УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА

УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА
НОВИ САД
Департман за рачунарство и аутоматику
Одсек за рачунарску технику и рачунарске комуникације

ИСПИТНИ РАД

Кандидат: **Алекса Арсић**
Број индекса: **PA119/2015**

Предмет: **Системска програмска подршка у реалном времену II**
Тема рада: **Енкрипција и декрипција података**

Ментор рада: **проф. др Мирослав Поповић**

Нови Сад, јануар, 2018.

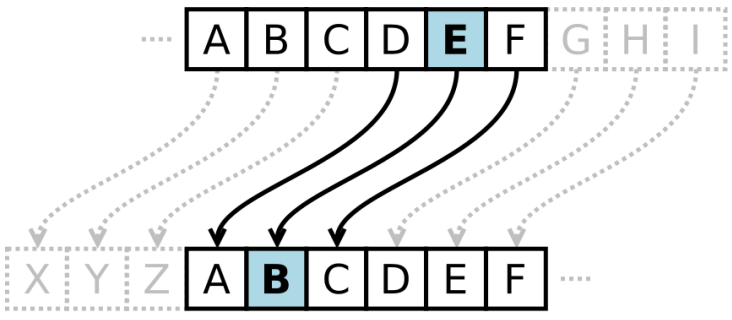
Sadržaj

Spisak slika

1. Uvod

Šifrovanje podataka kao problem javlja se od antičkih vremena usled potrebe razmene poverljivih informacija, nebitno radilo se to o vojnim, ličnim, ljubavnim ili informacijama nekakvog drugog karaktera. Na taj način informacije postaju nečitljive za osobe koje ne poseduju određeni ključ (znanje) potrebno za uspešno dešifrovanje podataka. S vremenom usled sve čećše potrebe za očuvanjem poverljivih informacija razvija se posebna nauka koja se bavi metodama očuvanja tajnosti informacija, kriptografija.

Jedna od najpoznatijih, najprostijih i najrasprostranjenijih metoda šifrovanja u kriptografiji je Cezarova šifra (eng. Caesar cipher). Ovaj metod koristi tip šifre zamenjivanja u kome se svako slovo otvorenog teksta menja određenim slovom abecede, pomerenim za određeni broj mesta. Na primer ako je pomak 3, A se zamenjuje slovom G, B sa D, itd. Ovaj metod dobio je ime po Juliju Cezaru, koji ga je koristio za razmenu poruka sa svojim generalima. Ona se takodje koristi kao korak u kreiranju složenijih načina šifrovanja. Ona se lako rabija i u praksi ne pruža nikakvu sigurnost u komunikaciji.



Slika 4 Cezarova šifra

1.1 Elementi enkripcije

- Algoritam: funkcija, obično sa jakom matematičkom osnovom, koja obavlja zadatak enkripcije podataka
- Ključevi: koriste se zajedno sa algoritmima enkripcije i određuju načina na koji su podaci šifrovani
- Dužina ključa: Enkripcioni ključevi imaju određenu dužinu. Dužina se meri brojem bitova, a što su ključevi duži, teži su za probijanje sistema enkripcije
- Otvoren tekst (eng. Plaintext): informacije koje želimo da šifrujemo
- Šifrovan tekst (eng. Ciphertext): informacije nakon šifrovanja

2. Zadatak

Potrebno je realizovati Linux rukovalac za Raspberry Pi 2 uređaj koji treba da omogući šifrovanje/dešifrovanje (eng. Encryption Decryption) podataka. Takođe je potrebno realizovati testni program za verifikaciju funkcionalnosti razvijenog rukovaoca. Rukovalac treba da omoći sledeću funkcionalnost:

- Postavljanje početne vrednosti (eng. Seed) generator slučajnih brojeva koji je dostupan u rukovaocu na osnovu vrednosti unseen kao parameter pri pokretanju modula (tajnog ključa) I pseudoslučajne vrednosti dobijene na osnovu trenutnog vremena u slučaju šifrovanja, odnosno unseen vrednosti u slučaju dešifrovanja (javni ključ). Generator slučajnih brojeva se koristi za generisanje elemenata pseudoslučajnog niza koji se koriste u procesu šiforvanja/dešifrovanja. Isti par (javni ključ, tajni ključ) koji se koristi prilikom šifrovanja otvorenog teksta se mora koristiti I prilikom šifrovanja zatvorenog teksta, da bi dešifrovanje uspelo.
- Stvaranje novog elementa iz generator slučajnih brojeva. Veličina elementa je 8 bita.
- Postavljanje podataka za šifrovanje/dešifrovanje.
- Iščitavanje šifrovanih/dešifrovanih podataka. Za šifrovanje/dešifrtovanje podataka koristiti logičku operaciju XOR nad parom (podatak, pseudo-slučajni broj)

U svrhu provere funkcionalnosti omogućiti I iščitavanje vrednosti pseudo slučajnih brojeva.

3. Analiza problema

3.1 Linux i kernel moduli

Da bi se uspešno realizovao kernel modul potrebno je pre svega poznavati operativni sistem Linux, način na koji on funkcioniše, kako upravlja modulima (drajverima), kako se oni ubacuju u njegov kernel i način funkcionisanja kernel modula.

Ako gledamo u paraleli C program I kernel modul možemo uočiti nekoliko razlika. C program mora da ima funkciju *main()* koja označava njegov početak i kraj, dok kernel moduli imaju dve funkcije, funkciju *module_init()* i funkciju *module_exit()* te one predstavljaju, respektivno, njegov početak i njegov kraj.

Funkcijom *module_init()* uređaj se registruje u operativni sistem te mu se ovde dodeljuje Major broj, pomoću kojeg operativni sistem razlikuje uređaje. Takođe se zauzimaju resursi potrebni za izvođenje i rad modula, dok se funkcijom *module_exit()* uređaj deregistruje iz kernela i osloboađaju se prethodno alocirani memorijski prostori, kao i Major broj pod kojim je on registrovan. Od nama bitnih funkcija važno je još pomenuti *read* i *write* funkcije.

Pošto sada radimo u kernel space-u potrebno je imati funkciju koja omogućava prenos informacija između uređaja i krajnjeg korisnika, a funkcija *read* nam upravo to mogućava. Ona pozivom funkcije *copy_to_user()* prenosi podatke iz kernel space-a u user space. Kada korisnička aplikacija želi da dobavi potrebne informacije od drajvera, ona će, takođe, pozivom funkcije *read* dobaviti potreban sadržaj. Kao što nam je od velike važnosti da dobavljamo podatke iz kernel space-a, od iste važnosti nam je važnosti predaja podataka drajveru radi dalje obrade. To je moguće postići funkcijom *write*. Pozivom ove funkcije korisnička aplikacije će proslediti podatke drajveru iz user space-a u kernel space, a drajver će ih iščitati i spremiti u predodređen memorijski prostor.

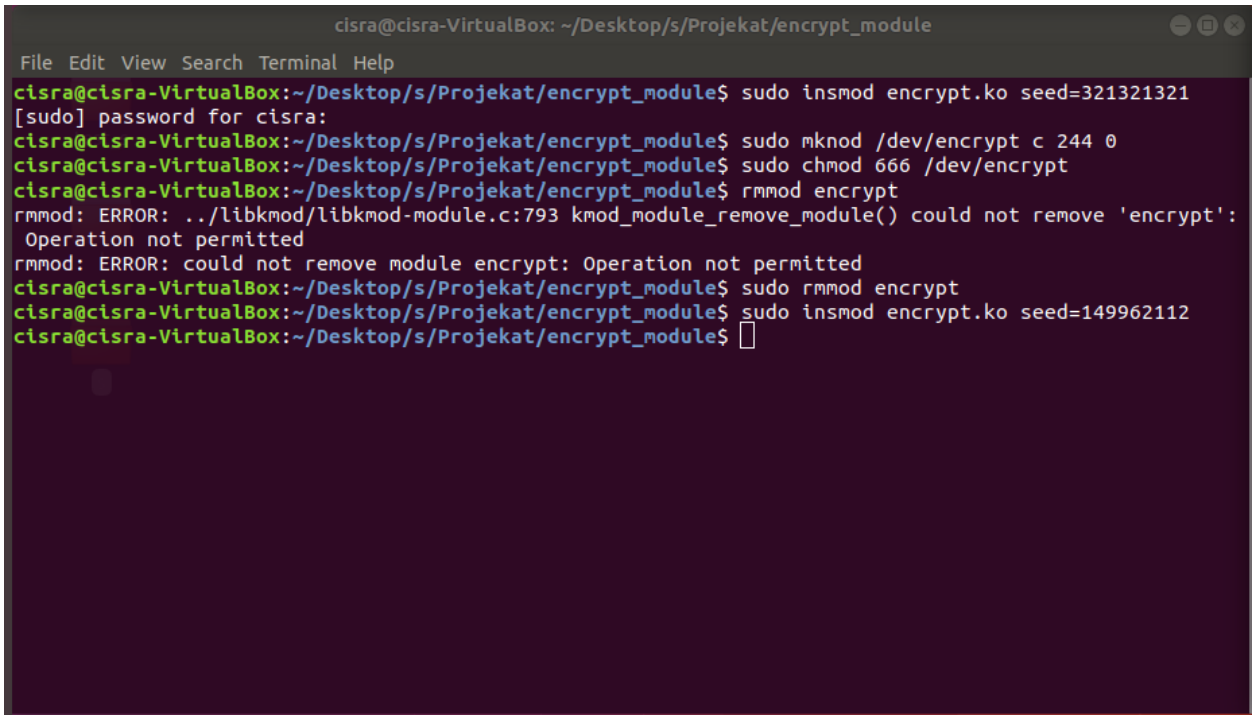
Drajver je na raspolaganju aplikaciji dok god je učitan u kernel operativnog sistema, ali dok god nema zahteva za nekakvu obradu on ne radi ništa. Sa stanovišta kernel modula, u implementaciji ovog primera i njegovom rešenju, ako korisnička aplikacija pozove samo funkciju *write* drajver će da učita podatke iz user space-a, ali ništa više se neće desiti. Tek kada korisnička aplikacija zatraži podatke od drajvera, putem funkcije *read*, on će da ih obradi, smesti u buffer i pošalje korisniku.

Treba napomenuti da zaglavlja koja uključujemo u C jeziku više ne važe i ne možemo da ih uključujemo kada pišemo kernel modul. Iako je velika većina C biblioteka implementirana za

programiranje kernel modula, neke ipak nisu te zbog toga gubimo mogućnost upotrebe *srand()* i *rand()* funkcija koje će nam biti potrebne za rešavanje datog problema.

Komande za rukovanje drajverom (potrebno pokrenuti sa kao super user):

- insmod – insertuje drajver u kernel operativnog sistema i, ako je to potrebno, prenosi njemu potrebne argumente
- rmmod – uklanja drajver iz kernel operativnog sistema
- mknod – pravi čvor drajvera pod tipom drajvera (c - char ili b – block), major i minor brojem
- cat /dev/kmesg – omogućava pregled dešavanja u kernel u real time-u



Slika 5 Primer inesertovanja modula u kernel operativnog sistema i pokušaj brisanja bez komande *sudo*

3.2 Logička operacija ekskluzivnog ILI (XOR)

U kriptografiji jednostavno šifrovanje operacijom ekskluzivno ILI predstavlja vrstu aditivnog šifrovanja koje se odvija po sledećim pravilima:

- A ^ 0 = A
- A ^ A = 0
- (A ^ B) ^ C = A ^ (B ^ C)
- (B ^ A) ^ A = B ^ 0 = B

gde ^ predstavlja bitwise operaciju ekskluzivnog ILI (XOR).

Za enkripciju dovoljno je samo XOR-ovati par (podatak, ključ), a za dekripciju ponoviti XOR operaciju nad parom (enkriptovan_podatak, ključ). Ova metoda šifrovanja podataka je uobičajena komponenta složenijih algoritama za enkripciju podataka.

4. Koncept rešenja

Koncept rešenja ostvaruje se kroz dva dela. Rešenje kernel modula i rešenje test programa za verifikaciju ispravnosti virtualnog uređaja, odnosno njegovog drajvera. Drajver implementira asimetričan metod šifrovanja podataka koji koristi javni ključ i tajni ključ prema kojima se generiše jedinstveni ključ za enkripciju/dekripciju podataka.

Tajni i javni ključ su celobrojne vrednosti tipa *unsigned char,* a jedinstveni ključ predstavljen je 8-bajtnim nizom takođe celobrojnih vrednosti tipa *unsigned char*.

4.1 Kernel modul

Prilikom pokretanja modula unosi se *seed* vrednost generatora slučajnih vrednosti. Ovako generisana vrednost naziva se tajni ključ. (Unešena kao parameter prilikom pokretanja modula). Dok god se modul nalazi unutar kernela operativnog sistema tajni ključ ostaje isti, a menja se jedino u slučaju kada je modul izbačen i ponovno ubačen u kernel sa drugim ulaznim parametrom. Ako dođe do toga da se ulazni parametar promeni, prethodno enkriptovani podaci sa prethodnim ulaznim parametrom postaju nečitljivi iako je korisnik uneo validan javni ključ. Oni postaju čitljivi i mogući za dešifrovanje samo u slučaju da se kernel modul ponovno pokrene sa istim parametrom pod kojim su oni šifrovani.

I javni i tajni ključ se generišu upotrebom *prng()* funkcije. Ona predstavlja pseudo-slučajan generator brojeva. Pošto C biblioteka *stdio.h* nije implementirana za programiranje kernel modula *prng()* funkcija predstavlja jednu prostu samostalno implementiranu funkciju pseudo-slučajnog generatora brojeva.

Javni ključ generiše se pomoć *seed* vrednosti prosleđenje od strane korisničke aplikacije (test aplikacije) u trenutku šifrovanja podataka.

4.1.1 ENKRIPCIJA

Kada korisnik zatraži putem test aplikacije da se podaci enkriptuju, kernel modul preuzima prosleđene podatke. Prosleđeni podaci sastoje se od jednog niza karaktera u kojem se nalaze, redom, opcija koju korisnik želi (enkripcija ili dekripcija), *seed* vrednost za generisanje javnog ključa i podaci za šifrovanje. Podaci su odvojeni karakterom '|'. Drajveri ih tada parsira i kreće u obradu.

Prvo se generiše javni ključ. Nakon toga generiše se jedinstveni ključ pozivom funkcije *unique_key()* kojoj se kao parametri prosleđuju tajni i javni ključ. Ova funkcija predstavlja još jednu samostalnu implementaciju pseudo-slučajnog generatora brojeva.

Kada je generisan jedinstveni ključ modul kreće da šifruje podatke. Prolazi se kroz podatke karakter po karakter i bitwise operacijom XOR se šifruju dok se ne dođe do kraja niza podataka. Jedinstveni ključ je, kako je već rečeno, predstavljen 8-bajtnim nizom celobrojnih vrednosti tipa *unsigned char* i paralelno se prolazi kroz niz podataka i niz jedinstvenog ključa. Kada se dođe do kraja ključa kreće se od njegovog početka. Pažnja se mora obratiti na rezultat operacije XOR para (podatak, jedinstveni ključ), ako je taj rezultat manji od 32, ponovo se radi XOR nad istim parom i podatak se ne menja. Ovaj proces se vrši zbog toga što su prvih 31 karakter u ASCII tabeli, eng. *Non-printable characters*, odnosno karakteri koji se ne mogu odštampati, a mogu napraviti problem (npr. NULL karakter – označiće kraj stringa, DELETE – obrisaće karakter na toj poziciji). Trideset i drugi karakter je karakter razmaka ‘ ‘ (eng. Space) te je on izuzet iz praktičnih razloga.

Nakon uspešnog šifrovanja podataka modul ponovno parsira podatke i šalje ih korisničkoj aplikaciji. Podaci koji se šalju su predstavljeni jednim nizom karaktera, a u koji je smešten javni ključ i šifrovani podaci, odvnojeni karakterom ‘|’.

4.1.2 DEKRIPCIJA

Kod dekripcije dolazi do sličnog procesa, osim što kada korisnička aplikacija zatraži dekripciju modul preuzima podatke po istom principu, ali ovoga puta na mestu *seed*-a u nizu karaktera nalazi se javni ključ.

Posle XOR-ovanja podataka, odnosno posle enkripcije podataka, oni se parsiraju ali se ovoga puta šalju samo dekriptovani podaci korisniku.

4.2 Test aplikacija

Test aplikacija korisniku nudi izbor enkripcije ili dekripcije podataka i to putem standardnog ulaza ili ulaznog fajla (“data.txt”).

U slučaju enkripcije aplikacija generiše *seed* vrednost potrebnu za stvaranje javnog ključa. Ona stavlja u jedan niz celobrojnih vrednosti i opciju (enkripcija ili dekripcija), *seed* vrednost i podatke te ih šalje modulu na dalju obradu. Kada modul obradi podatke, korisnička aplikacija čita podatke koji su joj upućeni i predstavlja ih korisniku putem standardnog izlaza i putem izlaznog fajla (“result.txt”).

U slučaju dekripcije program od korisnika zahteva da se unese javni ključ koji je dobijen nakon enkripcije podataka, učitava podatke, parsira i šalje modulu na obradu. Nakon što modul obradi podatke aplikacija ih prima i predstavlja korisniku putem standardnog izlaza i putem izlaznog fajla (“result.txt”).

5. Opis rešenja

5.1 Kernel modul

5.1.1 Inicijalzacija modula (encrypt_init)

int encrypt_init(void);

Funkcija za inicijalizaciju modula. Prilikom ubacivanja modula u kernel operativnog sistema ova funkcija je njegov početak. Vrši se registracija i dodeljivanje Major broja modulu kao i zauzimanje potrebnog memorijskog prostora.

Povratne vrednosti:

- 0 – uspešna alokacija memorijskog prostora
- result – neuspešno zauzimanje memorijskog prostora

5.1.2 Završetak modula (encrypt_exit)

void encrypt_exit(void);

Funkcija koja se poziva kada se modul izbacuje iz kernela operativnog sistema. Dealociraju se zauzeti memorijski prostori.

5.1.3 Otvaranje uređaja (encrypt_open)

static int encrypt_open(struct inode *inode, struct file *filp);

Funkcija otvaranja uređaja. Ispisuje poruku da je uređaj otvoren i tajni ključ. Ispis se vrši u /dev/encrypt. Inkrementira se vrednost Device_open.

Parametri:

- ne koriste se

Povratne vrednosti:

- EBUSY – resurs nedostupan
- SUCCESS – uspešno izvršena funkcija

5.1.4 Zatvaranje uređaja (encrypt_release)

static int encrypt_release(struct inode *inode, struct file *filp);

Dekrementira se vrednost Device_open. U slučaju da se u ovoj funkciji ta vrednost ne umanji ne možemo se rešiti uređaja.

Parametri:

- Ne koriste se

Povratne vrednosti:

- 0 - uspešno izvršena funkcija

5.1.5 Slanje podataka u user space-a (encrypt_read)

static ssize_t encrypt_read(struct file *filp, char *buf, size_t len, loff_t *f_pos)

Poziva funkciju *xor_data()* i *generate_end_buffer()*. Prebacuje podatke iz kernel space-a u user space funkcijom *copy_to_user()*.

Parametri:

- filp – struktura tipa “file”
- buf – bafer iy kojeg će user space funkcija primiti podatke
- len – brojač koji sadrži broj potrebnih bita koje treba poslati, a ima istu vrednost kao i običan brojač u user-space funkciji
- f_pos – pozicija odakle početi čitati fajl

Povratne vrednosti:

- EFAULT – greška
- data_size – veličina podataka
- 0 – uspešno izvršena funkcija

5.1.6 Primanje podataka iz user space-a (encrypt_write)

static ssize_t encrypt_write(struct file *filp, const char *buf, size_t len, loff_t *f_pos)

Pozivom funkcije *copy_from_user()*, prebacuje podatke iz user space-a u kernel space. Čita opciju koju je korisnička aplikacija prosledila (ENCRYPT ili DECRYPT) i na osnovu toga poziva, respektivno, *parse_buf_enc()* ili *parse_buf_dec()*.

Parametri:

- filp – struktura tipa “file”
- buf – bafer u koji će user space funkcija pisati podatke
- len – brojač koji sadrži broj potrebnih bita koje treba poslati, a ima istu vrednost kao i običan brojač u user-space funkciji
- f_pos – pozicija odakle početi čitati fajl

Povratne vrednosti:

- EFAULT – greška
- ERROR – greška
- len – brojač koji sadrži broj potrebnih bita koje treba poslati, a ima istu vrednost kao i običan brojač u user-space funkciji

5.1.7 Pseudo-slučajan generator brojeva (prng)

static unsigned char prng(const int seed)
Na osnovu vrednosti seed, generiše slučajnu vrednost.
Parametri:

- seed – vrednost za generisanje slučajnog broja

Povratne vrednosti:

- retVal – generisana vrednost

5.1.8 Generator jedinstvenog ključa (unique_key)

static void unique_key(const int s_key, const int p_key)
Na osnovu parametara generiše 8-bajtni celobrojni niz tipa *unsigned char*.
Parametri:

- s_key – tajni ključ
- p_key – javni ključ

Povratne vrednosti:

- /

5.1.9 Parsiranje u slučaju enkriptovanja (parse_buf_enc)

static int parse_buf_enc(char buffer[])
Parsira sadržaj ulaznog bafera iz funkcije *encrypt_wite()*. Smešta rezultate obrade u globalne promenljive: *int option, int p_seed, unsigned char buffer*. Generiše javni ključ pozivm *prng()*.
Parametri:

- buffer – ulazni bafer

Povratne vrednosti:

- SUCCESS – uspešno izvršenje parsiranja

5.1.10 Parsiranje u slučaju dekriptovanja (parse_buf_dec)

static int parse_buf_dec(char buffer[])
Parsira sadržaj ulaznog bafera iz funkcije *encrypt_wite()*. Smešta rezultate obrade u globalne promenljive: *int option, int p_key, unsigned char buffer*. Generiše javni ključ pozivm *prng()*.
Parametri:

- buffer – ulazni bafer

Povratne vrednosti:

- SUCCESS – uspešno izvršenje parsiranja

5.1.11 Generisanje završnog bafera (generate_end_buffer)

static int generate_end_buffer(void)

Prebacuje sadržaj globalnih promenljivih u bafer iz koje će user space funkcija čitati podatke na osnovu izabrane opcije (ENCRYPT ili DECRYPT)

Parametri:

- /

Povratne vrednosti:

- ERROR – greška u odabiru
- SUCCESS – uspešno smeštanje podataka u bafer

5.1.12 Logika kernel modula (xor_data)

static int xor_data(void)

Generiše jedinstveni ključ pozivom funkcije *unique_key()*. Logičkom operacijom ekskluzivnog ILI (XOR) obrađuje podatke iz globalne promenljive *buffer*. XOR-uje par (podatak, jedinstveni ključ).

Parametri:

- /

Povratne vrednosti:

- SUCCESS – uspešna obrada

5.1.13 Sadržaj bafera

Sadržaj bafera koji se šalje u user space nakon obrade

5.1.13.1 Enkripcija

Sadržaj bafera u slučaju enkripcije:

- Opcija
- Javni ključ
- Podaci

Format:

- opcija|javni_ključ|podaci

5.1.13.2 Dekripcija

Sadržaj bafera u slučaju dekripcije:

- Opcija
- Podaci

Format:

- opcija|podaci

5.2 Test aplikacija

5.2.1 Modul glavnog programa (main)

```
int main()
```

Sadrži sve funkcije i metode za testiranje ispravnosti kernel modula. Upisuje rezultate obrade iz bafera preuzetog iz kernel space-a u izlazni fajl “result.txt”.

Parametri:

- /

Povratne vrednosti:

- ERR – greška
- 0 – uspešno izvršavanje test programa

5.2.2 Start meni (start_menu)

```
int start_menu()
```

Nudi izbor enkripcije ili dekripcije korisniku.

Parametri:

- /

Povratne vrednosti:

- retVal – povratna vrednost opcije

5.2.3 Izbor načina učitavanja podataka (data_entry)

```
int data_entry()
```

Nudi izbor učitavanja podataka putem standardnog ulza ili ulaznog fajla korisniku.

Parametri:

- /

Povratne vrednosti:

- retVal – povratna vrednost opcije

5.2.4 Unos podataka preko standardnog ulaza (manual_entry)

```
int manual_entry(char buffer[])
```

Traži od korisnika unos podataka preko standardnog ulaza.

Parametri:

- buffer – bafer u koji se smeštaju podaci

Povratne vrednosti:

- 0 – podaci uspešno smešteni u bafer

5.2.5 Učitavanje podataka iz ulaznog fajla (file_entry)

```
int file_entry(char buffer[])
```

Otvara ulazni fajl “data.txt” i učitava podatke iz njega.

Parametri:

- buffer – bafer u koji se smeštaju podaci

Povratne vrednosti:

- 0 – podaci uspešno smešteni u bafer

5.2.6 Unos javnog ključa (request_p_key)

```
int request_p_key()
```

U slučaju izbora dekripcije, traži od korisnika unos javnog ključa.

Parametri:

- /

Povratne vrednosti:

- 0 – javni ključ uspešno zaprimljen

5.2.7 Generisanje bafera za slanje u kernel space (make_buf)

```
int make_buf(const int opt, const char buffer[])
```

Na osnovu izbora opcije (ENCRYPT ili DECRYPT) smešta podatke u bafer koji se šalje u kernel space.

Parametri:

- opt – ENCRYPT ili DECRYPT
- buffer – bafer u koji se smeštaju podaci

Povratne vrednosti:

- 0 – potrebni podaci uspešno upisani u bafer

5.2.8 Uklanjanje javnog ključa (remove_p_key)

```
void remove_p_key(char buffer[])
```

Uklanja javni ključ iz ulaznog bafera preuzetog iz kernel space-a.

Parametri:

- buffer – bafer iz kojeg se briše javni ključ

Povratne vrednosti:

- /

5.2.9 Upis javnog ključa u izlazni fajl (put_p_key_in_file)

```
void put_p_key_in_file(char p_key)
```

Upisuje javni ključ na kraj izlaznog fajla “result.txt”

Parametri:

- p_key – javni ključ

Povratne vrednosti:

- /

5.2.10 Sadržaj bafera

Sadržaj bafera koji se šalje u kernel space

5.2.10.1 Enkripcija

Sadržaj bafera u slučaju enkripcije:

- Opcija
- Seed za generisanje javnog ključa
- Podaci

Format:

- opcija|seed|podaci

5.2.10.2 Dekripcija

Sadržaj bafera u slučaju dekripcije:

- Opcija
- Javni ključ
- Podaci

Format:

- opcija|javni_ključ|podaci

6. Testiranje

Testiranje ispravnosti kernel modula vrši se pomoću test aplikacije i tri paralelno otvorena terminala. Da bi se test aplikacija uopšte mogla pokrenuti potrebno je pokrenuti kernel modul, to se vrši u nekoliko koraka:

- make – bilduju se izvršne datoteke sa ekstenzijom .ko
- insmod »encrypt.ko« seed=XXXXXXXXX- modul se ubacuje u kernel operativnog sistema, pod parametar seed unosimo proizvoljnu devetocifrenu vrednost za generisanje tajnog ključa
- mknod /dev/encrypt c MajorBr 0 – pravi se čvor kernel modula, c označava da je modul *eng. character device,* MajorBr se postavlja na vrednost dodeljenog major broja
- chmod 666 /dev/encrypt – menjaju se prava pristupa uređaju (read, write)

Napomena 1: prilikom insmod-ovanja modulu se automatski dodeljuje Major broj, a koji je ispisan u kernel log-u. Kernel log otvaramo komandom cat /dev/kmsg.

Napomena 2: sve komande se otvaraju komandom sudo

Kada je modul uspešno ubačen u kernel operativnog sistema možemo pokrenuti test aplikaciju. Testiranje se vrši kroz sledeće korake:

- make – bilduju se izvršne datoteke test aplikacije
- ./encrypt_test_app – pokretanje test aplikacije

Nakon što je aplikacija pokrenuta, lako je pratiti upustva za enkripciju/dekripciju.

Napomena 3: Nakon testiranja modul obavezno izbaciti iz kernela komandom *rmmod encrypt*

6.1 Test primeri

6.1.1 Unos podataka kroz standardni ulaz

Ako pratimo sve prethodno opisane korake spremni smo za testiranje. U fajl »data.txt« unećemo željenje podatke za enkripciju:

- Test?prekoSt4nd@RdnogU!!!laZa_#

Nakon obrade ovog stringa u »result.txt« fajlu dobićemo enkriptovane podatke:

- (WsPcC8@k]9Ph].e.\oCU!k!lS0E_#
- Javni ključ: 53

Ponovnim pokretanjem test aplikacije, izborom dekripcije i unošenjem javnog ključa u »result.txt« i na standardnom izlazu dobijamo:

- Test?prekoSt4nd@RdnogU!!!laZa_#

Napomena 1: Početna vrednost prilikom insmod-ovanja modula seed=321321321

Napomena 2: Rezultate obrade mozemo videti, kako u izlaznom fajlu, tako i na standardnom izlazu

Napomena 3: Fajlovi »data.txt« i »result.txt« nalaze se u ./data

6.1.2 »data.txt« kao ulazni stream podataka

Ako pratimo sve prethodno opisane korake spremni smo za testiranje. U fajl »data.txt« unećemo željenje podatke za enkripciju:

- Ovo j3 t3sT?_program_za 3NKRIPCiju, D3kripCIJU/podataka!#$

Nakon obrade ovog stringa u »result.txt« fajlu dobićemo enkriptovane podatke:

- pvBE*d 13syZ_'*XrLm_-Oe39f7IPm,Uu,EDdE7Vpn,JU/5PdLt!<Od#S
- Javni ključ: 66

Sada dobijeni string nakon enkripcije staviti u fajl »data.txt«. Ponovnim pokretanjem test aplikacije, izborom dekripcije i unošenjem javnog ključa u »result.txt« dobijamo:

- Ovo j3 t3sT?_program_za 3NKRIPCiju, D3kripCIJU/podataka!#$

Napomena 1: Početna vrednost prilikom insmod-ovanja modula seed=3213213213

Napomena 2: Rezultate obrade mozemo videti, kako u izlaznom fajlu, tako i na standardnom izlazu

Napomena 3: Fajlovi »data.txt« i »result.txt« nalaze se u ./data

6.1.3 Promena seed vrednosti

Koristićemo prethodni primer, u fajl »data.txt« postavićemo string:

- Ovo j3 t3sT?_program_za 3NKRIPCiju, D3kripCIJU/podataka!#$

Nakon enkripcije izlazni fajl »result.txt ima sadržaj:

- l-YmN3SY3(br{KrBD)W {Aa 3N}Rmk0DI.,m`3k_J+uInn\]L?W9EPa!#\0x7F
- Javni ključ: 124

Napomena : Početna vrednost prilikom insmod-ovanja modula seed=321321321

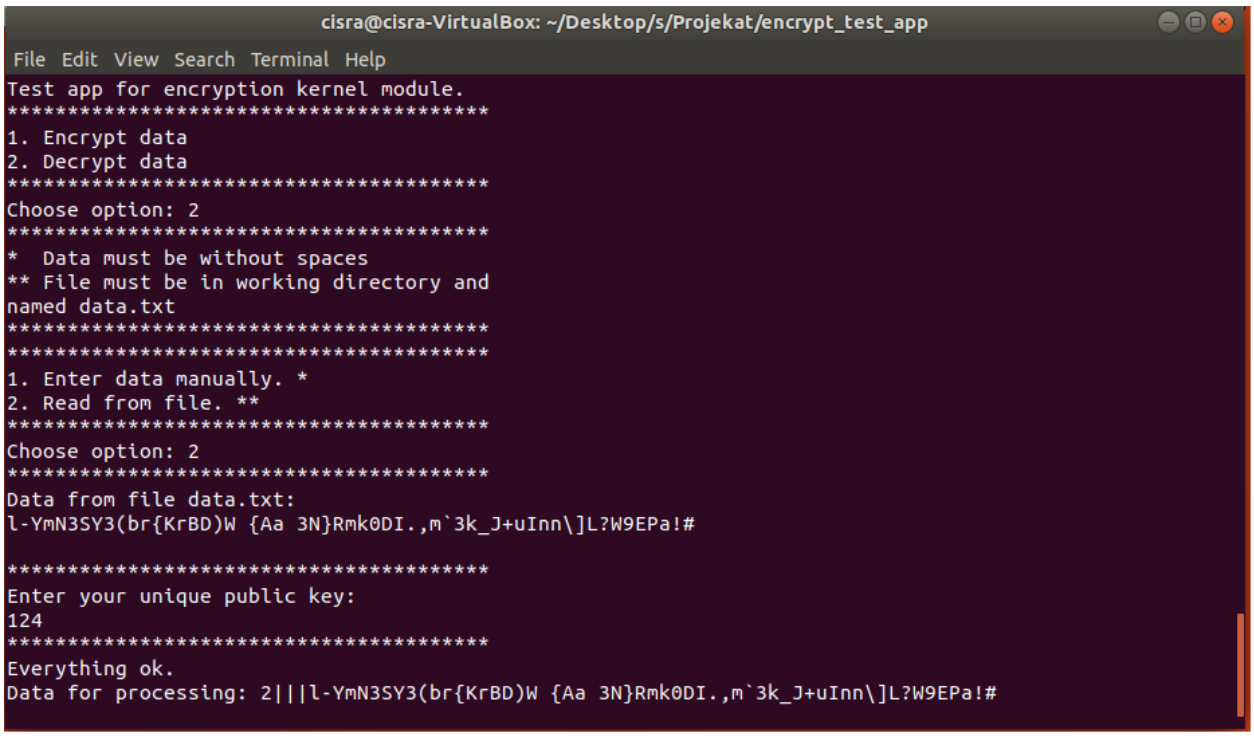
Sada ćemo uraditi rmmod encrypt.ko i ponovo inserotvati modul komandom: insmod encrypt.ko seed=149962112. Promenom ovog parametra menja se i tajni ključ.

Smeštanjem sadržaja »result.txt« u »data.txt«, ponovnim pokretanjem test aplikacije, odabirom dekripcije kroz ulazni fajl i unošenjem vrednosti 124 kao javnog ključa, sadržaj »result.txt« postaje:

- l-YGNJ`2R(1X:2A)%)W :8RKRr.x,k0/(.• G!JX4++&c/no6-?W9E)RJBC

Što očevidno neodgovara prvobitnom stringu, prema tome verifikacija kernel modula je uspešno obavljena.

Poruka se ne bi dekriptovala ni u slučaju da smo pogrešno uneli vrednost javnog ključa.



Slika 6 Dešifrovani podaci nakon promene seed vrednosti tajnog ključa

7. Zaključak

U ovom tekstu data je ideja generisanja dva ključa: jedan koji generiše sam modul na osnovu parametarske vrednosti seed I vrednosti koja se generiše na osnovu trenutnog vremena šifrovanja, te spajanjem te dve vrednosti kako bi se dobio jedinstveni niz vrednosti za šifrovanje. Ideja se zasniva na principu asimetričnog šifrovanja.

Za ozbiljnije šifrovanje podataka gde je bezbednost prioritet funkcije prng() i unique_key() ne garantuju bezbednost, ali za ovaj primer one su dovoljne za prikazivanje principa asimetričnog šifrovanja. U svrhu bezbednosti poželjno je koristiti pseudo-slučajne generatore koji garantuju bezbednu randomizaciju brojeva. Jedan od takvih pseudo-slučajnih generatora može se pronaći u C zaglavlju <sodium.h>, a koji nije dostupan za programiranje kernel modula.

Prikazan je i princip rada i programiranja kernel modula za operativni sistem Linux. Zbog manjka literature i dostupnih informacija ovo nije ni malo lak zadatak. Ne smemo zaboraviti da je jedan veći deo C biblioteka izveden za programiranje kernel modula, ali na programeru je da se što bolje snađe, pa čak i ispiše nove funkcije koje bi u C jeziku i “normalnom” programiranju predstavljale jednu liniju koda. Jedan pogrešan pokazivač u kernel space-u može da znači ponovo pokretanje celog sistema, stoga je ovo izazovan i mukotrpan proces.