UNIVERZITET SINGIDUNUM

Departman za poslediplomske studije
SAVREMENE INFORMACIONE TEHNOLOGIJE

MASTER RAD

ZAŠTITA PODATAKA U BAZI

mentor
prof.dr Mladen Veinović

kandidat
Đorđe Zečević
410591/2009

Beograd, 2011.

Zaštita podataka u bazi

Sažetak

U ovom radu je opisan model zaštite podataka u sistemima za upravljanje bazama podataka. Model je zasnovan na standardnim algoritmima za šifrovanje i na standardnim dodatnim servisima radi lakše primene u realnom okruženju. Detaljno je opisana i implementacija pomoću alata otvorenog koda.

Ključne reči: baza podataka, šifrovanje, podatak, servis direktorijuma, zaštita, sigurnost, otvoren kod.

Data security in database

Abstract

In this paper contains model for protecting data stored in database management systems. Model is based on standard cryptography algorithms and on standard services all to accommodate easy implementation in real world environment. Detail steps of inmplemetation with open source tools are described in second part of paper.

Key words: database, cryptography, information, directory service, protection, security, open source.

Sadržaj:

1. UVOD 3

1.1. BAZE PODATAKA 4
1.2. ZNAČAJ ZAŠTITE PODATAKA U BAZAMA 5
1.3. ZAŠTITA BAZA PODATAKA 6

2. PREDHODNA ISTRAŽIVANJA 8

2.1. MICROSOFT 8
2.2. ORACLE 10

3. METODOLOŠKI KONCEPT 11

3.1. PREDMET ISTRAŽIVANJA 11
3.2. CILJ I ZADACI ISTRAŽIVANJA 11
3.3. ISTRAŽIVAČKE HIPOTEZE 11
3.3.1. Generalna hipoteza 11
3.3.2. Posebne hipoteze 11
3.4. METODE ISTRAŽIVANJA I TOK ISTRŽIVAČKOG PROCESA 12

4. REZULTATI ISTRAŽIVANJA 13

4.1. MODEL ZAŠTITE PODATAKA 13
4.1.1. Bezbedno radno okruženje(SOE) 13
4.1.2. Sigurnosni dnevnik 13
4.1.3. Skraćeni nazivi funkcija 14
4.2. KONTROLA KORISNIKA 14
4.3. ŠIFROVANJE I UPRAVLJANJE KLJUČEVIMA 14
4.3.1. Šifrovanje sa sistemom javnih ključeva 16
4.3.2. Šifrovanje na nivou grupe korisnika 18
4.4. ŠIFROVANJE PODATAKA U MYSQL BAZI 20
4.4.1. Dodatne komponente 20
4.4.2. Sigurnosni katalog 20
4.4.3. Sertifikaciono telo 22
4.4.4. Server direktorijuma 23
4.4.5. CREATE TABLE upit 24
4.4.6. INSERT INTO upit 26
4.4.7. SELECT upit 27
4.4.8. ALTER TABLE upit 28
4.4.9. DELETE FROM upit 30
4.4.10. DROP TABLE upit 30
4.5. KLIJENTSKA APLIKACIJA 30

5. ZAKLJUČAK I PRIMENLJIVOST REZULTATA 31

5.1. PRIMENLJIVOST REZULTATA 32

6. LITERATURA 38

7. DODACI 39

7.1. KORIŠĆENI ALATI I SOFTWARE 39
7.2. IZVORNI KOD PROGRAMA KOJI KORISTI PROXY BAZE PODATAKA 39

1. Uvod

Ovaj Master rad se sastoji iz četiri celine. Prva celina je uvodna, i ona čitaoca upoznaje sa osnovnim pojmovima i temom rada. Opisan je pojam baze podataka i data je kratka istorija razvoja baza. Nakon toga je opisan značaj zaštite podataka, kroz karakterističan tok podataka u savremenim aplikacijama, sa posebnim osvrtom na sigurnosne probleme sa skladištenjem podataka. Detaljnije su opisani sigurnosni mehanizmi u savremenim bazama i analizirani problemi koji postoje.

U drugom delu rada prikazana su najnovija rešenja velikih proizvođača baza podataka i njihov način rešavanja problema sigurnosti podataka u bazi. Cilj ovog dela rada je da se uoče nedostaci trenutnih sistema zaštite i pronađu rešenja.

Kako bi pronašli rešenja, u trećem delu rada je opisan metodološki koncept istraživanja koje je urađeno. Iznešene su hipoteze koje će biti dokazane u nastavku rada, kroz istraživanje.

U četvrtom, najvažnijem, delu rada iznešeni su rezultati istraživanja. Opisan je teorijski model zaštite podataka u bazama zasnovan na radu inženjera, Jingmin He i Min Wang, iz IBM-ovog Centra za istraživanje. Model predviđa korišćenje šifrovanja da bi se zaštitili podaci u bazama podataka. Kako bi se šifrovanje primenilo, definisan je pojam Sigurnosnog kataloga, koji se upotrebljava za smeštanje informacija o šifrovanju podataka. U njemu se beleže kolone koje su šifrovane, prava korisnika da šifrovanje uključe ili isključe I ključeve kojima se podaci šifruju. Pored Sigurnosnog kataloga, razmotreni su i slučajevi šifrovanja na nivou grupe korisnika, kao i problematika primene šifrovanja nad podacima koji već postoje (naknadno uključivanje šifrovanja).

Drugi deo poglavlja o rezultatima istraživanja sadrži praktičnu primenu teoretskog modela. U ovom delu se nalazi najveći doprinos ovog rada problematici šifrovanja podataka u bazama. Detaljno je objašnjena primena teorijskog modela u zaštiti MySQL baze podataka. Uveden je koncept Proxy aplikacije kao cetralnog mehanizma za šifrovanje podataka. Proxy je aplikacija koja je podstavljena između klijenta i baze podataka. Ubacivanjem Proxy aplikacije, omogućena je primena nestandardnog SQL jezika, tj. uvođenje novih ključnih reči: **encrypt**, **key, user_list** i **update**. Pomoću ovih ključnih reči, korisniku se pruža mogućnost da upravlja šifrovanjem podataka iz standardne SQL klijentske aplikacije. Centralna uloga Proxy-a je da pregleda upite koje šalje klijent, po potrebi ih menja, izvršava izmene u Sigurnosnom katalogu i prosleđuje upite bazi podataka.

Osim Proxy-ja, za uspešan rad modela, potrebni su i dodatni servisi: server diretkorijuma i PKI infrastuktura. U rezultatima istraživanja se opisuje uloga i konfiguracija ovih servisa.

Proxy aplikacija sama ne menja upite koje šalje klijent, već pruža mogućnost programiranja. Na kraju ovog rada prikazan je programski kod koji je napisan u svrhu rada sa Sigurnosnim katalogom i manipulisanje sa korisničkim upitima.

1.1. Baze podataka

Baze podataka predstavljaju viši nivo rada sa podacima u odnosu na klasične programske jezike. Reč je o tehnologiji koja je nastala sa namerom da se uklone slabosti tradicionalne automatske obrade podataka iz 60-tih i 70-tih godina 20. veka. Ta tehnologija osigurala je veću produktivnost, kvalitet i pouzdanost u razvoju aplikacija koje se svode na memorisanje i pretraživanje podataka u računaru.

Baza podataka je skup međusobno povezanih podataka, snimljenjih u memoriji računara. Podaci su istovremeno dostupni raznim korisnicima i programima. Ubacivanje, promena, brisanje i čitanje podataka obavlja se posredstvom zajedničkog programa. Korisnici i aplikacije pritom ne moraju poznavati detalje fizičkog prikaza podataka, već se referenciraju na logičku strukturu baze. Sistem za upravljanje bazom podataka (Database Management System - DBMS) je veza sa bazom podataka. On oblikuje fizički prikaz baze u skladu sa traženom logičkom strukturom. Takođe, on obavlja u ime klijenata sve operacije sa podacima. Dalje, on je u stanju da podrži više baza, od kojih svaka može imati svoju logičku strukturu, ali u skladu sa istim modelom. Isto tako, brine se za sigurnost podataka, i automatizuje administrativne poslove s bazom. Podaci u bazi su logički organizovani u skladu s nekim modelom podataka. Model podataka je skup pravila koja određuju kako može izgledati logička struktura baze. Model čini osnovu za projektovanje i implementiranje baze. Dosadašnji DBMS-i obično su podržavali neki od sledećih modela:

Relacioni model. Zasnovan na matematičkom pojmu relacije. I podaci i veze između podataka prikazuju se tabelama.

Mrežni model. Baza je opisana usmerenim grafom. Čvorovi su tipovi zapisa, a lukovi definišu veze između tipova zapisa.

Hijerarhijski model. Specijalni slučaj mrežnog. Baza je predstavljena jednim stablom ili skupom stabala. Čvorovi su tipovi zapisa, a hijerarhijski odnos "nadređeni-podređeni" izražava veze među tipovima zapisa.

Objektni model. Inspirisan je objektno-orijentisanim programskim jezicima. Baza je skup objekata koji se sastoje od svojih internih podataka i "metoda" (operacija) za rukovanje s tim podacima. Svaki objekat pripada nekoj klasi. Između klasa se uspostavljaju veze nasleđivanja, agregacije, odnosno međusobnog korišćenja operacija.

Hijerarhijski i mrežni modeli bili su u uptrebi u 60-tim i 70-tim godinama 20. veka. Od 80-tih godina pa sve do danas preovladava relacioni model. Očekivani prelaz na objektni model za sada se nije desio, tako da kada danas pričamo o bazama podataka ustvari pričamo o relacionim bazama.

1.2. Značaj zaštite podataka u bazama

Bezbednosni propusti u računarskom sistemu mogu uticati na poverljivost, integritet ili dostupnost sistema i podataka. Gubitak poverljivosti servera baze podataka podrazumeva neovlašćen pristup podacima, gubitak integriteta podrazumeva neovlašćenu izmenu ili brisanje podataka u bazi, a gubitak dostupnosti podrazumeva da servisi baze podataka nisu dostupni. U zavisnosti od namene sistema, gubitak jednog od ovih elemenata može izazvati veću ili manju štetu. Korišćenje Interneta je postalo svakodnevnica za današnje društvo. Internet kupovina je postala veoma popularna, a elektronsko poslovanje je praktično izmenilo poslovanje iz korena.

Relacione baze podataka igraju veoma važnu ulogu u elektronskom poslovanju. Ogromne količine podataka se prikupljaju, skladište i obrađuju, što zahteva moćnu podršku baze podataka. Količina podataka se stalno povećava, potreba za većom brzinom obrade i pretrage, tj. izvlačenja korisnih informacija se stalno povećava.

Ako analiziramo jednu aktivnost kupovine na Internetu, a ovaj model se može primeniti i na skoro svaku aktivnost u elektronskom poslovanju, a posebno obratimo pažnju na tok podataka, videćemo da se mogući sigurnosni rizici mogu naći na jednom od dva mesta:

1. **Sigurni prenos podataka**. Kada korisnik pošalje svoje poverljive podatke (broj kreditne kartice, PIN broj,...) kroz Internet pretraživač ili drugu vrstu programa, ti podaci bi trebalo da ostanu poverljivi duž celog puta, tj. kroz Intenret mrežu, aplikacioni server, pa sve do baze podataka gde će biti snimljeni.
2. **Sigurno skladišetenje i pristup podacima**. Kada podaci dođu do baze podataka, potrebno ih je zaštititi na takav način da podacima mogu da pristupe samo osobe koje su prošle kroz proces provere identiteta i provere prava pristupa.

Problem sigurnog prenosa podataka je dosta dobro obrađen i podržan od većine proizvođača programa. Standardne zaštite pri prenosu, kao što su SSL (Secure Socket Layer) i TLS (Transport Layer Security), su podržani u većini Internet pretraživača i lako se ugrađuju u nove programe. Međutim, kada podaci dođu do baze, ne postoji efikasna zaštita od neovlašćenog pristupa.

Iako mehanizam kontrole pristupa postoji skoro isto toliko koliko i same baze podataka, sigurnost baze se uzimala u obzir samo ako se za to pojavi potreba, recimo kada su podaci već kompromitovani. Ovo je verovatno posledica više faktora:

- **Performanse baze podataka**. Koliko god da baza brzo obrađuje podatke uvek postoji potreba da se obrada ubrza, tako da uvođenje procedura u baze koje bi usporile obradu, se često ne razmatraju, čak su i nepoželjne.
- **Pozicija baze podataka u računarskoj mreži**. Standardno, baze podataka nisu dostupne sa Interneta i pristup iz unutrašnje računarske mreže je ograničen, pa se smatra da je rizik od ugrožavanja sigurnosti mali.

1.3. Zaštita baza podataka

Osnovna sigurnosna komponenta u bazama podataka je upravljanje korisnicima. Potrebno je napraviti korisnika za svakoga ko želi da pristupi resursima baze podataka. Sam postupak kreiranja korisnika je trivijalan, ali upravljanje korisnicima, posebno u velikim sistemima sa puno promena, je težak posao.

Administrator baze podataka (u nastavku teksta DBA[1]) je osoba koja je zadužena za kreiranje, izmenu i brisanje korisničkih naloga. On kreira korinika, određuje mu način provere identiteta, recimo korišćenjem lozinke. U ovom procesu se javlja prvi sigurnosni problem koji ćemo pokazati na primeru[2].

DBA kreira korisnika **Marko.**

```
CREATE USER MARKO
IDENTIFIED BY 'MARKOVALOZINKA';
```

Sada tabela sa korisnicima sadrži novi red.

```
SELECT Host, User, Password FROM user WHERE User = 'marko';
+------+-------+-------------------------------------------+
| Host | User  | Password                                  |
+------+-------+-------------------------------------------+
| *    | marko | *8DE0D5950A3F06F3A5D537720EC02D81C70B172E |
+------+-------+-------------------------------------------+
```

Slika 1 – Red u tabeli koji sadrži loinku za korisnika

Korisnička lozinka je smeštena u tabelu u šifrovanom obliku. DBA sada može da preuzme Markov identitet, bez ičijeg znanja, prateći sledeće korake:

[1] Skraćenica u engleskom jeziku za Database Administrator, u prevodu administrator baze podataka

[2] Svi primeri u ovom radu su izvedeni u MySQL bazi podataka

1. Izvrši upit nad bazom kako bi dobavio šifrovan oblik Markove lozinke, *8DE0D5950A3F06F3A5D537720EC02D81C70B172E.

2. Izmeni Markovu šifru izvršavajući sledeći upit:

```
UPDATE USER
SET PASSWORD=PASSWORD('DRUGALOZINKA')
WHERE USER = 'MARKO';
```

3. Pristupi sistemu kao korisnik Marko. Izvrši izmene za koje Marko ima prava.
4. Vrati Markovu lozinku na staru vrednost, izvršavajući sledeći upit:

```
UPDATE USER
SET PASSWORD='*8DE0D5950A3F06F3A5D537720EC02D81C70B172E'
WHERE USER = 'MARKO';
```

Kontrola pristupa predstavlja glavni sigurnosi mehanizam u bazama podataka. On se bazira na pojmu prava. Svaki korisnik može pristupiti bilo kom objektu baze ako za to poseduje odgovarajuća prava. DBA ima prava nad svim objektima u bazi, pa može u bilo kom trenutku i bez bilo čijeg znanja da ta prava izmeni. Isto tako može da uzme identitet drugog korisnika i da bez njegovog znanja napravi izmene u sistemu, za šta će snositi posledice korisnik čiji je identitet ukraden. Osim korisnika koji su ljudi, često se u bazama podataka mogu naći razni sistemski i aplikativni korsnici, čije naloge koriste aplikacije koje pristupaju bazi podataka.

Šifrovanje u bazama podataka je loše podržano. U ogromnoj većini slučajeva podaci se smeštaju kao čist tekst. Kao posledica ovog stanja, sigurnost poverljivih podataka korisnika zavisi od operativnog sistema koje je podloga bazi podataka, a on često može da bude nedovoljno zaštićen. Pristup osetljivim podacima može da kontroliše operativni sistem ili sistem za upravljanje bazom podataka. U oba slučaja, podaci su nedovoljno zaštićeni. Dva su načina da se ovaj problem prevaziđe. Prvi je da se podaci ne drže na serveru na kom se nalazi sistem baze podataka, a drugi da se šifruje ceo sadržaj tvrdih diskova na kojima se nalaze podaci. I jedan i drugi način dovode do javljanja novih problema.

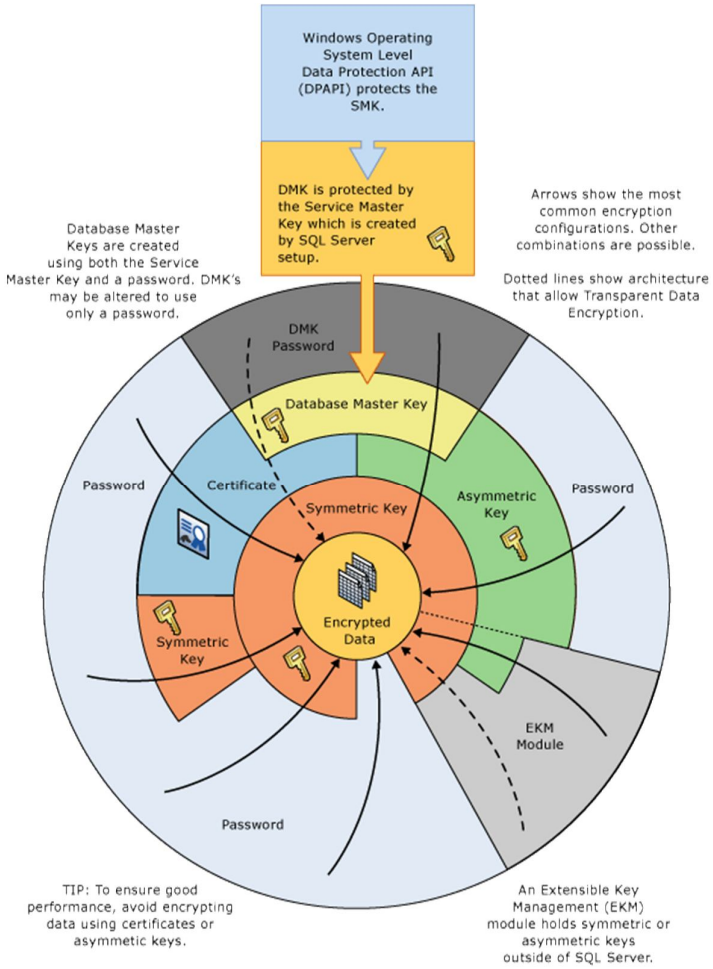
Ukoliko posmatramo administraciju baze podataka, administrator igra veoma važnu ulogu. On vodi računa da baza ima određene performanse, da baza ima rezervne kopije, rešava probleme sa pristupom podacima. Ukoliko posmatramo sigurnost baze podataka i privatnost podataka u njoj, administrator je osoba koja ima uvid u svačije podatke, koji mogu biti i lične prirode. U određenim slučajevima (npr. zdravstveni kartoni u bazi podataka), više nije pitanje poverenja koje se daje administratoru, već je i stvar principa, pa i kršenja zakona o poverljivosti podataka.

2. Predhodna istraživanja

U ovom poglavlju ćemo razmotriti principe koje primenjuju veliki proizvođači baza podataka kao što su Oracle i Microsoft. Takođe, ukazaćemo na slabosti koje mogu imati ovakvi sistemi.

2.1. Microsoft

Najnoviji proizvod za zaštitu baza podataka, firme Microsoft, je Transparent Data Encryption[3] (skraćeno TDE). TDE se pojavio sa verzijom SQL servera 2008. Kao što je prikazano na slici, Microsoft pruža šest različitih scenarija za zaštitu podataka. Svaka od strelica prikazuje jedan od načina za šifrovanje:



Slika 2 – Šema za zaštitu podataka u MS SQL Serveru

[3] Transparent data encryption je engleski termin koji se ne može bukvalno prevesti na srpski jezik. On označava zaštitu podataka bez dodatnih akcija korisnika.

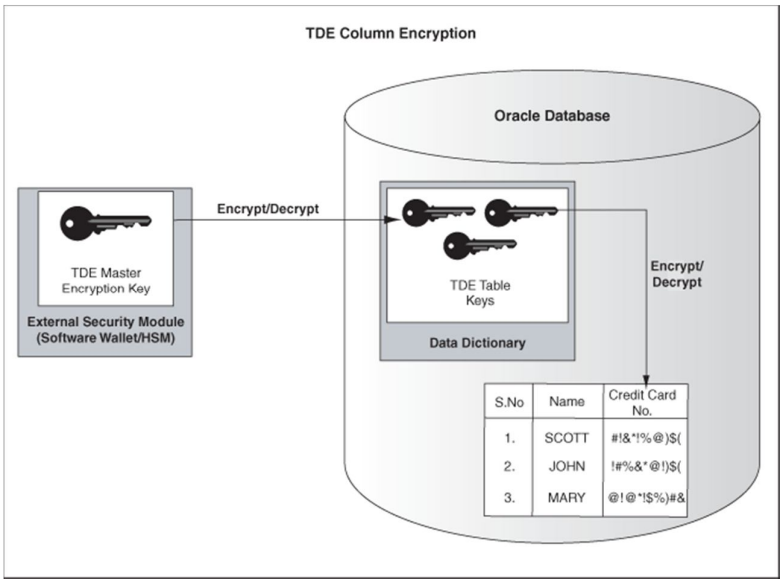
- Korišćenje lozinke za direktno šifrovanje podataka.
- Korišćenje dva simetrična ključa za šifrovanje podataka, od kojih jedan služi za zaštitu drugog, a drugi se štiti lozinkom.
- Sledeća dva načina su u suštini ista, a rade tako što se za zaštitu podataka koristi simetrični algoritam, ključ se štiti korisničkim sertifikatom ili asimetričnim algoritmom za šifrovanje, a asmetrični ključ, odnosno, sertifikat se štiti sa lozinkom.
- TDE koristi simtrični algoritam za šifrovanje podataka, korisničke sertifikate za zaštitu ključa za simetrično šifrovanje, Database Master Key (skraćeno DMK) za zaštitu sertifikata, i konačno DMK se štiti lozinkom.
- Poslenji način je da se podaci šifruju simetričnim algoritmom, a da se za zaštitu ključeva simetričnog algoritma koristi rešenje koje nije u samom SQL serveru. Onaj način ne podržava šifrovanje na nivou kolona.

Prvih pet načina šifrovanja se svodi na korišćenje lozinke, tako da je sigurnost celog sistema loša. Poslednji način se može iskoristiti za građenje dobrog sistema zaštite ali zahteva dodtne sisteme za upravljanje ključevima.

Dodatni problem je činjenica da podaci koji su u upotrebi, nisu zaštićeni, čak se, ukoliko postoji manjak slobodne memorije, mogu i upisati na tvrdi disk u otvorenom tekstu.

2.2. Oracle

Oracle, ima proizvod za šifrovanje podataka u bazi koji se, takođe, zove Transparent Data Encryption[4] (skraćeno TDE). Ovaj proizvod je uveden prvi put u verziji Oracle Database 10g release 2 (10.2).



Slika 3 – Dijagram šifrovanja u Oracle bazi

Postoje dve opcije pri uključenju šifrovanja podataka. Podaci se mogu šifrovati na nivou kolone i na nivou tablespace[5]. Za razliku od Microsoft SQL Servera, Oracle nudi samo jednu opciju za proces šiforvanja podataka. Podaci se šifruju simetričnim algoritmom, ključevi za simetričan algoritam se štite sistemom javnih i tajnih ključeva, dok se ovi ključevi štite drugim parom javnih i tajnih ključeva koji se nalaze van baze podataka.

Ovo je bolji pristup zaštiti podataka. Jedina zamerka je što su svi ključevi za zaštitu podataka na kraju zaštićeni jednim Master[6] ključem. Ako je Master ključ kompromitovan, svi podaci u bazi ostaju nezaštićeni.

[4] Transparent data encryption je engleski termin koji se ne može bukvalno prevesti na srpski jezik. On označava zaštitu podataka bez dodatnih akcija korisnika.

[5] Tablespace, je mesto gde Oracle baza smešta podatke. Tablespace se sastoji od datoteka, i nezavisan je od lokacije datoteke ili operativnog sistema na kom baza radi.

[6] Master ključ je čest naziv za ključ pomoću koga se može otključati sve što se štiti.

3. Metodološki koncept

3.1. Predmet istraživanja

Predmet ovog istraživanja je metod zaštite podataka u bazama pomoću šifrovanja.

3.2. Cilj i zadaci istraživanja

Cilj ovog istraživanja je pronalaženje mehanizma za šifrovanje podataka u bazama. Mehanizam za šifrovanje mora biti realizovan tako da:

- Koristi standardne algoritme za šifrovanje;
- Koristi alate koji već postoje, kako bi se smanjila cena implementacije;
- Pruži mogućnost šifrovanja na nivou kolone;
- Pruži mogućnost šifrovanja na novou grupe korinika;
- Pruži mehanizam za uključivanje i isključivanje šifrovanja u bilo kom trenutku;
- Ne traži izmene u postojećim sistemima za upravljanje bazama podataka;
- Reši probleme postojećih sistema za šifrovanje;
- Bude lako primenljiv u postojećim okruženjima.

3.3. Istraživačke hipoteze

3.3.1. Generalna hipoteza

Osnovna hipoteza ovog rada je pretpostavka da je moguće napraviti mehanizam za zaštitu podataka u sistemima za upravljanje bazama podataka, koji je nezavisan od proizvođača.

3.3.2. Posebne hipoteze

Mehanizam mora biti, pre svega, primenljiv u realnom okruženju, i mora koristiti proverene standardne algoritme za šifrovanje. Da bi ovaj mehanizam bio primenljiv mora ispuniti dodatne uslove koji su postavljeni u zadacima istraživanja.

Još jedna od pretpostavki je da se ovakav mehanizam može napraviti bez ulaganja u skupe programe i pakete programa koje nude svetski proizvođači, što će biti dokazano kroz detaljan opis jednog ovakvog mehanizma.

3.4. Metode istraživanja i tok istrživačkog procesa

Složenost predmeta istraživanja zahteva primenu više načina istraživanja. Prikupljanje podataka će biti relizovano:

- Analizom literature;
- Sekundarnom analizom dobijenih rezultata ranijih istraživanja;
- Analizom postojećih mehanizama za zaštitu podataka;
- Analizom problema od strane autora.

Nakon prikupljanja potrebne količine podataka, autor će metodama analize, dedukcije i apstrakcije pokušati da dokaže da su postavljene hipoteze tačne.

4. Rezultati istraživanja

4.1. Model zaštite podataka

Osim ugrađenih načina zaštite podataka u bazama, postoje slučajevi kada je potrebna dodatna zaštita. Model koji će biti predstavljen u ovom poglavlju rešava sigurnosne probleme koji su navedeni u predhodnom tekstu. Kako bi predstavili ovaj model potrebno je definisati nekoliko termina.

4.1.1. Bezbedno radno okruženje(SOE[7])

Operacije koje uključuju rad sa tajnim podacima potrebno je izvršavati u okruženju koje neće dovesti do toga da podaci budu ugroženi zbog samog okruženja, tj. koje neće imati sigurnosne propuste. Dobar primer za ovakvo okruženje je smart kartica. Korisnički privatni ključevi su na kartici i nikada je ne napuštaju u toku korišćenja. Ukoliko je potrebno izvršiti neku operaciju nad podacima, operacija se izvršava na kartici a samo rezultat napušta karticu.

Iako je baza podataka dosta složeniji sistem od smart kartice, ipak se jezgro baze može smatrati prilično bezbednim radnim okruženjem, a sigurnosni propusti nastaju slabim podešavanjem.

4.1.2. Sigurnosni dnevnik

Svi podaci, uključujući i one o korisničkim nalozima i pravima korisnika (sistemski katalog), nalaze se u tabelama baze podataka. Kao i ostali podaci oni mogu biti promenjeni direktnim izvršavanjem upita od strane administratora baze podataka.

Sigurnosni dnevnik je sličan sistemskom katalogu osim što izmene u sigurnosnom dnevniku može da izvrši samo proces baze podataka (za koji smo rekli da je bezbedno radno okruženje). To praktično znači da se korisnički nalozi mogu menjati samo pozivom sistemskih procedura.

Sigurnosni dnevnik se može kreirati na više načina. Može biti unutar baze podataka u vidu sistemskih procedura, ali može biti kreiran i kao aplikacija koja se nalazi van baze podataka ali u bezbednom radnom okruženju. Manje složen način podrazumeva kreiranje dnevnika unutar baze podataka i njega ćemo upotrebiti u ovom radu.

[7] Secure Operating environment(SOE)

4.1.3. Skraćeni nazivi funkciija

Kako bi u kasnijem tekstu izbegli pojašnjavanje osnovnih funkcija šifrovanja, i time se nepotrebno udaljili od suštine problema, sada ćemo ih definisati:

Funkcija E predstavlja operaciju šifrovanja. Funkcija D predstavlja operaciju dešifrovanja. Par funkcija (D,E) se može odnositi na simetrične algoritme za šifrovanje, kao što su 3DES i AES, kao i na asimtrične algoritme za šifrovanje, kao što je RSA. Otvoreni tekst ćemo obeležavati sa slovom m, šifrovani tekst sa slovom c, a ključeve za šifrovanje sa slovom K.

4.2. Kontrola korisnika

Da bi pristupio podacima, korisnik mora imati nalog u bazi. Administratora baze je zadužen za kontrolu korisničkih naloga. On kreira i briše korisnike, određuje način njihove provere identiteta i prava pristupa resursima baze. Postoji više načina provere identiteta, putem lozinke, putem PKI[8] infrastrukture, Kerberos protokola, itd. Sve metode se svode na tajni podatak koji je poznat samo korisniku.

Potrebno je da jedino korisnik ima pravo izmene svog naloga. Na primer, potrebno je da jedino korisnik može da promeni svoju lozinku, drugi korisnici (ili administrator) bi to mogli da urade samo ako im je dodeljeno određeno pravo. U današnjim sistemima za upravljanje bazom podataka, administrator može promeniti lozinku korisniku na njegov zahtev, ali i kada korisnik to nije zahtevao. Administrator, u stvari, može da promeni lozinku korisniku bez ičijeg znanja, što je opasan sigurnosni propust.

4.3. Šifrovanje i upravljanje ključevima

Najveći problem pri korišćenju šifrovanja je upravljanje ključevima. Sami algoritmi su dovoljno dobri. Kada pričamo o šifrovanju u bazama podataka, dve teme je potrebno obraditi kako bi se šifrovanje ugradilo u bazu podataka:

1. Potrebno je da korisnik ima mogućnost da odredi koji će podaci biti šifrovani pre nego što te podatke snimi u bazu.
2. Potrebno je da korisnik ima mogućnost (posredno ili neposredno) da odredi ključ kojim će se podaci šifrovati.

[8] Public Key Infrastructure – provera identiteta zasnovana na sertifikatima

Posmatrajući drugu stavku, ceo problem šifrovanja u bazi se svodi na upravljanje ključevima. Da bi rešili problem iz tačke jedan potebno je kreirati proceduru za kreiranje tabele koja će omogućiti da korisnik naznači kolone koje želi da šifruje:

```
CREATE TABLE KUPAC
(ID INTEGER PRIMARY KEY
IME VARCHAR(30),
BROJ_KARTICE VARCHAR(16) ENCRYPT UPDATE 1
);
```

Izvršavajući ovu proceduru korisnik je naznačio da kolona broj_kartice treba da bude šifrovana (ključna reč ENCRYPT), i server baze podataka će šifrovati podatke pre nego što ih upiše u bazu. Da bi server šifrovao podatke potrebno je da zna kojim ključem će podatke šifrovati a kasnije i dešifrovati. Polje UPDATE koje se postavlja na vrednost 1 služi za kontrolu prava nad šifrovanjem kolone.

Da bi imali evidenciju o kolonama koje su šifrovane, potrebno je da kreiramo novi sigurnosni katalog. Ovaj katalog bi sadržao polja **id_korisnika, ime_tabele, ime_kolone, izmena, izmenu_izvrsio**. Na primer:

```
(marko, kupac, broj_kartice, 1, marko)
```

Pretpostavimo da je korisnik sa korisničkim (i ličnim) imenom Marko izvršio proceduru kreiranja tabele. Samo postojanje reda u sigurnosnom katalogu znači da će kolona u tabeli biti šifrovana. Polje **izmena** će označiti prava nad ovim zapisom i može imati vrednosti 0, 1 ili 2. Vrednost 0 znači da niko ne može da menja ovaj zapis pa ni kreator zapisa, 1 znači da zapis može da menja samo kreator zapisa, tj. vlasnik tabele koja se nalazi u zapisu i 2 znači da zapis može da menja bilo ko, ko ima prava nad tabelom koja se nalazi u zapisu. U primeru koji smo naveli, samo Marko može da menja zapis koji je kreirao.

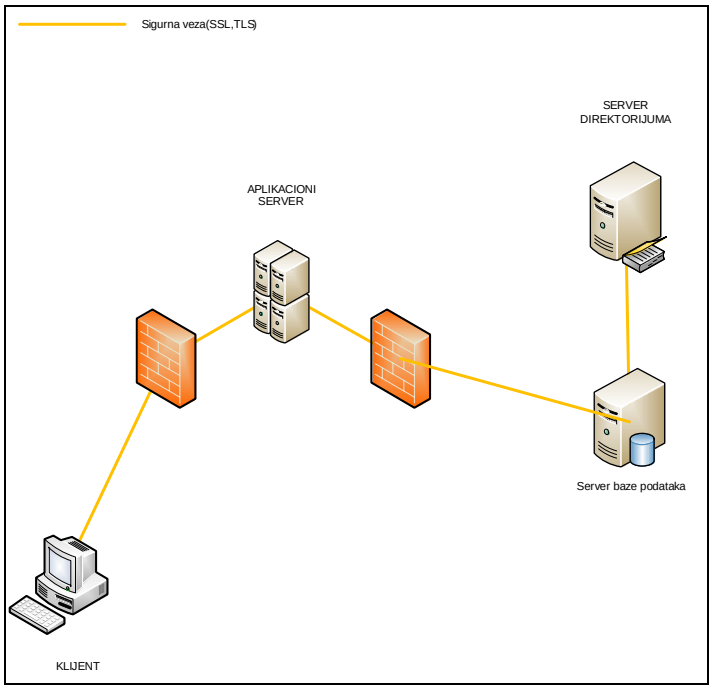
Pre nego što bi neko pokušao da promeni tabelu, baza podataka bi proverila sigurnosni dnevnik. Ukoliko ne postoje prava za promenu, izmena bi bila odbijena i pokušaj bi bio upisan u dnevnik (log) baze podatka. Kako bi ovaj sistem radio potrebno je kreirati i proceduru koja bi bila u mogućnosti da isključi šifrovanje nad određenom kolonom tabele:

```
ALTER TABLE KUPAC
MODIFY BROJ_KARTICE DROP ENCRYPT;
```

Ovde je potrebno razmotriti još jedan slučaj. Slučaj kada se promeni kolona u tabeli, bilo da je bila šifrovana pa je šifrovanje ukinuto, bilo da nije bila šifrovana pa je šifrovanje postavljeno. Dva su moguća rešenja, podaci koji su u tabeli ostaju neporomenjeni (šifrovani ili nešifrovani) i podaci u tabeli se menjaju. U ovom modelu ćemo primeniti slučaj kada se stari podaci menjaju, jer to smanjuje kompleksnost u vođenju sigurnosnog dnevnika.

4.3.1. Šifrovanje sa sistemom javnih ključeva

Da bi uspešno koristili model šifrovanja koristeći sistem javnih ključeva moramo pretpostaviti da već postoji server koji pruža uslugu direktorijuma [9](na primer LDAP[10] server) kojeg sistem za upravljanje bazom podataka može kontaktirati po potrebi. U ovom slučaju baza podataka bi kontaktirala server direktorijuma kako bi pribavila sertifikat[11] za određenog korisnika. Zajedno sa javnim ključem korisnika u direktorijumu se može nalaziti i privatni ključ korisnika koji je šifrovan lozinkom korinika, ali je bolja praksa da privatni ključ korisnika bude na smart kartici i uz korisnika.



Slika 4 – Model zaštite podataka u bazi - potrebni serveri

[9] Prevod od poznatijeg termina na engleskom jeziku - Directory service. Kao što mu samo ime kaže pruža uslugu centralnog direktorijuma (imenika) sa podacima o korisnicima. Među podacima se može naći i korisnički sertifikat.

[10] LDAP je skraćenica od Lightweight Directory Access Protocol. LDAP je protokol koji koristimo kako bi vršili upite i menjali podatke na serveru za usluge direktorijuma.

[11] Sertifikat je uobičajen naziv sa sertifikat o posedovanju i validnosti javnog ključa, kojim telo kojem se veruje, a nezavisno je od učesnika u komunikaciji, potvrđuje ispravnost javnog ključa.

Na primer, korisnik Marko poseduje sertifikat $CERT_A$ u serveru za direktorijum. Taj sertifikat potvrđuje verodostojnost javnog ključa PK_A, i u isto vreme PK_A ima odgovarajući privatni ključ SK_A. U serveru direktorijuma bi bio upisan ovakav skup podataka:

(marko, $CERT_A$)

Kada Marko želi da izvrši sledeći upit:

```
INSERT INTO KUPAC VALUES
(100,"PERA PERIĆ","1112221112")
;
```

Server baze podataka će izvršiti upit nad serverom za direktorijum i pribaviti sertifikat $CERT_A$ a samim tim i javni ključ PK_A. PK_A će biti upotrebljen kako bi se zaštitio radni ključ K i time obazbedilo da samo korisnik koji ima privatni ključ SK_A može da dešifruje radni ključ K i samim tim da jedini pročita podatke koji su zaštićeni ključem K. Radni ključ K generiše baza podataka i može se čuvati u bazi u šifrovanom obliku ili se može snimiti u server direktorijuma. Zapis bi izgledao ovako:

(marko, kupac, broj_kartice, $E(PK_A, K)$)

Kada Marko želi da šifruje novu kolonu, možemo primeniti jedan od dva slučaja. Prvi slučaj je da baza koristi isti ključ K koji je korišćen za šifrovanje prve kolone ili da generiše novi ključ K_1, njega šifruje Markovim javnim ključem i takvog ga snimi u server direktorijuma.

Kada Marko želi da pristupi podacima koji su šifrovani potrebno je da izvrši sledeću proceduru:

```
SELECT BROJ_KARTICE
FROM KUPAC
WHERE ID_KUPCA = 100 PRIVATE_KEY SK_A ;
```

Drugim rečima, Marko mora da eksplicitno navede svoj privatni ključ kako bi pristupio podacima. Sistem za upravljanje bazom podataka će prvo dohvatiti šifrovani radni ključ sa servera za direktorijum. Zatim će iskoristiti privatni ključ koji je Marko naveo u upitu kao bi dešifrovao radni ključ K, onda će iskoristiti radni ključ K kako bi dešifrovao podatke i vratio Marku rezultat.

Slična procedura se primenjuje i kada Marko želi da upiše podatak u bazu:

```
INSERT INTO KUPAC VALUES
(200, „ZORAN JOVANOVIC“, „123456789“) PRIVATE_KEY SK_A ;
```

4.3.2. Šifrovanje na nivou grupe korisnika

Podaci u bazi podataka često, ako ne i uvek, treba da budu dostupni većem broju korisnika. Do sada smo u radu bili skoncentrisani na zaštitu podataka za pojedinačne korisnike a u nastavku ćemo razmotriti model gde grupa korisnika treba da pristupi istim šifrovanim podacima.

Recimo, da Marko želi da kreira tabelu i u njoj zaštiti određenu kolonu. Sada je potrebno da i Ana pristupi i pročita te podatke. Najlakši način bi bio da Marko svoju lozinku, ili privatni ključ da Ani na korišćenje, ali se to kosi sa osnovnom sigurnosnom politikom koja zahteva da samo korisnik zna svoju lozinku.

Da bi omogućili šifrovanje, i dešifrovanje na nivou grupe, moramo generalizovati upotrebu javnih ključeva.

Sada, kada Marko kreira tabelu, mora prvo da navede listu korinika kojima će omogućiti da šifruju i dešifruju podatke. To će uraditi kroz novu proceduru koja bi izgleda ovako:

```
CREATE TABLE KUPAC
(ID INTEGER PRIMARY KEY
IME VARCHAR(30),
BROJ_KARTICE VARCHAR(16) ENCRYPT USER_LIST ANA,MIKA UPDATE 1
);
```

USER_LIST je lista korisnika koji mogu da pristupe šifrovanim podacima. Potrebno je da proširimo sigurnosni katalog, tako da on sada sadrži informacije o grupi korinika. Novi sigurnosni katalog bi sada sadržao ista polja, samo što bi za svakog korisnika postojao po jedan red. Sigurnosni katalog, za predhodni primer, bi izgledao ovako:

(marko, kupac, broj_kartice,1,marko,1)
(marko, kupac, broj_kartice,1,ana,1)
(marko, kupac, broj_kartice,1,mika,1)

Sada, kada Marko kreira ovu tabelu, radni ključ K će biti šifrovan tri puta i to sa Markovim, Aninim i Mikinim javnim ključevima. U server direktorijuma će biti upisani sledeći podaci:

- U delu za definisanje korisnika :
  (marko, $CERT_A$)
  (ana, $CERT_B$)
  (mika, $CERT_C$)

- U delu za čuvanje radnih ključeva:
  (marko, kupac, broj_kartice, $E(PK_A, K)$)
  (ana, kupac, broj_kartice, $E(PK_B, K)$)
  (mika, kupac, broj_kartice, $E(PK_C, K)$)

Kada Ana želi da pristupi zaštićenim podacima izvršiće sledeći upit:

```
SELECT BROJ_KARTICE
FROM KUPAC
WHERE ID_KUPCA = 100 PRIVATE_KEY SKB ;
```

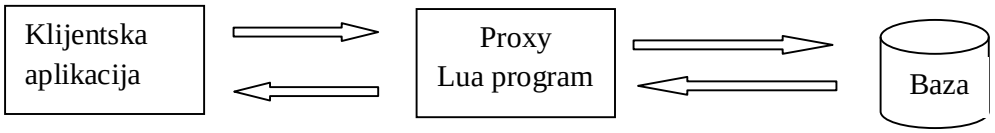
Baza podataka će proveriti da li se Ana nalazi u sigurnosnom katalogu za tabelu kupac i kolonu broj_kartice, ako jeste baza će izvršiti upit nad serverom direktorijuma i dobaviti radni ključ K koji je šifrovan sa Aninim javnim ključem $E(PK_B,K)$. Iskoristiće Anin privatni ključ koji je ona navela u upitu SK_B i dešifrovati radni ključ K. Sada će radni ključ biti upotrebljen kako bi se dešifrovali podaci. Podaci će onda biti vraćeni Ani.

Učestalo čitanje sa servera direktorijuma bi verovatno prilično usporilo izvršavanje upita nad bazom. Dobra ideja je da se zaštićeni radni ključevi drže u privremenoj memoriji (cache memoriji) radi poboljšanja performansi.

4.4. Šifrovanje podataka u MySQL bazi

U ovom poglavlju ćemo prikazati kako se opisani model primenjuje u realnom okruženju. Svi alati koji su korišćeni su otvorenog koda i besplatni i čine složeni sistem kojim se štiti baza podataka. Dok budemo detaljno opisivali postupak primene teorijskog modela, koristićemo i praktičan primer kako bi rad sistema bio jasniji.

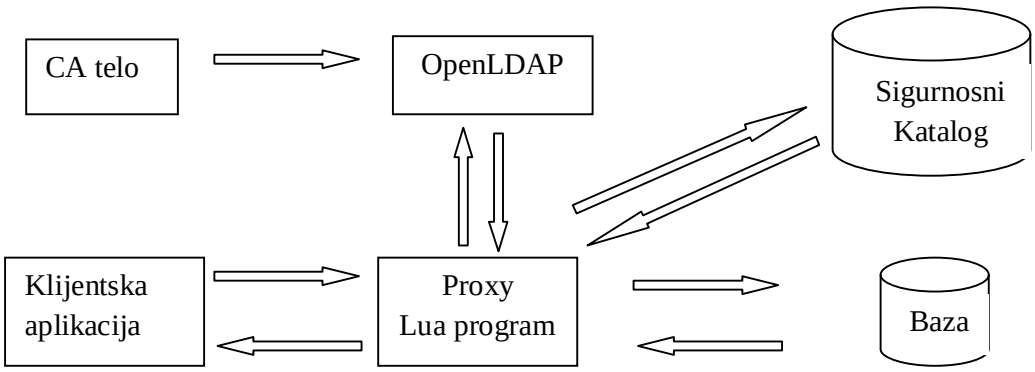
Baza podataka koju ćemo zaštititi je MySQL. Osnovni deo mehanizma koji će biti opisan je aplikacija MySQL-Proxy. MySQL-Proxy je pozicioniran između klijenta i baze, tako da presreće upite i po potrebi ih menja. Sve promene, u upitima ili u rezultatima upita, se vrše preko programa koji je napisan u programskom jeziku Lua.



Slika 5 – Uprošćena šema sistema za zaštitu baze podataka

4.4.1. Dodatne komponente

Kako bi sistem uspešno podržao opisani teorijski model, u opisano rešenje moramo uvesti još i server direktorijuma, CA telo i sigurnosni katalog.



Slika 6 – Model sistema za zaštitu podataka u bazama

4.4.2. Sigurnosni katalog

Sigurnosni katalog se sastoji iz dve tabele koje se nalaze na posebnoj bazi podataka i čija je izmena moguća samo iz programa Proxy aplikacije.

	Table
☐	ENC_COLUMNS
☐	ENC_KEYS

Slika 7 – Tabele u Sigurnosnom katalogu

Tabela ENC_COLUMNS sadrži imena kolona koje su šifrovane, kao i imena tabela kojima pripadaju, informaciju o korisniku koji je kreirao tabelu, pravima nad kolonom i informaciju o poslednjoj promeni. Tabela ima složeni primarmi ključ koji se sastoji od kolona **table_name** i **column_name.** Sva polja u ovoj tabeli su tekstualna.

	Field	Type
☐	table_name	varchar(30)
☐	column_name	varchar(30)
☐	creator	varchar(30)
☐	permition	varchar(10)
☐	modified_by	varchar(30)

Slika 8 – Struktura tabele ENC_COLUMNS

Tabela ENC_KEYS sadrži sesijske ključeve koji su šifrovani sa javnim ključevima korisnika koji imaju pravo šifrovanjai i dešifrovanja nad određenim tabelama. Tabela ima složeni primarmi ključ koji se sastoji od kolona **table_name**, **column_name** i **username.** Sva polja u ovoj tabeli su tekstualna.

	Field	Type
☐	table_name	varchar(30)
☐	column_name	varchar(30)
☐	username	varchar(30)
☐	enc_key	varchar(255)

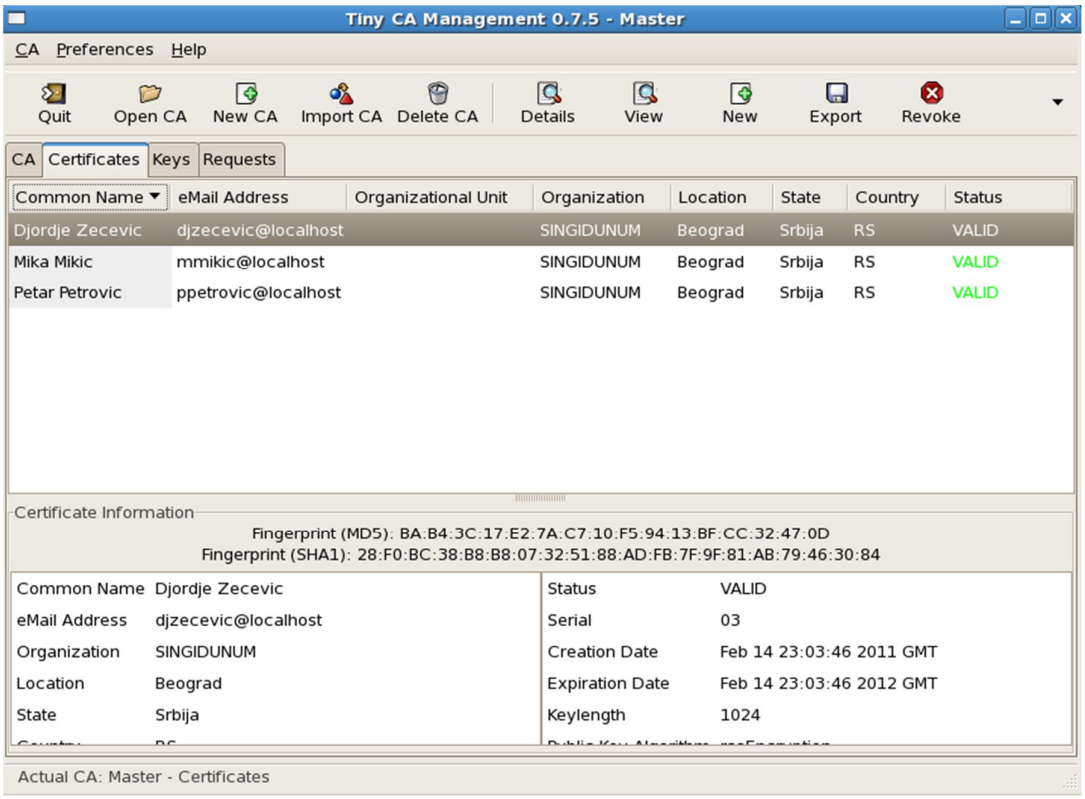
Slika 9 – Struktura tabele ENC_KEYS

4.4.3. Sertifikaciono telo

Uloga sertifakionog tela u ovom sistemu je jasna i važna. Sertifikaciono telo izdaje sertifikate i garantuje za njihov sadržaj.

Za sertifikaciono telo koristimo programski paket OpenSSL i to njegovu komponentu CA. OpenSSL je programski paket za kriptografiju, otvorenog koda, koji podržava sve poznate algoritme za simetrično i asimetrično šifrovanje, hešing[12] algoritme, rad sa sertifikatima, itd.

Radi lakšeg rada sa sertifikatima, koristićemo i dodatni program TinyCA. TinyCA je grafički alat za konfiguraciju CA komponente OpenSSL-a.

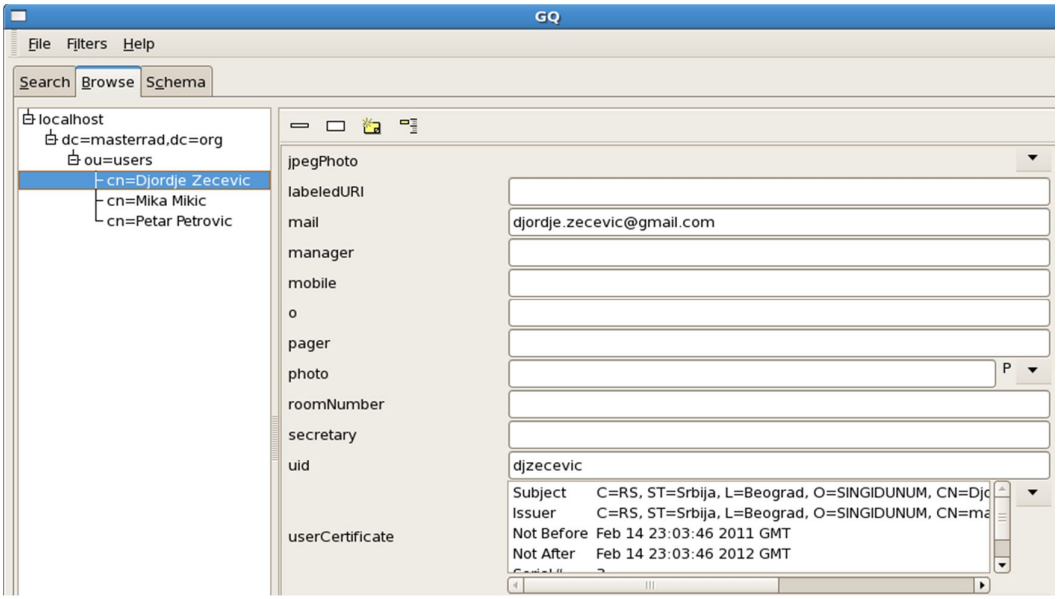


Slika 10 – Tiny CA – grafičko okruženje za CA komponentu OpenSSL-a

[12] Hašing je reč engleskog porekla koja nema pravi prevod na srpski jezik. Radi se o jednosmernoj funkcji koja se koristi za zaštitu integriteta podataka. Za svaki ulazni parameter, bilo koje veličine, funkcija daje jedinstveni izlaz fiksne dužine.

4.4.4. Server direktorijuma

Server direktorijuma je **OpenLDAP.** To je program koji razvija neprofitna firma The OpenLDAP Foundation još 1998. godine. Njegova uloga u ovom rešenju je da korsnički sertifikati, koji su kreirani od strane sertifikacionog tela, budu dostupni u svakom trenutku.



Slika 11 – GQ, grafičko okruženje za pristup OpenLDAP serveru

4.4.5. CREATE TABLE upit

Pri kreiranju tabele, potrebno je da korisnik može da navede kolone koje će biti šifrovane kao i listu korisnika koji će imati pravo da čitaju i upisuju šifrovane podatke. Ovi podaci se kasnije mogu izmeniti Alter table upitom.

Kada klijent pošalje upit za kreiranje tabele, Proxy server presreće upit. Pošto upit ne odgovara standardnoj SQL sintaksi (ima nove ključne reči), proxy će iz upita izvući sledeće informacije:

- koje kolone treba da budu šifrovane;
- koji korisnici mogu da čitaju ili menjaju kolone.

Na sledećem primeru ćemo bolje upoznati proces:

```
CREATE TABLE SIM(
ID INT,
IME VARCHAR(30),
BROJ VARCHAR(30),
STANJE INT,
PIN VARCHAR(30) ENCRYPT USER_LIST [MMIKIC PPETROVIC] UPDATE 1,
PUK VARCHAR(30) ENCRYPT UPDATE 2
);
```

Iz ovog upita Proxy će pronaći ključne reči **ENCRYPT** i **UPDATE,** i pomoću njih zaključiti da je potrebno šifrovati kolone PIN i PUK, i da su prava 1, odnosno 2.

Na osnovu ovih informacija Proxy će izvršiti sledeći niz operacija:

- Upit će izmeniti tako da on odgovara SQL sintaksi;

```
CREATE TABLE SIM(
ID INT,
IME VARCHAR(30),
BROJ VARCHAR(30),
STANJE INT,
PIN VARCHAR(30),
PUK VARCHAR(30),
);
```

- Za svaku kolonu koja je šifrovana generisaće ključ za simetrično šifrovanje (sesijski ključ);

- Za svakog korisnika kome je pravo dodeljeno će dohvatiti sertifikat sa njegovim javnim ključem;

- Sa svakim javnim kjučem će šifrovati sesijski ključ i takvog ga smestiti u sigurnosni katalog;

	table_name	column_name	username	enc_key
	sim	puk	djzecevic	KKRUYK9fQVpg3EWL9ycU/bh/jOqOtB8V40AlPfy4ec+uTBlRaG...
	sim	pin	djzecevic	HGnPb2/kl3AmwRsszDAgs1K62Q/1THtLRg/OVLiDKUE84cKOHA...
	sim	pin	mmikic	nW3WDoQH512Ul0S88dbF37o9+LzhPUtRM0M6FeoWLU6nuxZEi4...
	sim	pin	ppetrovic	qzDOVMtS2k6zzTPmPuBcxQtARpbWjJpQE1NYGX52Xz6rcyi8Oj...

Slika 12 – ENC_KEYS tabela u sigurnosom katalogu

- Svaku kolonu koja treba da bude šifrovana će označiti u sigurnosnom katalogu;

	table_name	column_name	creator	permition	modified_by
	sim	puk	djzecevic	2	djzecevic
	sim	pin	djzecevic	1	djzecevic

Slika 13 – ENC_COLUMNS tabela u sigurnosnom katalogu

- Izvršiće upit kreiranja tabele u bazi podataka;

	Field	Type
	id	int(11)
	ime	varchar(30)
	broj	varchar(30)
	stanje	int(11)
	pin	varchar(30)
	puk	varchar(30)

Slika 14 – Tabela SIM u bazi podataka

Nakon izvršavanja ove procedure, u bazi podataka će postojati željena tabela, a u sigurnosnom katalogu će postojati zapisi potRebni za njeno šifrovanje.

4.4.6. INSERT INTO upit

Pri ubacivanju podataka u tabelu, potrebno je da Proxy konsultuje Sigurnosni katalog i da utvrdi koje kolone treba da šifruje i da bazi podataka prosledi izmenjen originalni upit:

```
INSERT INTO SIM VALUES(
1,"DJORDJE ZECEVIC","064-123-1234",10000,"2222","113456789"
) KEY "SK";
```

Na osnovu dobijenih informacija iz upita Proxy izvršava sledeće korake:

- Konsultuje Sigurnosni katalog i proverava da li korisnik ima pravo da šifruje podatke u koloni, ukoliko nema, akcija se prekida;
- Konsultuje Sigurnosni katalog i saznaje koje kolone treba šifrovati;
- Uzima, za kolone koje su šifrovane, iz Sigurnosnog kataloga, ključeve za simetrično šifrovanje;
- Upit modifikuje tako što umesto imena kolone dodaje funkciju **aes_encrypt** i kao argumente prosleđuje ime kolone i ključ za simetrično šifrovanje;

```
INSERT INTO KORISNIK VALUES (
1,"DJORDJE ZECEVIC","111-222",
AES_ENCRYPT ("1111",'JSUMKMTNJIZQ17O/KA2LBW==')
)
```

- Ovako izmenjen upit se onda prosleđuje bazi na izvršavanje.

Nakon izvršavanja tabela u bazi izgleda ovako:

←T→	id	ime	broj	stanje	pin	puk
☐ ✎ ✕	1	Djordje Zecevic	064-123-1234	10000	Í□ñ¿ñPhäØ□Œ_Ùåê^	ˆ,oUðÎ□ü□n□□Z·`

Slika 16 – Zaštićeni podaci u bazi

4.4.7. SELECT upit

Procedura izvršavanja SELECT upita je slična izvršavanju INSERT INTO upita. Proxy prvo pronalazi kolone koje su šifrovane pa ih dešifruje sa ključevima iz Sigurnosnog kataloga. SELECT upit možemo posmatrati iz više uglova i u svakom pronaći različite slučajeve koje je potrebno obraditi.

Ako SELECT upit posmatramo iz ugla prava[13] korisnika nad tabelom, korisniku koji nema prava da šifruje će biti dozvoljeno da podatke pročita, ali će podatci koje dobije biti u šifrovanom obliku.

Ako posmatramo SELECT upit iz ugla izbora kolona, možemo pronaći dva slučaja. Slučaj kada korisnik navodi kolone koje mu trebaju i slučaj kada korisnik navodi specijalni znak *[14].

```
SELECT IME, BROJ, STANJE, PIN, PUK
FROM SIM KEY "SK";
```

```
SELECT *
FROM SIM KEY "SK";
```

Procedura za koju izvršava Proxy sastoji se iz sledećih koraka:

- Proxy presreće upit i ispituje kojem od dva tipa pripada;
- Ukoliko je upit sa specijalnim znakom, Proxy će znak zameniti sa listom kolona;
- Iz liste kolona pronalazi one koje su šifrovane;
- Iz Sigurnosnog kataloga dohvata ključeve za simetrično šifrovanje koji su šifrovani sa korisničkim javnim ključem;
- Koristi korisnički privatni ključ, koji je naveden u upitu, kako bi dešifrovao ključeve za simetrično šifrovanje;
- Upit menja tako da imena kolona koje su šifrovane, zemenjuje sa funkcijom za dešifrovanje, koja za argumente ima ime kolone i ključ sa kojim će rezultat biti dešifrovan;
- Ovako izmenjen upit se šalje bazi na izvršavanje.

[13] U ovom slučaju mislimo na prava za šifrovanje podataka, a ne na korisnička prava da izvrši SELECT upit.

[14] Znak * u standardnom SQL jeziku zamenjuje imena svih kolona u tabli.

Na primerima ćemo pokazati kako Proxy menja upite dok oni putuju od korisnika do baze podataka. Pretpostavimo da korisnik zadaje upit:

```
SELECT *
FROM SIM KEY "SK";
```

On će biti analiziran, izmenjen i takav poslat bazi na izvršavanje. Ovako izgleda izmenjen upit:

```
SELECT ID,IME,BROJ,STANJE,AES_DECRYPT(PIN,'8TVDRNDYZPR6PJIAQRH5IA==') AS
PIN, AES_DECRYPT(PUK,'ESROJXC1H91PUALGMR7MLW==') AS PUK
FROM SIM
```

Drugi primer pokazuje slučaj kada korisnik želi samo određene kolone:

```
SELECT IME, BROJ, STANJE, PIN, PUK
FROM SIM KEY "SK";
```

Nakon analiziranja i izmene upit se prosleđuje bazi u sledećem obliku:

```
SELECT IME,BROJ,STANJE,AES_DECRYPT(PIN,'8TVDRNDYZPR6PJIAQRH5IA==') AS
PIN,AES_DECRYPT(PUK,'ESROJXC1H91PUALGMR7MLW==') AS PUK
FROM SIM
```

4.4.8. ALTER TABLE upit

Da bi se promenila osobina kolone iz šifrovane u nešifrovanu ili obrunuto, koristi se specijalni oblik upita ALTER TABLE. ALTER TABLE upit se mora zadati u obliku:

```
ALTER TABLE IME_TABELE
SET IME_KOLONE [DROP] ENCRYPT KEY “SK”;
```

Ako se navodi ključna reč DROP, onda se šifrovanje isključuje sa kolone, a ukoliko se ne navodi, šifrovanje se uključuje za kolonu.

Procedura uključivanja i isključivanja šifrovanja je slična i sastoji se iz sledećih koraka:

- Proxy proverava da li korisnik ima pravo da izmeni osobinu šifrovanja za kolonu iz upita. Ukoliko nema, akcija se prekida;
- Proxy presreće upit i analizira koja kolona i iz koje tabele će biti izmenjena;

- Proxy kreira privremenu tabelu i u nju kopira tabelu iz koje se menja kolona;
- Briše sve redove iz tabele koja se menja;
- Ukoliko se uključuje šifrovanje za kolonu, u Sigurnosni katalog, Proxy upisuje potrebne podatke (izvršava procedure kao kod kreiranja tabele sa šifrovanim kolonama). Ukoliko se isključuje šifrovanje, iz Sigurnosnog kataloga se brišu redovi vezani za ovu kolonu;
- Kopira privremenu tabelu u originalnu tabelu, ali tako da se kolona koja predmet upita se šifruje ili dešifruje prilikom kopiranja;
- Privremena tabela se nakon toga briše.

Evo primera gde se sa kolone PUK iz tabele SIM uklanja šiforvanje:

```
ALTER TABLE SIM
SET PUK DROP ENCRYPT KEY "SK";
```

Tabela je, pre izvršavanja upita, izgledala ovako:

←T→	id	ime	broj	stanje	pin	puk
	1	Djordje Zecevic	064-123-1234	10000	ĂHbgÛ.ª g• ½¨¶É	¥1ì¿s,ì(Q¶ù ò
	2	Petar Petrovic	064-123-1234	10000	Kswf—v8Ëj×¬ ± h	b QÏ¶T¯^ ˜œAlÀèe
	3	Mika Mikic	064-123-1234	10000	!º¡—¶ëôÄ"« MHØ«	ÝB ß˜ýËjg9 ˆ6

Slika 16 – Sadržaj tabele pre izvršavanja ALTER TABLE upita

Nakon izvršavanja upita izgleda ovako:

←T→	id	ime	broj	stanje	pin	puk
	1	Djordje Zecevic	064-123-1234	10000	ĂHbgÛ.ª g• ½¨¶É	113456789
	2	Petar Petrovic	064-123-1234	10000	Kswf—v8Ëj×¬ ± h	122456789
	3	Mika Mikic	064-123-1234	10000	!º¡—¶ëôÄ"« MHØ«	123456799

Slika 17 – Sadržaj tabele nakon izvršavanja ALTER TABLE upita

4.4.9. DELETE FROM upit

DELETE FROM upit je jednak standardnom delete upitu, nema specijanih reči i bez izmena se izvršava na serveru baze podataka.

```
DELETE FROM SIM
;
```

4.4.10.DROP TABLE upit

Brisanje tabele je najjednostavniji upit u ovom praktičnom primeru. Pri brisanju tabele proxy će iz upita izvući ime tabele koja se briše, a onda će uraditi sledeće korake:

- Brisanje zapisa iz tabela Sigurnosnog kataloga;
- Izvršavanje samo upita nad bazom, tj. brisanje same tabele u bazi.

```
DROP TABLE SIM
;
```

4.5. Klijentska aplikacija

Iako se ovaj rad ne bavi klijentskim delom sistema za zaštitu, navešćemo preduslove koje treba da ispunjava klijentska aplikacija.

Aplikacija mora biti odgovorna za uspostavljanje sigurne veze sa bazom podataka i za siguran pristup korisničkim ključevima. Za sigurnu vezu se mogu koristiti neki od standardnih načina kao što su IPSEC ili SSL, dok se za čuvanje i pristup ključevima mogu koristiti Smart kartice ili USB Token uređaji.

5. Zaključak i primenljivost rezultata

U ovom radu opisali smo značaj zaštite podataka koji su smešteni u sistemima za upravljanje bazma podataka. Opisali smo trenutne sigurnosne mehanizme, kao i njihove slabe tačke koje dovode do sigurnosnih rupa. Navedeni su i načini na koje se slabosti baza podataka mogu zloupotrebiti. Takođe istaknuto je koliko se malo vodi računa o zaštiti i koje su opasnosti takve politike.

Kako bi prevazišli ove propuste, predstavljen je model zaštite koji se može, uz malo napora, primeniti u većini popularnih sistema za upravljanje bazama podataka. Model je zasnovan na primeni šifrovanja, kao proverene i standardne mere zaštite podataka. Vodili smo računa o činjenici da su baze podataka već dugo u upotrebi i da model mora biti takav da se može primeniti bez većih izmena u postojećim sistemima. Kako bi to postigli, model se oslanja sa standardne servise kao što je server direktorijuma koji se može lako povezati sa postojećim sistemima za upravljanje bazama podataka.

U trećem delu rada, opisana je i jedna praktična primena modela koji je opisan u radu. Model je izveden koristeći alate otvorenog koda i usmeren je ka zaštiti MySQL baze podataka.

5.1. Primenljivost rezultata

Primenljivost rezultata opisanog sistema ćemo proveriti kroz dva testna scenarija, koji u sebi sadrže sve operacije koje su definisane.

Scenario 1

Scenario jedan se sastoji iz niza sledećih upita:

```
DROP TABLE KORISNIK;

CREATE TABLE KORISNIK (
ID INT, IME VARCHAR(30),
BROJ_KARTICE VARCHAR(30),
PIN VARCHAR(30) ENCRYPT UPDATE 1
);

INSERT INTO KORISNIK VALUES
(1, "DJORDJE ZECEVIC", "111-222","1111") KEY SK;

INSERT INTO KORISNIK VALUES
(2, "PETAR PETROVIC", "111-222","1234") KEY SK;

INSERT INTO KORISNIK VALUES
(3, "MIKA MIKIC", "111-222","2222") KEY SK;

SELECT * FROM KORISNIK KEY "SK";

SELECT IME, BROJ_KARTICE, PIN FROM KORISNIK KEY "SK";

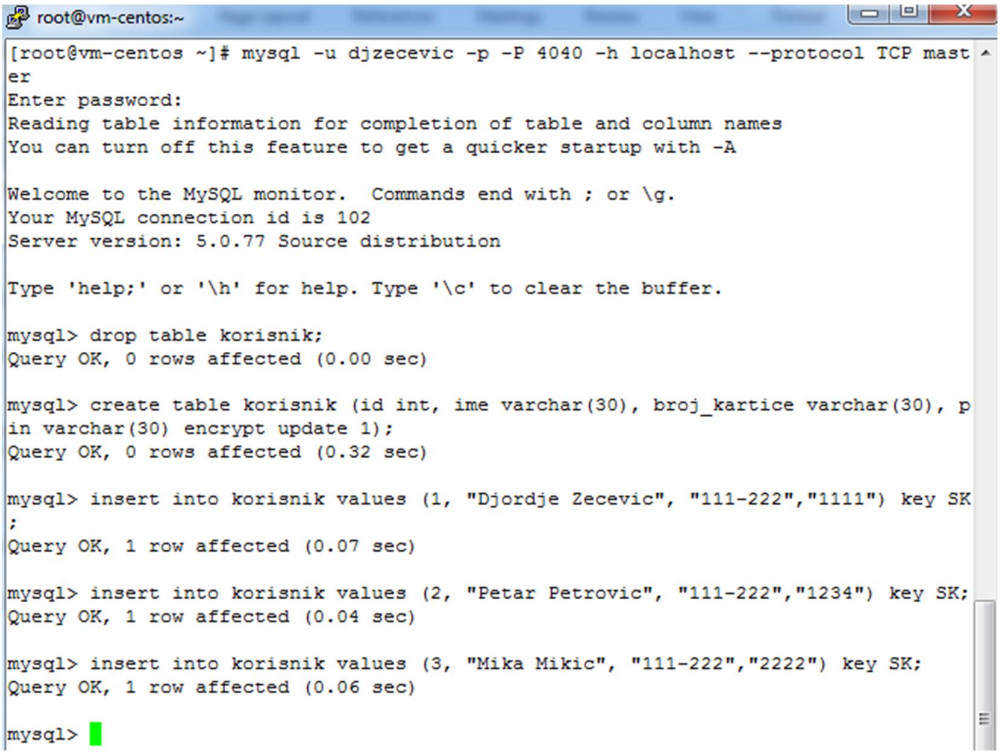
ALTER TABLE KORISNIK SET BROJ_KARTICE ENCRYPT KEY "SK";

SELECT * FROM KORISNIK KEY "SK";

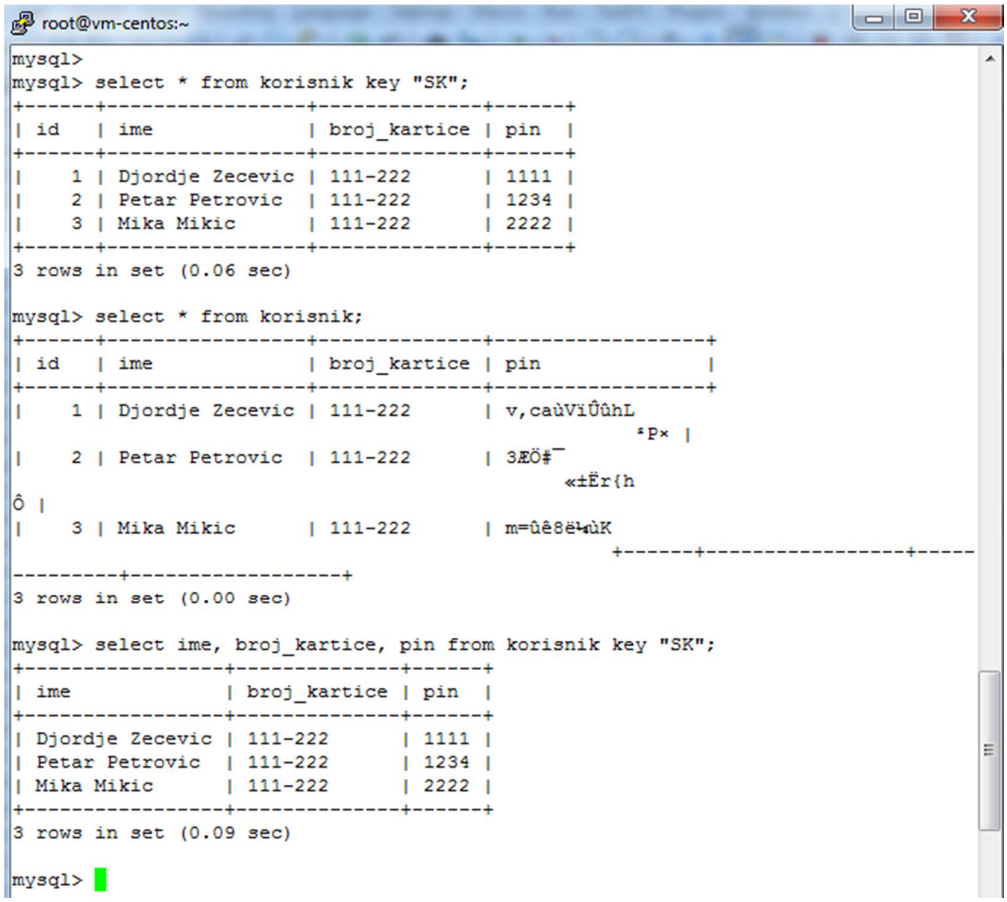
DELETE FROM KORISNIK;

DROP TABLE KORISNIK;
```

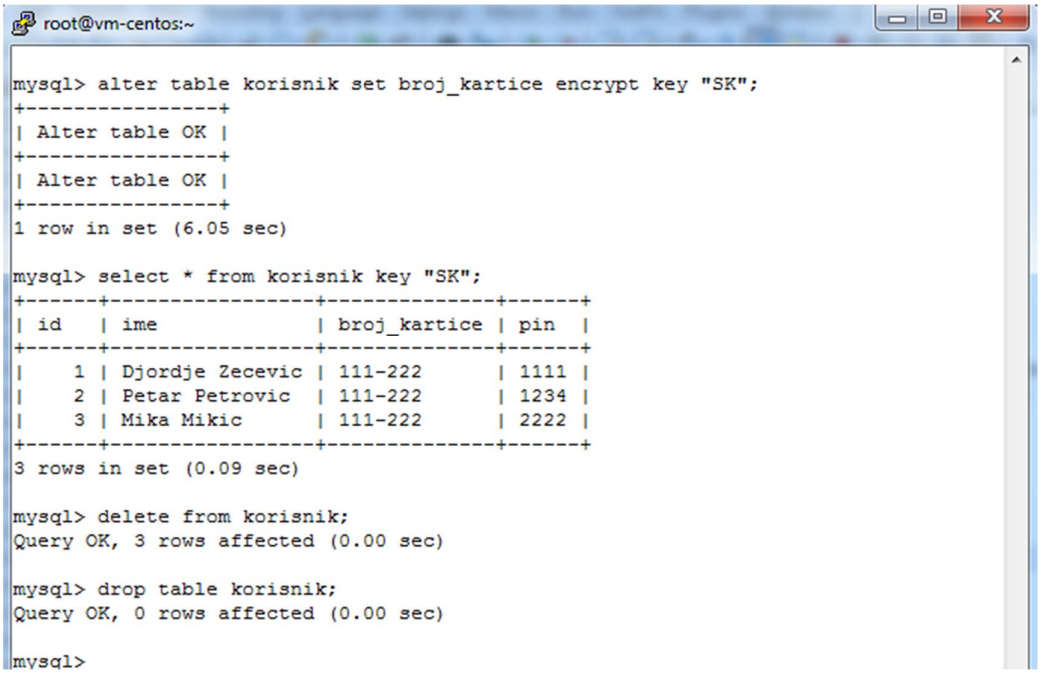
Rezultati scenarija 1:



Slika 18 – Rezultati scenarija 1



Slika 19 – Rezultati scenarija 1



Slika 20 – Rezultati scenarija 1

Scenario 2

Scenario dva se sastoji iz niza sledećih upita:

```
DROP TABLE SIM;

CREATE TABLE SIM(
ID INT, IME VARCHAR(30),
BROJ VARCHAR(30),
STANJE INT,
PIN VARCHAR(30) ENCRYPT USER_LIST [MMIKIC PPETROVIC] UPDATE 1,
PUK VARCHAR(30) ENCRYPT UPDATE 2
);

INSERT INTO SIM
VALUES(1,"DJORDJE ZECEVIC","064-123-1234",10000,"2222","113456789") KEY SK;

INSERT INTO SIM
VALUES(2,"PETAR PETROVIC","064-123-1234",10000,"2222","122456789") KEY SK;

INSERT INTO SIM
VALUES(3,"MIKA MIKIC","064-123-1234",10000,"2222","123456799") KEY SK;

SELECT * FROM SIM KEY "SK";

SELECT IME, BROJ, STANJE, PIN, PUK FROM SIM KEY "SK";

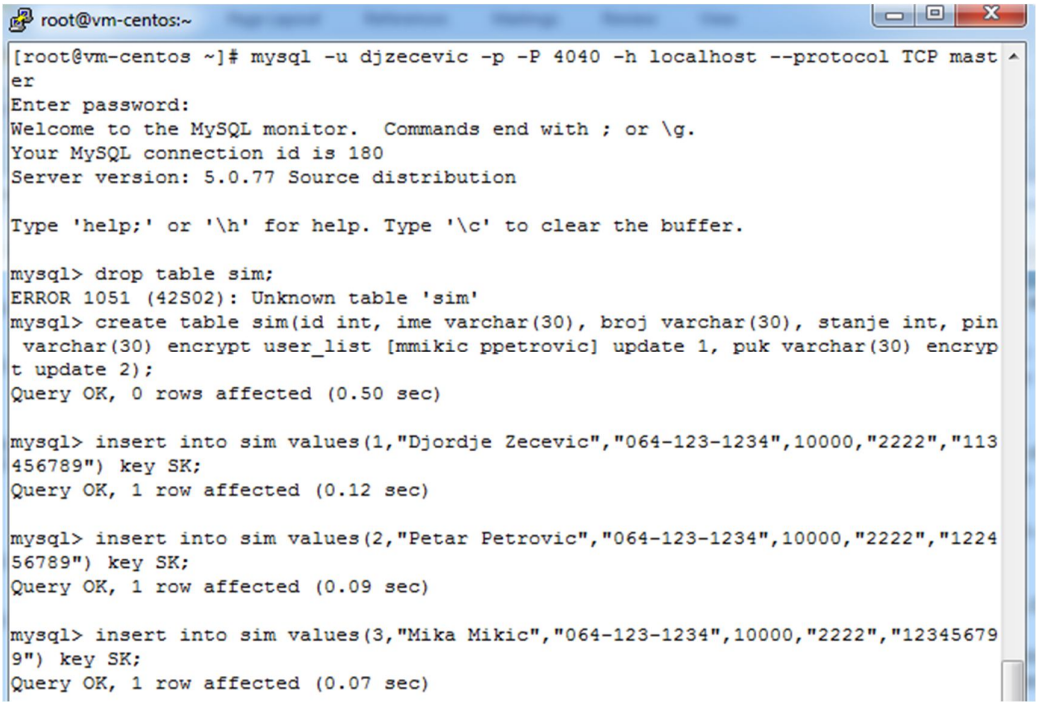
ALTER TABLE SIM SET PUK DROP ENCRYPT KEY "SK";

SELECT * FROM SIM;

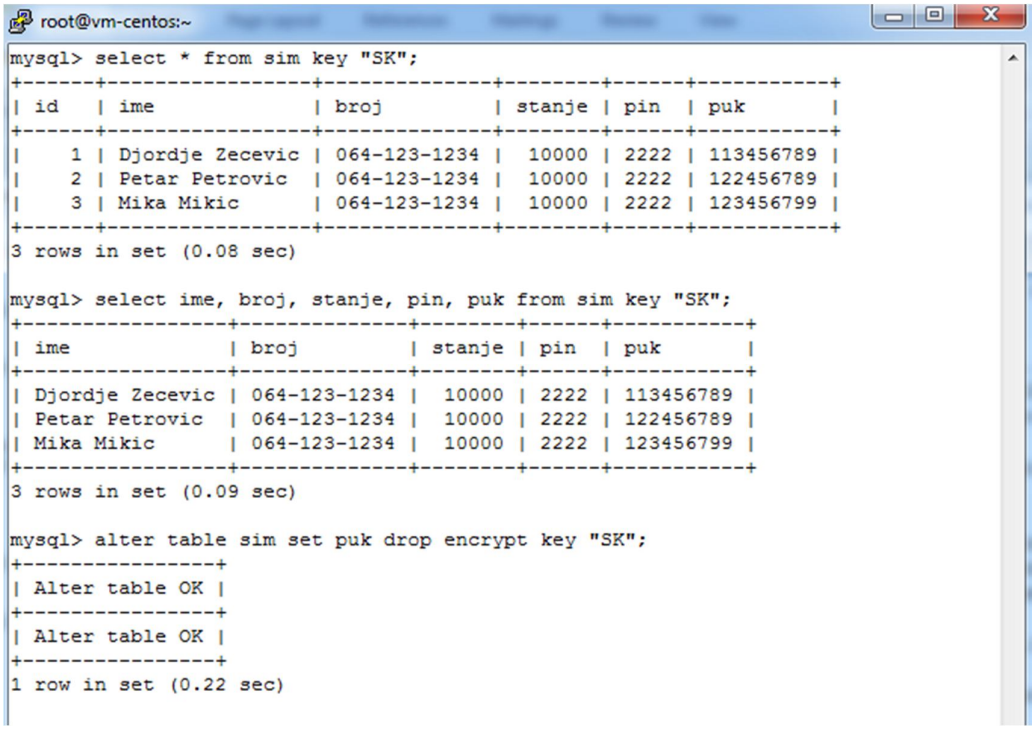
DELETE FROM SIM;

DROP TABLE SIM;
```

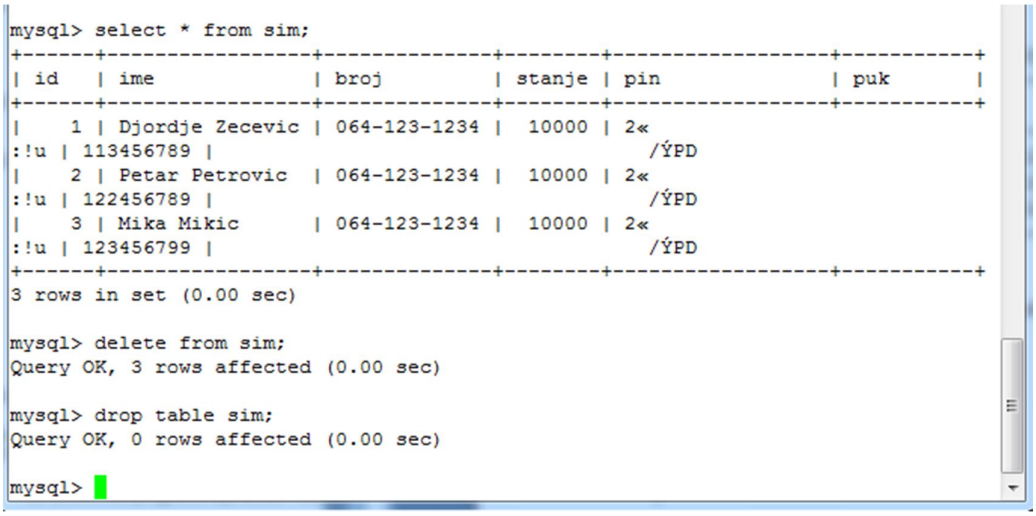
Rezultati scenarija 2:



Slika 21 – Rezultati scenarija 2



Slika 22 – Rezultati scenarija 2



Slika 23 - Rezultati scenarija 2

6. Literatura

[1] Pleskonjić Dragan, Maček Nemanja, Đorđević Borislav, Carić Marko: *Sigurnost u računarskim mrežama, Mikroknjiga*, 2007.

[2] CPNI Technical Note: *Understanding Database Security*, www.cpni.gov.uk/Docs/TN_Database_Security_Final.pdf, posećeno 30.08.2010.

[3] Tanenbaum Andrew: *Računarske mreže*, Mikroknjiga, 2005.

[4] Robert Manger: *Baze podataka*, Sveučilište u Zagrebu, Prirodoslovno matematički fakultet – Matematički odjel, 2008.

[5] *Securing Data at Rest: Developing a Database Encryption Strategy*, http://www.rsa.com/products/bsafe/whitepapers/DDES_WP_0702.pdf, posećeno 30.08.2010.

[6] *Database Encryption in SQL Server 2008 Enterprise Edition*, http://msdn.microsoft.com/en-us/library/cc278098%28v=sql.100%29.aspx#_Toc189384672, posećeno 30.08.2010.

[7] *OpenSSL Command-Line HOWTO*, http://www.madboa.com/geek/openssl/, posećeno 30.08.2010.

[8] *HOWTO: Creating your own CA with OpenSSL*, http://sandbox.rulemaker.net/ngps/m2/howto.ca.html, posećeno 30.08.2010.

[9] *Lua 5.1 Reference Manual*, http://www.lua.org/, posećeno 30.08.2010.

[10] *Getting started with MySQL Proxy*, http://dev.mysql.com/tech-resources/articles/proxy-gettingstarted.html, posećeno 30.08.2010.

[11] *Encryption and Compression Functions*, http://dev.mysql.com/doc/refman/5.5/en/encryption-functions.html, posećeno 30.08.2010.

[12] *OpenLDAP Software 2.4 Administrator's Guide* – www.ldap.org, posećeno 30.08.2010.

[13] *Publishing digital certificates with LDAP*, http://tldp.org/HOWTO/archived/LDAP-Implementation-HOWTO/certificates.html, posećeno 30.08.2010.

7. Dodaci

7.1. Korišćeni alati i software

[1] **Mysql** baza podataka verzije 5.0.77. Web adresa na kojoj se može pronaći više informacija je http://www.mysql.org.

[2] **OpenLDAP** server direktorijuma verzije 2.3.43. Web adresa na kojoj se može pronaći više informacija je http://www.openldap.org.

[3] **CA** server. OpenSSL – PKI server komponenta. Web adresa na kojoj se može pronaći više informacija je http://www.openssl.org.

[4] Apache2 - Server aplikacija (web server). Web adresa na kojoj se može pronaći više informacija je http://www.apache.org.

[5] **OpenSSL** verzije 0.9.8e. Komponente za rad sa simetričnim i asimetričnim algoritmima za šifrovanje. Web adresa na kojoj se može pronaći više informacija je http://www.openssl.org.

[6] **Lua** 5.1.4. Programski jezik za izmenu upita. Web adresa na kojoj se može pronaći veše informacija je http://www.lua.org.

[7] **Mysql-proxy** 0.8.1. Aplikacija koja stoji između baze podataka i korisnika. Web adresa na kojoj se može pronaći više informacija je http://dev.mysql.com/downloads/mysql-proxy.

[8] CentOS 5.5. Operativni sistem na kom je instaliran ceo rad. Web adresa na kojoj se može pronaći više informacija je http://www.centos.org

7.2. Izvorni kod programa koji koristi Proxy baze podataka

Izvorni kod je pisan u programskom jezuku Lua.

```
require ("luasql.mysql")
-- FUNCTIONS
-- Funkicija za logging
function log(msg)
      local log_file = '/var/log/mysql-proxy-app.log'
      local fh = io.open(log_file, "a+")
      fh:write( string.format("%s %6d -- %s \n",os.date('%Y-%m-%d
%H:%M:%S'), proxy.connection.server.thread_id, tostring(msg)))
      fh:flush()
end

-- RSAencrypt vraca putanju do datoteke sa sifrovanim kjucem
-- text je kljuc koji je potrebno sifrovati
function RSAencrypt(text,key)
      local f = io.popen("echo '".. text .. "' | openssl rsautl  -encrypt -
inkey " .. key .."  -certin  |  openssl enc -A -base64")
      return  f:read()
end
```

```
-- RSAdecrypt vraca kljuc K za simetricno sifrovanje
-- text je ime datoteke u kojoj se nalazi sifrovani kljuc
function RSAdecrypt(text,key)
      local f = io.popen("echo '".. text .. "' | openssl enc -A -base64 -d
| openssl rsautl -decrypt -inkey " .. key )
      return f:read()
end

-- dohvata korisnicki sertifikat koji sadrzi javni kljuc. Funkcija vraca
ime file-a u kom se nalazi sertifikat
function getLDAPKey(user)
      if(user==nil) then
            log("Prosledjen user nil u funkciju getldapkey")
            os.exit(1)
      end
      log("Ulazi u funkciju za dohvatanje sertifikata za korisnika " ..
user)
      local i = 0
      local filename = "/usr/local/masterrad/cert/" .. user ..
"/certificate.pem"
      log("Filename: " .. filename)
      --io.popen("umask u=rwx,g=rwx,o=rwx /tmp/mysql-cert/" .. user)
      --io.popen("rm -f ".. filename)
      --io.popen("touch " .. filename)
      io.popen("echo '' > " .. filename)
      local fh = io.open(filename, "w")
      local f = io.popen("ldapsearch -LLL -x -D
'cn=Manager,dc=masterrad,dc=org' -w admin '(uid=" .. user .. ")'
userCertificate")
      local rez = f:read()
      local rez = f:read()
      local rez = string.gsub(rez,"userCertificate;binary:: ","")
      fh:write("-----BEGIN CERTIFICATE-----\n")
      fh:write(rez)
      repeat
            local rez_komande = f:read();
            --log("kljuc red po red: " .. tostring(rez_komande))
            if (rez_komande) then
                  rez = "\n" .. string.gsub(rez_komande," ", "")
                  fh:write(rez)
            end
            i = i + 1
      until (i>25)
      fh:write("-----END CERTIFICATE-----\n")
      fh:flush()
      log("Izlazak iz funkcije za dohvatanje sertifikata...")
      return filename
end

-- generate128bit generise kljuc duzine 128 bita enkodovan u base64, koji
ce se koristiti za sifrovanje u bazi
function generate128bit()
      local f = io.popen("openssl rand  -base64 16") -- store the output in
a "file"
      local rez = f:read()
      return rez
end

function trim (s)
      return (string.gsub(s, "^%s*(.-)%s*$", "%1"))
```

```
end
 function printtable(tabela)
	local v = ""
	for i=1,#tabela do
		if(type(tabela[i])=="table") then
			print(#tabela[i])
			v = listToString(tabela[i])
		else
			v = tabela[i]
		end
		print (i .. " - " .. v)
	end
 end
-- iz "1,2,3,4" vraca niz {1,2,3,4}
function stringToList(lista)
	local rezultat = {}
	local i = 1
	local temp = ""
	if  (string.find(lista,",")) then
		repeat
			temp = trim(string.sub(lista,1,string.find(lista,",")-1))
			rezultat[i] = temp
			i = i + 1
			lista =
string.sub(lista,string.find(lista,",")+1,string.len(lista))
		until (string.match(lista,",") == nil)
	end
	upit = trim(lista)
	rezultat[i] = upit
	return rezultat
end

-- iz {1,2,3,4} vraca "1,2,3,4"
function listToString(lista)
	local rez = ""
	j = #lista
	i = 1
	if(j==1) then
		rez = lista[1]
	else
		repeat
			rez = rez .. lista[i] .. ","
			i = i +1
		until (j==i)
		rez = rez .. lista[i]
	end
	return rez
end

function numberOf (rec,deo)
	local i=0
	  while string.match(rec, deo) do
		i = i+1
		rec = string.gsub(rec,deo, '')
	  end
	  return i
end

function isSpecialQuery(query)
	local rez = false
	if string.match(query, "encrypt") or string.match(query, "key") then
```

```
            rez = true
      end
      return rez
end

-- proverava da li je upit tipa create table
function isCreateTable(query)
      local rez = false
      if string.match(string.upper(query), "CREATE TABLE") then
            rez = true
      end
      return rez
end

-- proverava da li je upit tipa drop table
function isDropTable(query)
      local rez = false
      if string.match(string.upper(query), "DROP TABLE") then
            rez = true
      end
      return rez
end
-- proverava da li je INSERT INTO upit
function isInsertInto(query)
      if string.match(string.lower(query), "insert into .+ key") then
            return true
      end
      return false
end

-- proverava da li je ALTER TABLE upit
function isAlterTable(query)
      if string.match(string.lower(query), "alter table .+ set .+ encrypt")
then
            return true
      end
      return false
end

-- proverava da li je SELECT upit
function isSelect(query)
      if string.match(string.lower(query), "select") then
            return true
      end
      return false
end
-- parse Select query
function parseSelect(query)
      local upit1 = ""
      local imeTabele  = getTableName(query)
      local k1,k2
      local kolone_sec={}
      local kolone = {}
      local sve_kolone = {}
      local se = {}
      local user = USER
      local schema = proxy.connection.client.default_db
      local new_query = ""
      local kol_sif, kol_nesif
      -- dohvatanje kolona koje treba da se sifruju u promenljivu
kolone_sec
```

```
	kolone_sec = getCryptCols(imeTabele)
	-- kreira se novi upit koji sadrzi funkciju za desifrovanje kolona
	local uslovi = ""
	uu,uslovi = string.find(query,'key.*%".+%"')
	uslovi = string.sub(query, uslovi+1, string.len(query))
	uslovi = trim(uslovi)
	kol_nesif = string.gsub(query,"select","")
	kol_nesif = trim(kol_nesif)
	local pos = string.find(kol_nesif,"from")
	kol_nesif = string.sub(kol_nesif,1,pos-2)
	kol_nesif = trim(kol_nesif)
	log("kolone nesifrovane .. " .. kol_nesif)
	if (kol_nesif == "*") then
		kol_nesif = getAsterisk(schema .. "." .. imeTabele)
	else
		kol_nesif = stringToList(kol_nesif)

	end
	sve_kolone = getAsterisk(schema .. "." .. imeTabele)
	local kk = getColNames(schema .. "." .. imeTabele)
	local num = #kk
	for i=1,num do
		se[i] = {}
		se[i][1] = kk[i]
		se[i][3] = sve_kolone[i]
		if(table.contains(kolone_sec,se[i][1])) then
			log("i=" .. i)
			se[i][2] = 1
			-- read key form SEC_CAT into k2
			upit1 = "SELECT enc_key FROM `SEC_CAT`.`ENC_KEYS` where
table_name = '" .. imeTabele .. "' and column_name = '".. se[i][1] .."' and
username = '".. user .. "'" ;
			log(upit1)
			k1 = readFromSEC(upit1)
			if(k1) then
				k2 = k1:fetch()
			else
				log("GRESKA! kolona nije sifrovana !")
				os.exit(2)
			end
			-- decrypt working key
			k2 = RSAdecrypt(k2,"/usr/local/masterrad/cert/" .. user
.. "/SK.pem")
			log(k2)
			-- encrupt v with woriking key
			se[i][3] = "AES_DECRYPT(" .. se[i][3] ..",'" .. k2 .. "')
as " .. se[i][3]
		else
			se[i][2] = 0
		end
		if (table.contains(kol_nesif,sve_kolone[i])) then
			table.insert(kolone, se[i][3])
		end
		log("kolone ".. i .. " = " .. tostring(kolone[i]) )
	end
	new_query = "select ".. listToString(kolone) .. " from " .. imeTabele
.. " " .. uslovi
	return new_query
end

-- PARSE INSERT INTO upit
```

```
function parseInsertInto(query)
	local ii = {}
	local kolone = {}
	local upit1 = ""
	local imeTabele  = getTableName(query)
	local k1,k2
	local kolone_sec={}
	local schema = proxy.connection.client.default_db
	local user = USER
	log("Korsnicka shema: " .. tostring(schema))
	local SK = "/usr/local/masterrad/cert/".. USER .."/SK.pem"
	-- vadjenje vrednosti kolona iz upita

	upit = string.gsub(query,"insert.*into .* values.*%(","")
	upit = string.gsub(upit,"%) key.+","")
	log(upit)
	upit = stringToList(upit)
	kolone_sec = getCryptCols(imeTabele)
	local kk = getColNames(schema .. "." .. imeTabele)
	local num  = #kk
	for i=1,num do
		ii[i] = {}
		ii[i][1] = kk[i]
		ii[i][3] = upit[i]
		if(table.contains(kolone_sec,ii[i][1])) then
			log("i=" .. i)
			ii[i][2] = 1
			-- read key form SEC_CAT into k2
			upit1 = "SELECT enc_key FROM `SEC_CAT`.`ENC_KEYS` where
table_name = '" .. imeTabele .. "' and column_name = '".. ii[i][1] .."' and
username = '".. user .. "'" ;
			log(upit1)
			k1 = readFromSEC(upit1)
			if(k1) then
				k2 = k1:fetch()
			end
			-- decrypt working key
			k2 = RSAdecrypt(k2,SK)
			-- encrupt v with woriking key
			ii[i][3] = "AES_ENCRYPT(" .. ii[i][3] ..",'" .. k2 ..
"')"
		else
			ii[i][2] = 0
		end
		kolone[i] = ii[i][3]
		log("kolone ".. i .. " = " .. tostring(kolone[i]) )
	end
	-- kreira se novi upit koji sadrzi sifrovane kolone
	new_query = "insert into " .. imeTabele .. " values (" ..
listToString(kolone) .. ")"
	log ("NQ = " .. new_query)
	return new_query
end
function getCryptCols(imeT)
	local k1,upit1
	local kolone_sec = {}
-- dohvatanje kolona koje treba da se sifruju u promenljivu kolone_sec
	upit1 = "select column_name from SEC_CAT.ENC_COLUMNS where table_name
= '" .. imeT .."'"
	k1 = readFromSEC(upit1)
	if(k1) then
```

```
            for u=1,k1:numrows() do
                  k2 = k1:fetch()
                  k2 = trim(k2)
                  table.insert(kolone_sec,k2)
                  --log("k2 - " .. tostring(k2))
            end
      else
            log("Greska! Tabela nema sifrovane kolone!")
            os.exit(1)
      end
      return kolone_sec
end

function getColNames(ImeTabele)
      -- dohvatanje imena kolona iz table u koju se ubacuje
      -- ime tabele u obliku shema.imetabele
      local upit1,k1
      upit1 = "select * from " .. ImeTabele
      k1 = readFromSEC(upit1)
      if(k1==false) then
            log("Greska! Tabela ne postoji !")
            os.exit(2)
      end
      return k1:getcolnames()
end

function CTelement(text)
      log("CTelement za -- " .. text .. " --")
      local element = text
      local el = {}
      local pravo = "12"
      local ime = ""
      local korisnici = {}
      --log(string.find(element,"update 1"))
      if (string.find(element,"update 1")) then
            pravo = "1"
      end
      if (string.find(element,"update 0")) then
            pravo = "0"
      end
      if (string.find(element,"update 2")) then
            pravo = "2"
      end
      element = string.gsub(element,"update .", "")
      element = string.gsub(element,"encrypt","")
      if(pravo=="12") then os.exit(12) end
      log("pravo=" .. pravo)
      log ("el - " .. element)
      if (string.match(element,"user_list")) then
            local upit1 = string.match(element,"user_list %[.+%]")
            upit1 = string.gsub(upit1,"user_list +%[","")
            upit1 = string.gsub(upit1,"%]","")
            --log("String koji se pretvara u listu: " .. upit1)
            repeat
                  local t = string.sub(upit1,1,string.find(upit1," "))
                  log("T - " .. t)
                  t = trim(t)
                  table.insert(korisnici,t)
                  upit1 = string.gsub(upit1,t,"")
                  upit1 = trim(upit1)
            until(string.find(upit1," ")==nil)
```

```
            log("T - " .. upit1)
            table.insert(korisnici,upit1)
            --log(element)
            element  = string.gsub(element,"user_list %[.+%]","")
            element = trim(element)
            --log("Upit posle skidanja liste - " .. element)
      end
      element = string.sub(element,1,string.find(element," "))
      ime = element
      table.insert(korisnici,USER)
      log("????" .. element .. " " .. tostring(korisnici) .. " " .. pravo)
      el = {ime,korisnici,pravo}
      return el
end
-- vraca niz kolona iz create table koje u sebi imaju encrypt
-- i listu korisnika kojima je dozvoljen pristup
function parseCreateTable(query)
      local CT = {}
      local i = 1
      local upit = string.lower(query)
      upit = trim(upit)
      local pocetak= string.find(upit,"%(")
      log ("pocetak = " .. tostring(pocetak))
      upit = string.sub(upit,pocetak+1,string.len(upit)-1)
      log(upit) -- deo upita izmedju zagrada
      if (string.match(query,",")) then
            repeat
                  local temp = ""
                  local ime = ""
                  temp = trim(string.sub(upit,1,string.find(upit,",")-1))
                  --log("kontrola - " .. temp)
                  if(string.find(temp,"encrypt")) then CT[i] =
CTelement(temp) i=i+1 end
                  upit =
string.sub(upit,string.find(upit,",")+1,string.len(upit))
                  --log(upit)
            until (string.match(upit,",") == nil)
      end
      upit = trim(upit)
      log("1kontrola:" .. upit .. i)
      if string.match(string.lower(upit),"encrypt") then
            if(string.find(upit,"encrypt")) then CT[i] = CTelement(upit)
end
      end
      return CT
end
--ALTER TABLE Functions
function parseAlterTable(query)
      local i = 1
      local add = {}
      local drop = {}
      query = string.lower(query)
      query = string.gsub(query,"alter table .+ set","")
      query = string.gsub(query,"encrypt","")
      query = string.gsub(query,"key .*","")
      local upit = stringToList(query)
      for j,v in ipairs(upit) do
            if(string.find(v,"drop")) then
                  v = trim(string.gsub(v," drop",""))
                  table.insert(drop,v)
                  --print(v)
```

```
            else
                  table.insert(add,v)
            end
      end
      return add,drop
end

-- izvrsava upit nad SEC katalogom
function writetoSEC(upit)
            local env = luasql.mysql()
            local konekcija =
env:connect("SEC_CAT","root","","localhost",3306)
            local kursor = konekcija:execute(upit)
            if (kursor==nil) then
                  return "upis u sec_cat neuspesan"
            end
            return "upis u sec_cat uspesan"
end

function readFromSEC(upit)
            local env = luasql.mysql()
            local konekcija =
env:connect("SEC_CAT","root","","localhost",3306)
            local kursor = konekcija:execute(upit)
            if (kursor==nil) then
                  return false
            end
            return kursor
end

function table.contains(table, element)
  for _, value in pairs(table) do
    if value == element then
      return true
    end
  end
  return false
end

-- vraca niz sa imenima kolona
function getAsterisk(tabela)
      local upit = "describe " .. tabela
      local kolona = {}
      local i = 1
      local rezultat = readFromSEC(upit)
      repeat
            local temp = rezultat:fetch()
            if(temp == nil) then
            else
                  kolona[i] = temp
                  --print (kolona[i])
            end
            i = i+1
      until (temp == nil)
      return kolona
end

-- skida kljucne reci iz upita
function dropKeyWords(text)
      text = string.gsub(text,"update 1","")
      text = string.gsub(text,"update 0","")
```

```
	text = string.gsub(text,"update 2","")
	text = string.gsub(text,"encrypt","")
	text = string.gsub(text,"user_list.*%[.*%]","")
	return text
end

-- vraca ime tabele iz upita
function getTableName(query)
	local upit = string.upper(query)
	if (string.match(upit,"CREATE TABLE %w+_%w+")) then
		upit = string.match(upit,"CREATE TABLE %w+_%w+")
		upit = string.gsub(upit,"CREATE TABLE","")
	elseif (string.match(upit,"CREATE TABLE %w+")) then
		upit = string.match(upit,"CREATE TABLE %w+")
		upit = string.gsub(upit,"CREATE TABLE","")
	end
	if (string.match(upit,"DROP TABLE %w+_%w+")) then
		upit = string.match(upit,"DROP TABLE %w+_%w+")
		upit = string.gsub(upit,"DROP TABLE","")
	elseif (string.match(upit,"DROP TABLE %w+")) then
		upit = string.match(upit,"DROP TABLE %w+")
		upit = string.gsub(upit,"DROP TABLE","")
	end
	if (string.match(upit,"INSERT INTO %w+_%w+")) then
		upit = string.match(upit,"INSERT INTO %w+_%w+")
		upit = string.gsub(upit,"INSERT INTO","")
	elseif (string.match(upit,"INSERT INTO %w+")) then
		upit = string.match(upit,"INSERT INTO %w+")
		upit = string.gsub(upit,"INSERT INTO","")
	end
	if (string.match(upit,"SELECT")) then
		local i = 0
		i = string.find(upit,"FROM")
		--print ("u1 - " .. upit)
		upit = string.sub(upit,i+5,string.len(upit))
		upit = trim(upit)
		upit = string.match(upit,"%a+")
	end
	if(string.match(upit,"ALTER TABLE .+ SET")) then
		upit = string.gsub(upit,"ALTER.+TABLE","")
		upit = string.sub(upit,1,string.find(upit,"SET")-2)
		--print (upit)
	end
	return string.lower(trim(upit))
end
function addColEnc(imetabele,imekolone,sk)
	local schema = "master"
	local imekorisnika = "djzecevic"
	local upit1 = "drop table IF EXISTS test.temp123"
	local USER = imekorisnika
	local tempRez = writetoSEC(upit1)
	log(upit1)
	log(tempRez)
	upit1="create table test.temp123 select * FROM " .. schema .. "." ..
imetabele
	local tempRez = writetoSEC(upit1)
	log(upit1)
	log(tempRez)
	local colnames = getAsterisk(schema .. "." .. imetabele)
	colnames = listToString(colnames)
	log(colnames)
```

```
	-- generate working key
	local KEY = generate128bit()

	colnames = string.gsub(colnames,imekolone,"AES_ENCRYPT(" .. imekolone
.. ",'" .. KEY .. "') as " .. imekolone)
	upit1 = "drop table " .. schema  .. "." .. imetabele
	local tempRez = writetoSEC(upit1)
	log(upit1)
	log(tempRez)
	upit1 = "create table " .. schema .. "." .. imetabele .. " select "
.. colnames .. " from test.temp123"
	local tempRez = writetoSEC(upit1)
	log(upit1)
	log(tempRez)
	upit1 = "drop table test.temp123"
	local tempRez = writetoSEC(upit1)
	log(upit1)
	log(tempRez)
	-- kreirati kljuc za trenutnog korisnika
	KEY = RSAencrypt(KEY,"/usr/local/masterrad/cert/" .. USER ..
"/certificate.pem")
	upit1 = "insert into  `SEC_CAT`.`ENC_KEYS` VALUES('" .. imetabele ..
"', '".. imekolone .."','" .. USER .. "','" .. KEY .. "')"
	local tempRez = writetoSEC(upit1)
	log(upit1)
	log(tempRez)
	upit1 = "insert into  `SEC_CAT`.`ENC_COLUMNS`VALUES('" .. imetabele
.. "' ,'".. imekolone .."','" .. USER .. "',1,'" .. USER .. "')"
	local tempRez = writetoSEC(upit1)
	log(upit1)
	log(tempRez)
end

function remColEnc(imetabele,imekolone,sk)
	local schema = proxy.connection.client.default_db
	local imekorisnika = USER
	local upit1 = "drop table IF EXISTS test.temp123"
	local tempRez = writetoSEC(upit1)
	log(upit1)
	log(tempRez)
	upit1="create table test.temp123 select * FROM " .. schema .. "." ..
imetabele
	local tempRez = writetoSEC(upit1)
	log(upit1)
	log(tempRez)
	local colnames = getAsterisk(schema .. "." .. imetabele)
	colnames = listToString(colnames)
	log(colnames)
	-- read key form SEC_CAT into k2
	upit1 = "SELECT enc_key FROM `SEC_CAT`.`ENC_KEYS` where table_name =
'" .. imetabele .. "' and column_name = '".. imekolone .."' and username =
'".. imekorisnika .. "'" ;
	log(upit1)
	local kk1 = readFromSEC(upit1)
	local kk2
	if(kk1) then
		kk2 = kk1:fetch()
		log(kk2)
	end
	-- decrypt working key
```

```
      kk2 = RSAdecrypt(kk2,"/usr/local/masterrad/cert/djzecevic/SK.pem")
      log("key: " .. kk2)
      colnames = string.gsub(colnames,imekolone,"AES_DECRYPT(" .. imekolone
.. ",'" .. kk2 .. "') as " .. imekolone)
      upit1 = "drop table " .. schema  .. "." .. imetabele
      local tempRez = writetoSEC(upit1)
      log(upit1)
      log(tempRez)
      upit1 = "create table " .. schema .. "." .. imetabele .. " select "
.. colnames .. " from test.temp123"
      local tempRez = writetoSEC(upit1)
      log(upit1)
      log(tempRez)
      upit1 = "drop table test.temp123"
      local tempRez = writetoSEC(upit1)
      log(upit1)
      log(tempRez)
      upit1 = "delete from  `SEC_CAT`.`ENC_KEYS` where table_name = '" ..
imetabele .. "' and column_name = '".. imekolone .."'"
      local tempRez = writetoSEC(upit1)
      log(upit1)
      log(tempRez)
      upit1 = "delete from  `SEC_CAT`.`ENC_COLUMNS` where table_name = '"
.. imetabele .. "' and column_name = '".. imekolone .."'"
      local tempRez = writetoSEC(upit1)
      log(upit1)
      log(tempRez)
end

function read_auth()
        USER = proxy.connection.client.username
end

function checkPrivileges(username,tblname,colname)
      local rezultat = false
      local temp = {}
      -- da li je kreator ?
      local upit1 = "SELECT creator,permition FROM `ENC_COLUMNS` where
`table_name` = '" .. tblname .. "' and `column_name` = '" .. colname .. "'"
      --print (upit1)
      local U = readFromSEC(upit1)
      if (U) then
            U:fetch(temp)
            log(temp[1] .. " " .. temp[2])
            log(username)
            if (temp[2]=="0") then
                  return false
            elseif (temp[1]==username) then
                  return true
            elseif(temp[2]=="2") then
                  return true
            end
      end
      return rezultat
end

function read_query( packet )
      if (packet:byte() ~= proxy.COM_QUERY) then
            return
      end
      local query = string.sub(packet, 2)
```

```
      -- proverava da li je drop table upit
      if (isDropTable(query)) then
            local imeT = getTableName(query)
            log(imeT)
            log("DELETE FROM `SEC_CAT`.`ENC_COLUMNS` WHERE table_name = '"
.. imeT .. "'")
            log("DELETE FROM `SEC_CAT`.`ENC_COLUMNS` WHERE table_name = '"
.. imeT .. "'")
            log(writetoSEC("DELETE FROM `SEC_CAT`.`ENC_COLUMNS` WHERE
table_name = '" .. imeT .. "'"))
            log(writetoSEC("DELETE FROM `SEC_CAT`.`ENC_KEYS` WHERE
table_name = '" .. imeT .. "'"))
            proxy.queries:append(1, string.char(proxy.COM_QUERY) .. query)
            return proxy.PROXY_SEND_QUERY
      end
      -- proverava da li u upitu ima kljucnih reci: encrypt, key
      if (not isSpecialQuery(query)) then
            return
      end
       log ("Special query;" .. query)
      -- proverava vrtu upita: create table, inset into, drop table, alter
table
      -- SELECT upit
      if (isSelect(query)) then
            query = parseSelect(query)
            log("SELECT query: " .. query)
            proxy.queries:append(1, string.char(proxy.COM_QUERY) .. query)
            return proxy.PROXY_SEND_QUERY
      end
      --CREATE TABLE
      if (isCreateTable(query)) then
            log("ulazak u create table")
            local kolone= parseCreateTable(query)
            --log("Pravi query pre drop key words- " .. query)
            query = dropKeyWords(query)
            local KEY = ""
            local imeTabele = getTableName(query)
            --log("Pravi query  posle drop key words- " .. query)
            log("Duzina niza  kolone " .. #kolone)
            for i=1,#kolone do
                  log("Duzina niza  useri " .. #kolone[i][2])
                  log("INSERT INTO `SEC_CAT`.`ENC_COLUMNS` VALUES ('" ..
imeTabele .. "','" .. kolone[i][1] .. "','" .. USER .. "','" ..
kolone[i][3] .. "','" .. USER .. "')")
                  tempRez = writetoSEC("INSERT INTO `SEC_CAT`.`ENC_COLUMNS`
VALUES ('" .. imeTabele .. "','" .. kolone[i][1] .. "','" .. USER .. "','"
.. kolone[i][3] .. "','" .. USER .. "')")
                  log("Rezultat ulita: " .. tempRez)
                  KEY = generate128bit()
                  for j=1,#kolone[i][2] do
                        log("kolona: " .. kolone[i][1] .. " # korisnik: "
.. kolone[i][2][j])
                        log ("KEY = " .. KEY)
                        local tempRez
                        -- potrebno je sifrovati kljuc sa javnim kljucem
korisnika
                        local ldapKEY = getLDAPKey(kolone[i][2][j])
                        log("LDAP KEY: " .. ldapKEY)
                        local KEY = RSAencrypt(KEY,ldapKEY,kolone[i][2][j])
                        log("korisnik: " .. kolone[i][2][j])
                        log ("Encrypted KEY = " .. tostring(KEY))
```

```
                        if (KEY) then
                              log("INSERT INTO `SEC_CAT`.`ENC_KEYS` VALUES
('" .. imeTabele .. "','" .. kolone[i][1] .. "','" .. kolone[i][2][j] ..
"','" .. KEY .. "')")
                              tempRez = writetoSEC("INSERT INTO
`SEC_CAT`.`ENC_KEYS` VALUES ('" .. imeTabele .. "','" .. kolone[i][1] ..
"','" .. kolone[i][2][j] .. "','" .. KEY .. "')")
                              log(tempRez)
                        else
                              log("Greska! - Nepravilno citanje korsnickog
sertifikata, molim vas pokusajte ponovo")
                              os.exit(1)
                        end
                  end
            end
            log(query)
            proxy.queries:append(1, string.char(proxy.COM_QUERY) .. query)
            return proxy.PROXY_SEND_QUERY
      end
      -- ALTER TABLE
      if (isAlterTable(query)) then
            log("alter table")
            rezultat = "select 'Alter table OK'"
            enc,no_enc = parseAlterTable(query)
            local imeTable = getTableName(query)
            SK = "/usr/local/masterrad/cert/".. USER .."/SK.pem"
            if  (#enc>0) then
                  log("Usao u  encrypt")
                  for j,v in ipairs(enc) do
                        -- obaraditi encrypt
                        addColEnc(imeTable,v,"SK")
                  end
            end
            log ("-------------")
            if  (#no_enc>0) then
                  log("Usao u no  encrypt")
                  for j,v in ipairs(no_enc) do
                        -- proveriti prava
                        if(checkPrivileges(USER,imeTable,v)) then
                              local upit1 = ""
                              remColEnc(imeTable,v,"SK")
                        else
                              rezultat = "select 'Access denied!'"
                        end
                  end
            end
            proxy.queries:append(1, string.char(proxy.COM_QUERY) ..
rezultat)
            return proxy.PROXY_SEND_QUERY
      end
      -- INSERT INTO TABLE
      if (isInsertInto(query)) then
            log("INSERT INTO FUNKCIJA POCINJE")
            query = parseInsertInto(query)
            proxy.queries:append(1, string.char(proxy.COM_QUERY) .. query)
            return proxy.PROXY_SEND_QUERY
      end

 end
```