|  |  |  |
| --- | --- | --- |
|  | УНИВЕРЗИТЕТ У НОВОМ САДУ  **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ** |  |

Филип Дутина

**Једно решење програмске подршке за безбедан пренос података путем BroadR-Reach интерфејса**

ДИПЛОМСКИ РАД

- Основне академске студије -

Нови Сад, 2018

|  |  |  |  |
| --- | --- | --- | --- |
| Редни број, **РБР**: | |  | |
| Идентификациони број, **ИБР**: | |  | |
| Тип документације, **ТД**: | | Монографска документација | |
| Тип записа, **ТЗ**: | | Текстуални штампани материјал | |
| Врста рада, **ВР**: | | Завршни (Bachelor) рад | |
| Аутор, **АУ**: | | Филип Дутина | |
| Ментор, **МН**: | | Доц. др Богдан Павковић | |
| Наслов рада, **НР**: | | Једно решење програмске подршке за безбедан пренос података путем BroadR-Reach интерфејса | |
| Језик публикације, **ЈП**: | | Српски / латиница | |
| Језик извода, **ЈИ**: | | Српски | |
| Земља публиковања, **ЗП**: | | Република Србија | |
| Уже географско подручје, **УГП**: | | Војводина | |
| Година, **ГО**: | | 2018 | |
| Издавач, **ИЗ**: | | Ауторски репринт | |
| Место и адреса, **МА**: | | Нови Сад; трг Доситеја Обрадовића 6 | |
| Физички опис рада, **ФО**: (поглавља/страна/ цитата/табела/слика/графика/прилога) | | **<уписати статистику>** | |
| Научна област, **НО**: | | Електротехника и рачунарство | |
| Научна дисциплина, **НД**: | | Рачунарска техника | |
| Предметна одредница/Кqучне речи, **ПО**: | | Безбедан пренос података, енкрипција/декрипција, BroadR-Reach интерфејс, IPv6 протокол | |
| **УДК** | |  | |
| Чува се, **ЧУ**: | | У библиотеци Факултета техничких наука, Нови Сад | |
| Важна напомена, **ВН**: | |  | |
| Извод, **ИЗ**: | | Задатак рада је дизајнирање и имплементација безбедног преноса података са наменске платформе на рачунар. Енкрипција података који се преносе је неопходна како би се осигурала безбедна и сигурна комуникација. Пренос података се врши путем BroadR-Reach интерфјеса и IPv6 протокола. | |
| Датум прихватања теме, **ДП**: | |  | |
| Датум одбране, **ДО**: | |  | |
| Чланови комисије, **КО**: | Председник: | **<име председника комисије>** |
|  | Члан: | **<име члана комисије>** | Потпис ментора |
|  | Члан, ментор: | **<име ментора>** |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Accession number, **ANO**: | |  | |
| Identification number, **INO**: | |  | |
| Document type, **DT**: | | Monographic publication | |
| Type of record, **TR**: | | Textual printed material | |
| Contents code, **CC**: | | Bachelor Thesis | |
| Author, **AU**: | | Filip Dutina | |
| Mentor, **MN**: | | Bogdan Pavković, Phd | |
| Title, **TI**: | | One software solution for safe data transmision via BroadR-Reach interface | |
| Language of text, **LT**: | | Serbian | |
| Language of abstract, **LA**: | | Serbian | |
| Country of publication, **CP**: | | Republic of Serbia | |
| Locality of publication, **LP**: | | Vojvodina | |
| Publication year, **PY**: | | 2018 | |
| Publisher, **PB**: | | Author’s reprint | |
| Publication place, **PP**: | | Novi Sad, Dositeja Obradovica sq. 6 | |
| Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes) | | **<upisati statistiku>** | |
| Scientific field, **SF**: | | Electrical Engineering | |
| Scientific discipline, **SD**: | | Computer Engineering, Engineering of Computer Based Systems | |
| Subject/Key words, **S**/**KW**: | | Safe data transmission, encryption/decription, BroadR-Reach interface, IPv6 protocol | |
| **UC** | |  | |
| Holding data, **HD**: | | The Library of Faculty of Technical Sciences, Novi Sad, Serbia | |
| Note, **N**: | |  | |
| Abstract, **AB**: | | This paper presents design and implementation of safe data transmission from embedded board to the PC. Data encryption is necessary in order to secure safe communication. Data transmission is done via BroadR-Reach interface and IPv6 protocol. | |
| Accepted by the Scientific Board on, **ASB**: | |  | |
| Defended on, **DE**: | |  | |
| Defended Board, **DB**: | President: | **<ime predsednika komisije>** |
|  | Member: | **<ime člana komisije>** | Menthor's sign |
|  | Member, Mentor: | **<ime mentora>** |  |

**Zahvalnost**

Srdačno se zahvaljujem mentoru Doc. dr Bogdanu Pavkoviću na stručnoj pomoći tokom izrade ovog rada.

Posebno se zahvaljujem Draganu Radanoviću na stručnoj pomoći, stalnoj podršci, strpljenju, utrošenom vremenu, razumevanju i savetovanju tokom izrade ovog diplomskog rada.

Takođe se zahvaljujem Institutu RT-RK na ukazanoj prilici da se bolje upoznam sa načinom rada u inženjerskom okruženju i na omogućenom usavršavanja znanja iz date oblasti.

Na kraju, najveće hvala Tatjani, Milošu, Vjeri, Mirku, Jeleni i svima onima koji su mi bili podrška, potpora i motivacija tokom školovanja.

**Sadržaj**

[1. Uvod 1](#_Toc517776823)

[2. Teorijske osnove 3](#_Toc517776824)

[2.1 zFAS razvojna ploča 3](#_Toc517776825)

[2.2 BroadR-Reach interfejs 3](#_Toc517776826)

[2.3 TCP protokol 4](#_Toc517776827)

[2.3.1 Uspostavljanje veze 7](#_Toc517776828)

[2.3.2 Prekid veze 8](#_Toc517776829)

[2.4 IPv6 protokol 8](#_Toc517776830)

[2.4.1 IPv6 adresa 8](#_Toc517776831)

[2.4.2 Struktura IPv6 paketa 9](#_Toc517776832)

[2.5 Asimetrična RSA enkripcija, osobine i primena 10](#_Toc517776833)

[2.5.1 Elementi enkripcije 10](#_Toc517776834)

[2.5.2 Osobine RSA enkripcije 11](#_Toc517776835)

[2.5.3 Algoritam RSA enkripcije 12](#_Toc517776836)

[3. Koncept rešenja 14](#_Toc517776837)

[4. Programsko rešenje 17](#_Toc517776838)

[4.1 Klijentska strana 17](#_Toc517776839)

[4.1.1 int main(void) 18](#_Toc517776840)

[4.1.2 void receiveFile(void) 18](#_Toc517776841)

[4.1.3 void decrypt(void) 19](#_Toc517776842)

[4.1.4 uint32\_t prime(uint\_32 pr) 19](#_Toc517776843)

[4.1.5 void ce(void) i uint32\_t cd(uint32\_t x) 19](#_Toc517776844)

[4.2 Serverska strana 19](#_Toc517776845)

[4.2.1 FUNC(void, RTE\_CTCDETHCOM\_APPL\_CODE) REthComInit(void) 20](#_Toc517776846)

[4.2.2 FUNC(void, RTE\_CTCDETHCOM\_APPL\_CODE) REthComCyclic(void) 20](#_Toc517776847)

[4.2.3 static void backgroundTask(void) 20](#_Toc517776848)

[5. Literatura 21](#_Toc517776849)

**Spisak slika**

Slika 2.1 *BroadR-Reach* interfejs 3

Slika 2.2 Neobložene uprede parice 4

Slika 2.3 Segmentacija 5

Slika 2.4 Izgled *TCP* segmenta 6

Slika 2.5 *TCP* uspostava veze 7

Slika 2.6 *TCP* prekid veze 8

Slika 2.7 Izgled *IPv6* paketa 9

Slika 2.8 Primer *IPv6* paketa sa uključenim zaglavljima 10

Slika 2.9 Prikaz *RSA* enkripcije i dekripcije 12

Slika 3.1 *Altera Cyclone* pete generacije 14

Slika 3.2 Izgled klijentskog terminala 15

**Spisak tabela**

Tabela 4.1 Funkcije na klijentskoj strani 17

Tabela 4.2 Funkcije na serverskoj strani 20

**Skraćenice**

**SoC** - *System on a Chip*, integisanokolo koje sadrži sve elektronske komponente nekog elektronskog sistema na jednom čipu

**TCP** - *Transmission Control Protocol*, mrežni protokol

**IPv6 -** *Internet Protocol Version 6,* šesta revizija Internet Protokola

**RSA -** *Rivest – Shamir – Adleman algorithm*, algoritam asimetrične enkripcije

**MIT -** *The Massachusetts Institute of Technology*, Tehnološki institut Masačusetsa

**RTOS** - *Real-time operating system*, operativni sistem koji radi u realnom vremenu

**ARM -** *Advanced RISC machine/Acorn RISC machine*, familija *RISC* čipova

**ACK** - *Acknowledgement*, paket koji potvrđuje neku tvrdnju

# Uvod

Da bi autonomni automobili preuzeli primat na putevima i postali deo svakodnevice, jedan od najvećih izazova sa inženjerske tačke gledišta svakako jeste prenos, skladištenje i manipulacija podacima. Navodno, devedeset minuta vožnje jednog autonomnog automobila predstavlja ekvivalet korišćenju milion pametnih telefona u rasponu od dvadeset i četiri časa po obimu obrađenih podataka. U savremenim automobilima može se nalaziti više desetina, sve češće i više stotina elektronskih sistema, koji međusobno komuniciraju preko automobilskih mreža (CAN, CAN HS, Ethernet, LIN itd.). Sa stanovništa bezbednosti od neželjenog pristupa, sigurnost automobilskih mreža nije na nivou kao u potrošačkoj industriji već mora biti na mnogo višem nivou. Ovde je potrebno obezbediti dodatnu zaštitu, kao i šifrovanje podataka, kako bi se onemogućila, ili makar znatno otežala manipulacija podacima koji se čuvaju u automobilu, kao što su dijagnostički podaci, rezultati očitavanja senzora, podaci o izvršenim popravkama i servisima, ukupna pređena kilometraža vozila, i slično.

U ovom radu biće opisano jedno rešenje programske podrške za bezbedan prenos podataka sa namenske ploče na računar, njihova enkripcija pre slanja i dekripcija po primanju, putem BroadR-Reach interfejsa i IPv6 protokola.

Rad je sačinjen od ----XY---- poglavlja. Prvo poglavlje sadrži kraći uvod u rad, motivaciju i opis samog rada.

U drugom poglavlju se nalaze teorijske osnove koje su neophodne za shvatanje rada – opis *zFAS* razvojne ploče, pojašnjenje *TCP* i *IPv6* protokola kao i nekoliko reči o asimetričnoj *RSA* enkripciji.

Treće poglavlje sadrži koncept rešenja.

Četvrto poglavlje daje detaljan opis programskog rešenja sa dubljim analizama napisanih funkcija i kritičnih sekcija koda.

U petom poglavlju su predstavljeni rezultati testnih slučajeva i njihovo tumačenje.

Šesto poglavlje sadrži kratak pregled onoga što je urađeno u ovom radu i ideje na koji način bi programska podrška mogla dalje da se razvija.

U sedmom poglavlju je dat spisak korišćene literature tokom izrade ovog rada.

# Teorijske osnove

## zFAS razvojna ploča

U nadolazećim generacijama automobila, centralni računar će polako ali sigurno preuzimati sve više i više raznih zadataka koji se tiču asistencije pri vožnji. *zFAS* znatno umanjuje broj elektronskih komponenti u vozilu jer u sebi ujedinjuje ono što su te komponente izvršavale. Najveći izazov je predstavljanje digitalnog modela okruženja koji zavisi od brzine i načina obrade velike količine

## BroadR-Reach interfejs



Slika 2.1 *BroadR-Reach* interfejs

*BroadR-Reach* tehnologija predstavlja standard fizičkog sloja koji je osmišljen za umrežavanje komponenti u automobilskoj industriji. *BroadR-Reach* pruža mogućnost istovremenog pristupa informacijama preko neobloženih upredenih parica.



Slika 2.2 Neobložene uprede parice

Jedna od najvećih prednosti koje proizvođačima pruža ova tehnologija je umnogome smanjena cena i težina kablova u vozilu.

Korišćenje *BroadR-Reach* tehnologije omogućava prelazak sa više zatvorenih mreža u automobilu na jednu otvorenu mrežu koja je bazirana na Eternet (eng. *Ethernet*) protokolu. Ovo dozvoljava proizvođačima da inkorporiraju veliki broj elektronskih sistema i uređaja u vozilo koji se tiču bezbednosti, udobnosti i razonode. Brzina prenosa podataka može da dostigne i 100Mbit/s, što prevazilazi standardnu Eternet konekciju. [1]

## TCP protokol

*TCP* predstavlja protokol transportnog nivoa. Ovaj protokol omogućava istovremeno dvosmernu pouzdanu komunikaciju između klijenta i servera. Komunikacija je realizovana u vidu konekcije koja se uspostavlja pomoću metode rukovanja (eng. *Handshaking*). Iz tog razloga ne podržava multicast.



Slika 2.3 Segmentacija

*TCP* koristi **segment** za jedinicu prenosa (jedinice podataka transportnog sloja). Proces podele originalne poruke aplikativnog nivoa na segmente naziva se **segmentacija**. *TCP* omogućava praćenje poslatih paketa, kontrolu toka komunikacije, kao i redosleda pristiglih segmenata.

*TCP* definiše uslugu pouzdane isporuke toka (engl. *stream*) korisničkih podataka. Osobine *TCP*-a su:

• Obavlja kontrolu toka podataka, *TCP* obezbeđuje komunikaciju sistema različitih brzina.

• Osnovna jedinica prenosa *TCP*-a je segment podataka. Segmenti se koriste za prenos upravljačke informacije (npr poruke za uspostavu i raskid veze), ili za prenos podataka.

• Format segmenta je izabran tako da je moguće potvrđivanje podataka iz jednog smera, njihovim uključivanjem u zaglavlje segmenata koji se šalju u drugom smeru.

• Kontrola toka je realizovana tako što prijemnik oglašava količinu podataka koju je spreman da primi.

• *TCP* takođe podržava poruke van opsega (engl. “*out of band*”), koje služe za slanje urgentnih podataka i za forsiranje isporuke korišćenjem (engl. “*push*”) podataka.



Slika 2.4 Izgled *TCP* segmenta

Na slici 2.2 je dat prikaz *TCP* segmenta, koji se sastoji iz zaglavlja i dela u kome se nalaze podaci. Zaglavlje se sastoji iz:

• Source port - izvorišni port (port pošiljaoca). Port ima ulogu u identifikaciji aplikacije.

• Destination port - odredišni port (port primaoca).

• Sequence number - broj prvog bajta segmenta u okviru toka podataka. Broj sekvence omogućava praćenje toka podataka. Inicijalni broj sekvence se bira nasumično, kako bi se otklonila mogućnost interferencije između različitih konekcija.

o syn = 1 - Inicijalni broj sekvence.

o syn = 0 - Akumulirani broj sekvence.

• Acknowledgement number - broj sekvence narednog segmenta koji se očekuje. Segmenti koji pristignu van redosleda, u zavisnosti od implementacije, mogu se odbaciti ili čuvati.

• Header Length - dužina zaglavlja. 3

• Reserved - rezervisano za buduću upotrebu.

• Control bits - ukazuje na funkciju i namenu segmenta.

o ack - ukazuje na validnost vrednosti potvrde (acknowledgment-a).

o syn, rst i fin - omogućavaju uspostavljanje i prekid konekcije.

o psh - ukazuje prijemnoj strani da se momentalno pošalju podaci višem sloju.

o urg - ukazuje na postojanje urgentnih podataka u segmentu.

• Window - maksimalan broj bajtova koje je moguće poslati, a da prethodno nije potvrđen njihov prijem.

• Checksum - koristi se za za proveru da li se desila greška poruke nad zaglavljem i podacima prilikom prenosa.

• Urgent - lokacija poslednjeg bajta koji je markiran kao urgentan.

• Options - dodatne opcije.

• Application data - podaci viših slojeva.

### Uspostavljanje veze

Komunikacija klijenta i servera preko *TCP* protokola zahteva uspostavu veze koja se ostvaruje tako što se između predajne i prijemne strane iz tri puta razmene poruke sa podešenih odgovarajućim kontrolnim bitima (*TCP* *three-way handshake*):

• Predajna strana A šalje poruku sa podešenim kontrolnim bitom SYN = 1, ostali su podešeni na 0, pri čemu nasumično odabere redni broj segmenta (SEQa).

• Prijemna strana B odgovara porukom sa kontrolnim bitima SYN i ACK = 1, takođe nasumično bira broj segmenta (SEQb), a za ACK uzima broj ACK = (SEQa) + 1. Na ovaj način je uspostavljena veza na liniji od predajne ka prijemnoj strain.

• Slanjem poruke sa podešenim kontrolnim bitom SYN = 1 od prijemne strane, ona zahteva da predajna strana potvrdi uspostavljanje veze od prijemne ka predajnoj strani. Predajna strana to čini slanjem poruke sa podešenim kontrolnim bitom ACK = 1 i 4 uzima vrednost ACK broja ACK = (SEQb) + 1. Na ovaj način je uspostavljena konekcija između klijenta i servera.



Slika 2.5 *TCP* uspostava veze

### Prekid veze

Prekid veze se ostvaruje u četiri koraka sledećim redosledom:

1. Klijent inicira prekid klijent-server konekcije,

2. Server potvrđuje zahtev za prekid klijent-server konekcije,

3. Server šalje zahtev za prekid server-klijent konekcije,

4. Klijent odgovara na zahtev za prekid server-klijent konekcije. [2]



Slika 2.6 *TCP* prekid veze

## IPv6 protokol

Internet protokol verzija 6 (*IPv6*) je protokol sloja mreže, naslednik internet protokola vrezije 4 (*IPv4*), osmišljen od strane IETF-a (engl. *Internet Engineering Task Force*).

### IPv6 adresa

*IPv6* je novi (ali ne još široko korišćen) standardni internet protokol, gde su adrese 128 bita široke, što bi, čak i sa velikim dodelama netblokova, trebalo da zadovolji blisku budućnost. Teoretski, postojalo bi tačno 2128, ili 3.403×1038 unikatnih adresa domaćinskih interfejsa. Kada bi zemlja bila sačinjena kompletno od zrna peska od 1cm³, onda bi mogla da se dodeli jedinstvena adresa svakom zrnu u 300 miliona planeta veličine zemlje. Ovaj veliki prostor za adrese teško da će ikada biti potpuno popunjen. Adresa verzije 6 se piše kao osam četvorocifrenih heksadecimalnih brojeva (8 puta po 16 bitova) odvojenih dvotačkama. Jedan niz nula po adresi može da se izostavi, pa je 1080::800:0:417A isto što i 1080:0:0:0:0:800:0:417A. Globalne adrese koje se šalju ka jednom odredištu se sastoje iz dva dela: 64-bitni deo za rutiranje i 64-bitni identifikator domaćina.

### Struktura IPv6 paketa

Jedan *IPv6* paket ima sledeću opštu formu:



Slika 2.7 Izgled *IPv6* paketa

Jedino potrebno zaglavlje se odnosi na *IPv6* zaglavlje. Ovo je fiksna veličina sa dužinom od 40 bajtova, u poređenju sa 20 bajtova pomoćnog dela *IPv4*. Sledeća produžena zaglavlja su definisana kao:

• Zaglavlje *Hop-by-Hop* opcija: Definišu specijalne opcije koje zahtevaju hop-by-hop procesiranje.

• Zaglavlje rutiranja: Obezbeđuje prošireno rutiranje, slično izvorišnom rutiranju u *IPv4*.

• Zaglavlje fragmenata: Sadrži informacije o fragmentaciji i ponovnom sklapanju.

• Zaglavlje autentičnosti: Ovezbeđuje integritet i autentičnost svakog paketa.

• Zaglavlje enkapsulacije sigurnosti podatka: Obezbeđuje privatnost.

• Zaglavlje odredišnih opcija: Sadrži fakultativne informacije koje će ispitati odredišni čvor.



Slika 2.8 Primer *IPv6* paketa sa uključenim zaglavljima

*IPv6* zaglavlje i svako produženo zaglavlje sadrže polje *sledeće zaglavlje*. Ovo polje identifikuje tip sledećeg zaglavlja. Ako je sledeće zaglavlje jedno od produženih zaglavlja, onda ovo polje sadrži identifikator tipa tog zaglavlja. U protivnom, ovo polje sadrži identifikator protokola za protokol višeg sloja koji koristi *IPv6* (obično protokol transportnog sloja), koriste se iste veličine kao i polja *IPv4* protokola. Na slici, protokol višeg sloja je *TCP*, tako da se podaci viših slojeva koje nosi *IPv6* paket sadrže od zaglavlja *TCP*-a koje prati blok aplikacionih podataka. [3]

## Asimetrična RSA enkripcija, osobine i primena

Enkripcija (engl. *encryption*) ili šifrovanje je proces u kriptografiji kojim se vrši izmena podataka tako da se podaci, ili poruke, učine nečitljivim za osobe koje ne poseduju određeno znanje (ključ). Na taj način se dobija šifrovana informacija. Da bi ovi podaci postali razumljivi i upotrebljivi, potrebno je da se dekodiraju. Dekodiranje se vrši procesom suprotnim od enkripcije koji se naziva dekripcija (engl. *decryption*).

### Elementi enkripcije

Svi sistemi enkripcije imaju u svojoj osnovi sledeće zajedničke elemente:

• Algoritam: Funkcija, obično sa jakom matematičkom osnovom, koja obavlja zadatak enkripcije podataka;

• Ključevi: Koriste se zajedno sa algoritmima enkripcije i određuju način na koji su podaci šifrovani;

• Dužinaključa: Enkripcioni ključevi imaju određenu dužinu u zavisnosti od toga koji enkripcioni sistemi se koriste. Dužina se meri brojem bitova ,a što su duži ključevi, teži su za oštećenje sistema enkripcije;

• Otvoren tekst (engl. *Plaintext*): Informacije koje želimo da šifrujemo;

• Šifrovan tekst (engl. *Ciphertext*): Informacije nakon šifrovanja. [4]

### Osobine RSA enkripcije

Postoje dve osnovne vrste enkripcije, simetrična i asimetrična enkripcija. Kod simetrične enkripcije se i za šifrovanje i za dešifrovanje koristi ista šifra (ključ), i samim tim što obe strane koriste isti ključ i za enkripciju i za dekripciju proboj sistema je znatno olakšan. Kod asimetrične enkripcije postoji poseban ključ samo za šifrovanje i drugi koji služi samo za dešifrovanje. Ova dva ključa nazivaju se još *javni* i *tajni* ključ. Tajni ključ se dodeljuje onda kada se vrši enkripcija i na osnovu njega se generiše javni ključ, koji koristi strana koja treba da pročita podatke. *RSA* (akronim od Rivest – Shamir – Adleman) predstavlja jedan od najčešće korištenih algoritama za asimetričnu enkripciju, razvijen od strane trojice stručnjaka sa *MIT*-a 1977. godine i u sledećem poglavlju će detaljno biti opisan način na koji funkcioniše.



Slika 2.9 Prikaz *RSA* enkripcije i dekripcije

### Algoritam RSA enkripcije

Kao što je već rečeno, kod *RSA* enkripcije se koriste dva ključa, javni i tajni. Javni ključ je dostupan svima i on se koristi za enkriptovanje poruka. Poruke koje su enkriptovane javnim ključem mogu se dekriptovati jedino tajnim ključem. Ovaj par ključeva se generiše na sledeći način:

1. Biraju se dva nasumična velika prosta broja i .
2. Računa se .
3. Računa se vrednost Ojlerove fi funkcije .
4. Bira se ceo broj takav da zadovoljava uslov i i ne smeju imati zajednički faktor osim broja **.** Par i predstavljaju javni ključ.
5. Na kraju se računa tako da zadovoljava uslov za neki ceo broj . Par i se čuvaju kao tajni ključ.

Formula koja se koristi za enkripciju poruke je , a da bi se iz enkriptovane poruke došlo do originalne poruke , neophodno je uraditi sledeće . [5]

Kod *RSA* enkripcije se postavlja pitanje zašto se koriste baš prosti brojevi? Zašto ne bilo koja dva velika broja? Odgovor leži u tome da je današnjim računarima veoma lako pomnožiti dva velika prosta broja i naći broj , ali ne postoji način na koji bi se efikasno odradila inverzna operacija kako bi se došlo do faktora i jer to direktno proizilazi iz osnovne teoreme algebre. Ona kaže da se svaki složeni broj, koji je veći od 1, može napisati na tačno jedan način kao proizvod prostih brojeva. Sa malim brojevima je lako, npr. ili , ali kada je dat problem koji izgleda kao , onda je izuzetno teško zaključiti da je i da je . Do današnjeg dana nije osmišljen algoritam koji ovo omogućava i ne preostaje ništa drugo no pokušati sa *brute-force* pretragom (sistematično nabrajanje svih mogućih kandidata za rešavanje problema i proveravanje da li svaki kandidat zadovoljava problem). Ako su brojevi i dovoljno veliki, srednje vreme *brute-force* pretrage može da iznosi nekoliko desetina, pa čak i nekoliko stotina godina, što svakako ide u prilog robusnosti i sigurnosti *RSA* algoritma.

# Koncept rešenja

U ovom poglavlju biće pojašnjen koncept rešenja. Rešenje prikazano u ovom radu se sastoji iz dva dela – prvi deo rešenja predstavlja klijentsku stranu i ona se izvršava na računaru, dok je drugi deo rešenja serverska strana koja se izvršava na namenskoj platformi. Obe strane su realizovane u programskom jeziku *C*. Radno okruženje koje je korišćeno prilikom izrade klijentske strane je *Visual Studio 2017 Professional*. Samo okruženje poseduje sve neophodne biblioteke i podešavanja kako bi se uspostavila komunikacija sa namenskom platformom. Operativni sistem koji se nalazi na računaru je *Windows 10*. Na namenskoj platformi se nalazi *RTOS VxWorks Wind River* operativni sistem sa svim neophodnim bibliotekama za razmenu podataka sa klijentskom stranom. Serverska strana je napisana u *Notepad++* alatu.

*SoC* koji se nalazi na namenskoj platformi i koji omogućava realizaciju rešenja sa serverske strane je *Altera Cyclone V Soc* sa integrisanim *ARM* procesorom.



Slika 3.1 *Altera Cyclone* pete generacije

Ideja zadatka je da se podaci sa namenske platforme izvuku bezbedno, i da samo vlasnici tajnog *RSA* ključa mogu da ih pročitaju u njihovom originalnom obliku. Kada bi neko sa malicioznim tendencijama pokušao da izvuče podatke sa platforme oni bi bili nečitljivi i njihovo vraćanje u originalni oblik bi bilo praktično nemoguće kao što je i objašnjeno u poglavnju *RSA* enkripcije.

Na samom početku rešenja neophodno je izabrati tajni i javni ključ. Oni se biraju na slučajan način na klijentskoj strani što svakako uvećava robusnost.

Komunikacija između računara i namenske platforme započinje tako što se najpre pošalje zahtev sa klijentske strane za početak komunikacije. To je realizovano tako što obe strane unapred znaju odgovarajuću reč (npr. *Start*) i proverom primljene i unapred poznate reči server lako utvrdi ko pokušava da pristupi namenskoj platformi. Ako je primljena reč potvrđena kao validna server šalje klijentu *ACK* paket i rešenje je spremno za bezbedan prenos informacija preko mreže.

Sledeći korak je slanje javnog ključa serveru kako bi se mogla izvršiti enkripcija podataka, dok tajni ključ ostaje u posedu klijenta radi dekripcije. Nakon toga sledi prikupljanje podataka iz željenog direktorijuma na namenskoj platformi, njihova enkripcija i slanje ka klijentu. Klijentska strana vrši dekripciju svakog paketa odmah nakon primanja i upisuje dekriptovani paket u datoteku. Rešenje je realizovano dinamički, što znači da broj datoteka koje se šalju ne mora biti unapred poznat već se vrši njihovo prebrojavanje nakon otvaranja željenog direktorijuma. Jedino što unapred mora biti poznato jeste ime željenog direktorijuma i putanja do njega.



Slika 3.2 Izgled klijentskog terminala

Na slici 3.2 su prikazani ispisi u terminalu koji demonstriraju sve što je prethodno objašnjeno, uz dodatak da klijentska strana vodi računa o tome na koji način se prima svaka datoteka, koliko celih paketa treba da dobije i koje je veličine poslednji paket o čemu će biti reči u narednom poglavlju.

# Programsko rešenje

Kao što je naglašeno ranije, programsko rešenje je podeljeno u dva modula – na serversku stranu koja se nalazi na namenskoj platformi i na klijentsku stranu koja se nalazi na računaru. U ovom poglavlju biće opisana oba modula detaljno, ali sa akcentom na funkcionisanje jednog modula kao celine, nezavisno od drugog (iako svakako zavise jedan od drugog, međutim ovaj pristup omogućuje dublji uvid u ideju i realizaciju iste zarad uspešnijeg rešenja problema).

## Klijentska strana

|  |  |
| --- | --- |
| *int main(void)* | *Main* funkcija |
| *void receiveFile(void)* | Primanje datoteke |
| *void decrypt(void)* | Dekripcija |
| *uint32\_t prime(uint32\_t pr)* | Provera da li je broj prost |
| *void ce(void)* | Funkcija u kojoj se računaju tajni i javni ključevi |
| *uint32\_t cd(uint32\_t x)* | Pomoćna funkcija za računanje tajnih ključeva |

Tabela 4.1 Funkcije na klijentskoj strani

U narednim glavama biće detaljno pojašnjena svaka napisana funkcija na klijentskoj strani rešenja.

### int main(void)

Izvršavanje klijentske strane počinje u *int main(void)* funkciji u kojoj se na početku na slučajan način, pomoću funkcija *void* *srand(unsigned int seed)* i *int* *rand(void)* iz biblioteke *time.h*, biraju dva prosta broja iz niza prostih brojeva. Nakon toga se pozivaju funkcije *void ce(void)* i *uint32\_t cd(uint\_t x)* zahvaljujući kojima se računaju tajni i javni ključevi pomoću *RSA* algoritma (poglavlje 2.5.3). Kada su ključevi izračunati sledeći korak je pravljenje *socket-a* za komunikaciju i popunjavanje strukture *sockaddr\_in6* odgovarajućim parametrima kako bi se uspešno uspostavila veza sa namenskom platformom. Sledi poziv funkcije *int connect(int socket, const struct sockaddr\* address, socklen\_t address\_len)* pomoću koje se klijentska strana povezuje sa serverskom stranom na platformi. Nakon uspešnog povezivanja sledi slanje inicijalne poruke, odnosno unapred dogovorene reči kako bi se verifikovala konekcija i kako bi usledilo slanje enkriptovanih podataka. Kada klijentska strana primi odobrenje od servera, sledeći korak je slanje javnog ključa kako bi serverska strana mogla da enkriptuje željene datoteke. Zatim se na klijentskoj strani bira iz kog direktorijuma namenska platforma treba da pročita podatke i dobija se odgovor od platforme koliko će datoteka biti poslato. Poslednje što treba uraditi u *int main(void)* funkciji je proći kroz *for* petlju onoliko puta koliko je platforma rekla da ima datoteka u željenom direktorijumu i za svaki prolaz pozvati funkciju *void receiveFile(void)*. Nakon izlaska iz petlje treba još zatvoriti *socket* i izvršavanje funckije je gotovo.

### void receiveFile(void)

U ovoj funkciji se odvija primanje jedne datoteke. Server najpre šalje ime i ekstenziju datoteke, pa zatim i veličinu. Na klijentskoj strani se otvara datoteka sa identičnim imenom i ekstenzijom pomoću funkcije *FILE\* fopen(const char\* filename, const char\* mode)*. Nakon toga sledi računanje broja paketa koji će biti veličine *512* (kao što je definisano u pretprocesorskoj direktivi) i računa se koliki će biti poslednji paket (najčešće je manji od *512*). Sledi *while* petlja u kojoj se vrši primanje paketa pomoću funkcije *ssize\_t recv (int s, void\* buf, size\_t len, int flags)*. Nakon primljenog paketa poziva se funkcija *void decrypt(void)* u kojoj se primljeni paket dekriptuje i odmah nakon toga upisuje u otvorenu datoteku pomоću funkcije *size\_t fwrite(const void\* ptr, size\_t size, size\_t nmemb, FILE\* stream)*. Sledi čišćenje svih bafera, provera da li je primljeni paket poslednji i ako jeste izlazi se iz *while* petlje. Nakon toga se zatvara otvorena datoteka i izlazi se iz *void receiveFile(void)* funkcije.

### void decrypt(void)

U funkciji *void decrypt(void)* vrši se dekripcija pristiglog paketa na osnovu algoritma koji je izložen u poglavlju *2.5.3*. Ova funkcija će biti pozvana onoliko puta koliko paketa stigne na klijentsku stranu.

### uint32\_t prime(uint\_32 pr)

Zadatak funkcije *uint32\_t prime(uint32\_t pr)* je da proveri da li je broj koji je prosleđen kao argument funkcije prost ili ne. Iako jednostavna po svojoj kompleksnosti, ima ključnu ulogu, jer kao što je već rečeno prosti brojevi su od suštinskog značaja *RSA* algoritma.

### void ce(void) i uint32\_t cd(uint32\_t x)

Uloga ovih funkcija jeste generisanje parova tajnih i javnih ključeva. U funkciji *void ce(void)* se bira vrednost ključa za enkriptivanje podataka, a pomoću funkcije *uint32\_t cd(uint32\_t x)* se bira vrednost tajnog ključa, odnosno ključa za dekripciju. Brojevi i koji se koriste za enkripciju i dekripciju podataka moraju biti prosti, pa se u rešenju za gornju granicu provere postavlja prvih *100* prostih brojeva (ovo ne mora da bude slučaj, tako je izabrano radi preglednosti rešenja).

## Serverska strana

|  |  |
| --- | --- |
| *FUNC(void, RTE\_CTCDETHCOM\_APPL\_CODE) REthComInit(void)* | Funkcija koja će se pozvati samo jednom pri pokretanju serverske strane |
| *FUNC(void, RTE\_CTCDETHCOM\_APPL\_CODE) REthComCyclic(void)* | Funkcija koja će se pozivati u pozadini nakon određenog vremena |
| *static void backgroundTask(void)* | *Callback* funkcija u kojoj se nalazi veći deo logike serverske strane rešenja |
| *static void receivePublicKeys(void)* | Primanje ključeva za enkripciju |
| *static void sendFile(const char fs\_name[])* | Slanje datoteke |
| *static int32\_t numOfFiles(void)* | Broj datoteka u direktorijumu |
| *static void encrypt(void)* | Enkripcija |

Tabela 4.2 Funkcije na serverskoj strani

Serverska strana nema *main* funkciju, pa će taj deo rešenja izgledati malo drugačije. Sledi detaljan opis svake funkcije ponaosob na serverskoj strani.

### FUNC(void, RTE\_CTCDETHCOM\_APPL\_CODE) REthComInit(void)

Funkcija koja se poziva samo jednom kada se namenska platforma upali. U njoj se nalazi poziv *callback* funkcije *static void backgroundTask()*. U funkciji je napravljen red poruka iz biblioteke *msgQLib.h* pomoću funkcije *MSG\_Q\_ID msgQCreate (int maxMsgs, int maxMsgLength, int options)* koji služi za kontrolu promene stanja. Tu se takođe nalazi funkcija pomoću koje se kreira *task* koji će pozvati *callback* funkciju.

### FUNC(void, RTE\_CTCDETHCOM\_APPL\_CODE) REthComCyclic(void)

Ova funkcija se poziva svaki put kada istekne predefinisano vreme (ciklična funkcija). U njoj će se poslati poruka koja će ući u prethodno napravljen red poruka, i na osnovu njene vrednosti se prelazi u odgovarajuće stanje, ili ako je vrednost poruke nepromenjena, stanje se ne menja.

### static void backgroundTask(void)

U ovoj funkciji je realizovana glavna logika serverske strane. Na početku je neophodno osposobiti ovu stranu rešenja za komunikaciju sa računarom. Kao i na klijentskoj strani, prva stvar koju treba uraditi jeste napraviti *socket*. Nakon toga se popunjava *sockaddr\_in6* struktura, poziva se *int bind(int socket, const struct sockaddr\* address, socklen\_t address\_len)* funkcija koja povezuje napravljeni *socket* sa željenom adresom. Sledeći je poziv *int listen(int socket, int backlog)* funkcije koja čeka konekcije i na kraju *int accept(int socket, struct sockaddr\* restrict address, sokclen\_t\* restrict address\_len)* funkcije koja prihvata komunikaciju sa uređajem koji pokušava da pristupi namenskoj platformi. Nakon što je komunikacija uspešno uspostavljena sledi utvrđivanje identiteta onoga ko pokušava da pristupi namenskoj platformi. Kao što je već rečeno, klijentska strana će poslati unapred dogovorenu reč kojom će se utvrditi da je reč o pouzdanom uređaju koji pokušava da izvuče resurse sa platforme. Ukoliko su reči identične komunikacija se nastavlja, u suprotnom dolazi do prekida. Sledi primanje javnog ključa za enkripciju, prebrojavanje datoteka u direktorijumu i slanje prebrojanog stanja. Zatim se poziva funkcija *static void sendFile(const char fs\_name[])* za svaku prebrojanu datoteku u kojoj se ista i šalje. Poslednje što treba uraditi jeste poslati poruku u red poruka kako se više ne bi ulazilo u ovaj deo koda.

### static void receivePublicKeys(void)

Ova funkcija služi za prijem javnog ključa koji se koristi za enkripciju željenih podataka. Ključ se prima iz dva dela jer kao što je već rečeno, par i predstavljaju javni ključ.

### static void sendFile(const char fs\_name[])

U *static void sendFile(const char fs\_name[])* funkciji nalazi se glavna logika za enkripciju i slanje datoteka. Najpre se klijentu šalje ime i veličina imena datoteke koja će biti poslata. Zatim se na serverskoj strani čita datoteka radi utvrđivanja njene veličine, pa se i veličina šalje klijentu. Nakon toga se datoteka raspoređuje u pakete, vrši se enkripcija svakog paketa i tako enkriptovani paketi se šalju klijentu. Veličina paketa je *512*, ali u suštini može biti bilo koja vrednost.

### static int32\_t numOfFiles()

U ovoj funkciji vrši se prebrojavanje datoteka u željenom direktorijumu. Povratna vrednost funkcije je broj datoteka koje će biti poslate klijentu.

### static void encrypt(void)

U funkciji *static void encrypt(void)* će biti izvršena enkripcija paketa po algoritmu iz poglavlja *2.5.3*.

# Testiranje i verifikacija

U okviru ovog poglavlja biće opisano testiranje i verifikacija rešenja. S obzirom da je akcenat u zadatku bio na enkripciji i dekripciji podataka, može se reći da je provera ispravnosti rešenja poprilično jednostavna, odnosno potrebno je utvrditi da li se enkriptovani podaci mogu vratiti u originalni oblik bez poznavanja tajnog ključa. Odgovor je da ne mogu. Jedini način je *brute-force* pretraga tajnog ključa, ali kao što je već rečeno u poglavlju 2.5.3 *brute-force* pretraga može da potraje i po nekoliko desetina godina.

Pri testiranju rešenja je isceniran slučaj presretanja komunikacije, gde ona strana koja presreće može samo da vidi enkriptovane podatke.

OVDE NPR SLIKA ORIGINALNOG TEKSTA

SAD OVDE STAVIM NPR SLIKU .TXT ENKRIPTOVANOG TEKSTA

Kao što se vidi na slici 5.2, podaci koji su izvučeni sa namenske platforme bez prethodnog poznavanja tajnog ključa su bezvredni. Slika 5.3 pokazuje da razlika između enkriptovanog i originalnog teksta nije konzistentna, odnosno da je razlika između enkriptovanih i originalnih bajta na različitim pozicijama potpuno stohastička (što na primer nije slučaj kod algoritma enkripcije koji se zove *Cezarova šifra*, svaki bajt enkriptovanog podatka pomera levo ili desno za unapred definisan broj mesta). Dakle kao što se i na datom primeru vidi, *RSA* algoritam enkripcije je praktično neprobojan.

Rešenje je, pored tekstualnih datoteka testirano i na slikama različitih formata. Kao što je i očekivano, enkriptovana slika ne može biti reprodukovana, dok se dekriptovana reprodujukuje bez ikakvih problema.

# Zaključak

# Literatura

1. *BroadR-Reach*, <https://en.wikipedia.org/wiki/BroadR-Reach>, jun 2018
2. *TCP*, <http://www.rt-rk.uns.ac.rs/predmeti/e2/orm-1-osnovi-računarskih-mreža-1>, jun 2018.
3. *IPv6*,<https://sr.wikipedia.org/sr-el/IPv6>, jun 2018.
4. *RSA*, <https://sr.wikipedia.org/wiki/Enkripcija>, jun 2018
5. *RSA*, <https://simple.wikipedia.org/wiki/RSA_algorithm>, jun 2018