# Algoritmi za obradu digitalnog potpisa

**Hmelik, Ivan**



**SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU**

**FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA**

**Sveučilišni studij**

# ALGORITMI ZA OBRADU DIGITALNOG POTPISA

**Diplomski rad**

**Ivan Hmelik**

**Osijek, 2019.**

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

**Obrazac D1: Obrazac za imenovanje Povjerenstva za obranu diplomskog rada**

**Osijek, 20.09.2019.**

**Odboru za završne i diplomske ispite**

# Imenovanje Povjerenstva za obranu diplomskog rada

| | |
|---|---|
| **Ime i prezime studenta:** | Ivan Hmelik |
| **Studij, smjer:** | Diplomski sveučilišni studij Računarstvo |
| **Mat. br. studenta, godina upisa:** | D 880 R, 26.09.2018. |
| **OIB studenta:** | 81084480009 |
| **Mentor:** | Doc.dr.sc. Tomislav Rudec |
| **Sumentor:** | Izv. prof. dr. sc. Alfonzo Baumgartner |
| **Sumentor iz tvrtke:** | |
| **Predsjednik Povjerenstva:** | Izv. prof. dr. sc. Krešimir Nenadić |
| **Član Povjerenstva:** | Dr. sc. Tomislav Galba |
| **Naslov diplomskog rada:** | Algoritmi za obradu digitalnog potpisa |
| **Znanstvena grana rada:** | **Procesno računarstvo (zn. polje računarstvo)** |
| **Zadatak diplomskog rada:** | Student će obraditi matematičke sheme digitalnog potpisa i opisati korištenih matematičkih algoritama RSA (Rivest - Shamir - Adleman), DSA (Digital Signature Algorithm), te Rabinovog algoritma. Praktični dio rada se zasniva na implementaciji u programskom jeziku C. Sumentor: Alfonzo Baumgartner |
| **Prijedlog ocjene pismenog dijela ispita (diplomskog rada):** | Izvrstan (5) |

| | |
|---|---|
| **Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:** | Primjena znanja stečenih na fakultetu: 3 bod/boda<br>Postignuti rezultati u odnosu na složenost zadatka: 2 bod/boda<br>Jasnoća pismenog izražavanja: 2 bod/boda<br>Razina samostalnosti: 3 razina |
| **Datum prijedloga ocjene mentora:** | 20.09.2019. |

| | |
|---|---|
| Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija: | Potpis: |
| | Datum: |

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

# IZJAVA O ORIGINALNOSTI RADA

**Osijek, 09.10.2019.**

| | |
|---|---|
| **Ime i prezime studenta:** | Ivan Hmelik |
| **Studij**: | Diplomski sveučilišni studij Računarstvo |
| **Mat. br. studenta, godina upisa:** | D 880 R, 26.09.2018. |
| **Ephorus podudaranje [%]:** | 13 |

Ovom izjavom izjavljujem da je rad pod nazivom**: Algoritmi za obradu digitalnog potpisa**

izrađen pod vodstvom mentora Doc.dr.sc. Tomislav Rudec

i sumentora Izv. prof. dr. sc. Alfonzo Baumgartner

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

# Sadržaj

# 1. UVOD

Živimo u vremenu u kojem su zahtjevi za digitalizaciju podataka sve veći, a komunikacija, trgovanje i razmjena podataka ne koristeći internet gotovo nezamisliva. Sukladno tome u pitanje se dovodi autentičnost, integritet i neporecivost podataka i komunikacije. Upravo nam digitalni potpis osigurava utvrđivanje identiteta pošiljatelja i provjeru originalnog sadržaja poruke, kao i nemogućnost primatelja da poriče sudjelovanje u komunikaciji. Digitalni potpis je ekvivalent ručnom potpisu, te ga je puno teže stvoriti od klasičnog vlastoručnog potpisa.[1] Korištenjem asimetrične kriptografije (šifriranja) omogućava se poruci siguran prolazak kroz nesigurne i ranjive komunikacijske kanale. Sigurnost se osigurava primjenom javnog i privatnog ključa što potencijalnim napadačima otežava dešifriranje presječenih poruka. Algoritmi koji se koriste za obradu digitalnog potpisa moraju biti implementirani na ispravan način jer u suprotnom se u pitanje dovodi sigurnost šifriranih poruka, dokumenata ili transakcija što može naštetiti organizacijama, poduzećima ili običnom čovjeku te njihovom materijalnom i intelektualnom vlasništvu.

Cilj diplomskog rada je opisati algoritme koji se koriste za obradu digitalnog potpisa te ih implementirati u programskom jeziku C.

U drugom poglavlju ovoga rada obrađuju se pojmovi digitalnog potpisa i kriptografije kao i povijest njihovog nastanka. U sljedećem će poglavlju detaljno biti opisani zadani matematički algoritmi DSA (Digital Signature Algorithm), RSA (Rivest – Shamir – Adleman) i Rabinov algoritam. Četvrto poglavlje se temelji na implementaciji navedenih matematičkih algoritama u programskom jeziku C, a završno peto poglavlje predstavlja zaključak diplomskog rada.

# 2. DIGITALNI POTPIS

Pojam digitalnog potpisa se prvi put pojavljuje 1976. godine kada su ga opisala dva američka kriptografa, Whitfield Diffie i Martin Hellman. Njihova metoda je danas zapamćena kao Diffie-Hellmanova metoda razmjene ključeva.[2] Njihov rad je bio inspiriran otkrićima Ralpha Merkela, a temeljio se na sigurnoj razmjeni kriptografskih ključeva putem javnih i nesigurnih komunikacijskih kanala. Samo godinu dana kasnije Ronald Rivest, Adi Shamir i Len Adleman osmišljavaju tzv. RSA algoritam čije ime predstavlja početna slova njihovih prezimena. Bio je to prvi siguran algoritam koji je bio prikladan za enkripciju i potpisivanje. Njegova prva komercijalna verzija puštena je u pogon 1989. godine pod imenom Lotus Notes. Nedugo nakon otkrića RSA algoritma, nastaje još nekoliko shema za digitalni potpis, a među najpoznatijima su Lamportov potpis, Merkleov potpis i Rabinov potpis koji će detaljno biti objašnjen u ovom radu. Ronald Rivest nije stao s proučavanjem digitalnog potpisa i kriptografije, pa je tako 1988. godine s kolegama Goldwasserom i Micalijem osmislio GMR shemu potpisa koja je također dobila ime prema prvim slovima njihovih prezimena. GMR algoritam je osmišljen tako da je siguran od ciljanih napada na poruke, a ako napadač dođe do potpisa od neke ciljane poruke neće ga moći krivotvoriti za neku dodatnu poruku. [1]

Standardizacija digitalnog potpisa počinje sredinom 1990-ih godina u Sjedinjenim Američkim Državama, dok je proces standardizacije u Europi započeo nekoliko godina kasnije.[2] Danas u nekoliko država svijeta i zemljama članicama Europske Unije digitalni potpis ima isti status kao onaj rukom potpisan, a to znači da sve što je digitalno potpisano zakonski obavezuje potpisanog na sve uvjete u njemu. U Hrvatskoj je 2002. godine izglasan zakon od elektroničkom potpisu u Republici Hrvatskoj, a dopunjen je 2008. godine. Zakonom je omogućeno postojanje pravno-valjanih dokumenata koji postoje isključivo u elektroničkom obliku. [3]

## 2.1. Shema digitalnog potpisa

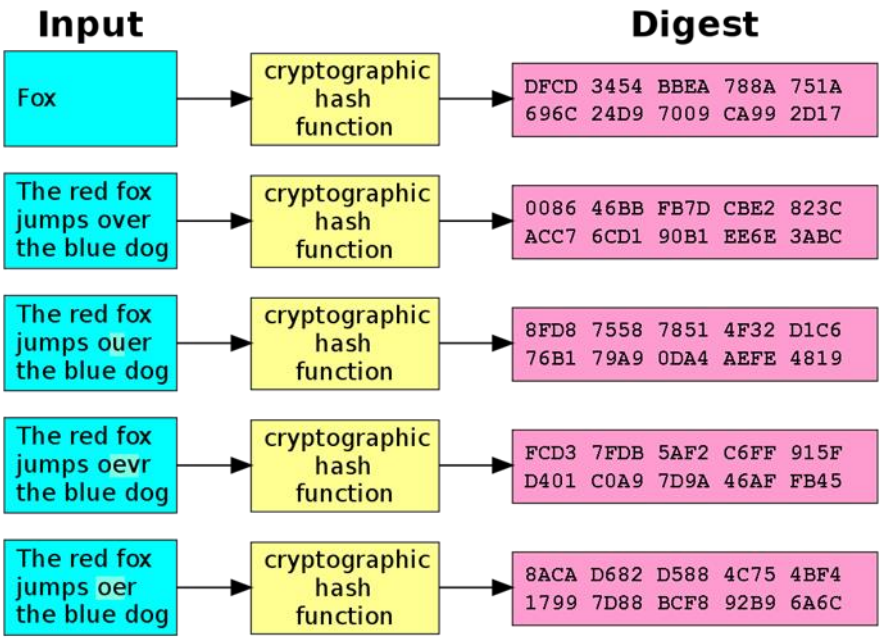
Prema [1], shema digitalnog potpisa se najčešće sastoji od tri algoritma:

- algoritam za generiranje ključeva
- algoritam za potpisivanje
- algoritam za provjeru potpisa

Algoritam za generiranje ključeva nasumično odabire privatni ključ iz skupa mogućih privatnih ključeva, te uz odabrani privatni šalje i adekvatan javni ključ. Kao što i sam naziv kaže, algoritam za potpisivanje na temelju poruke i privatnog ključa stvara jedinstven potpis. Algoritam za provjeru potpisa na temelju poruke, privatnog i javnog ključa odlučuje o autentičnosti poruke. DSA (Digital Signature Algorithm) algoritam koristi zasebne algoritme za elektroničko potpisivanje i šifriranje, dok RSA za obije operacije koristi isti algoritam.[3] U primjeru će biti pokazan protokol razmjene elektronički potpisane poruke između dvije osobe koji je isti bez obzira koji od ova dva navedena algoritma koristimo.

- Alice šifrira poruku pomoću svog privatnog ključa i tako stavlja potpis na dokument
- Alice šalje potpisanu poruku Bobu
- Bob dešifrira poruku koju je dobio od Alice pomoću njezina javnog ključa, te ujedno tako provjerava je li potpis od Alice autentičan.

Kriptografija javnog ključa se temelji na tajnosti privatnog ključa i povjerenju autentičnosti javnog ključa pošiljatelja poruke.[4] Prema [3], problem pri ovakvoj komunikaciji nastaje prilikom potpisivanja dugačkih dokumenata, te se zbog uštede vremena, protokoli elektroničkog potpisivanja se implementiraju pomoću tzv. *hash* funkcija. Kako je navedeno u [5], *hash* funkcija je funkcija koja ulazu proizvoljne duljine pridružuje izlaz fiksne duljine.



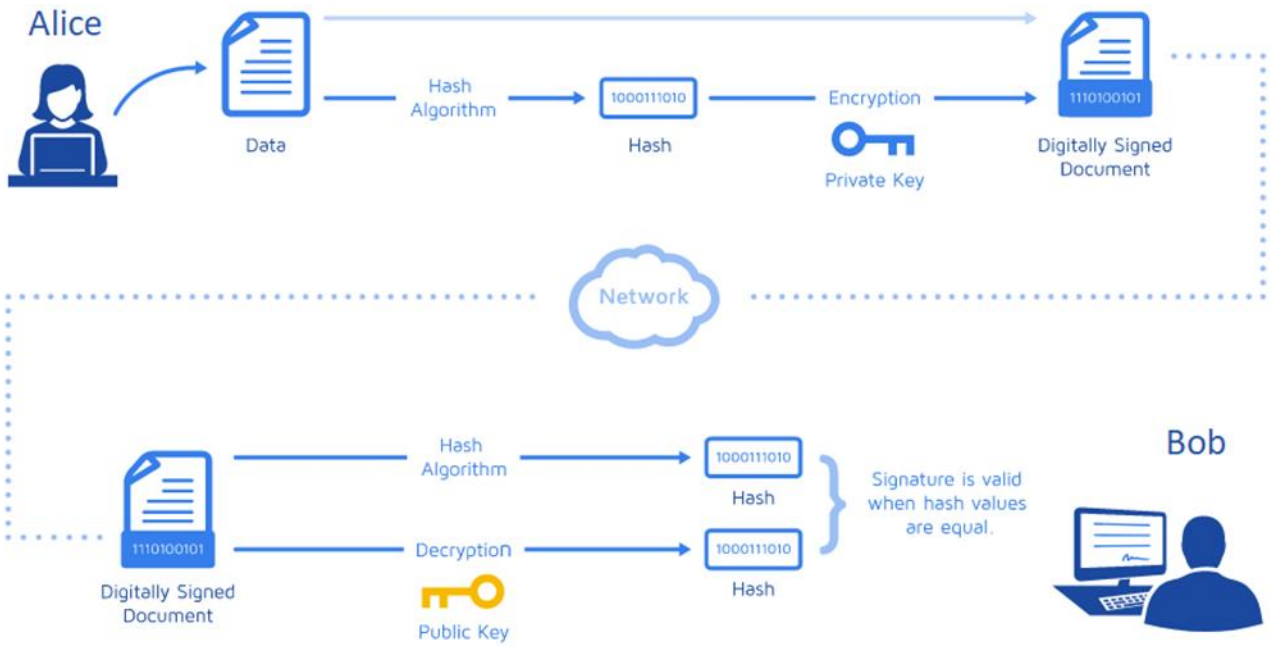

Slika 2.1. Primjer *hash* funkcije [4]

Na slici 2.1. možemo vidjeti da neovisno o broju znakova ulazne poruke dobijemo izlaz koji je uvijek jednake fiksne duljine. *Hash* funkcija je matematički jednostavna za izračunavanje, ali je teško pronaći njezinu inverznu funkciju.

Protokol za elektroničko potpisivanje koji koristi *hash* funkciju bit će praktično opisan na primjeru dvije osobe, Alice i Bob.

- Alice stvara *hash* dokumenta
- Alice šifrira *hash* pomoću privatnog ključa, te tako potpisuje dokument
- Alice šalje potpisani dokument Bobu

Bob dobije potpisani dokument i javni ključ od Alice pomoću kojeg dešifrira dokument. Ako se dva *hash*-a podudaraju, potpis je autentičan.




Slika 2.2. Protokol razmjene elektronički potpisane poruke između dvije osobe

Kako je navedeno u [2], provjeru autentičnosti digitalnog potpisa, odnosno njegovog autora i podataka također možemo provesti pomoću zaporki, ispitnog zbroja, CRC (Cyclic Redundancy Check) provjere, enkripcijom s privatnim ključem, enkripcijom s javnim ključem čiji je princip objašnjen na početku ovog potpoglavlja, te digitalnim certifikatima. Dokazivanje autentičnosti pomoću zaporki se temelji na korisničkom imenu i odgovarajućoj zaporci. Neispravni ispitni zbroj (eng. checksum) ukazuje na neovlaštenu izmjenu podataka, ali se prvenstveno koristi za provjeru ispravnosti primljenih podataka. CRC provjera se temelji na dijeljenju polinoma u svrhu utvrđivanja ispravnosti podataka, te je slična ispitnom zbroju. Enkripcija privatnim ključem se temelji na poznavanju privatnog ključa pošiljatelja. Privatni ključ se na pošiljateljevoj strani koristi u svrhu šifriranja poruke, a na primateljevoj strani za dešifriranje poruka. Najpraktičnije je prije uspostave komunikacije na svako računalo spremiti privatne ključeve računala od kojih se očekuje poruka. Digitalni certifikat je elektronička „lozinka“ koja omogućuje osobama ili organizacijama da na siguran način razmjenjuju podatke preko interneta koristeći PKI (eng. Public Key Infrastructure). Ovlaštena tijela djeluju kao posrednici u komunikaciji između korisnika ili računala. Spomenuta tijela potvrđuju njihove identitete i razmjenjuje njihove javne ključeve.

## 2.2. Primjena digitalnog potpisa u praksi

Korištenje digitalnog potpisa u praksi zahtjeva specifične komponente. Potrebne komponente su:

- Program za šifriranje na potpisničkoj strani
- Pametna kartica koja služi za pohranjivanje tajnog ključa i certifikata
- Računalo s instaliranim čitačem za navedenu karticu
- Program za provjeru autentičnosti na strani primatelja

Prema [3], proces razmjene dokumenta odvija se u jedanaest koraka:

1. Potpisnik umeće karticu u čitač na računalu
2. Označavanje željenih dokumenata koji će se digitalno potpisati
3. Pomoću algoritma kreira se *hash* poruke
4. *Hash* se šifrira pomoću privatnog ključa
5. Jedinstveni niz podataka dobiven šifriranjem se dodjeljuje dokumentu
6. Primatelj pomoću verifikacijskog programa na računalu i javnog ključa osobe ili ustanove provjerava vjerodostojnost potpisa
7. Ako se uspješno dešifrira digitalni potpis na certifikatu, primatelj potvrđuje vjerodostojnost potpisa i povezuje javni ključ s identitetom potpisnika
8. Primatelj zatim koristi isti *hash* algoritam na dokumentu kako bi stvorio vlastiti *hash*
9. Pomoću javnog ključa primatelj dešifrira digitalni potpis kako bi dobio *hash* poslanog dokumenta
10. Program za verifikaciju utvrđuje poklapaju li se vrijednosti poslanog i primljenog *hash*-a, te se provjerava je li javni odgovara tajnom ključu pošiljatelja
11. Pravna ispravnost dokumenta ili transakcije se potvrđuje ako su sve provjere uspješno završene

Digitalni potpis se u Republici Hrvatskoj može vidjeti na uslugama Privredne banke Zagreb, internetskoj stranici HITRO.HR koji predstavlja servis Vlade Republike Hrvatske za ubrzanu komunikaciju građana i poslovnih subjekata s državnom upravom, a svoje pametne kartice također posjeduju i ministri Vlade. [3]

# 3. ALGORITMI ZA OBRADU DIGITALNOG POTPISA

U ovom poglavlju će biti opisana tri algoritma za obradu digitalnog potpisa, DSA (Digital Signature Algorithm), RSA (Rivest – Shamir – Adleman) i Rabinov algoritam. Svaki od navedenih algoritama će biti detaljno opisan i pokazan na praktičnom primjeru.

## 3.1. DSA (Digital Signature Algorithm)

DSA algoritam je patentirao David W. Kravitz, 1991. godine. Predložen je od strane nacionalnog instituta za standarde i tehnologiju, a temelji se na matematičkom konceptu modularnog eksponenciranja i diskretnog logaritamskog problema.[6] DSA algoritam radi na temelju enkripcije s javnim ključem. Algoritam koristi dva ključa, javni i privatni. Privatni ključ se koristi za stvaranje digitalnog potpisa čija se autentičnost može provjeriti pomoću odgovarajućeg javnog ključa. Prema [5], kao i svaka shema digitalnog potpisa, tako se i DSA algoritam sastoji od algoritama za generiranje ključeva, potpisivanje i provjeru potpisa.

### 3.1.1. Generiranje ključeva

Algoritam za generiranje ključeva možemo podijeliti u dvije faze, prva faza je generiranje parametara, a druga faza je stvaranje ključeva.

A) Generiranje parametara

1. Odabrati odobrenu *hash* funkciju $H$.
2. $L$ i $N$ moraju imati jednu od navedenih vrijednosti: $(1024, 160)$, $(2048, 224)$ $ili$ $(3072, 256)$.
3. Odabrati $N$-bitni prosti broj $q$.
4. Odabrati L-bitni prosti broj $p$ takav da $p - 1$ predstavlja višekratnik broja $q$.
5. Nasumično odabrati cijeli broj $h$ iz intervala $\{2 \ldots p - 2\}$
6. Izračunati $g = h^{\frac{(p-1)}{q}} mod\ p\ > 1$. U rijetkim slučajevima kada je $g = 1$, postupak se ponavlja s drugačijom vrijednosti $h$. Često se odabire $h = 2$.

B) Stvaranje ključeva

1. Izabrati nasumičan cijeli broj *x* iz intervala $\{1 \ldots q - 1\}$
2. Izračunati $y = g^x\ mod\ p$

$X$ predstavlja privatni, a $y$ javni ključ. Prema [6], javni ključ bi trebao biti objavljen i proslijeđen primatelju putem sigurnog komunikacijskog kanala, dok bi privatni ključ trebao ostati tajan.

### 3.1.2. Potpisivanje poruke

1. Nasumično odabrati broj $k$ iz intervala $\{1\dots\ q-1\}$
2. Izračunati $r = (g^k \bmod p) \bmod q$. U specijalnim slučajevima kada je $r = 0$, postupak je potrebno ponoviti s drugačijom vrijednosti $k$.
3. Izračunati $s = (k^{-1}(H(m) + xr)) \bmod q$. U specijalnim slučajevima kada je $s = 0$, postupak je potrebno ponoviti s drugačijom vrijednosti $k$. $H(m)$ predstavlja kriptografsku *hash* funkciju koja je primijenjena na poruci $m$.
4. Potpis je $(r, s)$.

### 3.1.3. Provjera potpisa

1. Provjeriti jesu li zadovoljeni uvjeti: $0 < r < q\ i\ 0 < s < q$.
2. Izračunati $w = s^{-1} \bmod q$.
3. Izračunati $u_1 = H(m) \cdot w \bmod q$.
4. Izračunati $u_2 = (rw) \bmod q$.
5. Izračunati $v = \left((g^{u_1} y^{u_2}) \bmod p\right) \bmod q$.
6. Potpis je vjerodostojan ako vrijedi $v = r$.

### 3.1.4. Primjer digitalnog potpisa koristeći DSA algoritam

U praksi, $q$ predstavlja 160-bitni prosti broj, a $H$ je 160-bitni broj koji predstavlja SHA-1 (Secure Hash Algorithm 1) poruke $M$, ali algoritam i dalje funkcionira s malim brojevima.

1. Stvaranje ključeva

   Alice odabire proste brojeve $q = 17$ i $p = 52$. Alice odabire $h = 5$ iz intervala $\{2\dots p-2\}$ i izračunava $g = h^{\frac{(p-1)}{q}} \bmod p$, $g = 21$. Alice odabire nasumičan broj iz intervala $\{1\dots q-1\}$ što će predstavljati njezin privatni ključ, $x = 7$, te računa $y = g^x \bmod p, y = 5$. Javni ključ Alice je $y = 5$.
2. Potpisivanje poruke

   Alice nasumično odabire $k = 11$ iz intervala $\{1\dots q-1\}$. Alice računa $r = (g^k \bmod p) \bmod q$, te dobije da je $r = 5$. Budući da je $r \neq 0$, postupak se nastavlja. Alice računa $k^{-1} \bmod q = 30$ i

$H(M) = 13$. Alice računa $s = k^{-1}(h + xr)\ mod\ q, s = 12$. Alice šalje poruku $M$ i potpis $(r, s) = (5,12)$.

3. Provjera potpisa

   Bob provjerava jesu li zadovoljeni uvjeti $0 < r = 5 < 17\ i\ 0 < s = 12 < 17$, budući da je sve ispravno, proces se nastavlja. Bob računa $w = s^{-1}\ mod\ q, w = 10$. Bob pomoću iste *hash* funkcije računa $H(M) = 13$. Bob računa $u_1$ i $u_2$ iz sljedećih izraza: $u_1 = H(m) \cdot w\ mod\ q, u_2 = (rw)\ mod\ q;\ u_1 = 11, u_2 = 16$. Bob izračunava $v = ((g^{u_1} y^{u_2})\ mod\ p)\ mod\ q, v = 5$. Budući da je $v = r$, Bob potvrđuje vjerodostojnost potpisa.

## 3.2. RSA algoritam

Ron Rivest, Adi Shamir i Leonard Adleman su 1977. godine prvi puta opisali ovaj algoritam, a samo ime potječe od početnih slova njihovih prezimena.[2] RSA algoritam je jedan od prvih algoritama koji se temelji na šifriranju s javnim ključem i koji se koristio kao osiguranje razmjene podataka. Korisnik RSA algoritma mora napraviti, a zatim i objaviti javni ključ koji se temelji na dva velika prosta broja, koja moraju ostati tajna i pomoćnoj vrijednosti. Pomoću javnog ključa bilo tko može šifrirati poruku, ali dešifrirati je može samo osoba koja posjeduje te proste brojeve. Prema [7], RSA je relativno spor algoritam, te ga se rijetko koristi kod izravnog šifriranja podataka. Kako je objašnjeno u [8], RSA algoritam možemo podijeliti na više dijelova koji će biti objašnjeni u sljedećim odjeljcima.

### 3.2.1. Stvaranje ključa

1. Odabrati dva velika prosta broja $p$ i $q$ koji moraju ostati tajni. Iz sigurnosnih aspekata, dobro je da oba broja budu izabrana nasumično.
2. Izračunati $n = pq$, $n$ se objavljuje kao dio javnog ključa.
3. Izračunati $\lambda(n) = lcm(p - 1, q - 1)$, tj. pronaći najmanji zajednički višekratnik brojeva $p - 1$ i $q - 1$. U originalnim dokumentima RSA, $\lambda(n)$ se računao kao $\lambda(n) = (p - 1)(q - 1)$.
4. Odabrati cijeli broj $e$ tako da vrijedi $1 < e < \lambda(n)$, također $e$ i $\lambda(n)$ ne smiju imati zajedničkih djelitelja osim broja 1.
5. Izračunati $d$ kao $d \equiv e^{-1}\ (mod\ \lambda(n))$, odnosno $d \cdot e \equiv 1\ (mod\ \lambda(n))$.

### 3.2.2. Šifriranje poruke

Prema [8], ako osoba B želi poslati poruku osobi A, osoba B mora znati javni ključ osobe A kako bi mogla šifrirati poruku, dok osoba A mora iskoristiti svoj privatni ključ kako bi dešifrirala poruku. Osoba A šalje svoj javni ključ osobi B kroz sigurni komunikacijski kanal, dok privatni ključ osobe A mora ostati tajan.

1. Nakon što je osoba B zaprimila javni ključ osobe A, poruku $M$ pretvara u cijeli broj $m$.
2. Šifrirati poruku pomoću javnog ključa $e$, $c \equiv m^e \ (mod\ n)$, proces se unatoč velikim brojevima može ubrzati korištenjem modularnog eksponenciranja.

### 3.2.3. Dešifriranje poruke

Kako je navedeno u [8], osoba A primljenu poruku može dešifrirati koristeći svoj privatni ključ $d$.

1. Izračunati $c^d \equiv (m^e)^d \equiv m \ (mod\ n)$
2. Pomoću $m$, osoba A može inverznom metodom doći do originalne poruke $M$.

### 3.2.4. Primjer digitalnog potpisa koristeći RSA algoritam

1. Alice odabire dva prosta broja, $p = 43$ i $q = 67$. Izračunava $n = 2881$, prema formuli $n = pq$. Zatim se izračunava $\lambda(n) = lcm(p-1, q-1)$, $\lambda(n) = 462$. U idućem koraku Alice nasumično odabire $e_1$ takav da vrijedi $1 < e_1 < \lambda(n)$, $e_1 = 37$. Pomoću izraza $d \equiv e^{-1} \ (mod\ \lambda(n))$ računamo $d = 25$.
2. Javni ključ je (2881, 37). Alice želi šifrirati poruku $M$, koju je prvo potrebno pretvoriti u cijeli broj $m$, npr. $m = 77$. Za šifriranje koristi izraz $c = m^e \ (mod\ n)$, $c = 2094$.
3. Bob kako bi dešifrirao poruku izračunava $m = c^d \ mod\ n$, $m$ =77.

## 3.3. Rabinov algoritam

Rabinov algoritam je predložen od strane Michaela O. Rabina 1979. godine i predstavlja jednu od metoda kreiranja digitalnog potpisa. Rabinov algoritam je jedna od prvih shema digitalnog potpisa, te jedini koji je težinu krivotvorenja potpisa povezao s problemom cjelobrojne faktorizacije.[9]

Prema [9], ako kažemo da je $H$ neka *hash* funkcija, a $m$ poruka koja treba biti potpisana i

$H(m)^{\frac{p-1}{2}} \ mod = 1$ i $H(m)^{\frac{q-1}{2}} \ mod\ q = 1$

potpis $S$ je zadan jednadžbom:

$$S = \left(\left(p^{q-2}\, H(m)^{\frac{q+1}{4}} \bmod q\right)p + \left(q^{p-2}\, H(m)^{\frac{p+1}{4}} \bmod p\right)q\right) \bmod (p \cdot q).$$

Svatko može provjeriti

$H(m) = S^2\ mod\ (p \cdot q)$, ako je vrijednost $n = p \cdot q$ javno objavljena.

### 3.3.1. Stvaranje ključa

1. Odabrati dva prosta broja $p$ i $q$ koji su otprilike veličine $\frac{k}{2}$ bita, te izračunati $n = p \cdot q$.
2. Pošiljatelj nasumice odabire parametar $b$ iz intervala $\{1, \ldots, n\}$.
3. Javni ključ je $(n, b)$.
4. Privatni ključ je $(p, q)$.

### 3.3.2. Digitalni potpis

Kako je navedeno u [9], za potpisivanje poruke potrebno je slijediti navedene korake:

1. Kako bi potpisao poruku, pošiljatelj nasumično odabire nadopunu $U$ i izračunava $m \cdot U \bmod n$.
2. Pošiljatelj nakon toga rješava $x(x + b) \bmod n = m \cdot U \bmod n$.
3. Ukoliko rješenje ne postoji, pošiljatelj ponovno nasumično odabire nadopunu $U$ i računa iznova.
4. Potpis na poruci $m$ je $(U, x)$.

### 3.3.3. Provjera potpisa

1. Nakon što je poruka isporučena, primatelj računa $x(x + b) \bmod n$ i $m \cdot U \bmod n$, te provjerava jesu li isti.

Navedeni postupci se odnose na originalni Rabinov algoritam koji je nesiguran s obzirom na to da ne koristi kriptografsku *hash* funkciju. Sigurnija verzija algoritma koristi *hash* funkciju $H$ i pojednostavljen je tako da se na početku odabire $b = 0$. Prema [9], za implementaciju sigurnije i pojednostavljene verzije Rabinovog algoritma potrebno je slijediti korake koji će biti definirani u sljedećim odjeljcima.

### 3.3.4. Stvaranje ključa

1. Pošiljatelj odabire dva prosta broja $p$ i $q$ koji su otprilike veličine $\frac{k}{2}$ bita, takvi da vrijedi: $p \equiv q \equiv 3(mod\ 4)$ i računa $n = p \cdot q$.

2. Javni ključ je $n$.
3. Privatni ključ je $(p, q)$.

### 3.3.5. Šifriranje poruke

Prema [10], poruka $M$ se može šifrirati tako da se prvo pretvori u broj takav da vrijedi $m < n$, koristeći reverzibilno mapiranje. Šifrat se računa pomoću izraza $c = m^2 \ mod \ n$.

### 3.3.6. Digitalni potpis

1. Za potpis poruke $m$ pošiljatelj nasumično odabire nadopunu $U$ i računa $H(m, U)$.
2. Ako $H(m, U)$ nezadovoljava $x^2 \equiv H(m, U) \ mod \ n$, pošiljatelj odabire novu nadopunu $U$.
3. Pošiljatelj računa jednu vrijednost $x$, koja rješava jednadžbu $x^2 = H(m, U) \ mod \ n$.
4. Potpis na poruci je uređeni par $(U, x)$.

### 3.3.7. Dešifriranje poruke

Poruka $m$ se može rekonstruirati iz šifrata $c$ prateći korake navedene u [10]:

1. Potrebno je izračunati kvadratni korijen od $c \ modulo \ p \ i \ q$ koristeći formule: $m_p = c^{\frac{1}{4}(p+1)} \ mod \ p$ i $m_q = c^{\frac{1}{4}(q+1)} \ mod \ q$.
2. Pomoću proširenog Euklidovog algoritma pronaći $y_p$i $y_q$ tako da vrijedi $y_p \cdot p + y_q \cdot q = 1$.
3. Pomoću kineskog teorema o ostatcima pronaći četiri kvadratna korijena od $c \ modulo \ n$: $r_1 = (y_p \cdot p \cdot m_q + y_q \cdot q \cdot m_p) \ mod \ n$; $r_2 = n - r_1$; $r_3 = (y_p \cdot p \cdot m_q - y_q \cdot q \cdot m_p) \ mod \ n$; $r_4 = n - r_3$. Jedno od četiri rješenja predstavlja originalu poruku, ali bez dodatnih informacija ne može se sa sigurnošću utvrditi koje od njih sadrži originalnu poruku.

### 3.3.8. Provjera potpisa

1. Primatelj sada posjeduje poruku $m$ i potpis $(U, x)$. Kako bi provjerio vjerodostojnost potpisa računa $x^2 \ mod \ n$ i $H(m, U)$, te provjerava jesu li isti.

### 3.3.9. Primjer digitalnog potpisa koristeći originalni Rabinov algoritam

1. Alice želi poslati Bobu poruku. Nasumično odabire proste brojeve $p = 11$ i $q = 19$, te računa $n = p \cdot q = 209$. Nasumično odabire parametar $b = 17, b \in \{1, \dots, n\}$. Alice sada zna svoj javni $(209, 17)$ i privatni ključ $(11, 19)$.
2. Kako bi potpisala poruku $m = 34$, nasumično odabire nadopunu $U = 13$ i računa $m \cdot U \ mod \ n = 24$. Zatim izračunava jednadžbu $x(x + b) \bmod n = m \cdot U \bmod n$, te za rješenja dobija $x \in$

$\{63,129,272,338\}$. U slučaju da jednadžba nije imala rješenja, Alice bi nasumično izabrala novu nadopunu *U*, te ponovila postupak. Potpis koji Alice stavlja na poruku je uređeni par $(34, 129)$.

3. Bob dobije poruku $m$ zajedno s potpisom $(34, 129)$ i želi se uvjeriti kako mu je upravo Alice poslala tu poruku. Bob računa $x(x + b) \bmod n$ i $m \cdot U \bmod n$, te potvrđuje da mu je poruka stigla od Alice.

# 4. PROGRAMSKO RJEŠENJE

U ovom poglavlju bit će opisane tehnologije korištene prilikom izrade programskog rješenja, te će biti prikazana i objašnjena implementacija algoritama za obradu digitalnog potpisa (DSA, RSA i Rabinov algoritam) u programskom jeziku C.

## 4.1. Programski jezik C

Programski jezik C je proceduralni jezik, a napravio ga je Dennis Ritchie između 1972. i 1973. godine. Programski jezik C je osmišljen u svrhu rješavanja problema u operacijskom sustavu UNIX. U osamdesetima je postao popularan, a danas je jedan od najkorištenijih programskih jezika.[11] U računalnoj industriji, C je jedan od najvažnijih jezika te je do danas ostao jedini koji je prilagođen za sve računalne platforme. Prema [11], korištenje programskog jezika C zahtjeva od programera odlično razumijevanje rada procesora, ulazno-izlaznih sklopova i memorije. U današnje vrijeme C se najviše koristi za izradu sistemskih programa na strani poslužitelja, jezgre raznih operativnih sustava, tj. koristi ga se tamo gdje je brzina izvođenja, kontrola resursa i izravno upravljanje hardverom od velike važnosti.

Prema [12], programski jezik C sadrži 32 ključne riječi i to su riječi koje se ne mogu upotrijebiti za nazive varijabli ili u neke druge svrhe, a neke od njih su: int, long, else, case, char, if, while, return, do, itd. Program napisan u programskom jeziku C se sastoji od niza funkcija. Funkcija bez koje je ne moguće napraviti program je main ( ). Unutar C-a također postoje biblioteke koje se mogu pozvati na početku programa naredbom #include, a one sadrže razne funkcije koje možemo koristiti u našem programu budući da programski jezik C nema ugrađenih funkcija. [13]

Bitno je još naglasiti da nakon svake naredbe unutar programa napisanog u programskom jeziku C moramo staviti znak ; koji označava kraj naredbe.

```c
#include <stdio.h>

int main(void)
{
    printf("hello, world\n");
}
```

Slika 4.1. Primjer jednostavnog programa napisanog u programskom jeziku C

Kako je navedeno u [11], programski jezik C je imao veliki utjecaj na neke programske jezike kao što su C#, Java, JavaScript, Perl, PHP i Python. Neki programi kao što su Mathematica i MATLAB su cijeli ili gotovo cijeli napisani u programskom jeziku C. Bjarne Stroustrup tijekom 1980-ih godina proširuje programski jezik C tako da omogućuje objektno orijentirano programiranje, nazivajući novonastali programski jezik C++. Neke od mana C-a u odnosu na C++ je ta što C ne podržava učahurivanje (eng. encapsulation) i višeobličje (eng. polymorphism) , nije pogodan za skrivanje informacija za razliku od C++ kojemu učahurivanje to omogućuje. Postoji još nekoliko mana koje je potrebno navesti kao što je nemogućnost nasljeđivanja i rukovanje za izuzetcima. Unatoč navedenim manama, programski jezik C ima i prednosti u odnosu na C++. Statičko inicijaliziranje je puno sigurnije nego u C++, C nam daje veću kontrolu nad napisanim kodom kada ga izvršimo, a programski jezik C nam također omogućuje bolju suradnju s drugim programskim jezicima budući da većina dopušta direktno pozivanje funkcija napisanih u C-u. Potrebno je još naglasiti da je kompajliranje programa napisanog u C-u puno brže od programa napisanog u C++.

## 4.2. Implementacija DSA algoritma

U ovom potpoglavlju bit će prikazana implementacija DSA algoritma u programskom jeziku C. Algoritam je implementiran prateći korake koji su navedeni u [6].

```c
int main() {

    srand(time(NULL));
    generateDSAKeys();

    generateUserKeys();

    printf("Private key x = %u\n", x);
    printf("Public key y = %u\n", y);

    unsigned char data[] = "Diplomski rad.";

    printf("Poruka: %s\n", data);

    unsigned int length = sizeof(data) / sizeof(char);

    signing(data, length);

    printf("Potpis na poruci: (r = %u, s = %u)\n", r, s);

    bool valid = verifying(data, length);
    printf("Da li je poruka valjana?: %d\n", valid );

    return 0;
}
```

Kod 1: *Main* funkcija DSA algoritma

Unutar *main* funkcije pozivaju se funkcije pomoću kojih se generiraju DSA parametri, javni i privatni ključ, funkcija za potpisivanje i verifikaciju potpisa. Prema [6], prvi korak u implementaciji DSA algoritma je generiranje DSA parametara.

```
void generateDSAKeys() {
    do
        q = rand();
    while (!isPrime(q) || !hasBits(q, N));

    unsigned int p_minus_one = q;

    unsigned int mul = 1;
    while (true) {
        p_minus_one = q * mul;
        p = p_minus_one + 1;
        if (hasBits(p, L) && isPrime(p))
            break;
        ++mul;
    }

    printf("p = %u, q= %u\n", p, q);

    unsigned int h;
    while (true) {
        do
            h = rand();
        while (h < 2 || h > p - 2);
        printf("h = %u\n", h);

        g = modularExponentiationFast(h, mul, p);

        if (g != 1) break;
    }
    printf("g = %u\n", g);
}
```

Kod 2: Funkcija za generiranje DSA parametara

Kako je navedeno u [6], potrebno je izabrati $N$-bitni prosti broj $q$ i $L$-bitni prosti broj $p$ takav da je $p-1$ višekratnik od $q$. Definirano je da $L$ i $N$ trebaju imati jednu od zadanih vrijednosti $(1024, 160), (2048, 224)$ $ili$ $(3072, 256)$, ali zbog programskih ograničenja u ovome radu uzima se $L = 32, N = 8$. Budući da parametri $p$ i $q$ moraju biti prosti brojevi, potrebno je implementirati funkciju koja će provjeravati jesu li nasumično izabrani brojevi prosti.

```
bool isPrime(unsigned short n) {
    if (n <= 3)
        return n > 1;

    else if (n % 2 == 0 || n % 3 == 0)
        return false;

    unsigned short i = 5;
    while (i * i <= n) {
        if (n % i == 0 || n % (i + 2) == 0)
            return  false;
        i += 6;
    }
    return true;
}
```

Kod 3: Funkcija za provjeru prostih brojeva

Prosti brojevi su svi brojevi koji su djeljivi samo s brojem 1 ili sa samim sobom. Prema [14], svi prosti brojevi veći od 6 se mogu zapisati kao $6k \pm 1$, iz razloga jer se svi cijeli brojevi mogu zapisati kao $6k + i$, za neki cijeli broj k, te za $i = -1, 0, 1, 2, 3\ i\ 4$. Parametar h mora biti nasumično odabran cijeli broj iz intervala $\{2 \ldots p - 2\}$. Parametar g se računa prema formuli $g = h^{\frac{(p-1)}{q}} mod\ p$, te ako se za rješenje dobije $g = 1$, izračun je potrebno ponoviti za neki drugi h. Budući da se u kriptografiji koriste jako veliki brojevi čiji rezultat eksponenciranja prelazi programske granice, potrebno je implementirati funkciju za modularno eksponenciranje koja prema [15], računa ostatak kada se cijeli broj $b$ (baza) na potenciju $e$ (eksponent) podjeli s pozitivnim cijelim brojem $m$.

```
unsigned int modularExponentiationFast(long b, long e, long m) {
    long result = 1;

    while (e > 0)
    {
        if (e % 2 == 1)

            result = (result * b) % m;

        e = e >> 1;


        b = (b * b) % m;

    }

    return result;
}
```

Kod 4: Funkcija za brzo modularno eksponenciranje

Nakon uspješno generiranih DSA parametara, potrebno je generirati javni i privatni ključ.

```
unsigned int x, y;
void generateUserKeys() {
    do
        x = rand();
    while (x < 1 || x > q - 1);

    y = modularExponentiationFast(g, x, p);
}
```

Kod 5: Generiranje javnog i privatnog ključa

Budući da se za kreiranje javnog ključa $y$ koristi formula $y = g^x \ mod \ p$, potrebno je ponovno koristiti funkciju za modularno eksponenciranje. Kada su javni i privatni ključ kreirani, pozivamo funkciju za potpisivanje poruke koja će za ovaj primjer glasiti „Diplomski rad.“, kao što je vidljivo u kodu 1.

```
unsigned int r, s;
void signing(unsigned char* data, unsigned int length) {
    unsigned int k;
    unsigned char h;
    unsigned s2;
    while (true) {
        while (true) {
            do
                k = rand();
            while (k < 1 || k > q - 1);
            printf("k = %u\n", k);

            r = modularExponentiationFast(g, k, p) % q;
            if (r != 0) break;
        }

        h = hash(data, length);

        s = (modularExponentiationFast(k, q - 2, q) * (h + x * r) % q) % q;
        if (s != 0) break;
    }
}
```

Kod 6: Funkcija za potpisivanje poruke

Funkciji za potpisivanje poruke potrebno je predati sadržaj i duljinu poruke. Potpis na poruci činit će uređeni par $(r, s)$, stoga je potrebno pronaći njihove vrijednosti. Parametri $r$ i $s$ se računaju prema formulama $r = (g^k \ mod \ p) \ mod \ q$, $s = (k^{(q-2)}(H(m) + xr)) \ mod \ q$. Ako su parametri $r$ ili $s$ jednaki nuli, izračun je potrebno ponoviti s nekom drugom vrijednosti $k$. Prije izračunavanja parametra s, potrebno je pronaći *hash* vrijednost naše poruke. *Hash* funkcija je funkcija koja ulazu

proizvoljne duljine uvijek daje izlaz iste duljine kao što možemo vidjeti na slici 2.1. u ovome radu. U ovome primjeru koristi se SHA256 iz *openssl* biblioteke. Iako koristimo SHA256, nužno je da *hash* bude duljine $N = 8$, a ne 256 bitova, stoga u *hash* funkciji uzimamo najljeviji byte, tj. 8 najljevijih bitova.

```
unsigned char hash(const unsigned char* data, unsigned int length) {

    unsigned char* hashed = malloc(SHA256_DIGEST_LENGTH * sizeof(unsigned char));
    // unsigned char hashed[SHA256_DIGEST_LENGTH];
    SHA256(data, length, hashed);

    // as written on Wikipedia: only the leftmost N bits of the hash output are used
    unsigned char output = hashed[0];

    return output;
}
```

Kod 7: Funkcija za izračunavanje *hash* vrijednosti poruke

Nakon što je poruka potpisana, potrebno je provjeriti autentičnost potpisa.

```
bool verifying(unsigned char* data, unsigned int length) {
    if (!(0 < r < q && 0 < s < q)) return false;

    unsigned char h = hash(data, length);

    unsigned int w = multiplicativeInverse(s, q);
    printf("w = %u\n", w);
    unsigned int u1 = (h * w) % q;
    printf("u1 = %u\n", u1);
    unsigned int u2 = (r * w) % q;
    printf("u2 = %u\n", u2);
    unsigned int v = (((unsigned long)modularExponentiationFast(g, u1, p) * (unsigned
    long)modularExponentiationFast(y, u2, p)) % p) % q;
    printf("v = %u\n", v);
    return v == r;
}
```

Kod 8: Funkcija za provjeru autentičnosti poruke

Potpis je autentičan ako je zadovoljen uvjet $v = r$. Kako bi izračunali parametar $v$, potrebno je prvo izračunati parametre o kojemu ovisi. Za izračunavanje parametra $w$ koristi se modularni multiplikativni inverz. Prema [16], modularni multiplikativni inverz od $a$ se može definirati kao broj $x$ takav da vrijedi $ax \equiv 1\ (mod\ n)$. Modularni multiplikativni inverz postoji ako i samo ako je parametrima $a$ i $n$ jedini zajednički djelitelj broj 1. Za izračunavanje se koristi prošireni Euklidov algoritam.

```
unsigned int multiplicativeInverse(unsigned int a, unsigned int n) {
    long t = 0;
    long tNew = 1;
    long tTemp;
    long r = n;
    long rNew = a;
    long rTemp;

    long quotient;

    while (rNew != 0) {
        quotient = r / rNew;
        tTemp = t;
        t = tNew;
        tNew = tTemp - quotient * tNew;

        rTemp = r;
        r = rNew;
        rNew = rTemp - quotient * rNew;
    }

    if (t < 0) t += n;

    return t;
}
```

Kod 9: Funkcija za izračunavanje modularnog multiplikativnog inverza

Program na kraju uspoređuje parametre $v$ i $r$, ako su identični javit će kako je potpis valjan.

```
p = 2147489917, q= 227
h = 2015134303
g = 2127438325
Private key x = 51
Public key y = 1657748621
Poruka: Diplomski rad.
k = 86
Potpis na poruci: (r = 115, s = 157)
w = 107
u1 = 186
u2 = 47
v = 115
Da li je potpis valjan?: 1
```

Slika 4.2. Rezultat implementacije DSA algoritma

## 4.3. Implementacija RSA algoritma

U ovom potpoglavlju bit će prikazana implementacija RSA algoritma u programskom jeziku C. Algoritam je implementiran prateći korake koji su navedeni u [8].

```
int main() {
    srand(time(NULL));

    generateRSAKeys();

    printf("p = %u, q= %u\n", p, q);
    printf("n = %u\n", n);
    printf("lambda = %u\n", lambda);
    printf("e = %u\n", e);
    printf("d = %u\n", d);

    char plaintext[] = "Diplomski rad.";

    printf("Tekst poruke: %s\n", plaintext);

    unsigned int* cyphertext = RSAEncrypt(plaintext, sizeof(plaintext) / sizeof(char));
    unsigned int length = sizeof(plaintext) / sizeof(char);

    signing(plaintext, length);

    for (int i = 0; i < sizeof(plaintext) / sizeof(char); ++i)
        printf("%u\t", cyphertext[i]);
    printf("\n");

    unsigned char* restoredText = RSADecrypt(cyphertext, sizeof(plaintext) /
    sizeof(char));

    bool valid = verifying(cyphertext, length);

    printf("Originalna poruka: %s\n", restoredText);
    printf("Je li potpis na poruci valjan?: %d\n", valid);

    free(cyphertext);
    free(restoredText);

    return 0;
}
```

Kod 10: *Main* funkcija RSA algoritma

Unutar *main* funkcije pozivaju se funkcije pomoću kojih se generiraju RSA parametri, funkcije za šifriranje i dešifriranje originalne poruke, te funkcije za potpisivanje poruke i provjeru autentičnosti potpisa. Prema [8], prvi korak u implementaciji RSA algoritma je generiranje ključeva.

```
void generateRSAKeys() {
    // 1.
    do
        p = rand();
    while (!isPrime(p));

    do
        q = rand();
    while (!isPrime(q));

    // 2.
    n = (unsigned int)p * (unsigned int)q;

    // 3.
    lambda = lcm(p - 1u, q - 1u);

    // 4.
    while (true) {
        e = rand();
        if (1 < e && e < lambda && gcd(e, lambda) == 1)
            break;
    }

    // 5.
    d = multiplicativeInverse(e, lambda);
}
```

Kod 11: Funkcija za generiranje RSA ključeva

U prvom koraku budući da parametri $p$ i $q$ moraju biti nasumično izabrani prosti brojevi, koristimo funkciju prikazanu u kodu 3. Funkcija *rand* će nasumično odabirati brojeve dok se ne zadovolji zadani uvjet, a u idućem koraku je potrebno pomnožiti parametre $p$ i $q$. U trećem se koraku određuje parametar $\lambda$ pronalaženjem najmanjeg zajedničkog višekratnika od $(p-1, q-1)$.

```
unsigned int gcd(unsigned int p, unsigned int q) {
    unsigned int temp;
    while (q != 0) {
        temp = p % q;
        p = q;
        q = temp;
    }
    return p;
}


unsigned int lcm(unsigned short p, unsigned short q) {
    return (unsigned int)p * (unsigned int)q / gcd(p, q);
}
```

Kod 12: Funkcija za izračun najvećeg zajedničkog djelitelja i najmanjeg zajedničkog višekratnika

Kako je navedeno u [17], pomoću formule $lcm(a,b) = \frac{|a \cdot b|}{gcd(a,b)}$ problem najmanjeg zajedničkog višekratnika neka dva broja možemo svesti na problem određivanja najvećeg zajedničkog djelitelja ta dva broja. U idućem koraku je potrebno odrediti parametar $e$, takav da vrijedi $1 < e < \lambda(n)$, a budući da parametrima $e$ i $\lambda(n)$ jedini zajednički djelitelj smije biti broj 1, ponovno koristimo gore navedenu funkciju *gcd*. Za izračunavanje parametra $d$ u petom koraku generiranja RSA ključeva, potrebno je koristiti multiplikativni inverz prikazan u kodu 9. Nakon što su generirani RSA ključevi, potrebno je šifrirati poruku. Poruka koja će se koristiti je: „Diplomski rad.“, kao što je prikazano u kodu 10. Za šifriranje poruke se koristi funkcija *RSAEncrypt*.

```
unsigned int* RSAEncrypt(char* plaintext, unsigned long length) {

    unsigned int* cyphertext = malloc(length * sizeof(unsigned int));

    for (unsigned int i = 0; i < length; ++i)
        cyphertext[i] = modularExponentiationFast((unsigned int)plaintext[i], e, n);

    return cyphertext;
}
```

Kod 13: Funkcija *RSAEncrypt* za šifriranje poruke

Funkciji za šifriranje je potrebno predati poruku i njezinu duljinu, te je potrebno alocirati memoriju za šifrat. Kako je navedeno u [8], šifriranje se izvodi pomoću izraza $c = m^e \ (mod\ n)$. Šifrat predstavlja polje, te se svaki element unutar polja šifrira zasebno pomoću gore navedenog izraza. Budući da se u kriptografiji koriste jako veliki brojevi čiji rezultat eksponenciranja prelazi programske granice, potrebno je koristiti funkciju za modularno eksponenciranje koja je prikazana u kodu 4. Funkciji za modularno eksponenciranje predajemo tri parametra, parametar nultog elementa polja poruke koji je pomoću operatora eksplicitne konverzije (*eng. cast operator*) pretvoren u broj, te parametre $e$ i $d$ koji su generirani funkcijom *generateRSAKeys*. Nulti element polja predstavlja slovo 'D' te on poprima vrijednost 68 definiranu ASCII tablicom. Nakon što je poruka uspješno šifrirana, potrebno ju je potpisati.

```
unsigned char* hash(unsigned char* plaintext, unsigned int lenght) {
	unsigned char* hashed = malloc(SHA256_DIGEST_LENGTH * sizeof(unsigned char));
	SHA256(plaintext, lenght, hashed);
	return hashed;
}

unsigned int* signature;
void signing(unsigned char* plaintext, unsigned int length) {
	unsigned char* hashed = hash(plaintext, length);

	signature = malloc(SHA256_DIGEST_LENGTH * sizeof(unsigned int));

	for (int i = 0; i < SHA256_DIGEST_LENGTH; ++i)
	    signature[i] = modularExponentiationFast((unsigned int)hashed[i], d, n);

	free(hashed);
}
```

Kod 14: Funkcija za potpisivanje poruke

Funkciji za potpisivanje je potrebno proslijediti sadržaj i duljinu poruke. Potpis predstavlja polje *unsigned charova* koje alociramo unutar tijela funkcije za potpisivanje. Svaki znak iz *hasha* eksponenciramo s privatnim ključem $d\ modulo\ n$ pri čemu koristimo funkciju za modularno eksponenciranje prikazanu u kodu 4, te spremamo u polje *signature*.

```
char* RSADecrypt(unsigned int* cyphertext, unsigned long length) {

    char* plaintext = malloc(length * sizeof(char));

    for (unsigned int i = 0; i < length; ++i)
        plaintext[i] = (char)modularExponentiationFast(cyphertext[i], d, n);

    return plaintext;
}
```

Kod 15: Funkcija *RSADecrypt* za dešifriranje poruke

Prema [8], dešifriranje poruke se vrši pomoću izraza $c^d = m\ (mod\ n)$. Kod dešifriranja se također koristi funkcija za modularno eksponenciranje koja je prikazana u kodu 4, a za rezultat se dobije originalna poruka. Na kraju je potrebno provjeriti je li potpis na poruci autentičan.

```
bool verifying(unsigned int* cyphertext, unsigned int length) {

	unsigned char* restoredText = RSADecrypt(cyphertext, length);
	unsigned char* hashed = hash(restoredText, length);

	bool areEqual = true;
	for (int i = 0; i < SHA256_DIGEST_LENGTH; ++i)
		if (hashed[i] != modularExponentiationFast((unsigned int)signature[i], e,
		n)) {
			areEqual = false;
			break;
		}

	free(signature);
	free(restoredText);
	free(hashed);

	return areEqual;

}
```

Kod 16: Funkcija za provjeru autentičnosti potpisa

Za provjeru potpisa se koristi javni ključ, svaki znak potpisa se eksponencira s parametrom *e* modulo n, ako se rješenja podudaraju potpis je autentičan.

```
p = 1667, q= 55313
n = 92206771
lambda = 46074896
e = 27505001
d = 38339433
Tekst poruke: Diplomski rad.
46537978        67463733        24798909        12779613        63476815        24055131        25387334        39507425
  67463733        55768996        74993293        5450058 73435483        59196869        0
Originalna poruka: Diplomski rad.
Je li potpis na poruci valjan?: 1
```

Slika 4.3. Rezultat implementacije RSA algoritma

## 4.4. Implementacija Rabinovog algoritma

U ovom potpoglavlju bit će prikazana implementacija Rabinovog algoritma u programskom jeziku C. Algoritam je implementiran prateći korake koji su navedeni u [10].

```
int main() {
    srand(time(NULL));
    generateRabinKeys();

    printf("p = %u, q = %u\n", p, q);
    printf("n = %u\n", n);

    unsigned char plaintext[] = "Diplomski";

    printf("Originalna poruka: %s\n", plaintext);

    unsigned int length = sizeof(plaintext) / sizeof(unsigned char);


    unsigned int* cyphertext = RabinEncrypt(plaintext, length);

    for (int i = 0; i < sizeof(plaintext) / sizeof(char); ++i)
        printf("%u\t", cyphertext[i]);
    printf("\n");

    unsigned char** plainTextArr = RabinDecrypt(cyphertext, sizeof(plaintext) /
    sizeof(char));

    for (int i = 0; i < 4; ++i) {
        unsigned char* plainTextPtr = plainTextArr[i];
        for (int j = 0; j < length; ++j)
            printf("%c\t", plainTextPtr[j]);
        printf("\n");
    }


    free(cyphertext);
    for (int i = 0; i < 4; ++i)
        free(plainTextArr[i]);
    free(plainTextArr);



    return 0;
}
```

Kod 17: *Main* funkcija Rabinovog algoritma

Unutar *main* funkcije pozivaju se funkcije pomoću kojih se generiraju ključevi Rabinovog algoritma, te funkcije za šifriranje i dešifriranje originalne poruke. Prema [10], prvi korak u implementaciji Rabinovog algoritma je generiranje ključeva.

```
unsigned short p, q;
unsigned int n;
void generateRabinKeys() {
    while (n < 255) {

        while (true) {
            p = rand();
            if (isPrime(p) && (p % 4) == 3) break;
        }

        while (true) {
            q = rand();
            if (isPrime(q) && (q % 4) == 3) break;
        }

        n = (unsigned int)p * (unsigned int)q;
    }
}
```

Kod 18: Funkcija za generiranje Rabinovih ključeva

Parametri $p$ i $q$ moraju biti nasumično izabrani prosti brojevi tako da vrijedi $p \equiv 3\ mod\ 4, q \equiv 3\ mod\ 4$. Nakon što su uvjeti za $p$ i $q$ zadovoljeni, potrebno je odrediti javni ključ $n$. Kada su poznati javni i privatni ključ potrebno je šifrirati poruku, koja će za ovaj primjer glasiti „Diplomski“.

```
unsigned int* RabinEncrypt(unsigned char* plaintext, unsigned int length) {

    unsigned int* cyphertext = malloc(length * sizeof(unsigned int));

    for (unsigned int i = 0; i < length; ++i) {
        unsigned  int m = plaintext[i];
        cyphertext[i] = ((unsigned long)m * (unsigned long)m) % n;
    }
    return cyphertext;
}
```

Kod 19: Funkcija za šifriranje poruke

Funkciji za šifriranje potrebno je predati sadržaj i duljinu poruke, te je potrebno alocirati memoriju za šifrat. Šifrat predstavlja polje i svaki element polja se šifrira posebno. Prema [10], poruka $m$ se može izračunati iz šifriranog teksta $c$, računanjem kvadratnog korijena $c$ modulo $n$.

```
unsigned char** RabinDecrypt(unsigned int* cyphertext, unsigned long length) {

    unsigned char* plaintext1 = malloc(length * sizeof(unsigned char));
    unsigned char* plaintext2 = malloc(length * sizeof(unsigned char));
    unsigned char* plaintext3 = malloc(length * sizeof(unsigned char));
    unsigned char* plaintext4 = malloc(length * sizeof(unsigned char));

    for (unsigned int i = 0; i < length; ++i) {
        unsigned int mp = modularExponentiationFast(cyphertext[i], ((unsigned int)p + 1)
        / 4, (unsigned int)p);
        unsigned int mq = modularExponentiationFast(cyphertext[i], ((unsigned int)q + 1)
        / 4, (unsigned int)q);

        struct Pair y = extendedEuclideanAlgorithm((unsigned int)p, (unsigned int)q);
        long yp = y.s;
        long yq = y.t;

        long r1, r2, r3, r4;

        long t11 = yp * (long)p * (long)mq;
        long t12 = yq * (long)q * (long)mp;

        r1 = t11 % (long)n;
        r1 += t12 % (long)n;
        r1 %= (long)n;
        while (r1 < 0) r1 += n;

        r2 = n - r1;

        r3 = t11 % (long)n;
        r3 -= t12 % (long)n;
        r3 %= (long)n;
        while (r3 < 0) r3 += n;

        r4 = n - r3;

        plaintext1[i] = (unsigned char)r1;
        plaintext2[i] = (unsigned char)r2;
        plaintext3[i] = (unsigned char)r3;
        plaintext4[i] = (unsigned char)r4;

    }

    unsigned char** plainTextArr = malloc(4 * sizeof(unsigned char*));

    plainTextArr[0] = plaintext1;
    plainTextArr[1] = plaintext2;
    plainTextArr[2] = plaintext3;
    plainTextArr[3] = plaintext4;

    return plainTextArr;
}
```

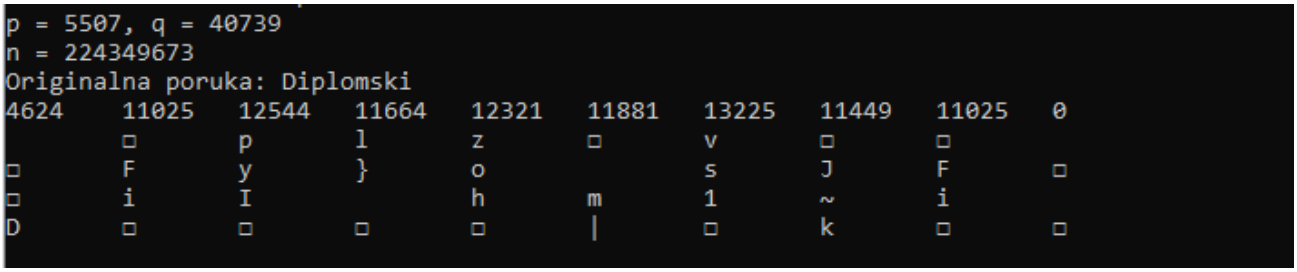
Kod 20: Funkcija za dešifriranje poruke

Prvo je potrebno pronaći parametre $m_p$ i $m_q$ prema formulama $m_p = c^{\frac{1}{4}(p+1)} \, mod \, p$, $m_p = c^{\frac{1}{4}(q+1)} \, mod \, q$. Zatim je potrebno izračunati $y_p$ i $y_q$ tako da vrijedi $y_p \cdot p + y_q \cdot q = 1$, koristeći prošireni Euklidov algoritam.

```
struct Pair extendedEuclideanAlgorithm(unsigned int a, unsigned int b) {
    long s = 0, oldS = 1, t = 1, oldT = 0, r = b, oldR = a;

    long quotient, rTemp, tTemp, sTemp;

    while (r != 0) {
        quotient = oldR / r;

        rTemp = oldR;
        oldR = r;
        r = rTemp - quotient * r;

        sTemp = oldS;
        oldS = s;
        s = sTemp - quotient * s;

        tTemp = oldT;
        oldT = t;
        t = tTemp - quotient * t;
    }

    struct Pair pair = {oldS, oldT};
    return pair;
}
```

Kod 21: Prošireni Euklidov algoritam

Prema [18], prošireni Euklidov algoritam se koristi u aritmetici i programiranju, te predstavlja proširenje Euklidovog algoritma. Osim što se pomoću njega računaju najveći zajednički djelitelji cijelih brojeva a i b, računaju se i Bézoutovi koeficijenti koji predstavljaju cijele brojeve x i y tako da vrijedi $a \cdot x + b \cdot y = gcd\,(a, b)$, gdje gcd predstavlja najveći zajednički djelitelj. Nakon što su izračunati parametri $y_p$ i $y_q$, potrebno je pronaći četiri kvadratna korijena od $c$ modulo $n$ za što će se koristiti kineski teorem o ostatcima. Prema [19] i [20], kineski teorem o ostatcima predstavlja teorem iz teorije brojeva koji kaže da ako su poznati ostatci Euklidske podjele od cijelog broja $n$ s nekoliko cjelobrojnih brojeva, tada se jedinstveno može odrediti ostatak dijeljenja $n$ s produktom tih cjelobrojnih brojeva pod uvjetom da je djeliteljima jedini zajednički djelitelj broj 1. Jedan od četiri dobivena rezultata predstavlja originalu poruku, ali bez dodatnih informacija se sa sigurnošću ne može utvrditi koji od navedenih predstavlja originalu poruku.




Slika 4.4. Rezultat implementacije Rabinovog algoritma

# 5. ZAKLJUČAK

Tema ovog diplomskog rada bila je opisati algoritme za obradu digitalnog potpisa i implementirati ih u programskom jeziku C. Na početku rada je opisana povijest nastajanja digitalnog potpisa i rani početci njihove implementacije kao i njihovo unaprjeđivanje tijekom vremena. Opisani su osnovni principi rada digitalnog potpisa, njihova primjena u praksi i zakonske regulative povezane s korištenjem digitalnog potpisa na službenim dokumentima u Republici Hrvatskoj. U sljedećem poglavlju su teorijski obrađeni DSA, RSA i Rabinov algoritam. Detaljno su opisani postupci njihove implementacije, te je za svaki algoritam napisan praktični primjer koristeći male brojeve kako bi funkcioniranje algoritama bilo jasnije i preglednije. U idućem poglavlju koristeći opisane korake iz prethodnog poglavlja, implementirani su algoritmi koristeći programski jezik C. Pomoću implementiranih algoritama se šifrira i potpisuje ulazna poruka, zatim se šifrirana poruka dešifrira pomoću privatnog ključa, a na kraju se potvrđuje autentičnost potpisa. Tijekom implementacije bilo je raznih programskih ograničenja, te je većina uspješno riješena pomoću teorema iz teorije brojeva, grane matematike koja se bavi proučavanjem cijelih brojeva. Problem korištenja brojeva od 2048 bita u implementaciji DSA algoritma je ostala neriješena budući da se unutar programskog jezika C takav broj ne može spremiti u varijablu, pa se sukladno tome u primjeru koriste brojevi do 8 bita. Digitalni potpis se već dugi niz godina koristi u razmjeni poruka, podataka i u novčanim transakcijama putem interneta, a u budućnosti se može očekivati još veći intenzitet korištenja, budući da se konstantno radi na poboljšanju njihove sigurnosti.

# 6. LITERATURA


[1] https://en.wikipedia.org/wiki/Digital_signature [15.06.2019]

[2] https://www.cis.hr/www.edicija/LinkedDocuments/CCERT-PUBDOC-2007-02-182.pdf [15.06.2019]

[3] https://bib.irb.hr/datoteka/481946.Zovkic-Vrbanec_-_Digitalni_potpis.pdf [16.06.2019]

[4] Christof Paar, Jan Pelzl - Understanding Cryptography: A Textbook for Students and Practitioners [17.06.2019]

[5] https://en.wikipedia.org/wiki/Cryptographic_hash_function [18.06.2019]

[6] https://en.wikipedia.org/wiki/Digital_Signature_Algorithm [18.06.2019]

[7] https://www.di-mgt.com.au/rsa_alg.html [20.06.2019]

[8] https://en.wikipedia.org/wiki/RSA_(cryptosystem) [20.06.2019]

[9] https://en.wikipedia.org/wiki/Rabin_signature_algorithm [22.06.2019]

[10] https://en.wikipedia.org/wiki/Rabin_cryptosystem [23.06. 2019]

[11] https://en.wikipedia.org/wiki/C_(programming_language) [20.07.2019]

[12] Brian W. Kernighan – C Programming Language, 2nd edition [20.07.2019]

[13] https://web.math.pmf.unizg.hr/~singer/Prog_Add/c.pdf [20.07.2019]

[14] https://en.wikipedia.org/wiki/Primality_test [31.08.2019]

[15] https://en.wikipedia.org/wiki/Modular_exponentiation [31.08.2019]

[16] https://en.wikipedia.org/wiki/Modular_multiplicative_inverse [01.09.2019]

[17] https://en.wikipedia.org/wiki/Least_common_multiple [05.09.2019]

[18] https://en.wikipedia.org/wiki/Extended_Euclidean_algorithm [10.09.2019]

[19] George E. Andrews – Number Theory, 1994. [14.09.2019]

[20] https://en.wikipedia.org/wiki/Chinese_remainder_theorem [10.09.2019]

# 7. SAŽETAK

Naslov: Algoritmi za obradu digitalnog potpisa

Digitalni potpis omogućuje utvrđivanje identiteta pošiljatelja i provjeru autentičnosti poslane elektroničke poruke ili dokumenta. Digitalni potpis je ekvivalentan ručnom potpisu. Matematička shema digitalnog potpisa se temelji na javnom i privatnom ključu, te se sastoji od algoritama za generiranje ključeva, potpisivanje poruke i provjeru potpisa. DSA, RSA i Rabinov algoritam su među poznatijim i korištenijim algoritmima za implementaciju digitalnog potpisa. DSA algoritam se temelji na matematičkom konceptu modularnog eksponenciranja i diskretnog logaritamskog problema, dok su RSA i Rabinov algoritam povezani s problemom cjelobrojne faktorizacije.

Ključne riječi: digitalni potpis, DSA, RSA, Rabinov algoritam, javni ključ, privatni ključ

## 8. ABSTRACT

Title: Digital Signature Processing Algorithm

A digital signature algorithm enables effective identification of a sender and the authenticity verification of a transmission. A performed digital signature algorithm is equal to a manual signature. The mathematical schema of a digital signature is based on mechanisms set forth by the concepts of a public and a private key, and implies key generation algorithms as well as message signing and signature verification mechanisms. The DSA, RSA and Rabin algorithms are one of widely known and implemented algorithms used for the implementation of the digital signature concept. The DSA algorithm is based on a mathematical concept that involves modular exponentiation and discrete logarithmic problem, whereas the RSA and Rabin signature algorithm are based on mathematical concepts that involve the problem of integer factorization.

Key words: digital signature, DSA, RSA, Rabin's signature algorithm, public key, private key

# 9. ŽIVOTOPIS

Ivan Hmelik je rođen 09. rujna 1993. godine u Virovitici. Po završetku osnovne škole Bilje u Bilju, upisuje Isusovačku klasičnu gimnaziju s pravom javnosti u Osijeku i maturira 2012. godine, a zatim upisuje preddiplomski studij računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija. 2016. godine završava preddiplomski studij i stječe zvanje inženjera prvostupnika računarstva, te upisuje diplomski studij računarstva na istoimenom fakultetu, smjer procesno računarstvo.

Vlastoručni potpis:

_____

Ivan Hmelik