za množenje u modularnoj aritmetici.

U nastavku su opisani navedeni problemi, nabrojani poznatiji algoritmi za njihovo rješavanje, te opisani algoritmi koji su uvršteni u implementaciju programskog rješenja koje je tema ovoga diplomskog rada.

4.2. Stepenovanje u modularnoj aritmetici

U odjeljcima 3.3 i 3.4 vidjeli smo da se postupak enkripcije i dekripcije kod RSA kriptosistema svodi na stepenovanje u modularnoj aritmetici (izrazi 3.3.1 i 3.4.1). Stepenovanje je računarski zahtjevan problem kako u običnoj, tako i u modularnoj aritmetici.

Neka imamo izraz $a = b^c \bmod d$. Računanju a možemo prići na dva načina.

4.2.1. Naivni pristup

Najjednostavniji pristup se sastoji iz dva koraka:

1. stepenovanje b na stepen c ;
2. traženje ostatka dijeljenja tako dobijenog broja sa d .

Iako je ovaj način stepenovanja u modularnoj aritmetici jednostavan, kako za razumijevanje tako i za implementaciju, postoji jedan **veliki problem**. U odjeljku 3.7 smo vidjeli da se za RSA trenutno preporučuje upotreba ključeva dužine 2048 bita, što znači da tajni eksponent može imati dužinu do 2048 bita (broj reda 10^{616}). Dizanjem bilo kojeg prirodnog broja većeg od 1 na taj stepen dobili bi broj koji ne može stati u računarsku memoriju (broj atoma u vidljivom dijelu Svemira se procjenjuje na oko 10^{80}).

4.2.2. Stepenovanje uzastopnim kvadriranjem (repeated squaring)

Stepenovanje uzastopnim kvadriranjem se zasniva na ideji da se manje memorije zauzima ako se stepenovanje vrši **uzastopnim kvadriranjem**, te se nakon svakog kvadriranja računanje nastavlja koristeći **modul trenutnog rezultata**.

Pseudokod 4.2.1 prikazuje pseudokôd ovog algoritma, kojim se računa $a = b^c \bmod d$, prilagođen iz [2].

```

STEPENOVANJE-UZASTOPNIM-KVADRIRANJEM(b, c, d)
1 neka je  $c_k, c_{k-1}, \dots, c_0$  binarna reprezentacija eksponenta c
2  $e \leftarrow 1$ 
3 for  $i \leftarrow k$  downto 0 do
4      $e \leftarrow (e \cdot e) \bmod d$ 
5     if  $c_i = 1$  then
6          $e \leftarrow (e \cdot b) \bmod d$ 
7 return e

```

Pseudokod 4.2.1: Stepenovanje uzastopnim kvadriranjem

Najprije je potrebno eksponent c predstaviti u binarnom obliku (linija 1) i inicijalizirati pomoćnu varijablu e, u kojoj će biti smješten konačni rezultat, na vrijednost 1 (linija 2). Zatim se, za svaki bit eksponenta c u binarnom obliku, od najznačajnijeg do najmanje značajnog, izvršavaju sljedeći koraci:

1. pomoćnoj varijabli e se dodijeli ostatak dijeljenja kvadrata varijable e (koji možemo lako izračunati množeći e sa samim sobom) sa vrijednošću d (linija 4);
2. ako trenutni bit eksponenta c ima vrijednost 1, tada se pomoćnoj varijabli e dodijeli vrijednost ostatka dijeljenja umnoška $e \cdot b$ sa vrijednošću d (linija 6).

Na samom kraju algoritma rezultat je smješten u varijabli e (linija 7). Efektivno, ovaj algoritam u i-tom koraku izračuna vrijednost b stepenovanu na broj koji se dobije kada se uzme najznačajnijih i bita eksponenta c. To radi tako što privremeni rezultat u svakom koraku diže na stepen dva (ako je trenutni bit 0) ili tri (ako je trenutni bit 1). Radi uštede memorije, **pri svakom stepenovanju se uzima modul privremenog rezultata** i sa istim se nastavlja računanje.

Kvadriranje koje se obavlja u liniji 4 objašnjava naziv “stepenovanje uzastopnim kvadriranjem”. Ako su ulazni argumenti b, c i d brojevi dužine n bita, tada je ukupan broj aritmetičkih operacija potrebnih za ovaj algoritam $O(n)$, dok je ukupan broj potrebnih operacija nad bitima $O(n^3)$.

4.3. Generisanje velikih prostih brojeva

U koraku 1 odjeljka 3.2 vidjeli smo da je za generisanje RSA ključeva potrebno najprije generisati dva prosta broja p i q, čiji umnožak n predstavlja modul javnog i tajnog RSA ključa. Da bi modul bio preporučene veličine od 2048 bita, brojevi p i q trebaju biti približno po 1024 bita

veličine. To znači da trebaju imati preko 300 cifri, te ih zato zovemo **velikim prostim brojevima**.

Da bi generisali velike proste brojeve, poslužit ćemo se činjenicom da su prosti brojevi prilično **gusto raspoređeni**, pa je dovoljno da generišemo slučajne brojeve odgovarajuće veličine i da ih testiramo na prostost. Funkcija raspodjele prostih brojeva $\pi(n)$ daje broj prostih brojeva manjih ili jednakih n . Npr., $\pi(11) = 5$, jer postoje četiri prosta broja manja ili jednaka 11: 2, 3, 5, 7 i 11. Izraz 4.3.1 (teorema o prostim brojevima) daje korisnu aproksimaciju vrijednosti $\pi(n)$ za neko n .

$$\lim_{n \rightarrow \infty} \frac{\pi(n)}{n / \ln(n)} = 1$$

Izraz 4.3.1: Teorema o prostim brojevima

Koristeći teoremu o prostim brojevima možemo odrediti vjerovatnoću da je slučajno izabran cijeli broj n prost kao $1 / \ln(n)$. To znači da ćemo u prosjeku testirati $\ln(n)$ cijelih brojeva izabranih slučajno iz blizine broja n , kako bi pronašli prost broj jednake dužine. Npr. kako bi pronašli slučajan broj veličine 1024 bita (što je čest zadatak kod rada sa RSA) potrebno je testirati otprilike $\ln(2^{1024}) \approx 710$ slučajno izabranih 1024-bitnih cijelih brojeva. (Ako testiramo samo neparne brojeve, onda je to polovina ukupnog iznosa).

Sada kada smo **sveli problem generisanja velikih prostih brojeva na problem testiranja prostosti**, možemo razmotriti postojeće algoritme koji rješavaju taj problem.

4.3.1. Naivni pristup

Najjednostavniji pristup testiranju prostosti broja n bio bi pokušaj dijeljenja n svim prirodnim brojevima od 2 do $\lfloor \sqrt{n} \rfloor$, izuzev parnih brojeva većih od 2. Ako se pokaže da je n djeljiv bilo kojim od tih brojeva, onda je n složen, u suprotnom n je prost broj. Ako pretpostavimo da se dijeljenje izvršava u vremenu $O(1)^2$, tada je najgore vrijeme $\theta(\sqrt{n})$, što zapravo predstavlja eksponencijsku složenost.

Dakle, ovaj pristup je praktičan samo kada je n jako malo ili ima bar jedan mali faktor. Međutim, prednost ovog pristupa je što se istim može, osim testiranja prostosti, otkriti i jedan faktor broja ako je isti složen [2], [D6]. To ipak za primjene generisanja velikih prostih brojeva nije bitno.

4.3.2. Miller-Rabinov probabilistički test prostosti

Miller-Rabinov (ili Rabin-Millerov) probabilistički test prostosti razvio je Rabin [6], a zasnovan je na idejama Miller-a [7]. Ovaj algoritam predstavlja brz metod za testiranje prostosti broja sa proizvoljno malom greškom. Algoritam se zasniva na generiranju slučajnih brojeva, i zbog te osobine spada u grupu algoritama koji se zovu **randomizirani algoritmi**. Ovo je također i **probabilistički algoritam**, što znači da dovodi do rješenja samo sa određenom vjerovatnoćom [11].

2 Ovo nije izvedivo ako dijelimo n -bitne brojeve na računaru sa širinom riječi koja je manja od n bita. Komercijalno dostupne računarske arhitekture danas imaju širinu riječi od 64 bita, dok se brojevi koje trebamo testirati na prostost kreću oko 1024 bita.

Kako bi opisali Miller-Rabinov test, najprije ćemo uvesti neke matematske tvrdnje. Za prost broj p , $2 < p$, ne postoji takav cijeli broj x , $x \neq \pm 1$, za koji vrijedi $x^2 \equiv 1 \pmod{p}$, tj. ne postoje netrivialni kvadratni korijeni broja 1 po modulu p (trivialni korijeni su 1 i -1).

Neka je p neparan prost broj. Tada $n - 1$ možemo rastaviti kao $2^s \cdot d$, gdje je s pozitivan cijeli broj, a d je neparan. Za svaki cijeli broj a , vrijedi jedan i samo jedan od izraza (4.3.2, 4.3.3):

$$ad \equiv 1 \pmod{n}, \text{ ili}$$

Izraz 4.3.2

$$a^{2^r \cdot d} \equiv -1 \pmod{n}, \text{ za neko } r, 0 \leq r \leq s-1.$$

Izraz 4.3.3

Ovo vrijedi prema **Maloj Fermatovoj teoremi**, po kojoj za svaki prost broj p vrijedi izraz 4.3.4:

$$a^{p-1} \equiv 1 \pmod{p}, 0 < a < p.$$

Izraz 4.3.4: Mala Fermatova
teorema

Prema navedenom, ako nastavimo računati kvadratne korijene iz a^{n-1} , rezultat će biti 1 ili -1. Ako dobijemo -1, tada je druga jednakost tačna i završili smo. Ako ne dobijemo kao rezultat -1 tada prva jednakost mora biti tačna.

Miller-Rabinov test se zasniva na kontrapozitivu gore navedene tvrdnje. Dakle, ako uspijemo naći takvo a da vrijede oba izraza (4.3.5, 4.3.6):

$$a^d \not\equiv 1 \pmod{n} \text{ i } a^{2^r \cdot d} \not\equiv -1 \pmod{n}, \text{ za svako } 0 \leq r \leq s-1,$$

Izraz 4.3.5

Izraz 4.3.6: Uslovi za
Miller-Rabin

tada testirani broj p sigurno nije prost, i broj a se naziva **svjedokom složenosti** broja p . Međutim, ako broj a zadovoljava izraz 4.3.6, to još uvijek nije garancija da je broj p prost broj. Tada se naziva **jakim lašcem**, a n **jakim pseudoprostim brojem** po bazi a [D5]. Termin “jaki lažac” se odnosi na slučaj kada n nije prost, ali je izraz 4.3.6 ispunjen kao kod prostog broja.

Za svaki neparan složen broj n postoji mnogo svjedoka a , međutim nije poznat jednostavan način generisanja istih. Zato je potrebno test učiniti **probabilističkim**: slučajno izaberemo a i provjerimo da li isti je svjedok složenosti n . Ako je n složen, većina izabranih a će biti svjedoci i test će proglasiti n složenim brojem sa velikom vjerovatnoćom. Međutim, postoji mala vjerovatnoća da je izabrani a jaki lažac za n . Vjerovatnoću takve greške možemo proizvoljno smanjiti ponavljanjem testa za više nezavisno izabranih a . [8] daje više informacija o tome.

Pseudokod 4.3.1 prikazuje dvije funkcije prilagođene iz [2] koje implementiraju opisani Miller-Rabin probabilistički algoritam za testiranje prostosti. Broj koji testiramo označen je sa n , broj iteracija sa z , dok se privremena varijabla korištena za stepenovanje u i -tom koraku algoritma označava sa x_i .

Funkcija **MILLER-RABIN** je probabilistički test dokaza da je n složen broj. Glavna petlja, koja počinje od linije 1, bira z slučajnih vrijednosti za a . Ako je izabrana vrijednost a svjedok složenosti broja n (što se testira u liniji 3 pozivom funkcije **SVJEDOK**), odmah se vraća rezultat

SLOŽEN, u što možemo biti sigurni. Ako se ne pronade svjedok u z pokušaja, tada **MILLER-RABIN** pretpostavlja da je broj n prost. Kao što ćemo vidjeti, vjerovatnoću prostosti broja n moguće je kvantificirati.

```

MILLER-RABIN( $n, z$ )
1 for  $j \leftarrow 1$  to  $z$  do
2    $a \leftarrow$  slučajan broj iz segmenta  $[1, n - 1]$ 
3   if SVJEDOK( $a, n$ ) then
4     return SLOŽEN ▷ Sigurno.
5 return PROST ▷ Skoro sigurno.

SVJEDOK( $a, n$ )
1 neka je  $n - 1 = 2^s d$ , gdje je  $s \geq 1$  i  $d$  je neparan broj
2  $x_0 \leftarrow$  STEPENOVANJE-UZASTOPNIM-KVADRIRANJEM( $a, d, n$ )
3 for  $i \leftarrow 1$  to  $s$  do
4    $x_i \leftarrow x_{i-1}^2 \bmod n$ 
5   if  $x_i = 1$  and  $x_{i-1} \neq 1$  and  $x_{i-1} \neq n - 1$  then
6     return TRUE
7 if  $x_s \neq 1$  then
8   return TRUE
9 return FALSE

```

Pseudokod 4.3.1: Miller-Rabin algoritam

Funkcija **SVJEDOK** računa $a^{n-1} \bmod n$ tako što najprije izračuna vrijednost $x_0 = a^d \bmod n$ u liniji 2, a zatim kvadrira taj rezultat s puta uzastopno u for petlji u linijama 3 do 6. Pri svakom kvadriranju u liniji 4, funkcija se može završiti ako se u liniji 5 otkrije netrivialni kvadratni korijen broja 1 po modulu n . U tom slučaju se vraća vrijednost TRUE. Linije 7 i 8 vraćaju TRUE ako je vrijednost izračunata za $x_i = a^{n-1} \bmod n$ različita od 1. U slučaju da nije vraćen TRUE, linija 9 vraća FALSE.

Što veći broj baza a testiramo, to je veća tačnost testa. Može se pokazati da za bilo koji neparan složen broj n , najmanje $3/4$ baza a predstavlja svjedoke složenosti [6]. Ako je n složen broj, tada Miller-Rabin test proglašava n vjerovlatnim prostim brojem **sa vjerovatnoćom od najviše 4^{-z}** [D5], gdje je z broj iteracija algoritma.

Ako je broj n dug m bita, a z broj iteracija, **MILLER-RABIN** zahtijeva $O(z \cdot m)$ aritmetičkih operacija i $O(z \cdot m^3)$ operacija sa bitima, jer asimptotski ne zahtijeva više posla od z stepenovanja po modulu. Zato je ovo najbrži poznati probablistički test prostosti, do u konstantan faktor [2].

4.4. Računanje inverznog elementa za množenje u modularnoj aritmetici

U koraku 4 odjeljka 3.2 govori se o potrebi računanja broja d kao inverznog elementa za množenje broju e , po modulu $\phi(n)$. Matematski, to možemo izraziti kao:

$$e \cdot d \equiv 1 \pmod{\phi(n)},$$

Izraz 4.4.1

odnosno $\phi(n) \mid (e \cdot d - 1)^3$. Ovi izrazi vrijede samo ako postoji cijeli broj $-x$, takav da je $e \cdot d - 1 = -\phi(n) \cdot x$, odnosno da je:

$$e \cdot d + \phi(n) \cdot x = 1$$

Izraz 4.4.2

Ovakav oblik jednačine naziva se **linearna Diofantova jednačina sa dvije nepoznate**, d i x , i ima rješenje ako i samo ako vrijedi $\text{NZD}(e, \phi(n)) \mid 1^4$, odnosno ako i samo ako je $\text{NZD}(e, \phi(n)) = 1$. Dakle, da bi pronašli d moramo riješiti odgovarajuću linearnu Diofantovu jednačinu [11].

4.4.1. Prošireni Euklidov algoritam

Jedan jednostavan algoritam kojim je moguće riješiti linearne Diofantove jednačine sa dvije nepoznate, čija je varijanta prezentovana i preporučena u [1], naziva se prošireni Euklidov algoritam. Ovaj algoritam se zasniva na Euklidovom algoritmu, čiji opis daje pseudokôd 4.4.1, prilagođen iz [2].

```
EUKLID(a, b)
1 if b = 0 then
2   return a
3 else
4   return EUKLID(b, a mod b)
```

Pseudokod 4.4.1: Euklidov algoritam

Euklidov algoritam je vrlo star (opisan je u Euklidovoj knjizi “Elementi” oko 300. g. p. n. e., ali je moguće da je bio poznat i prije Euklida), ali efikasan metod za nalaženje najvećeg zajedničkog djelioca. Zasniva se na činjenici da vrijedi $\text{NZD}(a, b) = \text{NZD}(b, a \bmod b)$, gdje $a \bmod b$ označava ostatak pri dijeljenju broja a brojem b , što je implementirano u liniji 4. Na osnovu ovoga moguće je rekursivno računati $\text{NZD}(a, b)$. Rekurzija se završava kada vrijedi uslov $b = 0$ (linija 1), s obzirom da je $\text{NZD}(a, b) < b$ i $\text{NZD}(a, 0) = a$ [11].

Kada Euklidov algoritam primijenimo na dva n -bitna broja, tada algoritam izvršava $O(n)$ aritmetičkih ili $O(n^3)$ binarnih operacija (pod pretpostavkom da množenje i dijeljenje n -bitnih brojeva traje $O(n^2)$ operacija) [2].

Sada kada imamo algoritam koji može da efikasno izračuna $\text{NZD}(e, \phi(n))$, proširićemo ga tako da nam kao rezultat vrati dodatne informacije d i x , pa da budemo u stanju da predstavimo $\text{NZD}(e, \phi(n))$ kao $e \cdot d + \phi(n) \cdot x$. Time ćemo, uz uslov $\text{NZD}(e, \phi(n)) = 1$, konačno dobiti rješenje jednačine 4.4.1, a samim tim i problema računanja inverznog elementa za množenje u modularnoj aritmetici. Pseudokod 4.4.2 prikazuje prošireni Euklidov algoritam, adaptiran iz [2] ⁵.

3 Oznaka “|” označava binarnu relaciju “dijeli”. $a \mid b$ znači da a dijeli b , tj. b je djeljivo sa a bez ostatka.

4 NZD označava najmanji zajednički djelilac.

5 Iako nam za generisanje RSA ključa treba samo d , ovdje ćemo u cijelosti predstaviti prošireni Euklidov algoritam, koji pored d računa i x i f , prema izrazu $f = e \cdot d + \phi(n) \cdot x$.

```

PROŠIRENI-EUKLID(e,  $\phi(n)$ )
1 if  $\phi(n) = 0$  then
2   return (e, 1, 0)
3 (f', d', x')  $\leftarrow$  PROŠIRENI-EUKLID( $\phi(n)$ , e mod  $\phi(n)$ )
4 (f, d, x)  $\leftarrow$  (f', d', d' -  $\lfloor e/\phi(n) \rfloor \cdot x'$ )
5 return (f, d, x)

```

Pseudokod 4.4.2: Prošireni Euklidov algoritam

Ulazni argumenti e i $\phi(n)$, zajedno sa izlaznim argumentima d i x zadovoljavaju izraz 4.4.2, tj. jednačinu 4.4.1. Funkcija **PROŠIRENI-EUKLID** je varijacija funkcije **EUKLID**. Obje funkcije imaju u liniji 1 ekvivalentan uslov za završetak rekurzije. Ako je $\phi(n) = 0$, tada **PROŠIRENI-EUKLID** u liniji 2 vraća ne samo $f = e$, već i koeficijente $d = 1$ i $x = 0$, tako da zadovolje izraz $e = e \cdot d + \phi(n) \cdot x$. Ako je $\phi(n) \neq 0$, tada **PROŠIRENI-EUKLID** prvo izračuna trojku (f', d', x') tako da vrijedi $f' = \text{NZD}(\phi(n), e \bmod \phi(n))$ i

$$f' = \phi(n) \cdot d' + (e \bmod \phi(n)) \cdot x'.$$

Izraz 4.4.3:

Što se tiče funkcije **EUKLID**, u ovom slučaju imamo $f = \text{NZD}(e, \phi(n)) = f' = \text{NZD}(\phi(n), e \bmod \phi(n))$. Kako bi izračunali d i x prema izrazu $f = e \cdot d + \phi(n) \cdot x$, uvrstimo u izraz 4.4.3 izraze

$$f = f'; a \bmod n = a - \lfloor a/n \rfloor \cdot n.$$

Tako dobijamo konačan izraz iz linije 4:

$$f = \phi(n) \cdot d' + (e - \lfloor e/\phi(n) \rfloor) \cdot x' = e \cdot x' + \phi(n) \cdot (d' - \lfloor e/\phi(n) \rfloor) \cdot x'.$$

Pošto je broj rekurzivnih poziva u funkciji **EUKLID** jednak broju rekurzivnih poziva u funkciji **PROŠIRENI-EUKLID**, vrijeme izvršavanja ovih funkcija je jednako do u konstantan faktor i iznosi $O(\log(b))$ [2].

4.5. Zaključak

Implementacija RSA algoritma, iako na prvi pogled jednostavna, zahtijeva razumijevanje i rješavanje nekoliko matematskih problema, kao što su generisanje velikih prostih brojeva, stepenovanje i računanje inverznog elementa za množenje u modularnoj aritmetici. Kvalitet implementacije RSA uveliko zavisi od odabira odgovarajućih (najboljih poznatih) algoritama za rješavanje ovih problema.

Problem stepenovanja u modularnoj aritmetici se javlja pri RSA enkripciji i dekripciji. Naivni algoritam zahtijeva mnogo više bajta računarske memorije nego što je procijenjeno da postoji atoma u vidljivom dijelu Svemira. **Algoritam uzastopnog kvadriranja (engl. repeated squaring)** efikasno rješava ovaj problem u vremenu $O(n^3)$.

Pri generisanju RSA ključa javlja se problem **generisanja velikih prostih brojeva** (tipično veličine 1024 bita ili više). Najbolji danas poznati pristup ovom problemu je generisanje velikih slučajnih brojeva i testiranje istih na prostost. Naivni pristup ima eksponencijalnu složenost, tako da ne zadovoljava za praktičnu primjenu. Predstavljeni **Miller-Rabin algoritam** je najbrži poznati probabilistički test prostosti, do u konstantan faktor, sa vremenom izvršavanja $O(n^3)$ po iteraciji.

Prilikom generisanja RSA ključa javlja se još jedan matematski problem, **problem računanja inverznog elementa za množenje u modularnoj aritmetici**. Ovaj problem se uspješno rješava **proširenim Euklidovim algoritmom**, prilagođenom verzijom jako starog Euklidovog algoritma, sa vremenom izvršavanja $O(n^3)$.

5. Rad sa velikim cijelim brojevima

5.1. Uvod

U prethodnim poglavljima vidjeli smo da se, sa napretkom kriptografije, povećava i dužina kriptografskih ključeva, te da je danas preporučena vrijednost RSA kriptografskog ključa 2048 bita ili više. Danas komercijalno dostupne računarske arhitekture imaju širinu riječi od najviše 64 bita, što znači da je najveći broj sa kojim je moguće vršiti aritmetičke operacije $2^{64}-1=18446744073709551615$, dok brojevi reda 2^{2048} imaju oko 617 cifri.

Ovo ograničenje je stvorilo potrebu za implementiranjem posebnih struktura podataka koje bi omogućile **rad sa cijelim brojevima proizvoljne veličine**, ograničene dostupnom računarskom memorijom. Osnovni zahtjevi nad ovim strukturama podataka, nametnuti potrebama kriptografije, su mogućnost memorisanja velikih cijelih brojeva i efikasna implementacija aritmetičkih operacija nad istim.

5.2. Memorisanje

Računarska memorija je organizovana kao sekvenca memorijskih jedinica koje se zovu bajti. **Bajt** je najmanja jedinica memorije koju računar može da adresira (pročita ili pohrani). Svaki bajt se sastoji od osam bita. **Bit** je minimalna jedinica memorije, koja može da ima dva stanja: 0 ili 1. Možemo ga poistovijetiti sa ciframa broja a predstavljenog u binarnom sistemu. **Riječ** možemo definirati kao najveću količinu memorije koju računar koristi pri aritmetičkim operacijama. Na modernim komercijalnim računarskim arhitekturama, riječ ima širinu od **32 ili 64 bita** (4 ili 8 bajta).

Sve informacije u računar u prikazane su **u binarnom obliku**. Npr. cijeli brojevi su predstavljeni u binarnom sistemu, brojevi sa pomičnim zarezom su predstavljeni zakodirano u binarnom obliku, znakovi su predstavljeni kodirani u binarnom obliku itd. Također, svi podaci koji ne mogu stati u jednu riječ (kao što je većina datoteka i sl.) moraju se protezati sekvencijalno kroz onoliko uzastopnih riječi koliko je potrebno za njihovo memorisanje. Bilo kakva implementacija velikih cijelih brojeva mora biti prilagođena nabrojanim ograničenjima.

S obzirom na ograničenje da računar ne može vršiti aritmetičke operacije nad brojevima koji imaju više bita od širine riječi na datoj arhitekturi, ugrađene računarske instrukcije za sabiranje, oduzimanje, množenje i dijeljenje (i možda druge operacije kao što su stepenovanje, šiftovanje ili računanje ostatka pri dijeljenju) se ne mogu primijeniti nad tim brojevima kao cjelinom, već samo nad pojedinačnim dijelovima tih brojeva. Zato je dobra ideja **predstaviti ove brojeve kao uzastopne cifre u nekom brojnom sistemu**. Jedan od načina koji je ljudima najbliži jeste u dekadnom brojnom sistemu (brojni sistem sa bazom 10). Npr. broj stotinu dvadeset i tri se u dekadnom brojnom sistemu može napisati kao $a = 123_{10} = 1 \cdot 10^2 + 2 \cdot 10^1 + 3 \cdot 10^0$. Za računarsku upotrebu, međutim, bolje je koristiti binarnu predstavu brojeva, pa bi tako broj a u binarnom sistemu bio zapisan kao $a = 1111011_2 = 2^6 + 2^5 + 2^4 + 2^3 + 0 \cdot 2^2 + 2^1 + 2^0$ [3].

Što se tiče predstavljanja negativnih brojeva, dovoljno je uvesti dodatnu logičku varijablu koja bi označavala predznak datog broja. Možemo slijediti konvenciju da vrijednost “tačno” ove varijable, nazovimo je “pozitivnost”, označava pozitivan, a vrijednost “netačno” negativan broj.

6 Indeks n pored broja A označava da je broj A zapisan u brojnom sistemu sa bazom n . U ovom slučaju, broj 123 je zapisan u dekadnom brojnom sistemu.

Nulu po konvenciji možemo uzimati kao pozitivan broj.

U nastavku su opisana dva načina memorisanja velikih cijelih brojeva: binarno i decimalno.

5.2.1. Binarno

Binarna reprezentacija se obično odnosi na predstavljanje podataka u binarnom brojnom sistemu, no ovdje se nećemo ograničiti samo na binarni brojni sistem, već na bilo koji **brojni sistem čija je baza stepen broja dva** izuzev jedinice, npr. 2, 4, 8, 16 itd.

Pretvaranje brojeva između ovih brojnih sistema se svodi na grupisanje cifri. Tako broj $a = 123_{10} = 1111011_2$ možemo u brojnom sistemu sa bazom 4 predstaviti grupisanjem po $\log_2 4 = 2$ cifre, od najmanje značajne do najznačajnije, i tako dobijemo $a = ((1)_2 (11)_2 (10)_2 (11)_2)_4 = (1323)_4$. Isti ovaj broj u bazi 16 možemo predstaviti grupisanjem po $\log_2 16 = 4$ cifre kao $a = ((111)_2 (1011)_2)_{16} = 7D_{16}$.

Činjenica da je jednostavno pretvarati brojeve između brojnih sistema čija je baza stepen broja dva, zajedno sa činjenicom da je najmanja količina podataka koju računar može adresirati jedan bajt, tj. 8 bita, upućuje na to da je jedan efikasan način pohranjivanja velikih brojeva u memoriju koristeći bazu $2^8 = 256$. Tako svaka cifra broja odgovara jednoj lokaciji (bajtu) u memoriji, a susjedne cifre mogu biti na susjednim lokacijama u memoriji. Isti princip se može iskoristiti i sa bazama 2^{32} i 2^{64} , gdje veličina baze odgovara širini riječi kod 32-bitne i 64-bitne arhitekture, respektivno. Tako **susjedne riječi odgovaraju susjednim ciframa broja**.

Bitno je napomenuti da od izbora baze zavisi i efikasnost manipulacije sa ciframa datog broja. Naime, ako npr. izaberemo bazu 16 (gdje 4 bita predstavljaju jednu cifru), možemo se odlučiti za dva načina pohranjivanja brojeva: sa poravnanjem i bez poravnanja.

Pohranjivanje sa poravnanjem znači da u jednom bajtu čuvamo jednu 4-bitnu cifru, dok je ostatak ispunjen binarnim nulama. Ovakav sistem je efikasan za aritmetičku upotrebu, jer svaki dohvaćeni bajt predstavlja cifru koja je spremna za računanje, no na taj način imamo 4 neiskorištena bita na svaku cifru velikog broja, tj. na svaki upotrijebljeni bajt. Tako 50% upotrijebljenog memorijskog kapaciteta ostaje neiskorišteno. Npr. broj 2^{1024} se može zapisati u $\log_{16} 2^{1024} = \log_{16} 16^{256} = 256$ bajta.

Pohranjivanje bez poravnanja znači da u jednom bajtu čuvamo dvije 4-bitne cifre. Time je iskorištenost memorije 100%. Npr. broj 2^{1024} se može zapisati u $(\log_{16} 2^{1024})/2 = 256/2 = 128$ bajta. No svaki put pri dohvaćanju i-te cifre, ako ona zauzima donja 4 bita unutar dohvaćenog bajta, moramo ispuniti nulama gornja četiri bita, jer računar ne može da vrši aritmetičke operacije nad dijelovima bajta, već samo nad cijelim bajtima. Ako i-ta cifra pak zauzima gornja četiri bita, tada je potrebno ta gornja četiri bita pomjeriti (šiftati) za četiri mjesta udesno, kako bi postigli isti efekat. Ovim se značajno povećava broj računarskih instrukcija potrebnih za jednu aritmetičku operaciju, pa samim tim i potrebno vrijeme.

Dilema “sa ili bez poravnanja” važi i za ostale baze koje su manje od 2^8 , tj. 2, 4 i 8. Ova dilema se rješava izborom 2^8 , 2^{32} ili 2^{64} za bazu, zavisno od arhitekture. Maksimalna moguća vrijednost je obično najbolje rješenje, jer 2^8 , iako iskorištava cijeli memorijski prostor, zahtijeva više vremena za izračunavanje. To se dešava zbog činjenice da je brzina instrukcija za aritmetičke operacije čiji su operandi 8, 32 ili 64-bitni brojevi obično ista kod date arhitekture, pa je brže odjednom računati sa 64 nego sa 8 bita.

Nedostatak ovakvog načina prezentacije velikih cijelih brojeva je u tome što je programeru

koji se bavi implementacijom istog **teško manipulirati npr. 64-bitnim ciframa**, u odnosu na standardne dekadne cifre na koje je navikao. Ovaj problem se javlja pri programiranju aritmetičkih operacija (koje moraju biti implementirane tako da rade na “cifra po cifra” sistemu), pri otklanjanju grešaka u kodu, kada je potrebno imati detaljan uvid u svaku cifru broja koji se računa, zatim pri predstavljanju ovih cifara u ljudima razumljivom obliku (npr. za čitanje) itd. Zato je potrebna značajna količina vremena za implementaciju binarnog načina prezentacije. Također, izvorni kod programa koji koristi binarnu reprezentaciju nije jasan kao izvorni kod npr. decimalne reprezentacije.

Sa druge strane, manipulacija najvećim mogućim ciframa na datoj arhitekturi daje ubjedljivo **najkraće vrijeme izvršavanja i optimalnu memorijsku iskorištenost**.

5.2.2. Decimalno

Decimalni način reprezentacije odnosi se na pohranjivanje velikih cijelih brojeva na takav način da se **u jedan bajt smjesti jedna decimalna cifra** datog broja. Naravno, decimalne cifre se pohranjuju u binarnom obliku, tako da se kreću između 0000_2 i 1001_2 . Kako to iznosi 4 bita, a jedan bajt se sastoji od 8 bita, memorijska iskorištenost je manja od 50% (oko 41.5%), jer se čak ni prva četiri bita ne iskoriste maksimalno (heksadecimalne cifre se ne koriste). Tako se broj 2^{1024} može predstaviti u $\log_{10} 2^{1024} \approx 310$ bajta. Brzina računanja sa ovako predstavljenim brojevima je nešto manja nego kod binarne reprezentacije u bazi 16 sa poravnanjem, jer je baza manja.

Iako niska memorijska iskorištenost i relativno mala brzina računanja čine ovakav pristup nepoželjnim za praktičnu upotrebu, programiranje decimalnog načina reprezentacije velikih brojeva je značajno jednostavnije u odnosu na binarno. To možemo zahvaliti upotrebi dekadnog brojnog sistema, koji je programeru najprirodniji. Također, vrijeme potrebno za analizu programa i traženje grešaka je znatno manje. Programski kod je **čitljiviji i lakše razumljiv**. Zato se autor ovog rada odlučio upravo na ovaj pristup pri implementaciji programa **rsa**.

5.3. Sabiranje

Kako smo vidjeli u odjeljku 5.2, veliki brojevi su u računarskoj memoriji predstavljeni u obliku sekvence cifri neke brojne baze. U nastavku ćemo opisati sabiranje i druge aritmetičke operacije nad strukturom podataka opisanom u odjeljku 5.2.2 jer ima dekadnu bazu i najjednostavnija je za razumijevanje. Rad sa ostalim nabrojanim strukturama bi zahtijevao neznatne izmjene.

Algoritam za sabiranje brojeva **cifru po cifru** je jednostavan za razumijevanje i implementaciju. Uči se u osnovnoj školi koristeći papir i olovku. Pseudokod 5.3.1 prikazuje jedan primjer takvog algoritma.

Funkcija **SABERI** sabira dva argumenta a i b , te vraća njihov zbir z . Pretpostavka je da je $a \geq b$ (tj. da je broj cifara broja a $(k+1)$ veći ili jednak broju cifara broja b $(l+1)$). Linija 3 dodjeljuje broju z vrijednost 0 sa $k + 2$ nula. U linijama 4 i 5 vrši se sabiranje brojeva a i b , cifru po cifru sa računanjem prenosa, pozivom funkcije **SABERI-CIFRE**. Ovo se ponavlja dok se ne saberu sve cifre broja b (primijetite da se prenos zbira zadnje cifre b i a upisuje u z_{l+1}). Zatim se, u linijama 6 i 7 dovršava sabiranje tako što se sabiraju cifre brojeva a i z . Ovo efektivno služi da se uračuna prenos sa cifre b_{l+1} , kao npr. u slučaju kada je $a = 1999$ i $b = 1$, kada se taj prenos prenosi sve do

cifre a_{k+1} . Funkcija **SABERI-CIFRE** računa i vraća zbir i prenos pri sabiranju dvije cifre.

```

SABERI(a, b)
1 neka je  $a_k, a_{k-1}, \dots, a_0$  rastava broja a na decimalne cifre
2 neka je  $b_l, b_{l-1}, \dots, b_0$  rastava broja b na decimalne cifre
3  $z \leftarrow 0..0 \triangleright z$  ima  $k + 2$  nula
4 for  $i \leftarrow 0$  to  $l$  do
5      $(z_i, z_{i+1}) \leftarrow \text{SABERI-CIFRE}(a_i, b_i)$ 
6 for  $i \leftarrow l + 1$  to  $k$  do
7      $(z_i, z_{i+1}) \leftarrow \text{SABERI-CIFRE}(a_i, z_i)$ 
8 return z

SABERI-CIFRE(a, b)
1  $(z, p) \leftarrow (a + b, 0)$ 
2 if  $z > 9$  then
3      $(z, p) \leftarrow (z \bmod 10, 1)$ 
4 return (z, p)

```

Pseudokod 5.3.1: Algoritam za sabiranje

Broj sabiranja pojedinačnih cifri kod ovog algoritma je **$O(n)$** , gdje je n broj cifri većeg od a, b .

5.4. Oduzimanje

Algoritam koji ćemo predstaviti za oduzimanje se također uči u osnovnoj školi. Pseudokod 5.4.1 prikazuje pseudokôd jednog takvog algoritma.

```

ODUZMI(a, b)
1 neka je  $a_k, a_{k-1}, \dots, a_0$  rastava broja a na decimalne cifre
2 neka je  $b_l, b_{l-1}, \dots, b_0$  rastava broja b na decimalne cifre
3  $r \leftarrow 0..0 \triangleright r$  ima  $k + 2$  nula
4 for  $i \leftarrow 0$  to  $l$  do
5      $(r_i, r_{i+1}) \leftarrow \text{ODUZMI-CIFRE}(a_i, b_i, r_i)$ 
6 for  $i \leftarrow l + 1$  to  $k$  do
7      $(r_i, r_{i+1}) \leftarrow \text{ODUZMI-CIFRE}(a_i, r_i, 0)$ 
8 return z

ODUZMI-CIFRE(a, b, c)
1  $r \leftarrow (a + 10 - b - c) \bmod 10$ 
2  $p \leftarrow 1 - ((a + 10 - b - c) / 10)$ 
3 return (r, p)

```

Pseudokod 5.4.1: Algoritam za oduzimanje

Funkcija **ODUZMI** vrši oduzimanje dva argumenta a i b , te vraća njihovu razliku r . Pretpostavka je da je $a \geq b$. U liniji 3, razlici r se dodjeljuje vrijednost nula sa $k+2$ nula, iako se efektivno koristi $k+1$ (to je učinjeno jer u liniji 7 dolazi do dodjele lokaciji r_{k+2} , tako da ova vrijednost mora postojati). U linijama 4 i 5 se vrši efektivno oduzimanje b od a tako što se $l+1$ puta poziva funkcija **ODUZMI-CIFRE** sa argumentima a_i, b_i i r_i .

Ova funkcija računa i vraća razliku $a - (b + c) = a - b - c$, kao i prenos pri oduzimanju. Na taj način je moguće istovremeno u formulu razlike $a - b$ uračunati i prenos iz prethodnog oduzimanja (c), bez da napravimo prebačaj⁷.

Petlja u linijama 4 i 5 ponavlja ovaj postupak za sve cifre broja b . Petlja u linijama 6 i 7 računa razlike ostalih cifri broja a i eventualnog prenosa koji je ostao od cifre b_{l+1} , sačuvan u r_{l+2} .

Broj oduzimanja pojedinačnih cifri kod ovog algoritma je **$O(n)$** , gdje je n broj cifri broja a .

5.5. Množenje

Poznato je mnoštvo algoritama za množenje cijelih brojeva, posebno velikih cijelih brojeva. Svaki od ovih algoritama ima određene prednosti i nedostatke, te se može preporučiti za rad sa velikim brojevima određene dužine (broja cifara). Lista nekih od poznatih algoritama za množenje, njihova kompleksnost, kao i aproksimacija broja cifara umnožaka za koji se ovi algoritmi smatraju efikasnim data je u tabeli 5.5.1.

Naziv	Kompleksnost	Opseg efikasne primjene ⁸
Školski algoritam [D13]	$O(n^2)$	od jedne do otprilike 100 decimalnih cifara ili 2^{320}
Podijeli pa vladaj (Karatsuba) algoritam [D12]	$O(n^{\log_2 3}) \approx O(n^{1.585})$	od 100 do između 10000 i 40000 decimalnih cifara, ili od 2^{320} do $2^{2^{15}}$ ili $2^{2^{17}}$
Toom – Cook algoritam [D11]	$\Theta(n^{\log(5)/\log(3)}) \approx \Theta(n^{1.465})^9$	od 100 do između 10000 i 40000 decimalnih cifara, ili 2^{320} do $2^{2^{15}}$ ili $2^{2^{17}}$
Schönhage – Strassen algoritam [D10]	$O(n \cdot \log(n) \cdot \log(\log(n)))$	od oko $2^{2^{15}}$ ili $2^{2^{17}}$ (10000 do 40000 decimalnih cifara) pa dalje
Fürer algoritam [9]	$n \cdot \log(n \cdot 2^{O(\log^* n)})$ ¹⁰	od oko $2^{2^{15}}$ ili $2^{2^{17}}$ (10000 do 40000 decimalnih cifara) pa dalje

Tabela 5.5.1: Algoritmi za množenje

Svi algoritmi iz tabele 5.5.1, izuzev školskog algoritma, dobili su ime po osobi/osobama koje su ih pronašle. Toom – Cook algoritam je jako sličan Karatsuba algoritmu i predstavlja generalizaciju istog. Njegova kompleksnost je nešto manja nego kod Karatsuba algoritma, zbog manjeg eksponenta (oko 1.465 u odnosu na oko 1.585 kod Karatsuba). Ovaj algoritam je

7 Efektivno, mogli smo napisati funkciju **ODUZMI-CIFRE**(a, b) tako da računa razliku između a i b , a kao argument b joj proslijediti zbir b_i i prijenosa iz prethodnog oduzimanja. Međutim, tada bi u jedinstvenom slučaju kada je $b_i = 9$ i postoji prijenos iz prethodnog oduzimanja napravili prebačaj, jer bi argumentu b pokušali dodijeliti $9 + 1 = 10$, što nije dekadna cifra.

8 Zavisno od implementacije. Neki podaci predstavljeni su u [D7] i [D8].

9 Ovaj podatak odnosi se na “Toom – 3” algoritam, jedan iz porodice Toom-Cook algoritama, koji se najčešće koristi u praksi.

10 “ $\log^* n$ ” se odnosi na operaciju iteriranog logaritma.

asimptotski brži od Karatsuba algoritma. Schönhage – Strassen i Fürer algoritmi zasnovani su na rekurzivnoj brznoj Fourierovoj transformaciji. Fürer algoritam je asimptotski brži od Schönhage – Strassen algoritma.

U nastavku su predstavljeni školski i Karatsuba algoritam. Ova dva algoritma su ujedno i implementirana u program **rsa**. Implementacija brzog algoritma za množenje je jako važna kod RSA kriptosistema, jer se generisanje ključeva, enkripcija i dekripcija zasnivaju na operacijama koje podrazumijevaju veliki broj množenja.

5.5.1. Školski algoritam (long multiplication)

Pseudokod 5.5.1 prikazuje pseudokôd školskog algoritma za množenje. Efektivno, ovaj algoritam množi svaku cifru broja b sa svakom cifrom broja a i u svakom koraku dodaje novi umnožak na postojeći rezultat, za razliku od običnog školskog algoritma, gdje se svi umnošci najprije potpišu jedan ispod drugog, a zatim sabiraju. Ovaj prilagođeni algoritam donosi **uštedu u upotrebi memorije**.

```

POMNOŽI(a, b)
1  neka je  $a_k, a_{k-1}, \dots, a_0$  rastava broja  $a$  na decimalne cifre
2  neka je  $b_l, b_{l-1}, \dots, b_0$  rastava broja  $b$  na decimalne cifre
3   $r \leftarrow 0..0$  ▷  $r$  ima  $k + l + 2$  nula
4   $(u, p) \leftarrow (0, 0)$ 
5  for  $i = 0$  to  $k$  do
6      for  $j = 0$  to  $l$  do
7           $u \leftarrow a_i \cdot b_j + r_{i+j} + p$ 
8           $r_{i+j} \leftarrow u \bmod 10$ 
9           $p \leftarrow u / 10$ 
10     if  $p \neq 0$  then
11          $r_{i+l+1} \leftarrow r_{i+l+1} + p$ 
12          $p \leftarrow 0$ 
13 return  $r$ 

```

Pseudokod 5.5.1: Školski algoritam za množenje

Funkcija **POMNOŽI** prima parametre a i b , koji imaju $k+1$, odnosno $l+1$ cifara, respektivno, te vraća njihov proizvod r . U liniji 3, rezultat r se inicijalizira na vrijednost 0 (sa $(k+1) + (l+1) = k+l+2$ cifara). U liniji 4 se inicijalizira privremeni proizvod dvije cifre, u , i prenos pri množenju, p , na nulu. Linije 5 i 6 označavaju da se prolazi kroz svaku cifru oba broja (indeks i odgovara broju a , a indeks j broju b). U liniji 7 se vrši samo množenje cifara a_i i b_j , uz računanje prenosa iz prethodnih iteracija (r_{i+j} i p). Linija 8 upisuje novu cifru rezultata, dok se u liniji 9 pamti prenos za sljedeću iteraciju. Kada for petlja linije 6 dostigne svoj kraj (kada se iskoriste sve cifre broja b), tada se u liniji 10 testira da li je ostao prenos, i on se u liniji 11 upisuje u rezultat, a u liniji 12 anulira.

Vremenska kompleksnost ovog algoritma iznosi $O(n^2)$.

5.5.2. Podijeli pa vladaj (divide and conquer) množenje

Ovaj algoritam otkrio je ruski matematičar Karatsuba 1960. godine, a objavljen je 1962. Osim naziva “**podijeli pa vladaj**”, još se zove i “**Karatsuba algoritam**”, “**binarna podjela**” i “**princip dihotomije**”.

Neka je dva n -bitna broja, a i b , koja je potrebno pomnožiti, moguće u brojnom sistemu sa bazom B napisati kao $a = a_1B^m + a_2$ i $b = b_1B^m + b_2$. Sada se, množenjem a i b , dobije :

$$ab = (a_1B^m + a_2)(b_1B^m + b_2) = a_1b_1B^{2m} + (a_1b_2 + a_2b_1)B^m + a_2b_2.$$

Standardni postupak bi bio da pomnožimo četiri umnoška zasebno i saberemo ih da bi dobili ab , što ima kompleksnost $O(n^2)$. Ako, zbog jednostavnosti, uvedemo smjene $X = a_1b_1$, $Y = a_2b_2$, $Z = (a_1b_2 + a_2b_1)$, dobijamo:

$$ab = XB^{2m} + ZB^m + Y$$

Izraz 5.5.1

Ubrzanje koje uvodi Karatsuba zasniva se na činjenici da je ova četiri produkta moguće izračunati koristeći samo tri operacije množenja, ako primijetimo da je Z moguće transformisati na sljedeći način:

$$Z = (a_1b_2 + a_2b_1) = (a_1b_1 + a_1b_2 + a_2b_1 + a_2b_2) - a_1b_1 - a_2b_2 = (a_1 + a_2)(b_1 + b_2) - X - Y.$$

Za računanje ova tri proizvoda (X , Y i Z), možemo ponovo upotrijebiti isti algoritam, rekursivno. Ostale aritmetičke operacije zahtijevaju linearno vrijeme izvršavanja, i **možemo ih zanemariti**. Množenje sa B^m i B^{2m} se može realizovati **šiftnjem ulijevo** za m ili $2m$ mjesta. Karatsuba množenje postiže najveće ubrzanje u odnosu na školski algoritam kada su umnošci a i b približno iste veličine u bitima. Iako se, teoretski, m može izabrati proizvoljno, najbolji rezultati se postižu kada je $m \approx n/2$. Također, kada, prilikom rekursivnih poziva, argumenti a i b postanu dovoljno mali, preporučuje se **upotreba školskog algoritma** za njihovo množenje, jer isti pokazuje bolje rezultate za brojeve do određene dužine, kao u tabeli 5.5.1.

Pseudokod 5.5.2 prikazuje ovaj algoritam. Zahtjev je da brojevi a i b budu n -bitni, i da je n paran broj, što ne umanjuje opštost, ali pojednostavljuje pseudokôd. Operacija šiftnja ulijevo je označena sa “ **$a \ll b$** ”, gdje je broj a šiftna ulijevo za b bita.

```

KARATSUBA(a, b)
1  neka su  $a_1, a_2$  gornjih, odnosno donjih  $n/2$  bita broja  $a$ 
2  neka su  $b_1, b_2$  gornjih, odnosno donjih  $n/2$  bita broja  $b$ 
3   $r \leftarrow 0$ 
4   $X \leftarrow 0$ 
5   $Y \leftarrow 0$ 
6   $Z \leftarrow 0$ 
7  neka je  $g$  granica do koje se za množenje koristi školski algoritam
8  if  $n < g$  then
9       $r \leftarrow \text{POMNOŽI}(a, b)$ 
10     return  $r$ 
11  $X \leftarrow \text{KARATSUBA}(a_1, b_1)$ 
12  $Y \leftarrow \text{KARATSUBA}(a_2, b_2)$ 
13  $Z \leftarrow \text{KARATSUBA}(a_1 + a_2, b_1 + b_2)$ 
14  $r \leftarrow (X \ll n) + (Z \ll n/2) + Y$ 
15 return  $r$ 

```

Pseudokod 5.5.2: Karatsuba algoritam

Linije 1 do 7 funkcije **KARATSUBA** predstavljaju inicijalizaciju parametara. Parametar g je konstanta koja treba prethodno biti definirana, i označava maksimalni broj bita za koji je odgovarajuća implementacija školskog algoritma množenja (kao npr. algoritam 5.5.1) brža od implementacije Karatsuba množenja. Kada brojevi a i b budu manji od tog broja bita, možemo pozvati funkciju **POMNOŽI** za školsko množenje, i vratiti rezultat. U suprotnom, vrši se računanje brojeva X , Y i Z rekurzivnim pozivima funkcije **KARATSUBA** u linijama 11 do 13. Na kraju, u liniji 14, izračuna se **konačan produkt r** prema izrazu 5.5.1 i u liniji 15 vrati kao rezultat funkcije.

5.6. Zaključak

Postojeće računarske arhitekture imaju ograničenje na izvršavanje aritmetičkih operacija nad operandima maksimalne dužine od 64 bita. Međutim, za potrebe RSA kriptosistema to je premalo, i potrebno je operisati sa brojevima dužine 2048 i više bita. To je stvorilo potrebu za razvojem takvih struktura podataka koje bi pružile mogućnost **memorisanja i efikasnog izvršavanja aritmetičkih operacija nad brojevima proizvoljne dužine**.

Od različitih predstavljenih tehnika koje se mogu primijeniti za memorisanje jako velikih cijelih brojeva, najboljom se pokazala tehnika **čuvanja velikih brojeva rastavljenih u cifre**, tako da je veličina pojedinačnih cifara u bitima jednaka širini računarske arhitekture na kojoj se operiše. Na taj način se stoprocentno iskorištava dostupni memorijski prostor i sposobnost računara da operiše sa ciframa veličine maksimalno jednake širini arhitekture.

Predstavljeni su algoritmi za sabiranje i oduzimanje u svojoj osnovnoj varijanti, koji zadovoljavaju potrebe rada sa velikim brojevima. Prikazana je usporedba pet različitih algoritama za množenje, od kojih su dva najjednostavnija detaljno opisana. Svaki od predstavljenih algoritama ima svoju **primjenu nad brojevima određenog opsega veličine**.

Iako su neki od algoritama upotrijebljenih u realizaciji programa **rsa** suboptimalne efikasnosti, izabrani su zbog svoje jednostavnosti kao najbolji **edukativni primjer** za onoga ko čita prateći izvorni kôd.

6. Implementacioni detalji

6.1. Uvod

Nakon upoznavanja sa svim teoretskim aspektima kriptografije, RSA kriptosistema, algoritama specifičnih za RSA, kao i rada sa jako velikim cijelim brojevima potrebnih za implementaciju RSA kriptosistema, možemo pristupiti i opisivanju implementacionih detalja samog programa **rsa**.

6.2. Korišteni alati

Realizacija **rsa** je urađena koristeći **programski jezik C++** i **objektno orijentisanu programsku paradigmu**¹¹. Poštovan je standard ANSI ISO/IEC 14882:1998 za programski jezik C++, što osigurava portabilnost izvornog kôda. Korištene su isključivo standardne biblioteke ovog jezika, i ništa osim njih. Razvojno okruženje koje je korišteno za izradu rada je **Eclipse Europa** (verzija 3.3.2), kompajler je **gcc** (verzija 4.2.3), a operativni sistem **Ubuntu** (u verzijama 7.10 i 8.04). Svi nabrojani alati su dostupni za slobodan download i upotrebu. Programski kôd je testiran pod razvojnim okruženjem Visual Studio 2005 sa MS VisualC++ kompajlerom na Windows XP SP2 operativnom sistemu i radi bez izmjena.

6.3. Izvorni kôd

Izvorni kôd programa **rsa** raspoređen je u **12 programskih jedinica**. Sastoji se od pet klasa koje sadrže funkcionalnost, jedne skupine testnih funkcija, te jedne datoteke sa glavnim programom, za potrebe demonstracije funkcionalnosti, kao u tabeli 6.3.1. Kompletan izvorni kôd je pisan na engleskom jeziku, uključujući i sve komentare.

11 Stogo rečeno, korišteno je objektno zasnovano programiranje (OBP) u odnosu na objektno orijentisano programiranje (OOP), koje, osim sakrivanja informacija i enkapsulacije, zahtijeva i nasljeđivanje i polimorfizam, koji u **rsa** nisu upotrijebljeni.

Jedinica	Datoteke	Funkcionalnost
RSA	RSA.h, RSA.cpp	Generisanje RSA ključeva, RSA enkripcija i dekripcija
PrimeGenerator	PrimeGenerator.h, PrimeGenerator.cpp	Generisanje velikih prostih brojeva
KeyPair	KeyPair.h, KeyPair.cpp	Čuvanje para RSA ključeva
Key	Key.h	Čuvanje jednog RSA ključa
BigInt	BigInt.h, BigInt.cpp	Čuvanje i rad sa cijelim brojevima
testne funkcije	test.h, test.cpp	Testiranje funkcionalnosti
glavni program	main.cpp	Demonstracija funkcionalnosti

Tabela 6.3.1: Izvorni kôd

6.3.1. RSA

Klasa **RSA** predstavlja implementaciju RSA kriptosistema. Podržava generisanje RSA ključeva, RSA enkripciju i dekripciju. Kôd 6.3.1 prikazuje javni interfejs ove klase.

```
static KeyPair GenerateKeyPair(    unsigned long int digitCount,
                                unsigned long int k);

static std::string Encrypt(const std::string &message, const Key &key);

static std::string Decrypt(const std::string &cypherText, const Key &key);

static void Encrypt(    const char *sourceFile,
                       const char *destFile,
                       const Key &key);

static void Decrypt(    const char *sourceFile,
                       const char *destFile,
                       const Key &key);
```

Kod 6.3.1: RSA, javni interfejs

Metoda **GenerateKeyPair** generiše i vraća par RSA ključeva sa $n = (2 \cdot \text{digitCount} - 1)$ cifara i sa vjerovatnoćom prostosti od $1 - 4^{-k}$. Metoda **Encrypt** sa argumentom tipa **std::string** šifruje poruku **message** koristeći ključ **key**, i vraća šifrovanu poruku kao rezultat, dok metoda **Encrypt** sa argumentima tipa **const char*** šifruje ulaznu datoteku **sourceFile** koristeći ključ **key** i rezultat spašava kao datoteku **destFile**. Metode **Decrypt** imaju analogno ponašanje, samo što vrše dekripciju. Klasa **RSA** za svoju funkcionalnost koristi usluge svih ostalih klasa. U ovoj klasi je implementiran prošireni Euklidov algoritam, opisan u odjeljku 4.4.1.

6.3.2. PrimeGenerator

Klasa **PrimeGenerator** služi za generisanje velikih slučajnih ili prostih brojeva. Kôd 6.3.2

predstavlja javni interfejs ove klase.

```
static void MakeRandom(BigInt &number, unsigned long int digitCount);  
  
static BigInt Generate(unsigned long int digitCount,  
                      unsigned long int k);
```

Kod 6.3.2: PrimeGenerator, javni interfejs

Metoda **MakeRandom** generiše slučajan broj dužine **digitCount** cifara, i vraća ga po referenci u argumentu **number**. Metoda **Generate** generiše prost broj dužine **digitCount** cifara, sa vjerovatnoćom prostosti od $1 - 4^{-k}$. Ovu metodu koristi klasa **RSA** kada generiše ključeve. U klasi **PrimeGenerator** implementiran je Miller-Rabinov probabilistički test prostosti, opisan u odjeljku 4.3.2.

6.3.3. KeyPair

Klasa **KeyPair** služi za čuvanje para RSA ključeva, koji se sastoji od javnog i tajnog ključa. Zato je ovo prilično jednostavna klasa. Kôd 6.3.3 predstavlja javni interfejs ove klase.

```
KeyPair(Key privateKey, Key publicKey);  
  
const Key &GetPrivateKey() const;  
  
const Key &GetPublicKey() const;  
  
friend std::ostream &operator <<(std::ostream &cout, const KeyPair &k);
```

Kod 6.3.3: KeyPair, javni interfejs

Jedini javni konstruktor, **KeyPair**, zahtijeva oba ključa kao parametar i tako osigurava da ne može biti kreiran objekat tipa **KeyPair** bez poznavanja oba ključa. Također, ne postoji način kako bi se mogli promijeniti ključevi, nakon što se jednom kreira ovaj objekat. Metode **GetPrivateKey** i **GetPublicKey** omogućavaju čitanje javnog i tajnog ključa, dok preklopljeni operator **<<** omogućava ispis informacija o ključu na standardni izlaz.

6.3.4. Key

Klasa **Key** je trivijalna i sastoji se od samo jedne datoteke. Njena funkcionalnost se sastoji u čuvanju jednog RSA ključa. Kôd 6.3.4 prikazuje javni interfejs ove klase. Ova klasa je analogna klasi **KeyPair** iz odjeljka 6.3.3, pa zato neće biti detaljnije opisivana.

```
Key(const BigInt &modulus, const BigInt &exponent);  
  
const BigInt &GetModulus() const;  
  
const BigInt &GetExponent() const;
```

Kod 6.3.4: Key, javni interfejs

6.3.5. BigInt

Klasa **BigInt** čini glavninu programa, i po obimu kôda je najveća klasa. Predstavlja implementaciju strukture podataka za čuvanje proizvoljno velikih cijelih brojeva i izvršavanje aritmetičkih operacija nad istim. Kôd 6.3.5 prikazuje javni interfejs ove klase.

```

BigInt();
BigInt(const char *charNum);
BigInt(unsigned long int intNum);
BigInt(const std::string &str);
BigInt(const BigInt &number);
BigInt &operator =(const BigInt &rightNumber);
~BigInt();
operator std::string() const;
friend std::ostream &operator <<(std::ostream &cout, const BigInt &number);
friend std::istream &operator >>(std::istream &cin, BigInt &number);
friend bool operator <(const BigInt &a, const BigInt &b);
friend bool operator <=(const BigInt &a, const BigInt &b);
friend bool operator >(const BigInt &a, const BigInt &b);
friend bool operator >=(const BigInt &a, const BigInt &b);
friend bool operator ==(const BigInt &a, const BigInt &b);
friend bool operator !=(const BigInt &a, const BigInt &b);
friend BigInt operator +(const BigInt &a, const BigInt &b);
BigInt &operator+();
BigInt &operator++();
BigInt operator++(int);
BigInt &operator+=(const BigInt &number);
BigInt operator-() const;
friend BigInt operator-(const BigInt &a, const BigInt &b);
BigInt &operator--();
BigInt operator--(int);
BigInt &operator--(const BigInt &number);
friend BigInt operator*(const BigInt &a, const BigInt &b);
BigInt &operator*=(const BigInt &number);
friend BigInt operator/(const BigInt &a, const BigInt &b);
BigInt &operator/=(const BigInt &number);
friend BigInt operator%(const BigInt &a, const BigInt &b);
BigInt &operator%=(const BigInt &number);
BigInt GetPower(unsigned long int n) const;
void SetPower(unsigned long int n);
BigInt GetPower(BigInt n) const;
void SetPower(BigInt n);
BigInt GetPowerMod(const BigInt &b, const BigInt &n) const;
void SetPowerMod(const BigInt &b, const BigInt &n);
unsigned char operator [] (unsigned long int n) const;
unsigned long int Length() const;
bool IsPositive() const;
bool IsOdd() const;
std::string ToString(bool forceSign = false) const;
bool EqualsZero() const;
BigInt Abs() const;

```

Kod 6.3.5: BigInt, javni interfejs

Zbog ograničenosti prostora, ukratko ćemo opisati navedene metode. Konstruktor bez parametara omogućava kreiranje objekta ovog tipa sa početnom vrijednošću nula. Implementirani su i konstruktori sa različitim parametrizacijama, tako da je moguća automatska konverzija iz tipova `unsigned long int`, `const char *`, ili `std::string`, kao i u tip `std::string`. Zahvaljujući implementaciji preklopljenog operatora dodjele i destruktora, rad sa objektima tipa **BigInt** je siguran. Preklopljeni operatori `<<` i `>>` omogućavaju unos i ispis sa standardnog ulaza i izlaza, respektivno. Implementirani su svi operatori poređenja (`<`, `<=`, `>`, `>=`, `==` i `!=`). Također, tu su i

operatori unarni plus (+, ne radi ništa) i unarni minus(–, mijenja predznak svom operandu), operator za sabiranje (binarni +, radi po algoritmu predstavljenom pseudokôdom 5.3.1), operator za oduzimanje (binarni –, radi po algoritmu predstavljenom pseudokôdom 5.4.1), prefiksni i postfiksni ++ operatori (oslanjaju se na rad binarnog + operatora), prefiksni i postfiksni -- operatori (oslanjaju se na rad binarnog – operatora), te operatori -= i += (koji se također oslanjaju na rad odgovarajućih binarnih operatora + i –, respektivno).

Množenje je implementirano u preklopljenom operatoru *(može se birati između dva algoritma opisana pseudokôdom 5.5.1 i 5.5.2), a operator *= se oslanja na operator *. Dijeljenje je implementirano preklapanjem operatora /, a koristi ga i operator /=. Tu je i operacija ostatka pri dijeljenju, implementirana operatorima % i %=.

Metode **GetPower** i **SetPower** implementiraju stepenovanje u standardnoj aritmetici sa različitim argumentima, dok metode **GetPowerMod** i **SetPowerMod** implementiraju stepenovanje u modularnoj aritmetici. Za posljednje dvije metode koristi se algoritam opisan pseudokôdom 4.2.1.

Preklopljeni operator [] sa argumentom *n* vraća *n*-tu cifru broja. Metoda **Length** vraća ukupan broj cifara. Metoda **IsPositive** vraća vrijednost **true** ako je broj pozitivan ili nula, u suprotnom **false**. Metoda **IsOdd** vraća **true** ako je broj neparan, u suprotnom **false**. Metoda **ToString** vraća broj u obliku tipa `std::string`. Metoda **EqualsZero** je brz način provjere da li je broj jednak nuli. Metoda **Abs** vraća apsolutnu vrijednost broja.

6.4. Zaključak

Program **rsa**, koji je nastao kao rezultat ovog diplomskog rada, implementira RSA kriptosistem. Osnovne funkcionalnosti programa su **generisanje RSA ključeva**, **RSA enkripcija i dekripcija**, dok je proširenu funkcionalnost predstavlja implementirana, **ponovno upotrebljiva struktura podataka za rad sa jako velikim cijelim brojevima**.

Programski kôd je organizovan u 12 datoteka. 9 datoteka se koristi za implementaciju 5 klasa: **RSA**, za generisanje ključeva, enkripciju i dekripciju; **PrimeGenerator**, za generisanje velikih slučajnih ili prostih brojeva; **KeyPair**, za čuvanje parova RSA ključeva; **Key**, za čuvanje RSA ključeva, te **BigInt**, za čuvanje i aritmetičke operacije nad proizvoljno velikim cijelim brojevima. Dvije datoteke služe za testove funkcionalnosti, dok je u jednoj datoteci sačuvan glavni program, u kojem se demonstrira navedena funkcionalnost.

7. Zaključak

U ovom diplomskom radu opisana je i realizirana programska **implementacija RSA kriptografskog algoritma sa automatskim generiranjem ključeva**.

Opisan je nastanak i teoretske osnove **kriptografije**, sa posebnim osvrtom na kriptografiju sa javnim ključem. Postavljena je teoretska osnova **RSA kriptosistema**, uključujući generisanje ključeva, enkripciju, dekripciju i digitalni potpis. Opisani su **karakteristični matematski problemi** koji se javljaju pri pratkičnoj implementaciji RSA, kao i algoritmi za njihovo rješavanje. Predstavljeno je nekoliko tipova struktura podataka za čuvanje **velikih cijelih brojeva** i vršenje aritmetičkih operacija nad istim. Konačno, realizirana je **potpuna i razumljiva** implementacija RSA kriptosistema u vidu programa **rsa**, te je u posljednjem poglavlju i opisana.

Kao mogući nastavak rada na ovu temu, preporučujemo teoretska razmatranja i praktičnu implementaciju **nekoliko novih funkcionalnosti** u okviru programa **rsa**.

Jedna takva funkcionalnost može biti modifikacija implementirane strukture podataka za rad sa velikim cijelim brojevima, na takav način da se **cifre memorišu kao 32-bitni ili 64-bitni cijeli brojevi**. U okviru ovog rada već su diskutirani teoretski aspekti ove implementacione varijante, koja čini vremenski i prostorno najefikasnije rješenje.

Implementacija jako brzih algoritama za množenje, sa mogućnošću izbora optimalnog algoritma za određenu veličinu umnožaka, kako je diskutovano u okviru rada, dodatno bi optimizirala rad programa.

Konačno, preporučujemo nastavak rada u pravcu **generisanja sigurnijih ključeva**. Posebno, potrebno je posvetiti pažnju generisanju takvog modula $n = pq$, čija se faktORIZACIJA opire nekim poznatim metodama faktORIZACIJE koje uspješno djeluju kada p i q nisu pažljivo izabrani.

Literatura

- [1] R. Rivest, A. Shamir and L. Adleman. *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*. Communications of the ACM, 21 (2), pp. 120-126, February 1978.
- [2] T. H. Cormen , C. E. Leiserson , R. L. Rivest and C. Stein . *Introduction to Algorithms, Second Edition*. The MIT Press , 2001.
- [3] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996 .
- [4] RSA Laboratories. *RSA Laboratories' Frequently Asked Questions About Today's Cryptography, Version 4.1*. RSA Security Inc., 2000.
- [5] A. Arnaut. *Kriptografija sa javnim ključem. RSA kriptosistem*. Prirodno-matematički fakultet u Sarajevu, 1999.
- [6] M.O. Rabin. *Probabilistic Algorithm for Primality Testing*. Journal of Number Theory. Vol. 12, pp. 128-138. 1980.
- [7] G.L. Miller. *Riemann's Hypothesis and Tests for Primality*. Journal of Computer Systems Science, Vol. 13, No. 3, pp 300-317. 1976.
- [8] B. C. Higgins. *The Rabin- Miller Probabilistic Primality Test: Some Results on the Number of Non-Witnesses to Compositeness*. 2003.
- [9] M. Fürer. *Faster integer multiplication*. Proceedings of the thirty-ninth annual ACM symposium on Theory of computing, pp. 57-66. 2007.
- [10] **International Standard 14882:1998 - Programming Language C++**. 1998.
- [11] Ž. Jurić. **Diskretna matematika**, Interna skripta (Udžbenik u pripremi), ETF Sarajevo. 2006.
- [12] W. T. Polk, D. F. Dodson, W. E. Burr. *Cryptographic Algorithms and Key Sizes for Personal Identity Verification*. NIST. 2007.

Dodatna literatura

- [D1] **History of cryptography**
http://en.wikipedia.org/w/index.php?title=History_of_cryptography&oldid=228300723
- [D2] **RSA**
<http://en.wikipedia.org/w/index.php?title=RSA&oldid=231467735>
- [D3] **RSA Algorithm**
http://www.di-mgt.com.au/rsa_alg.html
- [D4] **RSA Security**
<http://www.rsasecurity.com/>
- [D5] **Miller-Rabin primality test**
http://en.wikipedia.org/w/index.php?title=Miller%E2%80%93Rabin_primality_test&oldid=232066792
- [D6] **Primality test**

- http://en.wikipedia.org/w/index.php?title=Primality_test&oldid=229142415
- [D7] **Magma Arithmetic**
<http://magma.maths.usyd.edu.au/magma/Features/node86.html>
- [D8] L. C. C. García. *Can Schönhage multiplication speed up the RSA encryption or decryption?* University of Technology, Darmstadt. 2005.
<http://www.cdc.informatik.tu-darmstadt.de/%7Ecoronado/Vortrag/MoraviaCrypt-talk-s.pdf>
- [D9] **RSA Laboratories - RSA-200 is factored!**
<http://www.rsa.com/rsalabs/node.asp?id=2879>
- [D10] **Schönhage-Strassen algorithm**
http://en.wikipedia.org/w/index.php?title=Sch%C3%B6nhage-Strassen_algorithm&oldid=228786569
- [D11] **Toom–Cook multiplication**
http://en.wikipedia.org/w/index.php?title=Toom%E2%80%93Cook_multiplication&oldid=219564571
- [D12] **Karatsuba algorithm**
http://en.wikipedia.org/w/index.php?title=Karatsuba_algorithm&oldid=229091837
- [D13] **Multiplication algorithm**
http://en.wikipedia.org/w/index.php?title=Multiplication_algorithm&oldid=233069335