IZRADA WEB APLIKACIJE S ALGORITMIMA IZ MODULARNE ARITMETIKE

WEB APPLICATION DEVELOPMENT WITH ALGORITHMS BASED ON MODULAR ARITHMETIC

Vedad Varupa
Filozofski fakultet Univerziteta u Zenici
Zenica
Bosna i Hercegovina

Hermina Alajbegović
Mašinski fakultet Univerziteta u Zenici
Zenica
Bosna i Hercegovina

REZIME

Internet se koristi u skoro svim sferama društvenog života, nerijetko i za financijske transakcije i razmjenu ugovora između dva fizička ili pravna lica. Kako je internet nesiguran komunikacijski kanal, korisnici imaju sve veću potrebu da zaštite svoje podatke od mogućih napada. Kriptografske metode koriste se za provjeru autentičnosti poruke, autentičnosti korisnika, za digitalni potpis i čine našu komunikaciju putem interneta sigurnom i pouzdanom. U ovom radu ćemo prikazati osnovne korake pri izradi web aplikacije koja se može koristiti za šifriranje i dešifriranje poruke. Razvoj web aplikacije može se podijeliti na dva dijela: 1. Back-end (eng. Back-end development) 2. Front-end (eng. Front-end development). Serverski dio aplikacije (back-end) je urađen u Spring framework-u, koristeći programski jezik Java. Pojedinačne implementirane kriptografske algoritme, predstavljene kroz formu metoda, bilo je neophodno uvezati s odgovarajućim vizuelnim formama koje se nalaze na grafičkom interfejsu (klijentskoj strani aplikacije). Pomenuta veza je uspostavljena slijedeći REST principe, a sama implementacija grafičkog interfejsa je realizirana korištenjem Angular frameworka, zasnovanog na programskom jeziku JavaScript.

Ključne riječi: web aplikacije, Java, JavaScript, Angular, kriptografija, RSA, modularna aritmetika

ABSTRACT

The Internet is used in almost all spheres of social life, often for financial transactions and the exchange of contracts between two individuals or legal entities. As the internet is an insecure communication channel, users have an increasing need to protect their data from possible attacks. Cryptographic methods are used to verify the authenticity of the message, the authenticity of the user, for the digital signature and make our communication over the Internet secure and reliable. This paper will present the basic steps in creating a web application that can be used to encrypt and decrypt a message. Web application development can be divided into two parts: 1. Back-end development and 2. Front-end development. The server part of the application (back-end) is done in the Spring framework, using the Java programming language. The individual implemented cryptographic algorithms, presented through the form of methods, had to be linked to the corresponding visual forms located on the graphical interface (client's part of the application). The mentioned connection was established following the principles of REST and the implementation of the graphical interface was realized using the Angular framework, based on the JavaScript programming language.

Keywords: web applications, Java, JavaScript, Angular, cryptography, RSA, modular arithmetic

1. UVOD

Web aplikacija s algoritmima iz modularne aritmetike je nastala kao rezultat magistarskog rada (vidjeti [4]) u kojem su proučavani najpoznatiji algoritmi modularne aritmetike i RSA algoritam. Ova aplikacija ima prije svega edukativni karakter jer na slikovit način prikazuje rad svih algoritama objašnjenih u pomenutom magistarskom radu. O kojim sve algoritmima je riječ može se vidjeti iz menija web aplikacije u poglavlju 3. ovog rada. Aplikacija se također može koristiti i za šifriranje i dešifriranje poruka. Za izradu web aplikacije izabrani su programski jezici Java i Angular framework. Ovdje nećemo moći detaljno objasniti svaki korak izrade, ali zainteresovani čitalac može vidjeti čitav izvorni kod aplikacije na linku: https://github.com/vedad-varupa/crypto.git. U nastavku ćemo pokušati predstaviti ugrubo glavnu strukturu aplikacije kao i njen izgled i rad. O programskim jezicima Javi i Angular frameworku te korištenim algoritmima u aplikaciji, čitalac može više saznati u [1,2, … 8].

2. IZRADA WEB APLIKACIJE

Prilikom izrade aplikacije u Javi, pridržavali smo se u prvom redu SOLID principa, a nakon toga i DRY, KISS i YAGNI principa. Budući da je zamišljeno da aplikacija bude dostupna online u obliku Web stranice prirodno se nametnula činjenica da se mora koristiti neki 'Java-based' web framework. Nakon opsežnog istraživanja, izbor je sužen na Play i Spring framework, pri čemu je odabran Spring framework prvenstveno zbog kvalitetne dokumentacije. Korišten je REST arhitekturni stil jer je planirano da front-end web aplikacije bude urađen u Angular frameworku, što omogućava da se postigne visok nivo separacije na nivou back-end/front-end-a. Pomenuti nivo separacije je postignut tako što je kreiran specifičan REST API za svaki algoritam iz magistarskog rada ([4]), koje će klijent (u ovom slučaju to je Angular) konzumirati služeći se HTTP pozivima. Implementacija algoritama je i dalje ostala strogo na back-endu, a Angular nam je omogućio da služeći se Typescript-om, HTML-om i CSS-om adekvatno 'prezentujemo' te podatke u vidu odgovarajućeg grafičkog sučelja (GUI-a). Više algoritama je korišteno u okviru ove aplikacije, ali ovdje ćemo istaknuti RSA algoritam koji se koristi za šifriranje i dešifriranje poruke i *Prošireni Euklidov algoritam* koji računa ne samo $nzd(a,b)$ već i cijele brojeve x i y takve da je $ax + by = nzd(a,b)$:

RSA algoritam

1. *Osoba B slučajno odabire dva različita velika prosta broja* p *i* q *za koje vrijedi* $n = pq$ *i* $\varphi(n) = (p-1)(q-1)$. *Treba paziti da* n *bude otporan na metode faktorizacije.*
2. *Osoba B odabere slučajno broj* e *takav da je* $1 < e < \varphi(n)$ *i* $nzd\big(e, \varphi(n)\big) = 1$. *Zatim koristeći prošireni Euklidov algoritam računa* d *takav da je* $q < d < \varphi(n)$ *i* $ed \equiv 1\ (mod\ \varphi(n))$.
3. *Osoba B objavljuje* (n, e) *u nekoj javnoj bazi ključeva, dok zadržava* (p, q, d) *i* $\varphi(n)$ *kao tajnu.*
4. *Osoba A dohvaća javni ključ od osobe B iz baze ključeva.*
5. *Osoba A svoju poruku* m *šifrira i dobija šifrat* $c = m^e \bmod n$, *(koristeći algoritam "kvadriraj i množi") koji šalje osobi B. Ako je* $m > n$, m *se razbija u blokove.*
6. *Kada osoba B primi poruku* c, *koristi dekripcijski eksponent* d *i računa* $m = c^d \bmod n$ *i dobiva otvoreni tekst* m.

2.1. Back-end razvoj

Glavni zadatak *back-enda* je osigurati da aplikacija ispravno funkcioniše i da se aplikacija poveže s korisničkim interfejsom koji je razvijen od *front-end* developera. Serverski dio aplikacije (back-end) je urađen u Spring framework-u, koristeći programski jezik Java. Pojedinačne implementirane kriptografske algoritme, predstavljene kroz formu metoda, bilo je neophodno uvezati s odgovarajućim vizuelnim formama koje se nalaze na grafičkom interfejsu

(klijent strane aplikacije). Koristili smo Spring *open-source Java platformu* koja posjeduje izuzetno kvalitetnu podršku za razvoj web aplikacija. Zbog stroge separacije između klijenta (*front-enda*) i servera (*back-enda*) koristili smo *REST* softverski stil arhitekture weba. *REST* označava način komunikaciju između klijenta i servera prilikom korištenja mrežnih resursa pomoću *HTTP* protokola, a sastoji se od skupa principa koji definišu kako bi se trebali koristiti web standardi, poput HTTP-a i URI-ja. Sistemi koji poštuju *REST* restrikcije nazivaju se *RESTful* sistemi. U *back-end* dijelu aplikacije kreiran je određen broj *RESTful API-ja* koji bivaju konzumirani od strane *front-end* dijela aplikacije. U aplikaciji je ispoštovan princip modularnosti i slojevitosti tako što je glavni dio aplikacije podijeljen u tri modula/sloja. Prvi sloj je sloj *Controllera* koji sadrži skup *end-pointa* (URI) koji su namapirani na odgovarajuće metode koje služe za pozivanje *RESTful API-ja*.

```
package com.cripto.cripto;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;
import java.math.BigInteger;
import java.util.ArrayList;

@RestController
public class CriptoController {
    private CriptoService criptoService;
    public CriptoController(CriptoService criptoService) {
        this.criptoService = criptoService;
    }
    @GetMapping("/extended-euklid-gcd/{firstNumber}/{secondNumber}")
    public GeneralContainer exteuclidGCD(@PathVariable String firstNumber, @PathVariable
String secondNumber) {
        BigInteger bigInteger_a = new BigInteger(firstNumber);
        BigInteger bigInteger_b = new BigInteger(secondNumber);
        return criptoService.exteuclidGCD(bigInteger_a, bigInteger_b);
    }
    @GetMapping(value = "/rsa-algorithm/{textmessage}")
    public ArrayList<String> rsaAlgorithm(@PathVariable String textmessage) {
        return criptoService.rsaAlgorithm(textmessage);
    }
}
```

Slika 1. Primjer metode exteuclidGCD i rsaAlgorithm iz Controllera

U drugom sloju kojeg čini interfejs nazvan *CriptoService* navedene su deklaracije metoda koje bivaju implementirane u trećem sloju nazvanom *CriptoServiceImpl.* U biti sva implementacija kriptografskih algoritama se dešava u trećem sloju. Veza između prvog i drugog sloja se ostvaruje preko dependency injection ("ubrizgavanje zavisnosti"), što predstavlja jednu od glavnih karakteristika Spring frameworka. A uloga drugog sloja (*interfejsa*) služi za postizanje apstrakcije kao jedne od glavnih osobina OOP-a. Ukratko, treći sloj (konkretna klasa) je samo implementacija drugog sloja (interfejsa).

```
package com.cripto.cripto;
import java.math.BigInteger;
import java.util.ArrayList;
import java.util.List;
public interface CriptoService {
   public GeneralContainer exteuclidGCD(BigInteger firstNumber, BigInteger secondNumber);
   public ArrayList<String> rsaAlgorithm(String textmessage);
}
```

Slika 2. Primjer definicije metode exteuclidGCD i rsaAlgorithm iz CriptoService

```
package com.cripto.cripto;
import java.math.BigInteger;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import org.springframework.stereotype.Service;
@Service
public class CriptoServiceImpl implements CriptoService {
```

```
   @Override
   public GeneralContainer exteuclidGCD(BigInteger firstNumber, BigInteger secondNumber) {
      BigInteger[] x = new BigInteger[3];
      BigInteger[] y = new BigInteger[3];
      BigInteger[] t = new BigInteger[3];
      List<EuclidContainer> listOfEuclidContainers = new ArrayList<>();
      x[0] = BigInteger.ONE;
      x[1] = BigInteger.ZERO;
      x[2] = firstNumber;
      y[0] = BigInteger.ZERO;
      y[1] = BigInteger.ONE;
      y[2] = secondNumber;
      while (y[2] != BigInteger.ZERO) {
         BigInteger q = x[2].divide(y[2]);
         t[0] = x[0].subtract(y[0].multiply(q));
         t[1] = x[1].subtract(y[1].multiply(q));
         t[2] = x[2].subtract(y[2].multiply(q));
         check(firstNumber, secondNumber, t);
         x[0] = y[0];
         x[1] = y[1];
         x[2] = y[2];
         check(firstNumber, secondNumber, x);
         y[0] = t[0];
         y[1] = t[1];
         y[2] = t[2];
         check(firstNumber, secondNumber, y);
         listOfEuclidContainers.add(new EuclidContainer(q, x[0], x[1], x[2], y[0], y[1],
y[2]));
      }
      StringBuilder stringBuilder = new StringBuilder();
      // @formatter:off
      stringBuilder.append("(").append(x[0]).append(")*").append(firstNumber).append(" +
").append("(").append(x[1])
            .append(")*").append(secondNumber).append(" = ").append(x[2]);
      // @formatter:on
      return new GeneralContainer(listOfEuclidContainers, stringBuilder.toString());
   }
   public static void check(BigInteger a, BigInteger b, BigInteger[] w) {
      if (a.multiply(w[0]).add(b.multiply(w[1])) != w[2]) {
         System.out.println("*** Provjera nije uspjela: " + a + " " + b);
      }
   }
   @Override
   public ArrayList<String> rsaAlgorithm(String textmessage) {
      ArrayList<String> lista = new ArrayList<>();
      BigInteger p = largePrime(2048);
      BigInteger q = largePrime(2048);
      BigInteger n = n(p, q);
      BigInteger phi = getPhi(p, q);
      BigInteger e = genE(phi);
      BigInteger d = extEuclid(e, phi)[1];
      System.out.println("p: " + p);
      System.out.println("q: " + q);
      System.out.println("n: " + n);
      System.out.println("Phi: " + phi);
      System.out.println("e: " + e);
      System.out.println("d: " + d);
      lista.add("p: " + p);
      lista.add("q: " + q);
      lista.add("n: " + p);
      lista.add("Phi: " + p);
      lista.add("e: " + p);
      lista.add("d: " + p);
      String message = textmessage;
      BigInteger cipherMessage = stringCipher(message);
      BigInteger encrypted = encrypt(cipherMessage, e, n);
      BigInteger decrypted = decrypt(encrypted, d, n);
      String restoredMessage = cipherToString(decrypted);
      System.out.println("Original message\r\n" + ": " + message);
      System.out.println("Hash: " + cipherMessage);
      System.out.println("Encrypted: " + encrypted);
      System.out.println("Decrypted: " + decrypted);
      System.out.println("Restored: " + restoredMessage);
      lista.add("Original message\r\n" + ": " + message);
      lista.add("Hash: " + cipherMessage);
      lista.add("Encrypted: " + encrypted);
      lista.add("Decrypted: " + decrypted);
```

```
        lista.add("Restored: " + restoredMessage);
        return lista;
    }
    public static BigInteger stringCipher(String message) {
        message = message. "";
        int i = 0;
        while (i < message.length()) {
            int ch = message. toUpperCase();
        String cipherString = charAt(i);
            cipherString = cipherString + ch;
            i++;
        }
        BigInteger cipherBig = new BigInteger(String.valueOf(cipherString));
        return cipherBig;
    }
    public static String cipherToString(BigInteger message) {
        String cipherString = message.toString();
        String output = "";
        int i = 0;
        while (i < cipherString.length()) {
            int temp = Integer.parseInt(cipherString.substring(i, i + 2));
            char ch = (char) temp;
            output = output + ch;
            i = i + 2;
        }
        return output;
    }
     public static BigInteger getPhi(BigInteger p, BigInteger q) {
        BigInteger phi = p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE));
        return phi;
    }
    public static BigInteger largePrime(int bits) {
        Random randomInteger = new Random();
        BigInteger largePrime = BigInteger.probablePrime(bits, randomInteger);
        return largePrime;
    }
    public static BigInteger gcd(BigInteger a, BigInteger b) {
        if (b.equals(BigInteger.ZERO)) {
            return a;
        } else {
            return gcd(b, a.mod(b));
        }
    }
    public static BigInteger[] extEuclid(BigInteger a, BigInteger b) {
        if (b.equals(BigInteger.ZERO)) {
            return new BigInteger[]{a, BigInteger.ONE, BigInteger.ZERO}; // { a, 1, 0 }
        }
        BigInteger[] vals = extEuclid(b, a.mod(b));
        BigInteger d = vals[0];
        BigInteger p = vals[2];
        BigInteger q = vals[1].subtract(a.divide(b).multiply(vals[2]));
        return new BigInteger[]{d, p, q};
    }
    public static BigInteger genE(BigInteger phi) {
        Random rand = new Random();
          BigInteger e = new BigInteger(2048, rand);
        do {
            e = new BigInteger(2048, rand);
            while (e.min(phi).equals(phi)) {
                e = new BigInteger(2048, rand);
            }
        } while (!gcd(e, phi).equals(BigInteger.ONE));
        return e;
    }
     public static BigInteger encrypt(BigInteger message, BigInteger e, BigInteger n) {
          return message.modPow(e, n);
    }
    public static BigInteger decrypt(BigInteger message, BigInteger d, BigInteger n) {
        return message.modPow(d, n);
    }
    public static BigInteger n(BigInteger p, BigInteger q) {
        return p.multiply(q);
    }
}
```

Slika 3. Primjer metode exteuclidGCD i rsaAlgorithm iz CriptoServiceImpl

2.2. Front-end razvoj

Front-end obuhvaća sve ono što korisnik vidi pri korištenju web aplikacije. Dakle, *front-end* razvoj je fokusiran na klijentski dio. Klijentska strana aplikacije izrađena je korištenjem *Angular* razvojne tehnologije. To je strukturiran framework napravljen na JavaScriptu i služi za pravljenje dinamičkih web aplikacija. *Angular* se sastoji od tri stavke: komponente, servisa i modula. *Angular* je korišten u ovoj aplikaciji zbog svoje brzine rada te sistema koji je zamišljen kao SPA (eng. *Single Page Aplication*). *Angular* nije rješenje za svaku aplikaciju, ali za većinu CRUD aplikacija (Create, Read, Update, Delete) je sasvim dobar.

Osvrnimo se na protokol poziva eksternog *REST API-ja* koji se nalazi na *back-end/server* strani. Za Angular, koji je trenutno u ulozi klijenta, neophodno je da posjeduje importovan *Http* modul da bi bila omogućena konzumacija *REST API-ja.* Da bismo to ostvarili, služeći se ubrizgavanjem zavisnosti (Dependency injection) dodat ćemo u konstruktoru *HttpClient* i time kreirati objekt *http*. Pomenuta klasa *http* importovana je iz @angular/common/http. Prethodno pomenuti objekt posjeduje mnoštvo *HTTP* metoda, tipa: GET, POST, PUT, DELETE itd.

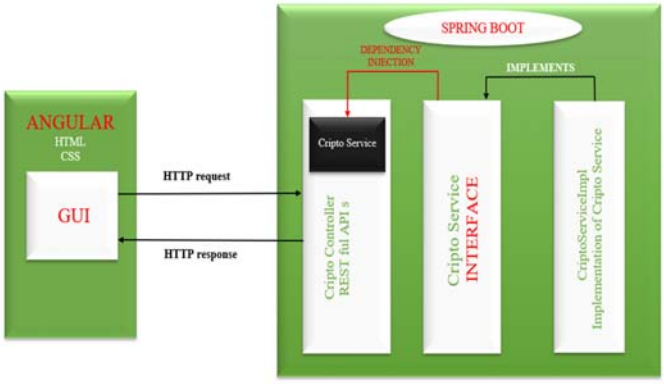
Za potrebe naše aplikacije dovoljna nam je GET metoda. GET metoda kao ulazne parametre mora primiti: URL *back-end* strane, endpoint *REST API-ja* i eventualne ulazne parametre metode koju želimo koristiti.

```
import { Component, OnInit } from '@angular/core';
import { HttpClient } from '@angular/common/http';
@Component({
  selector: 'app-exteuklid',
  templateUrl: './exteuklid.component.html',
  styleUrls: ['./exteuklid.component.css']
})
export class ExteuklidComponent implements OnInit {
  private firstNumber: String;
  private secondNumber: String;
  lista: any[];
  result: string;
  url = 'http://localhost:8081/';
  constructor(private http: HttpClient) { }
  ngOnInit() {}
  calculateExtEucGcd() {
    if (!this.firstNumber || !this.secondNumber) {
      alert("Molimo Vas da unesete oba broja.")
      return;    }
  this.http.get<'euclidContainers'>
 (this.url + 'extended-euklid-gcd/' + this.firstNumber + '/' + this.secondNumber).subscribe(
      (response: any) => {
        this.lista = JSON.parse(JSON.stringify(response.euclidContainers));
        this.result = response.result;
      });
}}
```

Slika 4. Poziv RESTful API-ja

```
import { Component, OnInit } from '@angular/core';
import { HttpClient, HttpHeaders, HttpParams } from '@angular/common/http';
@Component({
  selector: 'app-rsa',
  templateUrl: './rsa.component.html',
  styleUrls: ['./rsa.component.css']
})
export class RsaComponent implements OnInit {
  private firstNumber: any;
  private items: any;
  url = 'http://localhost:8081/';
  constructor(private http: HttpClient) { }
  ngOnInit() {
}
  rsaalgorithm() {
    this.http.get<any>(this.url + 'rsa-algorithm/' + this.firstNumber + "").subscribe(
      (response: any) => {
        this.items = response;
      });
  }}
```

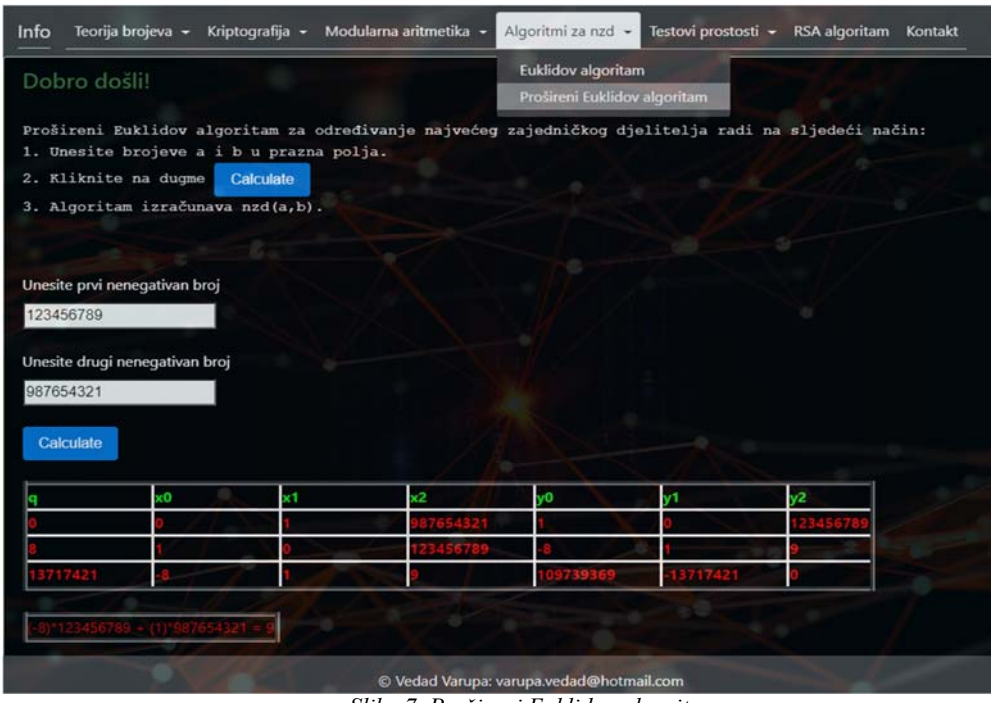
Slika 5. Poziv RESTful API-ja



Slika 6. Grafički prikaz strukture aplikacije

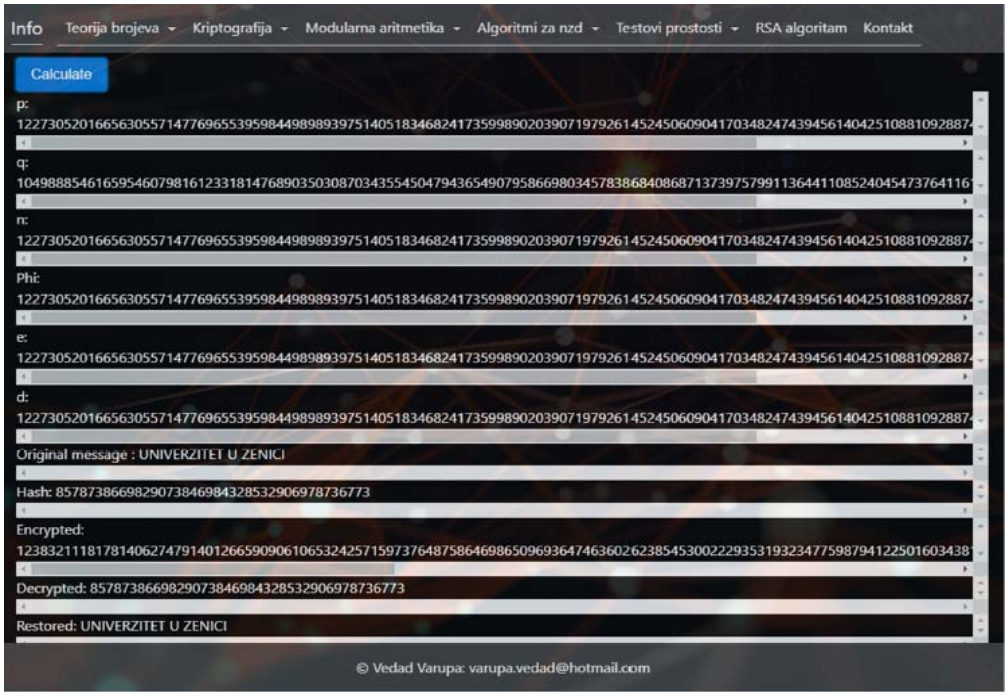
3. TESTIRANJE FUNKCIONALNOSTI WEB APLIKACIJE

Nakon što smo završili s programiranjem web aplikacije, na red je došlo i njeno testiranje. Pokazalo se da je aplikacija brza, tačna i jednostavna za korištenje. Meni web aplikacije se nalazi na vrhu stranice i sadržava podmenije: *Info*, *Teorija brojeva*, *Kriptografija*, *Modularna aritmetika*, *Algoritmi za NZD*, *Testovi prostosti*, *RSA algoritam* i *Kontakt* stranicu (slika 7.). Korisnik bira klikom iz menija algoritam koji želi koristiti. Ovdje je izabran *Prošireni Euklidov algoritam* koji računa ne samo $nzd(a,b)$ već i cijele brojeve x i y takve da je $ax + by = nzd(a,b)$. (slika 7.) Klikom na odgovarajući algoritam otvara se prozor na kojem se nalaze polja za unos podataka. Svakom polju za unos dodan je i njegov tekstualni opis. Klikom na dugme ***Calculate*** dobivamo traženo rješenje. Ako korisnik zaboravi unijeti neki podatak u polje za unos, dobit će na ekranu ispisanu poruku: "Molim Vas unesite sve ulazne podatke".



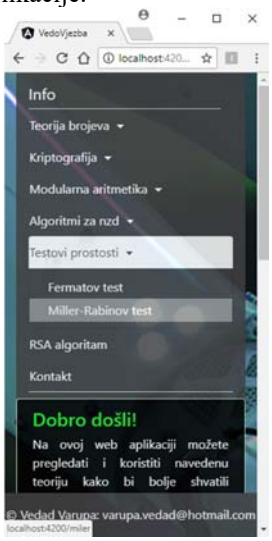
Slika 7. Prošireni Euklidov algoritam.

Na sljedećoj slici je testni primjer rada RSA algoritma. Algoritam je sam izabrao slučajne brojeve te je na osnovu njih šifrirao i dešifrirao poruku "UNIVERZITET U ZENICI".



Slika 8. Redom dobiveni rezultati: p, q, n, phi, e, hash, encrypted i decrypted

Na slici 9. je dat mobilni prikaz aplikacije.



Slika 9. Mobilni prikaz aplikacije

4. NA KOJI NAČIN SE MOŽE APLIKACIJA NADOGRADITI KAKO BI MOGLA ČITAV DOKUMENT ŠIFRIRATI ILI DEŠIFRIRATI?

Prije nego što odgovorimo na gore postavljeno pitanje, trebamo još nešto znati o RSA algoritmu. RSA kriptosistem je jedan od prvih, a ujedno i najpopularniji i najšire korišteni kriptosistem s javnim ključem. Sigurnost RSA je zasnovana u praktičnoj nemogućnosti faktorizacije proizvoda dva velika prirodna broja. Rastavljanje brojeva na proste faktore je fundamentalno težak problem, za čije rješenje računarima treba mnogo vremena koje zavisno od dužine brojeva, može da traje godinama. Svako može šifrirati poruku pomoću javnog ključa, ali dešifrirati je može samo osoba koja zna o kojim se prostim brojevima radi. RSA algoritam je u stanju šifrirati podatke čija dužina u bitovima nije veća od dužine ključa takođe izraženog u bitovima (u našem slučaju to je 2048 bitova). Preporučene veličine ključa se kreču u rasponu od 1024 do 4096 bitova. Dakle, veličina poruke je ograničena na veličinu ključa (ako imamo 2048-bitni ključ, možemo šifrirati poruke do 2048 bitova). Pa kao rezultat toga, često nije moguće direktno šifrirati dokumente s RSA (RSA nije za to ni dizajniran). Što znači da se kriptografija javnog ključa obično koristi u osiguranju povjerljivosti elektronskih komunikacija i pohrane podataka. Kada je potrebno šifrirati poruke veće dužine obično se generiše tajni ključ za neki algoritam iz simetričnih kriptosistema, kao što je AES, te se isti može razmijeniti upotrebom RSA algoritma, dok se onda dalja komunikacija ostvaruje korištenjem AES algoritma. RSA šifriranje je također mnogo sporije od AES šifriranje, pa se ova dva algoritma obično implementirana zajedno pri čemu se RSA koristi za razmjenu tajnog ključa. Otuda ovu aplikacija možemo nadograditi s AES algoritmom. RSA algoritam se koristi i za generisanje digitalnog potpisa te dolazimo do još jedne ideje za nadogradnju aplikacije. Jedna od najznačajnijih primjena kriptografije s javnim ključem je upravo generisanje digitalnih potpisa. Poruka se potpiše privatnim ključem, a onda validnost potpisa se provjerava odgovarajućim javnim ključem. Tako da se razmjena ključa za AES algoritam može obaviti na siguran način, a autentičnost i integritet se održavaju na ključu. Poruka čija je dužina (u bitovima) veća od dužine ključa (u bitovima) se može podijeliti na manje blokove jednakih dužina, tako da se svaki blok može šifrirati RSA algoritmom. Zadnji blok se po potrebi dopuni određenim brojem slova A kako bi i on bio dužine kao prethodni blokovi. Dobiveni šifrati od svakog bloka se redaju jedan pored drugog i zajedno čine šifrat polazne poruke. U tom smislu aplikacija se može

nadograditi tako da polaznu poruku ako je veća od dužine ključa podijeli na blokove na gore opisani način. Više o ovome čitalac može pročitati u [2]. No, kao što je već spomenuto zbog sporosti RSA algoritma više je u upotrebi prva opisana varijanta.

5. ZAKLJUČAK

Izrađena web aplikacija, osim primjene u kriptografiji, se može koristiti kao nastavno pomagalo na dodatnoj nastavu u srednjim školama i u okviru matematičkih i informatičkih predmeta na fakultetima. U tom svojstvu je već korištena na predmetu Kriptografija na II ciklusu studija na Odsjeku za matematiku i informatiku Filozofskog fakulteta Univerziteta u Zenici. Studenti i akademski istraživači će naći korist u ovoj web aplikaciji jer sadrži malu kolekciju najbitnijih i najfrekventnijih kriptografskih algoritama. Također aplikacija je besplatna i dostupna svim korisnicima. Aplikacija se može nadograditi drugim algoritmima, npr. onim koji su spomenuti u poglavlju 4., a tu nadogradnju bi uradili studenti Univerziteta u Zenici kroz svoje seminarske i magistarske radove. Oni bi tako produbljivali svoja znanja iz kriptografije i teorije brojeva te poboljšavali svoje programerske vještine. Spomenimo još da bi bilo interesantno aplikaciju nadograditi tako da se napravi *chat* s *end to end* enkripcijom. To je sistem komunikacije u kojem jedino korisnici koji komuniciraju mogu čitati poruke. U principu, sprječava potencijalne prisluškivače da pristupe kriptografskim ključevima potrebnim za dešifriranje razgovora. Poruke se šifriraju tako da ih samo jedinstveni primalac može dešifrirati, a nikako neko u sredini. Što dovodi do zaključka da osim korisnika koji komuniciraju niko drugi nema mogućnost čitanja tih poruka.

6. REFERENCE

[1] Damjanović, B., Osnove kriptografije s primjerima u programskom jeziku Java, Banja Luka: Besjeda, 2019.
[2] Ibrahimpašić, B., Kriptografija kroz primjere, Bihać, 2011.
[3] Menezes A. J.; Oorschot P. C.; Vanstone, S. A.: Handbook of Applied Cryptografy, CRC Press, Boca Raton, 1996.
[4] Varupa V., Algoritmi u modularnoj aritmetici s primjenama u kriptografiji s javnim ključem, Master rad, Univerzitet u Zenici, 2020.
[5] https://www.baeldung.com/
[6] https://getbootstrap.com/
[7] https://www.w3schools.com/java/
[8] https://www.w3schools.com/angular/