UNIVERZITET U BEOGRADU

FAKULTET ORGANIZACIONIH NAUKA

Boris Damjanović

ADAPTIBILNA PRIMJENA AES ALGORITMA KOD SAVREMENIH OPERATIVNIH SISTEMA

doktorska disertacija

Beograd, 2016

UNIVERSITY OF BELGRADE

FACULTY OF ORGANIZATIONAL SCIENCES

Boris Damjanović

Adaptive implementation of the AES algorithm in modern operating systems

Doctoral Dissertation

Belgrade, 2016

Mentor:

dr Dejan Simić,
redovni profesor Fakulteta organizacionih nauka u Beogradu

Članovi komisije:

dr Dušan Starčević,
redovni profesor Fakulteta organizacionih nauka u Beogradu

dr Boško Nikolić,
vanredni profesor Elektrotehničkog fakulteta u Beogradu

Datum odbrane:

Adaptibilna primjena AES algoritma kod savremenih operativnih sistema

Apstrakt

Današnji kriptografski algoritmi su dizajnirani tako da kombinuju složene matematičke postupke sa teorijom i praksom računarskih nauka da bi što više povećali vlastitu otpornost na kriptoanalizu. Navedena složenost kriptografskih algoritama uslovila je odgovarajuće zahtjeve u pogledu procesorske snage. Proizvođači hardvera se danas radi uvećanja performansi okreću ili višeprocesorskim sistemima, što od proizvođača sistemskog kao i kriptografskog softvera zahtjeva poseban način pisanja koda, ili se okreću hardverskim akceleratorima, kada proizvođači softvera treba da se prilagode posebnim drajverima ili posebnim skupovima instrukcija.

Kompleksan algoritam, čija je otpornost na kriptoanalizu u posljednjih 15 godina intenzivno testirana, je algoritam AES (*Advanced Encryption Standard*). Ovaj algoritam je postao *de facto* globalni standard za komercijalni i *open source* softver ali i za hardver, koji pored administracije SAD koristi i veliki broj institucija i pojedinaca u cijelom svijetu.

Cilj ovog rada je da istraži načine pomoću kojih izvršavanje algoritma AES može da se prilagodi promjenljivim hardverskim i softverskim mogućnostima na sistemskom nivou. U okviru istraživanja je primjenjen modularan pristup koji koristi kombinaciju kriptografskih algoritama, matematičkih, statističkih metoda kao i metoda mašinskog učenja te odgovarajućih programerskih metoda i tehnika. Navedene metode i tehnike su korišćene da bi se sagledale mogućnosti realizacije ideje adaptibilnog operativnog sistema koji se prilagođava promjenljivim hardverskim i softverskim kriptografskim resursima a pomoću virtuelnog sistema datoteka kao jedne njegove portabilne komponente.

Ključne riječi: kriptografija; algoritmi; strukture podataka i algoritmi; AES; zaštita računarskih sistema; adaptibilni operativni sistemi.

Naučna oblast: Tehničke nauke

Uža naučna oblast: Informacione tehnologije

Adaptive implementation of the AES algorithm in modern operating systems

Abstract

Today's cryptographic algorithms are designed in a way that they combine complex mathematical procedures and the theory and practice of computer science in order to improve resistance to cryptanalysis. The complexity of mentioned cryptographic algorithms is causing the substantial requirements in terms of processing power. In order to increase performance, hardware manufacturers are focused to multiprocessor systems or to special hardware accelerators. First condition requires a special way of writing code from the manufacturer of the system software and cryptographic software. Second condition is forcing software vendors to adapt to special drivers or special sets of instructions.

AES (Advanced Encryption Standard) algorithm is a complex cipher whose resistance to cryptanalysis has been extensively tested over the last 15 years. This algorithm has become the *de facto* global standard for commercial and open source software and hardware. In addition to the U.S. administration, a number of institutions and individuals around the world use this algorithm.

The aim of this work is to explore possible ways in which the execution of AES algorithm can adapt to changing hardware and software capabilities at the system level. As part of the research will be applied modular approach. This approach will use a combination of cryptographic algorithms, mathematical, statistical methods and machine learning methods and appropriate programming methods and techniques. These methods and techniques are used to reveal possibilities in realization of the idea of adaptable operating system that adapts to changing hardware and software cryptographic resources through a virtual file system as one of its portable components.

Key words: cryptography; algorithms; data structures and algorithms; AES, computer systems security; adaptable operating systems.

Scientific field: Technical sciences

Scientific subfield: Information technologies

Sadržaj

1 Uvod

Moderna kriptografija se u velikoj mjeri oslanja na kompleksne algoritme koji su zasnovani na matematičkoj teoriji i na praksi računarskih nauka. Današnji kriptografski algoritmi se dizajniraju oko binarnog formata podataka imajući pri tome u vidu i pretpostavku težine izračunavanja da bi što više otežali mogućnost njihovog razbijanja. Jedan od algoritama čija je otpornost na kriptoanalizu tokom prethodnih 15 godina intenzivno testirana je algoritam AES. Advanced Encryption Standard (AES) je standard za enkripciju elektronskih podataka koje je uspostavio NIST (National Institute of Standards and Technology) na osnovu javno objavljenog i održanog takmičenja. AES je nastao restrikcijom pobjednika ovog takmičenja, algoritma pod nazivom Rijndael, na blok veličine 128 bita. Od momenta njegovog prihvatanja za standard 2001. godine traju neprekidna testiranja i istraživanja njegove otpornosti na kriptoanalizu ali i testiranja i istraživanja skoncentrisana na poboljšanje njegovih performansi.

Prema Murovom (Moore) zakonu, broj tranzistora koji može da se smjesti na integrisanim kolima se duplira približno svake dvije godine. Dejvid Haus (David House), jedan od Intelovih rukovodilaca, iznio je predviđanje da će se performanse čipova udvostručavati svakih 18 mjeseci. Kalahan (David Callahan) je iznio tvrdnju da su se performanse mikroprocesora uvećavale u prosjeku za 52% godišnje u periodu od 1986. do 2002. godine. Ovakvo uvećanje performansi je značilo da proizvođači softvera nisu morali mnogo da brinu oko optimizacije koda. Često su jednostavno čekali sljedeću generaciju mikroprocesora koja je neminovno donosila ubrzanja. Međutim, od 2002. godine performanse jednoprocesorskih sistema su se uvećavale za svega 20% godišnje. Iako se svake godine na istu površinu smješta sve više i više tranzistora, njihova brzina se ne može više uvećavati zbog pregrijavanja. Umjesto toga, proizvođači mikroprocesora se okreću procesorima sa više jezgara. Ovakvi više-jezgreni čipovi ubrzavaju izračunavanja tako što iskorišćavaju paralelizam, jer upošljavaju više procesora da obavljaju jedan zadatak. Međutim, jednostavno dodavanje procesora neće magično poboljšati performanse većine serijski napisanih programa. Da bi na najbolji način bile iskorišćene mogućnosti višeprocesorskih sistema, današnji proizvođači softvera moraju da proizvode više-nitne ili više-procesne aplikacije koje su u stanju da iskoriste postojeći hardver. Osim povećanja broja jezgara na procesorima generalne namjene, proizvođači hardvera počinju da za ubrzanje određenih dijelova sistema dodaju posebne posvećene komponente kao što su grafičke, jedinice za obradu videa, kriptografske jedinice (kriptografski akceleratori) pa sve do Intelove hardverske podrške za paralelni JavaScript, koji olakšava skripting na strani klijenta i HTML-5.

Da bi mogli da iskoriste sve mogućnosti koje pruža ovakav razvoj hardvera, programeri prilikom kreiranja aplikacija moraju da budu svjesni postojanja različitih hardverskih komponenti u sistemu ali i njihove snage i sposobnosti da brže i kvalitetnije obavljaju neki posao. Svijest o postojanju određene hardverske komponente bi se lako mogla kodirati u aplikaciju npr. pomoću konfiguracionih datoteka ili pomoću jednostavne bulove logike i kao takva unijeti u neku aplikaciju kao napredna mogućnost. Međutim, ako bi u sistemu postojale dvije hardverske komponente koje mogu da obave isti zadatak, rezonovanje o tome koja od njih je bolja za obavljanje nekog zadatka postaje kompleksan zadatak. Pri donošenju odluke o tome koji dio je bolji za konkretan zadatak, ljudski način razmišljanja bi uključilo npr. mjerenja kvaliteta i brzine, eventualno izračunavanje numeričkih

podataka o datim pokazateljima, učenje o navedenim hardverskim komponentama i čitanje njihove dokumentacije te iskustvo i intuiciju. Pri dodavanju mogućnosti donošenja ovakvih odluka u softver, proizvođači softvera moraju da se oslanjaju na numeričke podatke i njihovu statističku reprezentaciju. Automatsko donošenje odluka je proces određivanja i provođenja akcija kao odgovora na neke ulazne podatke. Ulazni podaci bi u slučaju automatskog donošenja odluka morali da budu kombinacija numeričkih i deskriptivnih podataka. U skladu s tim, implementacija modula za donošenje odluka može da se oslanja na statističke metode i metode mašinskog učenja. Za obezbjeđenje čuvanja podataka numeričke prirode koji su potrebni za rad ovakvog modula rješenje se može oslanjati i na usluge nekog sistema za upravljanje bazama podataka (SUBP).

Pomjeranje fokusa proizvodnje hardvera sa brzine na paralelizam i na posebne hardverske module zadužene za različite specifične zadatke na jednoj strani, te velika kompleksnost algoritama i procesorska snaga koja je često potrebna za obradu podataka modernim kriptografskim rješenjima na drugoj strani, zahtjeva od proizvođača softvera da dizajniraju programe koji će voditi računa o okruženju u kojem će se izvršavati. Prilagođavanje kriptografskog softvera i posebno algoritma AES kao trenutno *de facto* najvažnijeg algoritma promjenjivim hardverskim i softverskim mogućnostima može se realizovati na takav način da svaki proizvođač softvera počne da razvija i koristi svoja vlastita rješenja. S druge strane, uočena potreba za prilagođavanjem kriptografskih rješenja navedenog algoritma hardverskom okruženju moguća je i na sistemskom nivou, kada je u okviru operativnog sistema neophodan multidisciplinaran pristup koji osim kriptografije kombinuje i matematičke i statističke metode, tehnike razvoja sistemskog softvera i metode mašinskog učenja.

1.1 Struktura rada

U uvodnom dijelu rada data su razmatranja vezana za postojanje zavisnosti između algoritma AES i heterogenih hardverskih i softverskih resursa koji mogu da se nađu u proizvoljnom računarskom sistemu. Predmet istraživanja rada obuhvata analizu iskorišćenosti raspoloživih hardverskih i softverskih kriptografskih resursa od strane modernih operativnih sistema kada se za enkripciju koristi algoritam AES. Centralni problem koji se razmatra u ovom radu je mogućnost adaptibilne primjene algoritma AES u okviru operativnog sistema u zavisnosti od dostupnih resursa. Primarni cilj ovog istraživanja je razvoj modela za adaptibilnu primjenu algoritma AES na nivou operativnog sistema. Sekundarni ciljevi odnose se na povećanje kvaliteta zaštite modifikacijom algoritma te na istraživanje postojećih i novih rješenja za paralelizaciju izvršenja algoritma AES koji će biti formalno reprezentovani i implementirani radi potvrde koncepta, te istraživanje mogućnosti za uključenje posebnih hardverskih sklopova (CUDA GPU) kao kriptografskih resursa operativnog sistema. U uvodnom dijelu su navedene i polazne hipoteze koje će biti provjeravane u okviru doktorske disertacije.

U drugom poglavlju data su uvodna objašnjenja i teoretske pretpostavke vezane za formalnu reprezentaciju algoritma AES. U ovom poglavlju se opisuju notacije i konvencije koje su karakteristične za AES, zatim matematičke pretpostavke, transformacije ovog algoritma i osnovne ideje vezane za optimizaciju njegovog izvršenja.

Nakon toga, u trećem poglavlju se predstavlja istraživanje korišćenih načina enkripcije u okviru operativnih sistema, gdje se prikazuju različite ideje, mjesta i

načini na koje je ona realizovana u različitim operativnim sistemima. Na osnovu istraživanja koje je predstavljeno u dijelu ovog poglavlja nastala je publikacija (Damjanović i Simić, InfoM47, 2013). U nastavku poglavlja predstavljen je određen broj radova koji je istraživao podršku kriptografskim akceleratorima u operativnim sistemima, te adaptibilnost softverskih resursa hardverskim mogućnostima računara. Na kraju trećeg poglavlja data je komparativna analiza kriptografskih rješenja u operativnim sistemima u kojoj je dat uporedni prikaz dijela rješenja koja se i danas koriste u modernim operativnim sistemima.

U četvrtom poglavlju dat je pregled metoda i tehnologija primijenjenih u razvoju rješenja. Od potrebnih tehnologija za razvoj najprije je prikazana Java sa JCA/JCE skupom API-ja, a zatim i Oracle/SUN JCE kriptografska biblioteka. Nakon toga predstavljen je Intel-ov Advanced Encryption Standard New Instructions (AES-NI) skup instrukcija. Sljedeća tehnologija koja je predstavljena u okviru ovoga poglavlja je CUDA arhitektura i paralelno programiranje. Nakon toga je predstavljen virtuelni fajl sistem Apache Commons koji je namijenjen da služi kao kontejner za adaptibilni virtuelni kriptografski fajl sistem. U nastavku je predstavljen koncept mašinskog učenja sa metodama regresione analize, M5' i *Hill-climbing*. Nakon toga, predstavljene su tehnologije soketa te imenovanih i neimenovanih cijevi. Na kraju su predstavljeni kriptografski načini rada koji su uticali na neka rješenja prikazana u radu.

U petom poglavlju razmatra se razvoj modela za adaptiblnu primjenu AES algoritma u savremenim operativnim sistemima. Model je pozicioniran unutar virtuelnog fajl sistema unutar kojeg je smješten modul za koordinaciju. U okviru ovog modula logički su smješteni sistemi za trening i kategorizaciju te za selekciju najefikasnijeg resursa, veze sa odabranim sistemom za upravljanje bazama podataka, sa ulazno izlaznim podsistemom, konfiguracija sistema i određen broj kriptografskih resursa (modula) koji su u stanju da vrše šifrovanje i dešifrovanje pomoću algoritma AES. U nastavku se vrši pozicioniranje prethodno navedenih tehnika i tehnologija, koje iako logički spadaju u modul za mašinsko učenje, fizički mogu i moraju biti sasvim drugačije realizovani. U ovom poglavlju se razmatraju i zajedničke karakteristike implementiranih kriptografskih modula i utvrđuju granice razvoja sistema.

Šesto poglavlje donosi opis arhitekture rješenja za adaptibilnu primjenu AES algoritma na nivou operativnog sistema. U ovom dijelu su logičke komponente koje su predstavljene u modelu sada prikazane kao zasebni fizički moduli koji ne moraju, a u slučajevima kriptografskih modula i ne trebaju, da budu u sastavu modula za mašinsko učenje. Ovdje su predstavljeni Apache Commons VFS, te sistemi za trening i kategorizaciju, sistem za selekciju najefikasnijeg resursa sa podsistemima za klasifikaciju i nominaciju, veze sa bibliotekom programa za mašinsko učenje, sa odabranim sistemom za upravljanje bazama podataka, ulazno-izlazni modul, korišćeni SUBP, modeli komunikacije te podsistem za određivanje praga iskoristivosti.

U sedmom poglavlju razmatraju se osnovne karakteristike implementiranih jednonitnih kriptografskih modula. Najprije se razmatraju originalne jednonitne implementacije u programskom jeziku Java koje se oslanjaju na ideje autora algoritma (EAESG). Nakon toga predstavlljene su implementacije omotača oko kriptografske biblioteke Oracle/SunJCE. Potom je predstavljeno jednonitno rješenje implementirano pomoću Visual C/C++ kompajlera. Na kraju predstavljanja jednonitnih modula je prikazano rješenje koje se oslanja na AES-NI skup instrukcija. Dio ovog poglavlja koji istražuje implementaciju omotača oko *third party* biblioteka doveo je do rezultata koji su predstavljeni u publikaciji (Damjanović and Simić, 2013).

U nastavku, u poglavlju 8 slijedi opis kriptografskog modula koji paralelizuje izvršenje AES algoritma uz pomoć CUDA arhitekture, kombinacijom CUDA C kompajlera i Visual C/C++ kompajlera, gdje se CUDA GPU pojavljue kao resurs operativnog sistema. Među prikazanim modulima ovaj se, uz modul koji se oslanja na speciijalizovano hardversko AES-NI šifrovanje, izdvaja svojim velikim potencijalom za brzu obradu podataka.

Nakon toga su, u okviru poglavlja 9, predstavljeni načini modifikacije algoritma AES koji su pronađeni u dostupnoj literaturi, te vlastite modifikacije ovog algoritma. Dio istraživanja koja su predstavljena u ovom poglavlju potekao je iz Master rada (Damjanović, 2010), a drugi dio je predstavljen u publikacijama (Damjanović and Simić, 2011) i (Damjanović i Simić, InfoM46, 2013). Kombinovanje prethodnih modifikacija sa različitim kriptografskim režimima rada dovelo je do novih ideja vezanih za paralelizaciju izvršenja algoritma AES.

U desetom poglavlju su predstavljene ideje za paralelizaciju izvršenja AES algoritma korišćenjem modifikovanog OFB načina rada. Rezultati ovoga istraživanja koje svoje izvore ima u objavljenim publikacijama (Damjanović and Simić, 2011), (Damjanović i Simić, InfoM46, 2013) te konačno u (Damjanović and Simić, 2015).

U jedanaestom poglavlju daje se evaluacija prikazanih rješenja, sa međusobnim poređenjem karakterističnih modela pojedinih kriptografskih resursa, uz poređenje tačnosti predviđanja ovakvih modela i tačnosti određivanja optimalnog kriptografskog resursa od strane adaptibilnog virtuelnog kriptografskog fajl sistema. Na kraju ovog poglavlja se daje provjera hipoteza u cilju potvrde svrhe istraživanja, koja se ogleda u uticaju koji na performanse ima adaptacija operativnih sistema raspoloživim hardverskim i softverskim kriptografskim resursima.

U zaključnim razmatranjima istaknut je značaj istraživanja problema iskorišćenja raspoloživih kriptografskih resursa, najznačajniji doprinosi disertacije i smjernice za dalji rad.

1.2 Definisanje problema i predmeta istraživanja

Kriptografija je osnovni alat za zaštitu velikih količina digitalnih informacija koje se danas smještaju u modernim računarima ili distribuiraju kroz mrežu. Moderna kriptografija se oslanja na intenzivna izračunavanja koja, kada se radi o softverskoj enkripciji, često predstavljaju usko grlo za tok informacija. Zbog toga su proizvođači hardvera počeli da proizvode s jedne strane različite specijalizovane hardverske komponente od kojih su neke bile kreirane isključivo za ubrzanje kriptografskih operacija dok su s druge strane za ubrzanje enkripcije i dekripcije korišćene i jedinice opšte namjene.

Jedan dio specijalizovanih komponenti koje pružaju kriptografske mogućnosti se mogu naći u sastavu novijih mikroprocesora, poput mogućnosti izvršavanja Intelovog AES-NI skupa instrukcija, zatim VIA PADLOCK te CLMUL skupova instrukcija. Drugi dio specijalizovanih komponenti je predstavljen tzv. kriptografskim akceleratorima. Kriptografski akceleratori su kreirani sa namjerom da izvršavaju specifične operacije ili kompletne algoritme mnogo brže nego mikroprocesori opšte namjene (CPU), a javljaju se u vidu koprocesora, poput IBM-4764, Freescale ili kartica poput Oracle-ove SUN Crypto Accelerator 6000 PCIe kartice.

Treba napomenuti da, kada je u pitanju poboljšanje performansi algoritma AES, postoje i rješenja koja pokušavaju da ubrzaju izvršenje koristeći jedinice opšte namjene poput procesora sa više jezgara i modernih GPU jedinica.

Implementacije AES algoritma koje se oslanjaju na nove mogućnosti koje pružaju moderni GPU procesori koriste činjenicu da noviji grafički procesori nisu više okrenuti isključivo floating point operacijama, već da su u stanju da izvršavaju i aritmetičke i logičke operacije nad cjelobrojnim podacima. Međutim, kako se GPU jedinice često sastoje od nekoliko stotina ili hiljada jezgara, njihovo programiranje često zahtjeva prilagođenje implementacije i korišćenje odgovarajućih programskih biblioteka. Ovdje naravno treba pomenuti i prilagođene implementacije koje koriste veći broj jezgara neke CPU jedinice za poboljšanje performansi AES algoritma. Različiti prijedlozi za prilagođenje implementacija algoritma AES mogućnostima paralelne obrade kako pomoću CPU tako i GPU do sada su objavljeni u većem broju naučnih radova.

Da bi aplikacije mogle da koriste pomenute skupove instrukcija poput AES-NI skupa, izvorni kod njihovih kriptografskih biblioteka u kojima se nalazi implementacija npr. AES algoritma mora da se prilagođava - ako procesor podržava navedeni skup instrukcija, aplikacija treba da ga koristi, a u suprotnom treba da koristi tradicionalnu implementaciju AES algoritma. Slično vrijedi i za višejezgrene procesore - da bi aplikacije mogle da koriste njihovo postojanje, njihov izvorni kod mora biti napisan tako da podržava paralelno izvršavanje. S druge strane, kada su u pitanju kriptografski akceleratori, kriptografske biblioteke ne mogu da im pristupaju bez posebnih upravljačkih programa (*drivers*).

Danas u svijetu postoji veliki broj kriptografskih biblioteka u okviru kojih je između ostalih implementiran algoritam AES u različitim programskim jezicima. Pored njih, u velikom broju projekata otvorenog koda često se javljaju aplikacije u okviru kojih je implementiran samo ovaj algoritam. Pojedine ovakve implementacije često imaju integrisane mogućnosti adaptacije dostupnim hardverskim resursima u nekom računarskom sistemu.

Navedene kriptografske biblioteke i softverske implementacije AES algoritma, kao i hardverski sklopovi koji se sa odgovarajućim drajverima ili bez njih koriste za

poboljšanje performansi ovog algoritma, se po volji i u skladu sa potrebama vlasnika računarskog sistema instaliraju ili uklanjaju sa njega. Zbog toga se pomenuti računarski sistem mora posmatrati kao heterogen i rekonfigurabilan sistem.

Pomenuta heterogenost i rekonfigurabilnost hardverskih i softvreskih komponenata koje se koriste za poboljšanje performansi algoritma AES nameće potrebu postojanja jedinstvenog interfejsa u okviru operativnog sistema za pristup kriptografskim mogućnostima hardvera i softvera sa standardizovanim i uniformnim načinima inicijalizacije i pozivanja pojedinih komponenti. U takvim kompleksnim sistemima se pored pomenutog problema mapiranja, uslijed heterogenosti hardverskih sklopova i softverskih rješenja javlja i potreba traženja što efikasnijih načina korišćenja raspoloživih hardverskih i softverskih resursa. Navedena rekonfigurabilnost hardverskih resursa i softverskih rješenja stvara još i potrebu postojanja mogućnosti prilagođenja operativnog sistema i donošenje odluke koju od trenutno raspoloživih komponenata koristiti za dobijanje najboljih performansi.

Predmet istraživanja ovog rada je analiza iskorišćenosti raspoloživih kriptografskih hardverskih i softverskih resursa od strane modernih operativnih sistema kada se za enkripciju koristi AES algoritam. Mogućnost adaptibilne primjene algoritma AES u okviru operativnog sistema u zavisnosti od dostupnih resursa predstavlja centralni problem koji se istražuje u ovom radu.

1.3 Ciljevi istraživanja

Razvoj modela za adaptibilnu primjenu algoritma AES na nivou operativnog sistema radi poboljšanja performansi računarskog sistema pri izvršavanju ovog algoritma predstavlja primarni cilj ovog istraživanja. Model se ostvaruje kao kombinacija jedinstvenog i jednostavnog interfejsa za pristup resursima i njihovo mapiranje i posebnog algoritma koji odluku o korišćenju raspoloživih resursa donosi prije samog pokretanja funkcije šifrovanja/dešifrovanja, a na osnovu prethodno memorisanih prediktivnih modela koji su nastali na osnovu rezultata izvršenja pojedinih hardverskih i softverskih komponenata kao i pojedinih posebnih uslova koje zadaju korisnici.

Sekundarni ciljevi koji su postignuti istraživanjem adaptibilne primjene algoritma AES u okviru operativnih sistema su:

- razvoj rješenja za kreiranje i čuvanje prediktivnih modela koji opisuju ponašanje pojedinih kriptografskih resursa;
- razvoj rješenja za donošenje odluke o izboru najefikasnijeg kriptografskog resursa na osnovu više nezavisnih varijabli;
- razvoj postupka za evaluaciju modela nekog kriptografskog resursa koji može da koristi više nezavisnih varijabli;
- razvoj uniformnog modela za razmjenu podataka između operativnog sistema i pojedinih kriptografskih modula;
- razvoj modela za mapiranja pojedinih kriptografskih resursa;
- razvoj postupka modifikacije algoritma AES izmjenom broja bita ključa;
- razvoj rješenja za paralelizaciju algoritma AES korišćenjem GPU jedinica;
- razvoj algoritma za paralelizaciju algoritma AES pomoću različitih kriptografskih načina rada (*modes of operation*).

Sa naučnog aspekta istraživanje treba da pokaže da li adaptacija operativnog sistema raspoloživim resursima utiče na poboljšanje performansi i kvalitet zaštite algoritmom AES. Pri tome naučna deskripcija kako metode za kreiranje prediktivnih

modela koji opisuju ponašanje kriptografskih resursa i tako i postupka za prezentaciju i evaluaciju navedenih prediktivnih modela predstavljaju bitan preduslov za uspješnu adaptaciju operativnog sistema raspoloživim kriptografskim resursima. Treba napomenuti naučni značaj prikazanih rješenja za modifikaciju algoritma AES, rješenja za paralelizaciju izvršenja algoritma AES koje se oslanja na istraživanja modifikacije kriptografskih načina rada (*modes of operation*), kao i na u radu prikazanu mogućnost ubrzanja paralelizacije korišćenjem CUDA GPU uređaja. Sa društvenog aspekta istraživanje treba da obezbjedi specifična znanja vezana za problematiku projektovanja i implementacije kako algoritma koji odlučuje o primjeni određenog hardverskog ili softverskog rješenja, tako i o implementaciji pojedinih rješenja koja se, kao potvrda koncepta, oslanjaju na (u datom momentu) dostupne hardverske komponente.

Postizanje navedenih ciljeva je multidisciplinaran zadatak jer se oslanja na znanja iz kriptografije, softverskog inženjerstva, arhitekture računara, statistike i mašinskog učenja.

Imajući u vidu ciljeve istraživanja, kao najvažniji zadaci se javljaju:

- razvoj modela za adaptibilnu primjenu algoritma AES na nivou operativnog sistema radi poboljšanje performansi računarskog sistema koji se oslanja na:
    - izgrađeno rješenje za kreiranje i snimanje prediktivnih (prognostičkih) modela koji predviđaju brzinu pojedinih kriptografskih resursa;
    - razvijen postupak za evaluaciju podataka dobijenih od tako kreiranih prediktivnih modela;
    - razvijen uniformni model za razmjenu podataka između VFS i pojedinih kriptografskih modula;
    - razvijen model za mapiranje pojedinih kriptografskih resursa;
- razvoj metoda i tehnika za paralelizaciju algoritma AES koji se oslanja na
    - paralelizaciju pomoću modifikacije kako različitih kriptografskih načina rada (*modes of operation*) tako i
    - paralelizaciju korišćenjem GPU jedinica;
- razvoj metoda i tehnika za modifikaciju i ojačanje algoritma AES;
- primjena navedenih rješenja u okviru adaptibinog virtuelnog kriptografskog fajl sistema radi potvrde koncepta.

Konačni rezultati će dati doprinos deskripciji postupka i klasifikaciji mogućih rješenja za adaptibilnu primjenu algoritma AES u modernim operativnim sistemima.

1.4 Polazne hipoteze

Uzevši u obzir da je predmet istraživanja analiza iskorišćenosti raspoloživih kriptografskih hardverskih i softverskih resursa od strane modernih operativnih sistema kada se za enkripciju koristi algoritam AES i da se u istraživanju ispituje mogućnost adaptibilne primjene algoritma AES u okviru operativnog sistema u zavisnosti od dostupnih resursa, postavlja se generalna (opšta) hipoteza ovog istraživanja:

H0: Postoji uticaj koji se na poboljšanje performansi i kvaliteta zaštite algoritmom AES ostvaruje putem adaptacije operativnih sistema raspoloživim hardverskim i softverskim kriptografskim resursima.

Na osnovu generalne hipoteze, a prema predmetu istraživanja, izdvajaju se posebne hipoteze:

H0.1: Ako se kreiraju metoda za klasifikaciju i algoritam za nominaciju koji uz pomoć ranije kreiranih prediktivnih modela (i pomoću metoda mašinskog učenja) donose odluku o izboru najefikasnijeg raspoloživog kriptografskog resursa u datim uslovima, tada je moguće kreirati elastičan model koji se na sistemskom nivou prilagođava raspoloživim hardverskim i softverskim kriptografskim resursima.

H0.2: Mogućnost izbora najefikasnijeg raspoloživog resursa zavisi od mogućnosti mapiranja i komunikacije različitih kriptografskih resursa sa sistemom.

Iz posebnih hipoteza se daljim preciziranjem izvlače pojedinačne hipoteze:

H0.1.1: Realizacija postupka kreiranja prediktivnih modela koji predviđaju ponašanje pojedinih kriptografskih resursa predstavlja preduslov za projektovanje algoritma za donošenje odluke o izboru najefikasnijeg kriptografskog resursa.

H0.1.2: Kreiranjem kategorizovanih empirijskih trening podataka se stvaraju pretpostavke za primjenu metoda mašinskog učenja radi kreiranja perzistentnih prediktivnih modela koji predviđaju brzinu izvršenja pojedinih kriptografskih resursa u određenim uslovima.

H0.1.3: Ako se modeluju postupci za kategorizaciju, evidentiranje i pretraživanje empirijskih podataka koji opisuju različite kriptografske resurse (module), tada se stvaraju pretpostavke za kreiranje trening podataka koji će biti korišćeni od strane metoda za mašinsko učenje.

H0.2.1: Kreiranjem modela za mapiranje kriptografskih resursa od strane sistema ili korisnika stvaraju se pretpostavke za uključenje raspoloživih kriptografskih resursa u sistem.

H0.2.2: Kreiranjem standardnog modela za komunikaciju sistema sa okruženjem stvaraju se pretpostavke za korišćenje raspoloživih kriptografskih resursa.

1.5 Metode istraživanja

U ovom istraživanju će biti primijenjene različite opšte naučne metode (Sotirović i Adamović, 2005), (Mihailović 2012) i (Miljević 2007). Najprije će kao opšte naučna metoda biti korišćena metoda modelovanja, a zatim, statistička i analitičko-deduktivna metoda te metoda naučne deskripcije. Metoda modelovanja je teorijsko-empirijska metoda u čijoj osnovi su metode tipologizacije, apstrakcije i konkretizacije. Modelovanje će najprije biti predstavljeno pomoću tipologizacije (klasifikacije) različitih kriptografskih resursa, zatim pomoću apstrakcije navedenih kriptografskih resursa i konkretizacije algoritma za odlučivanje o izboru najboljeg resursa. U toku modelovanja će biti korišćen modelni eksperiment koji se kao praktična i empirijska metoda u velikoj mjeri oslanja na statističke metode. Dedukcija se shvata i razmatra kao oblik zaključivanja, prvenstveno silogističkog. Dedukcijom se analitički misaono - logički iz premisa - već formiranih zaključaka po utvrđenoj proceduri izvode novi zaključci.

Kao istraživačka metoda, u ovom radu je korištena metoda naučnog eksperimenta. Eksperiment koji će biti proveden predstavlja organizovano izvođenje procesa analize podataka u unaprijed određenim uslovima (virtuelni fajl sistem i kriptografski resursi) u cilju utvrđivanja svojstava navedenog procesa i u cilju provjere hipoteza o ovom procesu. Ovakav eksperiment zahvaljujući posebnim uslovima njegovog izvršavanja ima u sebi elemente laboratorijskog eksperimenta. U eksperimentalnom dijelu će biti posmatrana i mjerena uspješnost prilagođenja u svrhu potvrđivanja osnovne hipoteze istraživanja.

U radu će biti predstavljeno empirijsko istraživanje performansi pojedinih kriptografskih modula, te međusobna komparacija dobijenih rezultata kao i empirijska procjena performansi adaptacije operativnog sistema dostupnim resursima, zbog čega će se u njoj koristiti i komparativna metoda.

Rezultati istraživanja će biti prikazani u tekstualnom obliku, zatim u obliku tabela i grafikona, slika i dijagrama te pomoću isječaka koda. U toku implementacije različitih verzija AES algoritma koji se oslanjaju na različite hardverske i softverske resurse korišćene su metode i tehnike iz oblasti kriptografije, softverskog inženjerstva i arhitekture računara. Algoritam pomoću kojega se operativni sistem prilagođava raspoloživim resursima obuhvata korišćenje metoda statistike, mašinskog učenja i softverskog inženjerstva, dok sam proces izrade modela pripada metodologiji modelovanja.

Radi potvrde koncepta biće korišćen Apache Commons virtuelni fajl sistem sa kojim će biti povezani različiti kriptografski paketi poznatih proizvođača, kao i vlastita rješenja koja mogu da koriste jedan procesor (prema idejama autora algoritma) ili rješenja koja mogu da koriste više procesora. U toku istraživanja će za implementaciju u najvećoj mjeri biti korišćene tehnike i tehnologije koje se oslanjaju na programske jezike Java i C/C++. Pored ovoga će za upotrebu specifičnih AES-NI skupova instrukcija biti korišćen programski jezik C++, a za programiranje CUDA uređaja programski jezik CUDA C i odgovarajući kompajler.

2 Algoritam AES

Algoritam AES (FIPS197, 2001) ili *Advanced Encryption Standard* je nastao zbog potrebe da se zamijeni već pomalo nesigurni DES algoritam.

U januaru 1997. godine, NIST (*National Institute of Standards and Technology*) objavio je početak inicijative za razvoj algoritma koji će postati sljedeći kriptografski standard američke vlade (Daemen and Rijmen, 2002). Za razliku od procesa selekcije za algoritme DES, SHA-1 i DSA, NIST je objavio da će takmičenje za izbor najboljeg algoritma biti otvoreno. Kako se navodi u (Roback and Dworkin, 1998) 21 algoritam je prijavljen za takmičenje do 15.juna, do kada su trajale prijave. Međutim, među prijavama je bilo samo 15 kandidata koji su zadovoljili stroge ulazne kriterijume: CAST-256, CRYPTON, DEAL, DFC, E2, FROG, HPC, LOKI97, MAGENTA, MARS, RC6, Rijndael, SAFER+, Serpent i Twofish. U mjestu Ventura u Kaliforniji u avgustu 1998. godine održana je prva konferencija posvećena izboru ovog algoritma na kojoj su kandidati predstavljeni. U martu 1999. godine u Rimu u Italiji je održana druga konferencija čiji rezultati su kombinovani sa rezultatima radionice pod nazivom *Fast Software Encryption Workshop*, koja je takođe održana u Rimu. U avgustu 1999. godine, lista kandidata je smanjena na 5: MARS, RC6, Rijndael, Serpent i Twofish. Slijedeća runda testiranja održana je u aprilu 2000. godine u Nju Jorku na konferenciji koja je bila posvećena ovom algoritmu. Konferencija je ponovo bila kombinovana sa rezultatima sljedeće radionice pod nazivom *Fast Software Encryption Workshop*, koja je takođe održana u Nju Jorku.

Krajem 2000. godine, NIST je proglasio pobjednika – algoritam zvan Rijndael (po imenima pronalazača Vincent Rijmen i Joan Daemen, a izgovara se kao "*Rhine Dahl*", "*Rain Doll*" ili "*Reign Dahl*"). Algoritam AES koristi SP (supstituciono-permutacionu) mrežu za razliku od svog prethodnika, algoritma DES, koji koristi Feistelovu mrežu. Rijndael (Daemen and Rijmen, 2002) dozvoljava da veličinu ključa kao i veličinu bloka biramo iz skupa {128, 160, 192, 224, 256}. Treba reći da je američka vlada dio Rijndaela koji je prihvatila nazvala *Advanced Encryption Standard* (AES), te da je u dokumentu FIPS-197 koji definiše AES algoritam navedeno da dužina bloka mora uvjek biti 128 bita, dok dužine ključeva mogu biti 128, 192 ili 256 bita. Transformacije ovog algoritma - SubBytes, ShiftRows, MixColumns, AddRoundKey i kao njihove inverzne verzije se obavljaju nad matricom veličine 4x4 bajta koja se naziva Stanje (State).

Sva navedena istraživanja i testiranja ovog algoritma koja su nastavljena do današnjih dana su pokazala da ovaj algoritam nema javno poznatih slabosti. Treba reći da je tokom prvih deset godina postojanja ovog algoritma zabilježen određen broj napada, ali samo na oslabljene i redukovane verzije algoritma AES, zbog čega je Vinsent Rijmen napisao ironičan članak pod nazivom "*Practical-Titled Attack on AES-128 Using Chosen-Text Relations*" (Rijmen, 2010). Tek 2011. godine Bogdanov, Khovratovich i Rechberger (Bogdanov et al. 2011) uspijevaju da izvrše *key-recovery* napad pomoću kojeg je moguće otkriti 128-bitni ključ pomoću $2^{126.1}$ operacija a 256-bitni ključ pomoću $2^{254.4}$ operacija. Iako se u kriptografiji probojem smatra bilo koje rješenje koje je iole brže od *brute force* napada, ipak se za ovakve napade ne može reći da su u velikoj mjeri oslabili algoritam.

2.1 Specifikacija i formalna reprezentacija

Rijndael je blokovski kriptografski algoritam sa varijabilnom dužinom bloka i varijabilnom dužinom ključa. Dužina bloka i dužina ključa mogu biti nezavisno specificirane kao umnošci broja 32, s tim da je najmanji dozvoljeni umnožak 128, a najveći dozvoljeni 256 bita (Daemen and Rijmen, 2002). Generalno, dužine ulaznog i izlaznog bloka mogu da uzimaju bilo koju od navedenih vrijednosti, ali u algoritmu AES jedina dozvoljena dužina bloka je 128 bita (Gladman, 2007).

2.1.1 Standardom definisani AES algoritam i njegove transformacije

Advanced Encryption Standard (AES) zasnovan je na Rijndael algoritmu. AES je simetrični blokovski algoritam koji obrađuje blokove podataka veličine 128 bita i koji pri tome koristi ključeve dužine 128, 192 i 256 bita. Rijndael je originalno napravljen da obrađuje blokove različitih veličina i drugačije dužine ključeva, iz skupa {128, 160, 192, 224, 256} bita, ali različite veličine blokova u standardu nisu prihvaćene. Pošto algoritam može biti korišten sa različitim dužinama ključeva, različite verzije algoritama u literaturi se navode kao "AES-128", "AES-192", and "AES-256".

2.1.1.1 Notacije i konvencije

2.1.1.1.1 Termini

U sljedećoj tabeli dati su termini koji će biti korišteni pri definisanju algoritma AES:

AES	Advanced Encryption Standard
Afina transformacija	Transformacija koja se sastoji od množenja matricom kojoj slijedi dodavanje vektora
Bit	Binarna cifra (0 ili 1)
Blok	Sekvenca binarnih bita koja čini ulaz, izlaz, stanje i ključ runde
Bajt	Grupa od osam bita
Inicijalni ključ	Tajni, kriptografski ključ koji se koristi u rutini za ekspanziju ključeva da bi proizveo skup ključeva runde. Može se predstaviti kao pravougaoni niz bajta sa četiri reda i Nk kolona.
Rutina za ekspanziju ključeva (*Key Expansion*)	Rutina koja se koristi za generisanje niza ključeva rundi iz ključa.
Ključ runde (*Round Key*)	Ključevi runde se izvode iz ključa pomoću rutine za ekspanziju ključeva. Primjenjuju se na Stanja u algoritmu za šifrovanje i dešifrovanje.
Proširenи ključ, Lista ključeva	Kompletan niz koji je proizveden pomoću rutine za ekspanziju ključeva
Stanje (*State*)	Trenutni međurezultat šifrovanja koji se može predstaviti kao pravougaoni niz bajta sa četiri reda i Nb kolona.
S-box	Supstituciona kutija, nelinearna supstituciona tabela koja se koristi u nekoliko transformacija nad bajtima i u rutini za

	ekspanziju ključeva
***Word* ili Riječ**	Grupa od 32 bita koja se tretira ili kao jedinstven entitet ili kao niz od četiri bajta.

Tabela 1: Termini

2.1.2 Parametri, simboli i funkcije algoritma AES

U algoritmu se koriste sljedeći paramerti, simboli i funkcije:

AddRoundKey ()	Transformacija kojom se u toku šifrovanja (i dešifrovanja) ključ runde dodaje Stanju pomoću XOR operacije.
InvMixColumns()	Transformacija kojom se, tokom dešifrovanja, vrši operacija inverzna MixColumns() transformaciji
InvShiftRows()	Transformacija kojom se, tokom dešifrovanja, vrši operacija inverzna ShiftRows() transformaciji
InvSubBytes()	Transformacija kojom se, tokom dešifrovanja, vrši operacija inverzna SubBytes() transformaciji
K	Inicijalni ključ
MixColumns()	Transformacija kojom se, tokom šifrovanja, uzimaju sve kolone jednog Stanja i miješaju njihovi podaci, da bi se dobile nove kolone.
Nb	Broj kolona (32-bitnih riječi) koji opisuje Stanje. Za ovaj standard, Nb = 4.
Nk	Broj 32-bitnih riječi koji opisuje inicijalni ključ. U ovom standardu, Nk = 4, 6 ili 8.
Nr	Broj rundi koji je u funkciji Nk i Nb. U ovom standardu, Nr = 10, 12 ili 14.
Rcon[ ]	Niz konstanti veličine jedne riječi (*Word*)
RotWord()	Funkcija koja se koristi pri ekspanziji ključeva (*Key Expansion*) i koja uzima Riječ (*Word*) od 4 bajta i nad njom izvodi cikličnu permutaciju.
ShiftRows()	Transformacija koja, tokom šifrovanja, obrađuje Stanje tako da ciklično pomjera (šifta) tri zadnja reda Stanja za različite pomake.
SubBytes()	Transformacija koja, tokom šifrovanja, obrađuje Stanje koristeći S-Box. Ova transformacija obradi svaki bajt Stanja nezavisno od drugih.
SubWord()	Funkcija koja se koristi prilikom ekspanzije ključeva (*Key Expansion*) koja kao ulazni parametar uzima Riječ (*Word*) od 4 bajta I primjenjuje S-box na svaki od ta 4 bajta da bi proizvela izlaz (ponovo *Word*).
XOR	Ekskluzivni OR
$\oplus$	Ekskluzivni OR

$\otimes$	Množenje dva polinoma (od kojih je svaki stepena manjeg od 4) na koje se onda primjeni operacija Modulo (ostatak pri dijeljenju) sa $X^4 + 1$
•	Množenje konačnih polja

Tabela 2: Parametri, simboli i funkcije

2.1.2.1.1 Ulazi i izlazi

Ulazi i izlazi u AES algoritam sastoje se od nizova od po 128 bita. Ove sekvence se još nazivaju i blokovi, a broj bita od kojih se oni sastoje naziva se još i njihova dužina. Ključ AES algoritma je niz od 128, 192 ili 256 bita (FIPS197, 2001).

Biti u ovakvom nizu su pobrojani od nule pa do *n-1*, gdje je n dužina takvog niza.

Broj ***i*** koji je pridružen svakom bitu naziva se indeks. On se nalazi u jednom od sljedećih rangova:

$0 \leq i < 128$,

$0 \leq i < 192$ ili

$0 \leq i < 256$.

Prema alternativnoj reprezentaciji (Gladman, 2007) ulaz, izlaz i ključ su jednodimenzioni nizovi bajta gdje se bajt *n* sastoji od bita na pozicijama *8n* do *8n+7*, tako da se n-ta sekvenca (n-ti bajt) može mapirati pozicijama *8n+i*, za *0<=i<8*.

2.1.2.1.2 Bajt

Bajt je osnovna jedinica obrade podataka u AES algoritmu. Ulazi, izlazi i inicijalni ključevi se obrađuju tako da se podijele u nizove bajta (FIPS197, 2001).

Ako imamo sekvencu bajta označenu sa a, tada n-ti bajt referenciramo preko a_n ili a[n], pri čemu n leži u jednom od rangova 0 <= n < 16, 0 <= n < 24 ili 0 <= n < 32 (Gladman, 2007).

Svaki bajt se u AES algoritmu predstavlja kao niz bita između vitičastih zagrada

$$\{ b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0\}. \quad (1)$$

Ovi biti se interpretiraju i kao elementi konačnog polja i prikazuju pomoću polinoma (FIPS197, 2001):

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0x^0. \quad (2)$$

Na primjer, bajt {01100011} je polinom

$$x^6 + x^5 + x^1 + 1. \quad (3)$$

2.1.2.1.3 Niz bajta

Niz bajta je u standardu predstavljen u slijedećoj formi:

$$a_0a_1a_2 \ldots a_{15} \quad (4)$$

Redoslijed bajta i bita iz ulazne sekvence od 128 bita

$b_0\, b_1\, b_2 \ldots b_{126}\, b_{127}$ (5)

se izvodi kako slijedi:

$a_0 = \{b_0, b_1, b_2, \ldots, b_7\}$ (6)

$a_1 = \{b_8, b_9, b_{10}, \ldots, b_{15}\}$

...

$a_{15} = \{b_{120}, b_{121}, b_{122}, \ldots, b_{127}\}$

Sljedeća tabela prikazuje kako su pobrojani biti u okviru svakog bajta:

Ulazna sekvenca	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	....
Broj bajta	0								1								2								....
Brojevi bita u bajtu	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	....

Tabela 3: Biti u okviru bajta

2.1.2.1.4 Niz Stanje (State)

Kako se navodi u (FIPS197, 2001), sve operacije ovog algoritma obavljaju se nad dvodimenzionalnim nizovima bajta koji se nazivaju Stanje (State). Svaki niz Stanje se sastoji od 4 reda. Svaki red za AES algoritam sadrži Nb = 4 bajta, gdje je

Nb = dužina bloka / 32 = 128 / 32 = 4 (7)

Ako bafer Stanje označimo sa *s*, svaki bajt stanja ima dva indeksa *r* i *c*, za koje važi:

$0 \le r < 4$ i (8)

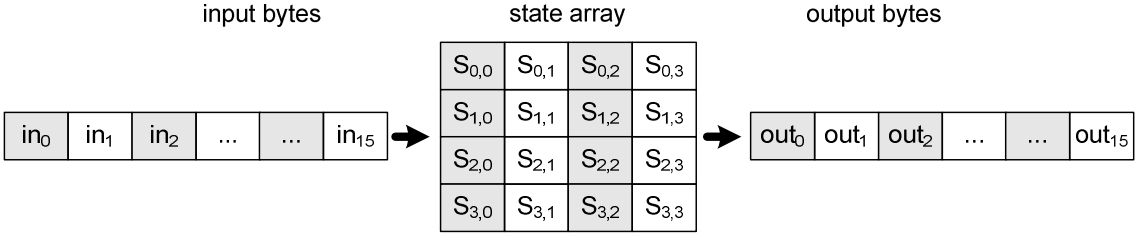
$0 \le c < Nb$

Na ovaj način, svakom bajtu Stanja moguće je pristupati pomoću $s_{r,c}$ ili s[r, c]. Pošto je u standardu definisana dužina bloka od 128 bita, u AES-u je Nb = 4.

Ulazni bajti se mapiraju u niz stanje prema sljedećem redoslijedu (Daemen and Rijmen, 2002):

s[0, 0], s[1, 0], s[2, 0], s[3, 0], s[0, 1], s[1, 2], s[2, 1], s[3, 1],... (9)

Prema (FIPS197, 2001), u slučaju šifrovanja, ulazni niz bajta in0, in1, … in15 se kopira u niz Stanje. Zatim se sve operacije obavljaju nad stanjem, a onda se rezultat kopira u izlazni niz kao out_0, out_1, … out_{15}.



Slika 1: Kopiranje podataka u Stanje i iz Stanja

Na početku, prilikom šifrovanja ili dešifrovanja, ulaz se kopira u stanje po sljedećem obrascu (Gladman, 2007):

s[r, c] = in[r + 4c], gdje je (10)

$0 \leq r < 4$ i

$0 \leq c < Nb$

Na kraju operacije šifrovanja ili dešifrovanja, Stanje se kopira u izlazni niz po sljedećem obrascu (Gladman, 2007):

out[r + 4c] = s[r, c], gdje je (11)

$0 \leq r < 4$ i

$0 \leq c < Nb$

2.1.2.1.5 Nizovi 32-bitnih riječi u sastavu kolona i kao dijelovi ključa

Četiri bajta u svakoj koloni niza Stanje mogu da se posmatraju ili kao niz od četiri bajta indeksiran prema broju reda r ili kao jedna 32-bitna riječ. Zbog toga niz Stanje može da se posmatra i kao jednodimenzioni niz riječi za koje broj kolone predstavlja indeks u nizu (Gladman, 2007). Ako broj reda r iskoristimo kao indeks za ovakvu riječ, imamo (FIPS197, 2001):

$w_0 = s_{0,0}s_{1,0}s_{2,0}\ s_{3,0}$ (12)
$w_2 = s_{0,2}s_{1,2}s_{2,2}\ s_{3,2}$
$w_1 = s_{0,1}s_{1,1}s_{2,1}\ s_{3,1}$
$w_3 = s_{0,3}s_{1,3}s_{2,3}\ s_{3,3}$

za $0 <= r < 4$.

Prošireni ključ ovog algoritma se predstavlja i kao niz 32-bitnih riječi koji se kreira na osnovu inicijalnog ključa prema redoslijedu koji je određen sa *r* i to tako da se bajt *4n+r* inicijalnog ključa kopira u bajt *r* riječi *w[n]* proširenog ključa. Algoritam pri izvršenju prolazi kroz odgovarajući broj rundi tokom kojih koristi *Nk* riječi iz proširenog ključa (Gladman, 2007).

2.1.2.2 Matematičke pretpostavke

Svi bajti algoritma AES se interpretiraju kao elementi konačnih polja pomoću notacije koja je predstavljena jednačinama 4, 5 i 6 (FIPS197, 2001). Veliki dio algebarskih operacija algoritma AES odvija se u konačnim ili Galoa poljima. Da bi definisali Galoa polja moramo se prisjetiti i pojmova prstena, Abelove grupe i polja.

Prema (Roman, 2006) prsten (R, +, *) je algebarska struktura sastavljena od skupa R i operacija „+" i „*" (običnog sabiranja i množenja) koja ima slijedeće osobine:

- (R, +) je Abelova grupa sa neutralnim elementom 0. (Abelova grupa je grupa u kojoj važi komutativnost - a•b=b•a).
- Operacija „•" je asocijativna (a•b)•c = a•(b•c).
- Za svako a $\in$ R postoji neutralni element 1 za operaciju „•"
- Operacija „•" je distributivna u odnosu na „+": a• (b+c) = (a•b) + (a•c)

Ako za element a prstena R postoji b $\in$ R tako da važi a•b=1, onda b nazivamo inverznim elementom elementa a.

Polje (Roman, 2006) je komutativni prsten u kome svaki element (osim nule) ima multiplikativni inverzni element. Konačno polje je polje koje sadrži konačan broj elemenata (Pleskonjić et al. 2007).

Konačna polja se definišu na sljedeći način (Jacobson, 2009):

- Red, ili broj elemenata konačnog polja daje se u formi p^n, gdje je p prost broj koji se naziva karakteristika polja, a n je pozitivan cijeli broj.
- Za svaki prost broj i i pozitivan cijeli broj n, postoji konačno polje sa p^n elemenata.
- Bilo koja dva konačna polja sa istim brojem elemenata su izomorfna.

Ovakva polja označavamo sa F_{p^n} ili $GF(p^n)$ (Pleskonjić et al. 2007).

Svaki bajt podataka u AES algoritmu se tretira kao niz bita koji su elementi konačnog polja, po slijedećoj notaciji:

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0 = \sum_{i=0}^{7} b_i x^i \qquad (13)$$

U AES algoritmu, elementi konačnih polja se mogu dodavati i množiti, ali po pravilima različitim od onih koja se koriste za normalne operacije množenja i sabiranja dva broja.

2.1.2.2.1 Sabiranje

Sabiranje dva elementa konačnog polja u AES algoritmu ostvaruje se „dodavanjem“ odgovarajućih eksponenata polinoma u dva elementa (FIPS197, 2001). Dodavanje se u GF(2) izvodi pomoću XOR operacije (označava se sa $\oplus$) tako da je:

$\oplus$	0	1
0	0	1
1	1	0

Tabela 4: XOR operacija

Na primjer:

$(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) = x^7 + x^6 + x^4 + x^2$, kao polinom: (14)
{01010111} $\oplus$ {10000011} = {11010100} , binarno
{57} $\oplus$ {83} = {d4} , heksadecimalno

2.1.2.2.2 Množenje

Množenje u konačnim poljima je mnogo teže nego sabiranje a postiže se množenjem dva polinoma i prikupljanjem odgovarajućih eksponenata u rezultatu (Gladman, 2007). AES-ovo množenje se u $GF(p^n)$ označava sa „ • “ (FIPS197, 2001). Ovo množenje se izvodi tako da se dva polinoma pomnože, a zatim se pronađe njihov moduo po nesvodivom polinomu stepena 8. Polinom je nesvodiv ako je djeljiv samo sa 1 i sa samim sobom. Za AES algoritam, nesvodiv polinom je:

$m(x) = x^8 + x^4 + x^3 + x + 1$ (15)

Ovaj polinom se u heksadecimalnoj notaciji može zapisati kao:

{01}{1b} (16)

Ako se na ovaj način pomnoži 57h i 83h, dobija se (Gladman, 2007):

{57} • {83} = {c1} (17)

1010111 • 10000011 = 11000001

$(x^6 + x^4 + x^2 + x + 1)(x^7 + x + 1) = x^{13} + x^{11} + x^9 + x^8 + x^7 +$

$x^7 + x^5 + x^3 + x^2 + x +$

$x^6 + x^4 + x^2 + x + 1$

$= x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1$

Na kraju pronalazimo moduo:

$(x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1) \bmod (x^8 + x^4 + x^3 + x + 1) =$ (18)

$x^7 + x^6 + 1$

Dijeljenje sa ovim polinomom obezbjeđuje da će rezultat uvijek biti binarni polinom stepena manjeg od 8 tako da ga možemo predstaviti jednim bajtom.

2.1.2.2.3 Množenje sa x ili množenje ponovljenim šiftom

Prema (FIPS197, 2001), ako treba da pomnožimo sljedeći polinom:

$b(x) = b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0$ (19)

sa polinomom x, dobićemo rezultat:

$b_7x^8 + b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0x^1$ (20)

Rezultat množenja xb(x) se zatim redukuje na stepen 8 pomoću modula po nesvodivom polinomu m(x). Ako je $b_7 = 0$, polinom je već u redukovanoj formi. Ako je $b_7 = 1$, redukovanje se obavlja oduzimanjem (pomoću xor operacije) polinoma m(x).

Znači, na nivou bita, množenje sa x ({00000010} ili {02}) može se obaviti pomoću lijevog šifta nakon kojeg (uslovno) slijedi xor sa {00011011} ili {1b}.

Ova operacija primjenjena na bajtu podataka prema (FIPS197, 2001) se naziva xtime(). Množenje višim stepenima od x može se implementirati ponovljenom primjenom xtime() operacije.

Na primjer, ako treba da pomnožimo {57} • {13} = {fe}, koristićemo slijedeću tabelu:

(21)

{57} • {01} = {57}	1
{57} • {02} = xtime({57}) = {ae}	1
{57} • {04} = xtime({ae}) = {47}	0
{57} • {08} = xtime({47}) = {8e}	0
{57} • {10} = xtime({8e}) = {07}	1

Prema ovome, imajući u vidu da je {13h} = {10011b} ako treba da pomnožimo {57} • {13} = {fe}, koristićemo:

{57} • {13} = {57} • ({01} ⊕{02} ⊕{10}) (22)

= {57} ⊕{ae} ⊕{07}

= {fe}

Alternativno opis množenja sa x daje se u (Gladman, 2007), gdje se navodi da je element konačnog polja *{0000 0010}* polinom *x*. To znači da će množenje bilo kog drugog polinoma sa *x* uvećati sve njegove eksponente za jedan. Ovo je ekvivalentno šiftanju njegove reprezentacije pomoću niza bita za jedan bit ulijevo, tako da će bit sa pozicije *i* biti šiftan na poziciju *i+1*. Ako je prije šifta bio postavljen bit na krajnjoj lijevoj poziciji (poziciji 7) njegov šift će izazvati prekoračenje (*overflow*). U tom slučaju se rezultatu dodaje modularni polinom da bi se rezultat vratio u okvire jednog bajta.

Na primjer, kada se *{11001000}* množi sa *x* odnosno sa *{00000010}* dobija se rezultat *1{10010000}*. Ovo prekoračenje se uklanja dodavanjem modularnog polinoma *1{00011011}*, tako da se dobije konačni rezultat *{10001011}* (Gladman, 2007).

2.1.2.2.4 Množenje konačnih polja pomoću tabela

Prema (Gladman, 2007), ako se određeni elementi Rijndael-ovih (konačnih, Galoa) polja, koji se nazivaju generatori, uzastopno množe sami sobom, oni generišu svih 255 nenultih vrijednosti u polju. Dakle, u Rijndael-ovim Galoa poljima eksponenti se dobijaju ponovljenim množenjem istim brojem, a svi brojevi polja (osim nule, znači brojevi 1-255) se mogu dobiti na ovaj način – ponovljenim množenjem istim brojem. Ovo važi samo za neke, ne za sve elemente polja.

Npr. ako broj 3 množimo 255 puta, imamo:

1 • 3 =	3^1	= 3
3 • 3 =	3^2	= 5
5 • 3 =	3^3	= 15
15 • 3 =	3^3	= 17
17 • 3 =	3^5	= 51
51 • 3 =	3^6	= 85
85 • 3 =	3^7	= 255
255 • 3 =	3^8	= 26
...		
..		
82 • 3 =	3^{254}	= 246
246 • 3 =	3^{255}	= 1
1 • 3 =	*$3^{1\,(256)}$*	*= 3*

Tabela 5: Eksponenciranje za broj 3

Kada kreiramo listu stepena nekog generatora g^n, kreirali smo svih 255 nenultih elemenata Galoa polja (Gladman, 2007). Nakon što n dosegne vrijednost 256, postupak se ponavlja, jer je g^{255} jednako {01}.

Ako ovako dobijene rezultate sortiramo, dobićemo brojeve od 1 do 255. Naravno, ovo važi samo za određene elemente polja koji se, kako je navedeno, nazivaju generatori. Ako npr. broj 2 množimo 255 puta, nećemo dobiti sve brojeve od 1 do 255, već brojeve koji se ponavljaju u grupama po 5.

U Rijndael-ovim Galoa poljima postoje sljedeći generatori:

```
3 5 6 9 11 14 17 18 19 20 23 24 25 26 28 30 31 33 34 35 39 40 42 44 48
49 60 62 63 65 69 70 71 72 73 75 76 78 79 82 84 86 87 88 89 90 91 95
100 101 104 105 109 110 112 113 118 119 121 122 123 126 129 132 134 135
136 138 142 143 144 147 149 150 152 153 155 157 160 164 165 166 167 169
170 172 173 178 180 183 184 185 186 190 191 192 193 196 200 201 206 207
208 214 215 218 220 221 222 226 227 229 230 231 233 234 235 238 240 241
244 245 246 248 251 253 254 255
```

Tabela 6: Generatori

Prema (Gladman, 2007), vrijednosti eksponenata koje formiraju svaki element polja mogu se posmatrati kao logaritmi, što omogućava da se množenje prevede u sabiranje. Dva elementa $a=g^{\alpha}$ i $b=g^{\beta}$ daju proizvod $a \bullet b = g^{\alpha+\beta}$. Pomoću tabele "logaritama" u kojoj bi bili izlistani svi eksponenti generatora za svaki element konačnog polja mogu da se dobiju eksponenti α i β koji odgovaraju elementima a i b, a uvrštavanje tih vrijednosti daje eksponent od g kao rezultat. Inverzna tabela može se koristiti da se pronađe element proizvoda.

e5	0	1	2	3	4	5	6	7	8	9	a	**b**	c	d	e	f
0?	--	00	c8	08	91	10	d0	36	5a	3e	d8	**43**	99	77	fe	18
1?	23	20	07	70	a1	6c	0c	7f	62	8b	40	**46**	c7	4b	e0	0e
2?	eb	16	e8	ad	cf	cd	39	53	6a	27	35	**93**	d4	4e	48	c3
3?	2b	79	54	28	09	78	0f	21	90	87	14	**2a**	a9	9c	d6	74
4?	b4	7c	de	ed	b1	86	76	a4	98	e2	96	**8f**	02	32	1c	c1
5?	33	ee	ef	81	fd	30	5c	13	9d	29	17	**c4**	11	44	8c	80
6?	**f3**	**73**	**42**	**1e**	**1d**	**b5**	**f0**	**12**	**d1**	**5b**	**41**	**a2**	d7	2c	e9	d5
7?	59	cb	50	a8	dc	fc	f2	56	72	a6	65	2f	9f	9b	3d	ba
8?	7d	c2	45	82	a7	57	b6	a3	7a	75	4f	ae	3f	37	6d	47
9?	61	be	ab	d3	5f	b0	58	af	ca	5e	fa	85	e4	4d	8a	05
a?	fb	60	b7	7b	b8	26	4a	67	c6	1a	f8	69	25	b3	db	bd
b?	66	dd	f1	d2	df	03	8d	34	d9	92	0d	63	55	aa	49	ec
c?	bc	95	3c	84	0b	f5	e6	e7	e5	ac	7e	6e	b9	f9	da	8e
d?	9a	c9	24	e1	0a	15	6b	3a	a0	51	f4	ea	b2	97	9e	5d
e?	22	88	94	ce	19	01	71	4c	a5	e3	c5	31	bb	cc	1f	2d
f?	3b	52	6f	f6	2e	89	f7	c0	68	1b	64	04	06	bf	83	38

Tabela 7: Tabela "logaritama"

U ovoj tabeli (tabela "logaritama"), vrijednost na presjeku nekog reda i kolone (a2) predstavlja eksponent broja e5 (koliko puta smo pomnožili e5xe5xe5...), a zaglavlje reda i kolone (6? i b = 6b) čine rezultat podizanja broja e5 na eksponent a2. Naravno, prvo polje u tabeli je prazno – logaritam broja nula ne postoji.

Primjer:

$0xe5^{\mathbf{0xa2}}=0x6b$; heksadecimalno ili
$229^{\mathbf{162}}=107$; decimalno

Dakle, tabela logaritama nam prikazuje koliko puta moramo pomnožiti broj e5 sa samim sobom da bi dobili bilo koji (ne nulti) broj u Rijndael-ovom Galoa polju.

Kada bilo koji generator višestruko eksponenciramo, nakon 255 eksponenciranja dostižemo originalni broj, kao u slijedećoj tabeli (u heksadecimalnoj notaciji):

e5	0	1	**2**	3	4	5	6	7	8	9	a	b	c	d	e	f
0?	**01**	e5	**4c**	b5	fb	9f	fc	12	03	34	d4	c4	16	ba	1f	36
1?	05	5c	**67**	57	3a	d5	21	5a	0f	e4	a9	f9	4e	64	63	ee
2?	11	37	**e0**	10	d2	ac	a5	29	33	59	3b	30	6d	ef	f4	7b
3?	55	eb	**4d**	50	b7	2a	07	8d	ff	26	d7	f0	c2	7e	09	8c
4?	1a	6a	**62**	0b	5d	82	1b	8f	2e	be	a6	1d	e7	9d	2d	8a

5?	72	d9	**f1**	27	32	bc	77	85	96	70	08	69	56	df	99	94
6?	a1	90	**18**	bb	fa	7a	b0	a7	f8	ab	28	d6	15	8e	cb	f2
7?	13	e6	**78**	61	3f	89	46	0d	35	31	88	a3	41	80	ca	17
8?	5f	53	**83**	fe	c3	9b	45	39	e1	f5	9e	19	5e	b6	cf	4b
9?	38	04	**b9**	2b	e2	c1	4a	dd	48	0c	d0	7d	3d	58	de	7c
a?	**d8**	**14**	**6b**	87	47	e8	79	84	73	3c	bd	92	c9	23	8b	97
b?	95	44	dc	ad	40	65	86	a2	a4	cc	7f	ec	c0	af	91	fd
c?	f7	4f	81	2f	5b	ea	a8	1c	02	d1	98	71	ed	25	e3	24
d?	06	68	b3	93	2c	6f	3e	6c	0a	b8	ce	ae	74	b1	42	b4
e?	1e	d3	49	e9	9c	c8	c6	c7	22	6e	db	20	bf	43	51	52
f?	66	b2	76	60	da	c5	f3	f6	aa	cd	9a	a0	75	54	0e	01

Tabela 8: Tabela eksponenata

U prethodnoj tabeli, koju ćemo nazvati tabela eksponenata, zaglavlje reda (npr. a?) i zaglavlje kolone (npr.2) formiraju eksponent (koliko puta množimo – npr. 0xa2 = 162 puta). Na presjeku navedenog reda i kolone nalazi se rezultat podizanja na dati eksponent u Rijndael-ovim Galoa poljima. Naravno, u prethodnoj tabeli, na presjeku reda 0? i kolone 0 nalazi se vrijednost 01 – x^0 je uvijek 1.

Primjer:

$0xe5^{\mathbf{0xa2}}=0x6b$; heksadecimalno ili
$229^{\mathbf{162}}=107$; decimalno

2.1.2.2.5 Polinomi čiji su koeficienti u GF(2^8)

Kako se navodi u (FIPS197, 2001), u AES algoritmu se mogu definisati i polinomi stepena manjeg od 4, čiji su koeficijenti elementi GF(2^8):

$$a(x) = a_3x^3 + a_2x^2 + a_1x^1 + a_0 \quad (23)$$

Ovaj polinom se može predstaviti i kao Riječ (Word) u formi [a_0, a_1, a_2, a_3], pri čemu, prema (Gladman, 2007) indeksi u ovakvoj notaciji rastu s lijeva u desno.

Treba primijetiti da se ovako definsani polinomi ponašaju nešto drugačije nego polinomi korišteni pri definsanju elemenata konačnih polja. Ovi koeficijenti su u stvari elementi konačnih polja (ranije su cijeli polinomi predstavljali elemente konačnih polja). Znači ovdje su to bajti umjesto bita. Kako se navodi u (FIPS197, 2001), množenje ovakvih polinoma zahtijeva drugačiji polinom za redukciju, dok se kao oznaka ovako definisanog množenja koristi simbol $\otimes$ (x u krugu).

Prema (Gladman, 2007) ako pored prethodno definisanog polinoma a(x) imamo i polinom:

$$b(x) = b_3x^3 + b_2x^2 + b_1x^1 + b_0 \quad (24)$$

operacija sabiranja se može izvesti pomoću dodavanja odgovarajućih koeficijenata konačnih polja, što odgovara izvođenju XOR operacije nad odgovarajućim bajtima svake riječi ili XOR operaciji nad cijelim riječima. Treba imati u vidu da se ovdje varijabla x ne koristi u istom kontekstu kao u ranijem opisu pojedinih elemenata konačih polja.

Operacija sabiranja ova dva polinoma se, prema (FIPS197, 2001), svodi na sabiranje koeficijenata uz iste eksponente (pomoću xor operacije, dakle sabiranje u GF(2^8).

$$a(x) + b(x) = (a_3 \oplus b_3)x^3 + (a_2 \oplus b_2)x^2 + (a_1 \oplus b_1)x^1 + (a_0 \oplus b_0) \quad (25)$$

Prema (FIPS197, 2001), množenje se izvodi u dva koraka. Najprije se koristi algebarska ekspanzija:

$$\begin{aligned} a(x) \bullet b(x) &= (a_3x^3 + a_2x^2 + a_1x^1 + a_0) \bullet (b_3x^3 + b_2x^2 + b_1x^1 + b_0) \\ = \quad & (a_3 \bullet b_3)x^6 + (a_2 \bullet b_3)x^5 + (a_1 \bullet b_3)x^4 + (a_0 \bullet b_3)x^3 + \\ & (a_3 \bullet b_2)x^5 + (a_2 \bullet b_2)x^4 + (a_1 \bullet b_2)x^3 + (a_0 \bullet b_2)x^2 + \\ & (a_3 \bullet b_1)x^4 + (a_2 \bullet b_1)x^3 + (a_1 \bullet b_1)x^2 + (a_0 \bullet b_1)x^1 + \\ & (a_3 \bullet b_0)x^3 + (a_2 \bullet b_0)x^2 + (a_1 \bullet b_0)x^1 + (a_0 \bullet b_0) \end{aligned} \tag{26}$$

Zatim se koeficijenti istog stepena skupe, da bi dali:

$$c(x) = c_6x^6 + c_5x^5 + c_4x^4 + c_3x^3 + c_2x^2 + c_1x^1 + c_0 \tag{27}$$

gdje je:

$$\begin{aligned} c_0 &= (a_0 \bullet b_0), \\ c_1 &= (a_0 \bullet b_1) \oplus (a_1 \bullet b_0), \\ c_2 &= (a_0 \bullet b_2) \oplus (a_1 \bullet b_1) \oplus (a_2 \bullet b_0), \\ c_3 &= (a_0 \bullet b_3) \oplus (a_1 \bullet b_2) \oplus (a_2 \bullet b_1) \oplus (a_3 \bullet b_0), \\ c_4 &= (a_1 \bullet b_3) \oplus (a_2 \bullet b_2) \oplus (a_3 \bullet b_1), \\ c_5 &= (a_2 \bullet b_3) \oplus (a_3 \bullet b_2), \\ c_6 &= (a_3 \bullet b_3). \end{aligned} \tag{28}$$

pri čemu su • i $\oplus$ množenje i sabiranje (XOR) u konačnim poljima.

Pošto dobijeni rezultat, c(x), ne predstavlja četvorobajtnu riječ, u drugom koraku moramo izvršiti redukovanje ovakvog polinoma na polinom čiji je stepen manji od četiri. U AES algoritmu (FIPS197, 2001) i Rijndael algoritmu (Gladman, 2007) to se izvodi pomoću sljedećeg polinoma:

$$x^4 + 1 \tag{29}$$

tako da je:

$$x^i \bmod (x^4 + 1) = x^{i \bmod 4} \tag{30}$$

Sada je modularni proizvod $a(x) \otimes b(x)$ dat slijedećim polinomom:

$$d(x) = d_3x^3 + d_2x^2 + d_1x^1 + d_0 \tag{31}$$

pri čemu je:

$$\begin{aligned} & d(x) = c(x) \bmod (x^4 + 1) = (c_6x^6 + c_5x^5 + c_4x^4 + c_3x^3 + c_2x^2 + c_1x^1 + c_0) \bmod (x^4 + 1) = \\ & c_6x^{6 \bmod 4} + c_5x^{5 \bmod 4} + c_4x^{4 \bmod 4} + c_3x^{3 \bmod 4} + c_2x^{2 \bmod 4} + c_1x^{1 \bmod 4} + c_0x^{0 \bmod 4} = \\ & c_6x^2 + c_5x^1 + c_4x^0 + c_3x^3 + c_2x^2 + c_1x^1 + c_0x^0 = \\ & c_3x^3 + c_6x^2 + c_2x^2 + c_5x^1 + c_1x^1 + c_4x^0 + c_0x^0 = \\ & d_3x^3 + d_2x^2 + d_1x^1 + d_0 \end{aligned} \tag{32}$$

gdje je:

$$d_0 = c_0 \oplus c_4 = (a_0 \bullet b_0) \oplus (a_1 \bullet b_3) \oplus (a_2 \bullet b_2) \oplus (a_3 \bullet b_1), \tag{33}$$

$d_1 = c_1 \oplus c_5 = (a_0 \bullet b_1) \oplus (a_1 \bullet b_0) \quad \oplus \quad (a_2 \bullet b_3) \oplus (a_3 \bullet b_2)$,

$d_2 = c_2 \oplus c_6 = (a_0 \bullet b_2) \oplus (a_1 \bullet b_1) \oplus (a_2 \bullet b_0) \quad \oplus \quad (a_3 \bullet b_3)$,

$d_3 = c_3 = (a_0 \bullet b_3) \oplus (a_1 \bullet b_2) \oplus (a_2 \bullet b_1) \oplus (a_3 \bullet b_0)$.

Ako je a(x) fiksni polinom, izračunavanje modularnog proizvoda može se predstaviti u matričnoj formi:

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Slika 2: Modularni proizvod a(x) $\otimes$ b(x)

Kako je polinom $x^4 + 1$ nesvodiv u GF(2^8), rezultat množenja ovakvim polinomom ne mora obavezno imati inverzni multiplikativni element. Prema (FIPS197, 2001), AES algoritam navodi fiksni polinom koji ima inverzni element:

$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$ (34)

$a^{-1}(x) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\}$

Još jedan polinom koji se koristi u AES algoritmu (u RotWord() funkciji) ima koeficijente:

$a^0a^1a^2 = \{00\}$ i (35)

$a^3 = \{01\}$

što daje polinom x^3. Ako ovaj polinom uvrstimo u prethodnu matricu, imamo:

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} 0_0 & 1_3 & 0_2 & 0_1 \\ 0_1 & 0_0 & 1_3 & 0_2 \\ 0_2 & 0_1 & 0_0 & 1_3 \\ 1_3 & 0_2 & 0_1 & 0_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 0*b_0 + 1*b_1 + 0*b_2 + 0*b_3 \\ 0*b_0 + 0*b_1 + 1*b_2 + 0*b_3 \\ 0*b_0 + 0*b_1 + 0*b_2 + 1*b_3 \\ 1*b_0 + 0*b_1 + 0*b_2 + 0*b_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_0 \end{bmatrix}$$

Slika 3: Transformacija RotWord

što daje efekat rotacije bajta jedne riječi, odnosno polinom [b0, b1, b2, b3] se transformiše u [b1, b2, b3, b0] (Gladman, 2007).

2.1.2.2.6 Dijeljenje

Imajući u vidu način realizacije množenja konačnih polja pomoću tabela predstavljen u (Gladman, 2007) i sekciji 2.1.2.2.4, operacija dijeljenja brojeva *a* i *b* nad Rijndael-ovim Galoa poljima realizuje se pronalaženjem logaritma od *a* nakon čega slijedi oduzimanje logaritma od *b* po modulu 255, kako slijedi:

a/b = (ltable[a] – ltable[b]) mod 255 (36)

2.1.2.2.7 Multiplikativni inverzni elementi

Prema (Carrell, 2005) za svako *a* <> 0 u polju *F*, postoji element a^{-1} koji se naziva multiplikativni inverzni element takav da je $axa^{-1}=1$. Multiplikativni inverzni element računa se kao:

mul_inv = 1/a (37)

U nastavku slijedi pseudo kod za pronalaženje multiplikativnog inverznog elementa za konačna (Galoa) polja GF(2^8)

```
galoa_mul_inverzni(byte in, byte out)
begin
      if (in = 0) then
        out = 0
      else
        out = exptable[(255 - ltable[a])];
end
```

Listing 1: Multiplikativni inverzni element

Slijedi tabela svih multiplikativno inverznih elemenata u Rijndael-ovim Galoa poljima:

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
00	--	01	8d	f6	cb	52	7b	d1	e8	4f	29	c0	b0	e1	e5	c7
10	74	b4	aa	4b	99	2b	60	5f	58	3f	fd	cc	ff	40	ee	b2
20	3a	6e	5a	f1	55	4d	a8	c9	c1	0a	98	15	30	44	a2	c2
30	2c	45	92	6c	f3	39	66	42	f2	35	20	6f	77	bb	59	19
40	1d	fe	37	67	2d	31	f5	69	a7	64	ab	13	54	25	e9	09
50	ed	5c	05	ca	4c	24	87	bf	18	3e	22	f0	51	ec	61	17
60	16	5e	af	d3	49	a6	36	43	f4	47	91	df	33	93	21	3b
70	79	b7	97	85	10	b5	ba	3c	b6	70	d0	06	a1	fa	81	82
80	83	7e	7f	80	96	73	be	56	9b	9e	95	d9	f7	02	b9	a4
90	de	6a	32	6d	d8	8a	84	72	2a	14	9f	88	f9	dc	89	9a
a0	fb	7c	2e	c3	8f	b8	65	48	26	c8	12	4a	ce	e7	d2	62
b0	0c	e0	1f	ef	11	75	78	71	a5	8e	76	3d	bd	bc	86	57
c0	0b	28	2f	a3	da	d4	e4	0f	a9	27	53	04	1b	fc	ac	e6
d0	7a	07	ae	63	c5	db	e2	ea	94	8b	c4	d5	9d	f8	90	6b
e0	b1	0d	d6	eb	c6	0e	cf	ad	08	4e	d7	e3	5d	50	1e	b3
f0	5b	23	38	34	68	46	03	8c	dd	9c	7d	a0	cd	1a	41	1c

Tabela 9: Multiplikativni inverzni elementi u Rijndael-ovim Galoa poljima

2.1.2.3 Transformacije algoritma AES

Rijndael je simetrični blokovski algoritam sa iteracijom ključeva (Daemen and Rijmen, 2002). On se sastoji od uzastopne primjene transformacije runde na blok podataka Stanje. Broj rundi se označava sa Nr a zavisi od veličine bloka i dužine ulaznog ključa.

U standardnom algoritmu AES, dužine ulaznog, izlaznog bloka i stanja su 128 bita (FIPS197, 2001). Ovo se predstavlja pomoću Nb = 4, a odnosi se na broj 32-bitnih riječi (broj kolona) u Stanju.

U ovom algoritmu, dužine inicijalnog ključa su 128, 192 i 256 bita. Dužina ključa predstavlja se sa Nk = 4, 6 ili 8, što se odnosi na broj 32-bitnih riječi u inicijalnom ključu (FIPS197, 2001).

Kako ćemo kasnije vidjeti, da bi algoritam bio funkcionalan, inicijalni ključ se mora razviti do određenog broja bajta, koji zavisi od tipa algoritma (AES-128, AES-192 ili AES-256).

Broj rundi u algoritmu AES zavisi od dužine ključa. Broj rundi predstavlja se sa Nr, gdje je

Nr = 10 kada je Nk = 4,

Nr = 12 kada je Nk = 6,

Nr = 14 kada je Nk = 8.

Sve kombinacije odnosa ključeva, blokova i broja rundi za algoritam AES date su u sljedećoj tabeli:

	Dužina ključa (Nk riječi)	Dužina bloka (Nb riječi)	Broj rundi
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

Tabela 10: Odnos dužina inicijalnog ključa, bloka i broja rundi

I za šifrovanje i za dešifrovanje, AES koristi funkciju runde koja se sastoji od četiri transformacije:

1. Supstitucija bajta pomoću supstitucione tabele (**S-box**), zatim

2. pomjeranje (šiftanje) redova Stanja za različite pomake,

3. miješanje podataka u okviru svake kolone u Stanju,

4. dodavanje Ključa runde Stanju.

Dodavanje ključa runde stanju se vrši jedan put više od broja rundi, što se označava kao (10+1, 12+1 i 14+1).

2.1.2.3.1 Šifrovanje (Transformacija Cipher)

Funkcija šifrovanja u sebi obuhvata veći broj AES-ovih transformacija. Na početku rada pri procesu šifrovanja ulazni podaci se kopiraju u blok Stanje na način kako je to predstavljeno jednačinom 10 (Gladman, 2007).

Tada se inicijalni ključ runde kombinuje sa navedenim podacima, a zatim se Stanje dalje transformiše iterativno u funkciji runde pomoću odgovarajućeg broja iteracija. Po završetku šifrovanja, finalni blok stanje se ponovo kopira na izlaz prema konvenciji koja je data jednačinom 11. Funkcija runde kao parametar uzima dio proširenog ključa koji se sastoji od 32-bitnih riječi od kojih su najmanje 4, 6 ili 8 riječi inicijalizovane po konvenciji koja je opisana u sekciji 2.1.2.1.5. Za Rijndael, dužina ulaznog, izlaznog bloka i stanja uvijek je umnožak broja 32-bitnih riječi Nb, odnosno 4, 6 ili 8, ali AES standard dozvoljava samo dužinu od 4 ovakve riječi za navedene blokove podataka. Dužina ključa je, mjerena u 32-bitnim riječima, za oba algoritma jednaka 4, 6 ili 8 (Gladman, 2007).

Ključevi runde koji se koriste u funkciji runde dobijaju se pomoću Rutine za ekspanziju ključeva (*KeyExpansion*) u obliku niza 4-bajtnih Riječi. Pseudo kod funkcije šifrovanja može se predstaviti u sljedećih nekoliko linija (FIPS197, 2001):

```
/*      Ulaz in         Izlaz out       Kljucevi w      */
Cipher (byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
      byte state[4, Nb]
      state = in
      /* inicijalno dodavanje kljuca */
      AddRoundKey( state, w[0, Nb-1])
```

```
      /* „obicne“ runde */
      For round = 1 step 1 to Nr-1
            SubBytes(state)
            ShiftRows(state)
            MixColumns(state)
            AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
      End for

      /* zavrsna runda */
      SubBytes(state)
      ShiftRows(state)
      AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

      Out = state
End
```

Listing 2: Pseudo kod funkcije šifrovanja

Prema (FIPS197, 2001), individualne transformacije *SubBytes(), ShiftRows(), MixColumns()* i *AddRoundKey()* obrađuju stanje tokom svake runde. Kako se iz pseudo koda vidi, samo završna runda se razlikuje od prethodnih Nr rundi u tome što ne vrši funkciju MixColumns(). U (Daemen and Rijmen, 2002) se navodi da su autori izmjenili nazive nekih transformacija prema sugestijama dr B.Gladmana u odnosu na originalno podneseni dokument.

2.1.2.3.2 Transformacija SubBytes()

Ova transformacija je nelinearna supstitucija koja se vrši na svakom bajtu Stanja nezavisno od drugih bajta. Pri ovome, SubBytes() koristi supstitucionu tabelu (S-Box). Ovaj S-box je invertibilan – ulazni podatak koji pomoću S-boxa proizvede neki izlaz, može se ponovo jednoznačno rekonstruisati na osnovu tog izlaza. S-box se konstruiše pomoću sljedeće dvije transformacije (FIPS197, 2001):

1. Uzimamo multiplikativni inverzni element u konačnom polju GF(2^8). Pri tome se element {00} mapira sam u sebe.
2. Primjenjujemo Afinu transformaciju u GF(2^8), na slijedeći način:

$$b'_i = b_i \oplus b_{(i+4) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+6) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus c_i \qquad (38)$$

za svako i iz $1 \le i < 8$, gdje je

b_i - i^{ti} bit bajta,

c_i - i^{ti} bit bajta c koji ima vrijednost {63h} ili {01100011b}.

U matričnoj formi, jedan element S-box-a može se prikazati na sljedeći način (FIPS197, 2001):

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Slika 4: Element S-Box -a

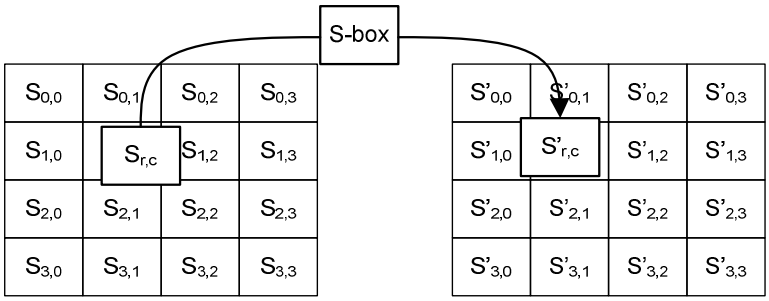
Komplenta supstituciona tabela data je u heksadecimalnoj formi:

	y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
x	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Tabela 11: Izračunata tabela S-box-a

Prema (FIPS197, 2001), ako je $s_{1,1}$ = {53}, onda se vrijednost ove transformacije dobija na presjeku reda sa indeksom 5 i kolone sa indeksom 3 iz supstitucione tabele. Dakle, $s'_{1,1}$ imaće vrijednost {ed}.

Efekat *SubBytes()* transformacije dat je na slijedećoj slici (FIPS197, 2001):



Slika 5: Transformacija SubBytes

Pseudo kod za izračunavanje svih elemenata S-Boxa dat je u slijedećem listingu:

```
function sbox_calc(byte in, byte out)
begin
```

```
  byte c, s;
  s := galoa_mul_inverzni(in);
  out := s;
  for c = 0 step 1 to 3
    s := (s shl 1) or (s shr 7);
    out := out XOR s;
  end for
  out := out XOR 99;  // 0x63
end
```

Listing 3: Pseudo kod za računanje S-Boxa

U (Gladman, 2007) navodi se pseudo kod transformacije *SubBytes()*:

```
SubBytes (byte state[4, Nb])
begin
  for r = 0 step 1 to 3
    for c = 0 step 1 to Nb - 1
      state[r,c] = Sbox[state[r,c]]
    end for
  end for
end
```

Listing 4: Pseudo kod transformacije SubBytes()

2.1.2.3.3 Transformacija ShiftRows()

Prema (FIPS197, 2001), u ovoj transformaciji, bajti zadnja tri reda Stanja se ciklički pomjeraju (šiftaju) za različit broj bajta (*offset*). Prvi red, r = 0, se ne pomjera.

Transformacija *ShiftRows()* se izvodi na slijedeći način:

$$S'_{r,c} = S_{r,\,(c + shift(r, Nb))\ mod\ Nb} \tag{39}$$

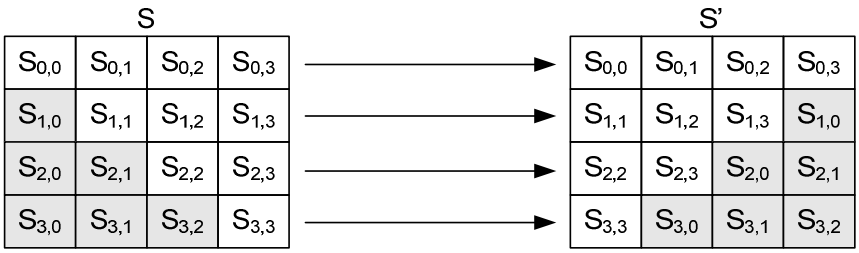
za $0 < r < 4$ i

$0 \le c < Nb$,

gdje vrijednost shift(r, Nb) za algoritam AES zavisi od broja reda:

$$shift(1, 4) = 1;\ \ shift(2, 4) = 2;\ shift(3, 4) = 3; \tag{40}$$

Slijedeća slika opisuje *ShiftRows()* transformaciju algoritma AES (FIPS197, 2001):



Slika 6: Transformacija ShiftRows

U (Gladman, 2007) dat je pseudo kod transformacije ShiftRows():

```
ShiftRows(byte state[4,Nb])
begin
  byte t[Nb]
  for r = 1 step 1 to 3
    for c = 0 step 1 to Nb - 1
      t[c] = state[r, (c + h[r,Nb]) mod Nb]
    end for
    for c = 0 step 1 to Nb - 1
      state[r,c] = t[c]
```

```
    end for
  end for
end
```

Listing 5: Pseudo kod transformacije ShiftRows ()

2.1.2.3.4 Transformacija MixColumns()

Prema (FIPS197, 2001), ova transformacija obrađuje niz Stanje kolonu po kolonu, tretirajući svaku kolonu kao polinom trećeg stepena. Ove kolone se posmatraju kao polinomi u konačnom polju GF(2^8) i kao takvi se zatim množe fiksnim polinomom

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\} \quad (41)$$

po modulu $x^4 + 1$.

Ovo se može predstaviti kao množenje matrica. Ako je s(x) stanje, ova transformacija može se predstaviti kao:

$$s'(x) = a(x) \otimes s(x) \quad (42)$$

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

Slika 7: Transformacija MixColumns kao matrica

za $0 \le c < Nb$.

Kao rezultat ovog množenja, četiri bajta jedne kolone Stanja biće zamijenjeni sa (FIPS197, 2001):

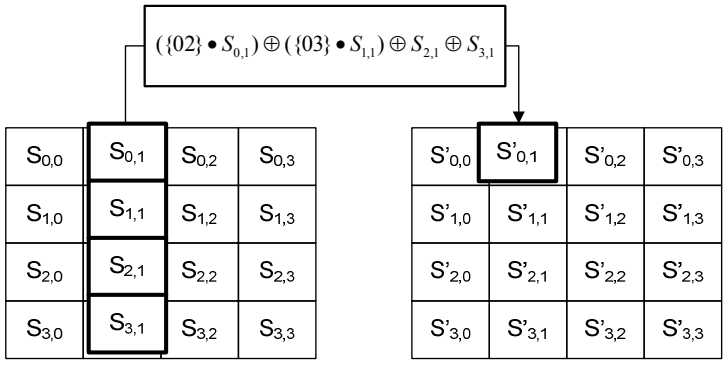
$$S'_{0,c} = (\{02\} \bullet s_{0,c}) \otimes (\{03\} \bullet s_{1,c}) \otimes (\{01\} \bullet s_{2,c}) \otimes (\{01\} \bullet s_{3,c}) \quad (43)$$

$$S'_{1,c} = (\{01\} \bullet s_{0,c}) \otimes (\{02\} \bullet s_{1,c}) \otimes (\{03\} \bullet s_{2,c}) \otimes (\{01\} \bullet s_{3,c})$$

$$S'_{2,c} = (\{01\} \bullet s_{0,c}) \otimes (\{01\} \bullet s_{1,c}) \otimes (\{02\} \bullet s_{2,c}) \otimes (\{03\} \bullet s_{3,c})$$

$$S'_{3,c} = (\{03\} \bullet s_{0,c}) \otimes (\{01\} \bullet s_{1,c}) \otimes (\{01\} \bullet s_{2,c}) \otimes (\{02\} \bullet s_{3,c})$$

Slijedeća slika ilustruje *MixColumns()* transformaciju (FIPS197, 2001):



Slika 8: Transformacija MixColumns - jedan bajt

U (Gladman, 2007) prikazan je pseudo kod transformacije *MixColumns(),*pri čemu FFmul(x, y) vraća proizvod dva elementa konačnih polja (Gladman, 2007)::

```
MixColumns(byte state[4,Nb])
begin
  byte t[4]
  for c = 0 step 1 to Nb - 1
    for r = 0 step 1 to 3
      t[r] = state[r,c]
    end for
    for r = 0 step 1 to 3
      state[r,c] = FFmul(0x02, t[r]) xor
      FFmul(0x03, t[(r + 1) mod 4]) xor
      t[(r + 2) mod 4] xor t[(r + 3) mod 4]
    end for
  end for
end
```

Listing 6: Pseudo kod transformacije MixColumns ()

2.1.2.3.5 Transformacija AddRoundKey()

Transformacija *AddRoundKey()* se (Gladman, 2007) još naziva i *XorRoundKey()*. U okviru ove transformacije, Ključ runde se dodaje Stanju XOR operacijom. Svaki Ključ runde sastoji se od Nb riječi koje dobija iz liste ključeva na osnovu funkcije proširenja ključa. Ovih Nb riječi se pojedinačno dodaju Stanju, tako da vrijedi (FIPS197, 2001):

$$[s'_{0,c}, s'_{1,c}, s'_{2,c}, s'_{3,c}] = [s_{0,c}, s_{1,c}, s_{2,c}, s_{3,c}] \oplus [w_{round*Nb+c}] \quad (44)$$

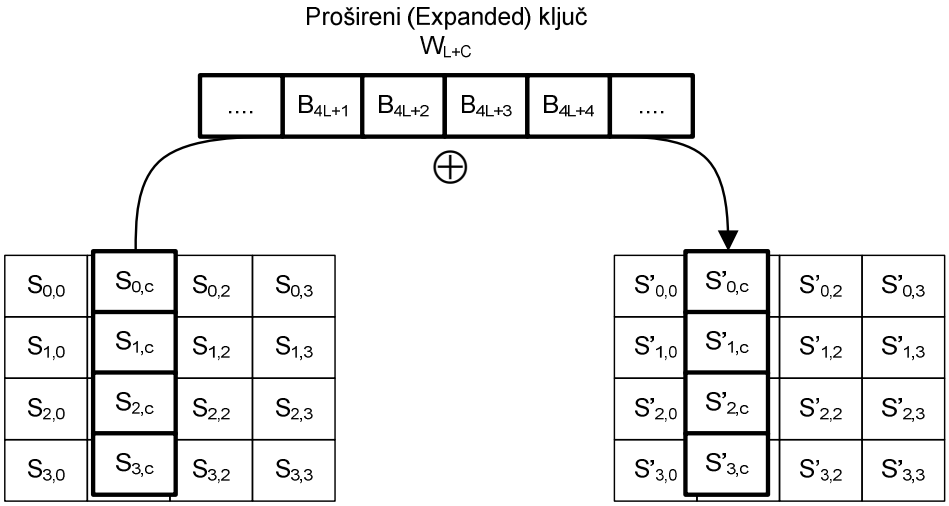
za $0 \le c < Nb$,

gdje su [wi] riječi iz liste ključeva,

dok je runda (*round*) vrijednost u opsegu $0 \le round \le Nr$.

Prilikom šifrovanja, inicijalno dodavanje ključa runde dešava se kada je round = 0, prije prve primjene funkcije runde. U funkciji runde, transformacija AddRoundKey() se izvršava kada je $1 \le round \le Nr$.

Sljedeća slika ilustruje ovu transformaciju:



Slika 9: Dodavanje ključa runde

U (Gladman, 2007) prikazan je pseudo kod transformacije *AddRoundKey()*, koja je tu nazvana *XorRoundKey*:

```
XorRoundKey(byte state[4,Nb], word rk[])
begin
  for c = 0 step 1 to Nb – 1
    for r = 0 step 1 to 3
      state[r,c] = state[r,c] xor xbyte(r, rk[c])
    end for
  end for
end
```

Listing 7: Pseudo kod transformacije XorRoundKey()

2.1.2.3.6 Ekspanzija ključeva

Prema (Gladman, 2007), ključevi runde se izvode iz inicijalnog ključa pomoću funkcije ekspanzije ključeva. Ova funkcija za svaku rundu proizvodi potrebnih Nb riječi koje zajedno sa inicijalnim skupom čine Nb(Nr+1) riječi. Prošireni ključ sastoji se od linearnog niza 4-bajtnih riječi koje se označavaju sa w_i ili w[i], kada i leži u rangu 0 <= i < Nb(Nr + 1).

Prema (FIPS197, 2001), algoritam AES zahtijeva inicijalni ključ K, odnosno zahtijeva inicijalni skup od Nb riječi i onda primjenjuje rutinu ekspanzije ključa da bi generisao prošireni ključ (listu ključeva).

Pseudo-kod koji opisuje ekspanziju ključeva, dat je u sljedećem listingu (FIPS197, 2001):

```
KeyExpansion( byte key[4*Nk], word w[Nb*(Nr + 1)], Nk)
begin
  word temp
  i = 0
  while (i < Nk)
    w[i] = word( key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
    i = i + 1
  end while
  i = Nk
  while ( i < Nb*(Nr + 1) )
    temp = w[i-1]
    if ((i mod Nk) = 0)
      temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
      else if ( (Nk > 6) and ((i mod Nk) = 4)
        temp = SubWord(temp)
    end if
    w[i] = w[i - Nk] xor temp
    i = i + 1
  end while
end
```

Listing 8: Pseudo kod za expanziju ključeva

2.1.2.3.7 Dešifrovanje (transformacija InvCipher)

Funkcija dešifrovanja u sebi obuhvata transformacije *InvShiftRows(), InvSubBytes(), InvMixColumns()* i *AddRoundKey()* (Gladman, 2007). Ova transformacija je inverzna šifrovanju. Prema (FIPS197, 2001), ako joj na ulaz dovedemo šifrat (izlazni podatak iz transformacije *Cipher()*, transformacija *InvCipher()* će restaurirati originalni otvoreni tekst.

Pseudo kod za ovu transformaciju dat je u (FIPS197, 2001):

```
/*         Ulaz in       Izlaz out       Kljucevi w      */
InvCipher (byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
      byte state[4, Nb]
      state = in
```

```
      /* inicijalno dodavanje kljuca */
      AddRoundKey( state, w[Nr*Nb, (Nr+1)*Nb-1])

      /* „obicne" runde */
      For round = Nr-1 step -1 downto 1
            InvShiftRows(state)
            InvSubBytes(state)
            AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
            InvMixColumns(state)
      End for

      /* zavrsna runda */
      InvShiftRows(state)
      InvSubBytes(state)
      AddRoundKey(state, w[0, Nb-1])

      Out = state
End
```

Listing 9: Pseudo kod transformacije InvCipher()

2.1.2.3.8 Transformacija InvShiftRows()

Ova transformacija je inverzna transformaciji *ShiftRows*. Kako se navodi u (Gladman, 2007), operacija *InvShiftRows* se izvršava na zadnja tri reda bloka podataka Stanje pri čemu se oni ciklički pomjeraju (šiftaju) za različit broj bajta (*offset*), ali na suprotnu stranu od prethodno opisane transformacije *ShiftRows*(). Prvi red, r = 0, se ne pomjera.

Transformacija ShiftRows() se izvodi na slijedeći način (FIPS197, 2001):

$$S'_{r,c} = S_{r,\,(c + shift(r, Nb))\ mod\ Nb} \qquad (45)$$

za $0 < r < 4$ i

$0 \le c < Nb$,

Pseudo kod transformacije *InvShiftRows*() predstavljen je u (Gladman, 2007) na sljedeći način:

```
InvShiftRows(byte state[4,Nb])
  byte t[Nb]
  for r = 1 step 1 to 3
    for c = 0 step 1 to Nb - 1
      t[(c + h[r,Nb]) mod Nb] = state[r,c]
    end for
    for c = 0 step 1 to Nb - 1
      state[r,c] = t[c]
    end for
  end for
end
```

Listing 10: Pseudo kod transformacije InvShiftRows ()

2.1.2.3.9 Transformacija InvSubBytes()

U ovoj transformaciji koja je inverzna prethodno opisanoj *SubBytes*() transformaciji, na svaki bajt Stanja se primjenjuje inverzni S-Box. Inverzni S-Box se može dobiti primjenom inverzne afine transformacije na pojedine elemente nakon čega se pronalazi multiplikativni inverzni element u GF(2^8) (FIPS197, 2001).

Za praktičnu primjenu još je jednostavnije izvođenje inverznog S-Boxa na osnovu prethodno formirane tabele za S-Box.

Tabela koja prikazuje inverzni S-Box data je u nastavku (FIPS197, 2001):

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
00	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
10	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
20	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
30	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
40	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
50	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
60	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
70	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
80	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
90	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
a0	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
b0	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
c0	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
d0	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
e0	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
f0	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

Tabela 12: Izračunata tabela za Inverzni S-Box

2.1.2.3.10 Transformacija InvMixColumns()

Prema (FIPS197, 2001), ova transformacija je inverzna prethodno opisanoj *MixColumns*() transformaciji. I ova transformacija obrađuje niz Stanje kolonu po kolonu, tretirajući svaku kolonu kao polinom trećeg stepena. Ove kolone se posmatraju kao polinomi u konačnom polju GF(2^8) i kao takvi se zatim množe fiksnim polinomom

$$a^{-1}(x) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\} \quad (46)$$

po modulu $x^4 + 1$.

Ovo se može predstaviti kao množenje matrica. Ako je s(x) stanje, ova transformacija može se predstaviti kao:

$$s'(x) = a^{-1}(x) \otimes s(x) \quad (47)$$

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

Slika 10: Transformacija InvMixColumns kao matrica

za $0 \leq c < Nb$.

Kao rezultat ovog množenja, četiri bajta jedne kolone Stanja biće zamijenjeni sa:

$$S'_{0,c} = (\{0e\} \bullet s_{0,c}) \otimes (\{0b\} \bullet s_{1,c}) \otimes (\{0d\} \bullet s_{2,c}) \otimes (\{09\} \bullet s_{3,c}) \quad (48)$$

$$S'_{1,c} = (\{09\} \bullet s_{0,c}) \otimes (\{0e\} \bullet s_{1,c}) \otimes (\{0b\} \bullet s_{2,c}) \otimes (\{0d\} \bullet s_{3,c})$$

$$S'_{2,c} = (\{0d\} \bullet s_{0,c}) \otimes (\{09\} \bullet s_{1,c}) \otimes (\{0e\} \bullet s_{2,c}) \otimes (\{0b\} \bullet s_{3,c})$$

$S'_{3,c} = (\{0b\} \bullet s_{0,c}) \otimes (\{0d\} \bullet s_{1,c}) \otimes (\{09\} \bullet s_{2,c}) \otimes (\{0e\} \bullet s_{3,c})$

Pseudo kod transformacije *InvMixColumns*() predstavljen je sljedećim listingom pseudo koda, pri čemu FFmul(x, y) vraća proizvod dva elementa konačnih polja (Gladman, 2007):

```
InvMixColumns(byte block[4,Nb])
begin
  byte t[4]
  for c = 0 step 1 to Nb - 1
    for r = 0 step 1 to 3
      t[r] = block[r,c]
    end for
    for r = 0 step 1 to 3
      block[r,c] =
        FFmul(0x0e, t[r]) xor
        FFmul(0x0b, t[(r + 1) mod 4]) xor
        FFmul(0x0d, t[(r + 2) mod 4]) xor
        FFmul(0x09, t[(r + 3) mod 4])
    end for
  end for
end
```

Listing 11: Pseudo kod transformacije InvShiftRows ()

2.1.2.3.11 Inverzna transformacija AddRoundKey ()

Prema (FIPS197, 2001), transformacija *AddRoundKey*() je sama sebi inverzna, pošto uključuje samo XOR operaciju.

2.1.2.3.12 Dopuna (Padding)

Blok algoritmi, poput DES ili AES algoritma zahtjevaju da na ulazu dobiju podatak koji je jednak nekom umnošku dužine njihovog bloka.

Npr., ako je dužina jednog bloka nekog algoritma 16 bajta, onda na ulazu treba da se pojavi podatak dužine 16, 32, 48, 64 itd. bajta. Ako na ulazu u bilo koju rundu nemamo dovoljan broj bajta, tada moramo da koristimo određene metode kojima dopunjavamo nedostajuće bajte (*padding*).

Standardizovane i često korišćene metode kojima dopunjavamo sadržaj ulaznog podatka su:

1. PKCS7: dopunjavaju se neophodni bajti pomoću vrijednosti koja odgovara broju bajta koje dopunjavamo (RFC 5652, 2009). Npr. ako treba da se dopuni 3 bajta, sva tri bajta treba da imaju vrijednost 3. Ako treba da se dopuni 5 bajta, svih 5 treba da imaju vrijednost 5:

   ```
   ...f  o  r
   ...66 6F 72 05 05 05 05 05
   ```

2. ISO/IEC 7816-4: drugi metod navodi da je neophodne bajte moguće dopuniti karakterom 0x80 (ili 128 decimalno) iza koga slijede NULL karakteri do kraja bloka:

   ```
   ...f  o  r
   ...66 6F 72 80 00 00 00 00
   ```

3. ANSI X.923: je treći metod koji navodi da bajte treba popuniti nulama sve do kraja bloka, osim zadnjeg bajta, koji je jednak broju dopunjenih bajta:

```
...f  o  r
...66 6f 72 00 00 00 00 05
```

2.1.2.4 *Osnovne pretpostavke za softversko poboljšanje performansi algoritma AES*

U prethodnom tekstu su predstavljene teoretske osnove na kojima funkcioniše algoritam AES kao i pretpostavke za njegovu implementaciju. Implementacija algoritma AES na osnovu teoretskih pretpostavki koje su date u (FIPS197, 2001) i u prethodnom tekstu je veoma kompleksna i predstavlja obiman posao i za najiskusnije programere. Međutim, ako se ne bi posvetila pažnja poboljšanju njegovih performansi, ovako definisan algoritam bi u implementaciji bio prilično spor u obradi većih količina podataka. Algoritam AES je danas, osim po svojoj pouzdanosti, poznat i po svojoj efikasnosti i brzini obrade podataka.

U cilju poboljšanja performansi ovog algoritma, kriptografska zajednica je neprekidno pokušavala da unaprijedi mogućnosti njegove implementacije. U nastavku teksta će biti predstavljene neke osnovne metode i mogućnosti za poboljšanje performansi algoritma - kako osnovne tako i veoma napredne.

Arhitektura modernih računarskih sistema je takva da se i programi i podaci nad kojima se oni izvršavaju nalaze u operativnoj memoriji (von Neumann, 1945). Kada se vrši šifrovanje neke veće datoteke koja se nalazi na hard disku, zbog fizičkih ograničenja uređaja i softverskih ograničenja operativnih sistema, navedena datoteka se ne učitava u memoriju kao cjelina. Ona se podijeli na fragmente odgovarajuće dužine, pa se pojedini fragmenti uzastopno učitavaju u operativnu memoriju i obrađuju jedan po jedan u memoriji, a zatim se rezultati obrade zapisuju na hard disk. U daljem tekstu će fragmenti u koje se učitava datoteka biti navođeni pod imenom kriptografski bafer, interni bafer ili samo bafer.

2.1.2.4.1 *Elementarno ubrzavanje algoritma množenja*

Operacija množenja ponovljenim šiftom predstavljena u 2.1.2.2.3 nad Rijndael-ovim Galoa poljima je dosta složena. Kada se implementira, zahtijeva osam prolazaka kroz istu petlju u kojoj je uključeno nekoliko testova, XOR operacija i pomjeranje (šiftanje) bita lijevo i desno. U (Gladman, 2007) dat je pseudo kod za multiplikaciju u konačnim poljima:

```
byte FFmul(const byte a, const byte b)
begin
  byte aa = a, bb = b, r = 0, t
  while (aa <> 0)
    if ((aa & 1) <> 0)
      r = r ^ bb
    endif
    t = bb & 0x80
    bb = bb << 1
    if (t <> 0)
      bb = bb ^ 0x1b // najviši bit polinoma (0x11b) nije potreban
    endif            // jer bb je 8-bitna vrijednost
    aa = aa >> 1
  endwhile
  return r
end
```

Listing 12: Pseudo kod za multiplikaciju u GF(2^8)

Ako se unaprijed pripreme tabele eksponenata i logaritama, one mogu poslužiti da se ubrza proces množenja u Rijndael-ovim Galoa poljima. Ako se koriste pomoćne lookup tabele sa logaritmima i eksponentima koje su prikazane u 2.1.2.2.4 tada se uz pomoć dva 256-bitna niza može ubrzati množenje u konačnim poljima pomoću slijedećeg pseudo koda (Gladman, 2007):

```
byte FFlog[256] // array from table 2
byte FFpow[256] // array from table 3

byte FFmul(const byte a, const byte b)
begin
  if ((a <> 0) and (b <> 0))
    word t = FFlog[a] + FFlog[b]
    if(t >= 255)
      t = t - 255
    endif
    return FFpow[t]
  else
    return 0
  endif
end
```

Listing 13: Pseudo kod za multiplikaciju pomoću tabela

Ako se udvostruči dužina niza FFpow i ako se elementima od 255 do 509 daju iste vrijednosti kao elementima od 0 do 254, tada se pseudo kod može dodatno poboljšati (Gladman, 2007):

```
byte FFmul(const byte a, const byte b)
begin
  if ((a <> 0) and (b <> 0))
    return FFpow[ FFlog[a] + FFlog[b] ]
  else
    return 0
  endif
end
```

Listing 14: Pseudo kod za multiplikaciiju pomoću tabela

2.1.2.4.2 Poboljšavanje performansi AES algoritma pomoću T-tabela

Različite operacije koje se provode tokom jedne Rijndaelove runde mogu se iskombinovati u nekoliko prolazaka kroz unaprijed pripremeljene tabele, čime postaju moguće veoma brze implementacije na procesorima koji imaju 32 bita ili više (Daemen and Rijmen, 1999).

Neka je ulaz u jednu transformaciju runde označen sa a i neka je izlaz iz transformacije *SubBytes* označen sa b (Daemen and Rijmen, 2002):

$$b_{i,j} = S_{RD}[a_{i,j}], \quad 0 <= i < 4; \quad 0 <= j < Nb \tag{49}$$

Neka je izlaz iz ShiftRows transformacije označen sa c, a izlaz iz transformacije MixColumns sa d:

$$\begin{bmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{bmatrix} = \begin{bmatrix} b_{0,j+C0} \\ b_{1,j+C1} \\ b_{2,j+C2} \\ b_{3,j+C3} \end{bmatrix}, 0 \le j < N_b \tag{50}$$

$$\begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \cdot \begin{bmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{bmatrix}, 0 \le j < N_b \quad (51)$$

Dodavanje indeksa u jednačini 50 mora se obaviti po modulu Nb. Jednačine 49-51 mogu se iskombinovati u jednu:

$$\begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \cdot \begin{bmatrix} S_{RD}\left[a_{0,j+C0}\right] \\ S_{RD}\left[a_{1,j+C1}\right] \\ S_{RD}\left[a_{2,j+C2}\right] \\ S_{RD}\left[a_{3,j+C3}\right] \end{bmatrix}, 0 \le j < N_b \quad (52)$$

Matrična multiplikacija može da se interpretira kao linearna kombinacija četiri vektora kolona:

$$\begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} = \begin{bmatrix} 02 \\ 01 \\ 01 \\ 03 \end{bmatrix} \cdot S_{RD}\left[a_{0,j+C0}\right] \oplus \begin{bmatrix} 03 \\ 02 \\ 01 \\ 01 \end{bmatrix} \cdot S_{RD}\left[a_{1,j+C1}\right] \oplus \quad (53)$$

$$\begin{bmatrix} 01 \\ 03 \\ 02 \\ 01 \end{bmatrix} \cdot S_{RD}\left[a_{2,j+C2}\right] \oplus \begin{bmatrix} 01 \\ 01 \\ 03 \\ 02 \end{bmatrix} \cdot S_{RD}\left[a_{3,j+C3}\right], 0 \le j < N_b$$

Na osnovu ovoga, u (Daemen and Rijmen, 2002) autori definišu četiri T-tabele:

$$T_0[a] = \begin{bmatrix} 02 \cdot S_{RD}[a] \\ 01 \cdot S_{RD}[a] \\ 01 \cdot S_{RD}[a] \\ 03 \cdot S_{RD}[a] \end{bmatrix}, T_1[a] = \begin{bmatrix} 03 \cdot S_{RD}[a] \\ 02 \cdot S_{RD}[a] \\ 01 \cdot S_{RD}[a] \\ 01 \cdot S_{RD}[a] \end{bmatrix} \quad (54)$$

$$T_2[a] = \begin{bmatrix} 01 \cdot S_{RD}[a] \\ 02 \cdot S_{RD}[a] \\ 03 \cdot S_{RD}[a] \\ 01 \cdot S_{RD}[a] \end{bmatrix}, T_1[a] = \begin{bmatrix} 01 \cdot S_{RD}[a] \\ 01 \cdot S_{RD}[a] \\ 03 \cdot S_{RD}[a] \\ 02 \cdot S_{RD}[a] \end{bmatrix} \quad (55)$$

T-tabele se sastoje od po 256 elemenata dužine 4 bajta i zauzimaju 4kB prostora. Pomoću ovih tabela, jednačina 55 se pretvara u:

$$\begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} = T_0\left[a_{0,j+C0}\right] \oplus T_0\left[a_{1,j+C1}\right] \oplus T_2\left[a_{2,j+C2}\right] \oplus T_3\left[a_{3,j+C3}\right] \quad (56)$$

$$0 \le j < N_b$$

Kako se navodi u (Gladman, 2007), svaka kolona u bloku Stanje može se izračunati pomoću 4 XOR operacije u koju se uključene jedna riječ iz tabele ključeva i četiri riječi iz T-tabela. Riječi iz T-tabela se mogu indeksirati pomoću četiri bajta iz ulaznog bloka.

U finalnoj rundi se ne koristi transformacija *MixColumns*, zbog čega se mora koristiti S-Box umjesto T-tabela (Daemen and Rijmen, 2002).

Prema 56 i 57, pseudo kod za generisanje tabele T1 dat je u sljedećem listingu:

```
byte genT1()
begin
  for r = 0 step 1 to 255
    print gMul(SBox[t], 02)
    print gMul(SBox[t], 01)
    print gMul(SBox[t], 01)
    print gMul(SBox[t], 03)
  endfor
end
```

Listing 15: Pseudo kod za generisanje tabele T1

2.1.2.4.3 Bertonijeve ideje za poboljšanje performansi pomoću transponovanja matrice Stanje

Prva ideja za poboljšanje performansi algoritma zasnovana na idejama autora prikazana je u (Daemen and Rijmen, 2002), dok su njene najpoznatije i najčešće citirane implementacije urađene od strane dr Brajena Gladmana (Gladman, 2007). Ona u svom radu koristi osam tabela veličine po 4KB svaka, četiri za proces šifrovanja i četiri za proces dešifrovanja. U ovim tabelama se nalaze unaprijed izračunati međurezultati djelovanja više AES-ovih transformacija odjednom. Ova implementacija obrađuje bafer Stanje kolonu po kolonu.

Prema alternativnom rješenju koje je zasnovano je na Bertonijevim (Bertoni et al. 2002) idejama, matrica Stanje se najprije transponuje a zatim obrađuje red po red. Ova implementacija u svom radu ne zahtijeva postojanje ranije pomenutih osam tabela već samo dva S-Boxa, te ima manje memorijske zahtjeve, zbog čega je interesantna za konstrukciju različitih hardverskih sklopova.

Prema ovoj ideji, transformacija *MixColumns* je u velikoj mjeri izmijenjena. Treba imati u vidu da je sama matrica Stanje transponovana i da je rutina za proširenje ključa nešto izmijenjena prije obrade podataka u svakoj rundi koja se izvršava po Bertonijevoj ideji.

Kako se navodi u (Bertoni et al. 2002) ako sa x_i, za *0 <= i <= 3*, označimo 32.-bitne riječi transponovane matrice Stanje prije primjene *MixColumns* transformacije, ako sa y_i, za *0 <= i <= 3*, označimo 32-bitne riječi nakon primjene *MixColumns* transformacije, izmijenjena verzija *MixColumns* transformacije može biti predstavljena pomoću sljedećeg skupa jednačina:

$$y_0 = \{02\}\bullet x_0 \oplus \{03\}\bullet x_1 \oplus x_2 \oplus x_3 \quad (57)$$

$$y_1 = x_0 \oplus \{02\}\bullet x_1 \oplus \{03\}\bullet x_2 \oplus x_3$$

$$y_2 = x_0 \oplus x_1 \oplus \{02\}\bullet x_2 \oplus \{03\}\bullet x_3$$

$$y_3 = \{03\}\bullet x_0 \oplus x_1 \oplus x_2 \oplus \{02\}\bullet x_3$$

Varijable y_i i x_i sadrže po 4 bajta na pozicijama i u kolonama normalne, matrice, prije njenog transponovanja. Treba imati u vidu da operator množenja koji se ovdje koristi nije operator modularnog proizvoda ($\otimes$) kakav se koristi u standardnoj *MixColumns* transformaciji, već da je to skup od četiri paralelna AES-ova proizvoda nad Galoa poljima GF(2^8).

Standardna operacija *MixColumns* po jednoj koloni prikazana sljedećom jednačinom:

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = 02 \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \oplus 03 \begin{bmatrix} s_{3,c} \\ s_{0,c} \\ s_{1,c} \\ s_{2,c} \end{bmatrix} \oplus \begin{bmatrix} s_{2,c} \\ s_{3,c} \\ s_{0,c} \\ s_{1,c} \end{bmatrix} \oplus \begin{bmatrix} s_{1,c} \\ s_{2,c} \\ s_{3,c} \\ s_{0,c} \end{bmatrix} \quad (58)$$

Efekat nove transformacije *MixColumns* vidljiv ja na slijiedećim jednačinama, gdje je ona prikazana kao varijanta matričnog množenja, pri čemu treba imati u vidu da se ova transformacija služi AES-ovim proizvodom nad poljem GF(2^8).

$$\begin{bmatrix} s'_0 & s'_4 & s'_8 & s'_{12} \\ s'_1 & s'_5 & s'_9 & s'_{13} \\ s'_2 & s'_6 & s'_{10} & s'_{14} \\ s'_3 & s'_7 & s'_{11} & s'_{15} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \otimes \begin{bmatrix} s_0 & s_4 & s_8 & s_{12} \\ s_1 & s_5 & s_9 & s_{13} \\ s_2 & s_6 & s_{10} & s_{14} \\ s_3 & s_7 & s_{11} & s_{15} \end{bmatrix} \quad (59)$$

$$\begin{bmatrix} s'_0 & s'_4 & s'_8 & s'_{12} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \end{bmatrix} \otimes \begin{bmatrix} s_0 & s_4 & s_8 & s_{12} \\ s_1 & s_5 & s_9 & s_{13} \\ s_2 & s_6 & s_{10} & s_{14} \\ s_3 & s_7 & s_{11} & s_{15} \end{bmatrix}$$

Pseudo kod transformacije MixColumns koja je dodatno modifikovana radi poboljšanja performansi dat je u slijedećem listingu:

```
byte MixColumns(int w0, w1, w2, w3)
begin
  const m1 = 0x80808080
  const m2 = 0x7f7f7f7f
  int x23, x01, x12, x03, x2
  x23 = w2 xor w3
  x01 = w0 xor w1
  x12 = w1 xor w2
  x03 = w0 xor w3
  x2 = w2

  w0 =  (w1 xor x23 xor
        (((x01 and m2) shl 1) xor (((x01 and m1) shr 7) * 0x1b)));
```

```
  w1 =  (w2 xor x03 xor
        (((x12 and m2) shl 1) xor (((x12 and m1) shr 7) * 0x1b)));
  w2 =  (w3 xor x01 xor
        (((x23 and m2) shl 1) xor (((x23 and m1) shr 7) * 0x1b)));
  w3 =  (x2 xor x01 xor
        (((x03 and m2) shl 1) xor (((x03 and m1) shr 7) * 0x1b)));
end
```

Listing 16: Pseudo kod modifikovane rutine MixColumns

U prethodnom listingu pseudo koda kao ulazi u ovu transformaciju se javljaju 32-bitne riječi w0, w1, w2, w3 transponovane matrice stanje koje su već prošle kroz operacije *SubBytes*() i *ShiftRows*(), dok se na njenom izlazu ponovo nalaze istoimene 32-bitne varijable.

Kako se navodi u (Bertoni et al. 2002), dio navedenog rada je već tada bio u procesu patentiranja, tako da o ovoj ideji u literaturi ima vrlo malo tragova. Treba reći i da se na Web prezentaciji kompanije EnSilica Ltd može naći nepotpisan članak u kom autor ili autori tvrde da su modifikovali Bertonijevu ideju i da je njihova implementacija brža od Gladmanove implementacije.

3 Načini enkripcije unutar operativnih sistema

Pristupe enkripciji unutar operativnih sistema možemo uslovno podijeliti na mrežno orijentisane načine enkripcije (*network encryption*) i načine orijentisane ka medijumu za čuvanje podataka (*storage encryption*).

U nastavku će biti prikazana osnovna softverska kriptografska rješenja koja koriste *block based* i file *based enkripciju* koja se koriste u operativnim sistemima, kao i najvažnija mrežno orjentisana rješenja.

3.1 Načini enkripcije orijentisani ka medijumu za čuvanje podataka (*storage encryption*)

Danas postoje dva osnovna pristupa softverskoj enkripciji podataka na medijumu unutar operativnih sistema: na nivou bloka (*block based*) i na nivou datoteke (*file based*, još i *disk based*).

U slučaju enkripcije na nivou bloka, mehanizam za šifrovanje se smiješta između sistema datoteka i blok uređaja. U momentu kada fajl sistem treba da upiše podatke na blok uređaj, ti podaci se prije fizičkog zapisivanja na disk šifruju. Fajl sistem je nesvjestan da je došlo do enkripcije podataka. Prednost ovoga pristupa ogleda se u činjenici da je on jednostavan, transparentan i da šifruje sve što se nalazi na uređaju. Njegov osnovni nedostatak je nepostojanje granularne kontrole šifrovanja. Kod šifrovanja na nivou bloka nije moguće tretirati jednu datoteku drugačije od druge. Ovakva vrsta enkripcije je jedan od najboljih načina za zaštitu podataka, jer ne samo da je zaštićena svaka datoteka, već je zaštićen i privremeni (*temporary*) sadržaj koji može sadržavati neke dijelove datoteka.

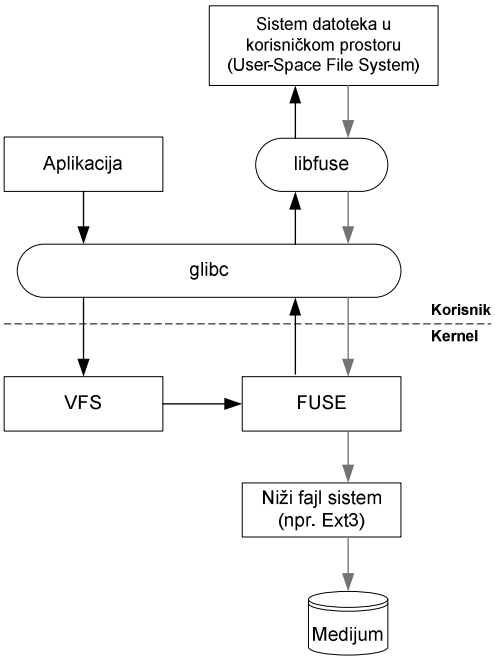
Za razliku od šifrovanja na nivou bloka, kriptografski fajl sistem vrši enkripciju na nivou datoteka. Podaci se ovdje šifruju prije nego što dođu do upravljačkih programa (*device driver*) blok uređaja. Ovakav način omogućava mnogo veću kontrolu načina na koji će datoteke biti šifrovane. Najveći nedostatak ovakvog načina šifrovanja je kompleksnost implementacije i nemogućnost zaštite prostora za straničenje (*swap*).

3.1.1 Sistemi datoteka u korisničkom prostoru (*user space file systems*) u službi kriptografije

Konvencionalni operativni sistemi obično dijele virtuelnu memoriju na prostor koji pripada kernelu i korisnički prostor. Prostor dodjeljen kernelu rezervisan je za izvršavanje kernela, njegovih dodataka i upravljačkih programa. S druge strane, korisnički prostor je dio memorije u kojem se izvršavaju sve korisničke aplikacije i koji se po potrebi može straničiti (*swapping*).

Mnogi sistemi datoteka u korisničkom okruženju koriste usluge FUSE (Szeredi, 2008) modula. FUSE (*File System in Userspace*) je modul koji se može učitati u jezgro nekog UNIX-u sličnog operativnog sistema. Pomoću ovog modula korisnici mogu da kreiraju svoje vlastite sisteme datoteka bez izmjena izvornog koda jezgra operativnog sistema. Ovaj modul je vrlo koristan za pisanje virtuelnih fajl sistema koji, za razliku od tradicionalnih, ne smiještaju podatke direktno na medij. FUSE presreće pozive

između korisničkog i virtuelnog sistema datoteka te kernela dok im dodaje bezbjednosne funkcije.



Slika 11: FUSE modul (Diesburg, 2010)

Primjeri sistema datoteka koji koriste FUSE modul u funkciji zaštite podataka su EncFS i CryptoFS.

EncFS (Gough, 2008) predstavlja šifrovani sistem datoteka u korisničkom prostoru. On koristi FUSE biblioteku i Linux kernel da bi obezbijedio interfejs ka sistemu datoteka. Namijenjen je za zaštitu podataka od *off-line* napada, a za šifrovanje podataka oslanja se na OpenSSL biblioteku. Zadnje izmjene na github projektu objavljene su sredinom 2015. godine. EncFS je dostupan i na Android platformi u okviru open source aplikacije pod imenom cryptonite, a njegova najnovija verzija 1.8.1. objavljena je početkom 2016. godine.

CryptoFS (Hohmann, 2008) može da koristi FUSE i LUFS - *Linux Userland FileSystem* (Malita, 2013) za čuvanje šifrovanih podataka. CryptoFS ne zahtijeva administrativni pristup ni bilo kakavo komplikovano podešavanje (*setup*) za kreiranje šifrovanog sistema datoteka. Poput EncFS sistema i CryptoFS čuva datoteke i njihova imena u šifrovanim direktorijumima. Od korisnika oba sistema zahtijevaju da montiraju (*mount*) šifrovane direktorijume sa korektnom lozinkom da bi pristupili datotekama u njemu.

3.1.2 Lokalni sistemi datoteka koji koriste NFS mehanizme za šifrovanje i dešifrovanje

Network File System (NFS) (Sandberg, 1985) je distribuirani fajl sistem protokol koji je originalno razvijen u kompaniji SUN Microsystems 1984. godine, koji je dozvoljavao klijentskim kompjuterima da pristupaju datotekama kroz mrežu na način sličan lokalnom pristupu. Mrežni sistem datoteka (NFS) je dizajniran na *Open Network Computing Remote Procedure Call* (ONC RPC) protokolu, koji je otvoreni standard definisan u IETF RFC publikaciji zbog čega bilo ko može slobodno da ga implementira. Pošto NFS preusmjerava neke zahtjeve upućene operativnom sistemu prema korisničkom prostoru, neki lokalni sistemi datoteka koriste njegove usluge da

olakšaju razvoj. Dva lokalna sistema koji koriste njegove usluge za obezbjeđenje privatnosti su *Cryptographic File System* (CFS) i *Transparent Cryptographic File System* (TCFS).

CFS (*Cryptographic File System*) (Blaze, 1993) predstavlja inetrfejs ka transparentnom UNIX sistemu datoteka koji omogućava automatsku enkripciju sistema direktorijuma pomoću datih lozinki. Korisnici pomoću jednostavnih komandi priključuju kriptografski ključ nekom direktorijumu. Korisnik zatim može da koristi standardne alate i usluge operativnog sistema, a datoteke će automatski biti šifrovane odnosno dešifrovane prilikom pisanja odnosno čitanja. CFS koristi jedino prilično zastarjeli DES kao svoj osnovni algoritam.

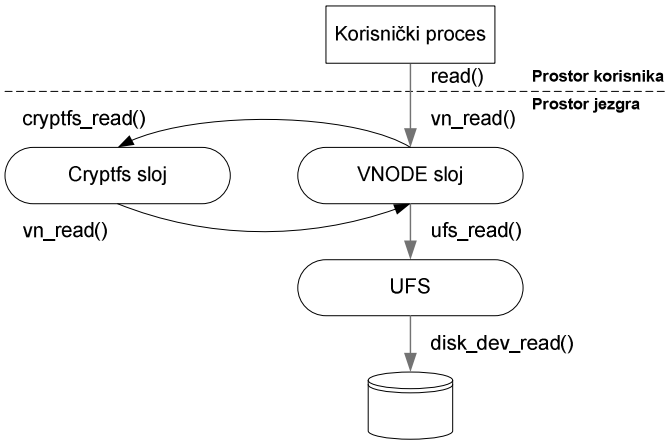
TCFS (*Transparent Cryptographic File System*) (Ermelindo, 1997) je razvijen na odsjeku Dipartimento di Informatica ed Applicazione univerziteta di Salerno u Italiji a trenutno je dostupan za operativni sistem Linux. TCFS predstavlja proširenje NFS sistema koje se ponaša kao NFS ali dozvoljava korisnicima da zaštite svoje datoteke pomoću enkripcije. I ovaj sistem koristi jedino DES algoritam za obezbjeđenje povjerljivosti podataka, zbog čega se oba pomenuta sistema mogu smatrati zastarjelima.

3.1.3 Složeni (*stackable*) kriptografski sistemi datoteka

Složeni (*stackable*) sistemi datoteka (Wright, 2004) predstavljaju kompromis između sistema datoteka koji funkcionišu na nivou kernela i mrežnih sistema datoteka. Ovakvi sistemi datoteka mogu da funkcionišu iznad bilo kakvog fajl sistema. Oni ne moraju da prenose podatke između prostora jezgra i korisničkog prostora, a portabilni su između većeg broja operativnih sistema. Ovakvi sistemi datoteka se slažu (nadodaju) na postojeću arhitekturu operativnog sistema da bi proširili neku njegovu (obično kriptografsku) funkcionalnost. Oni presreću sistemske pozive i preusmjeravaju ih prema jednom ubačenom sistemskom sloju. Složeni (*stackable*) sistemi datoteka se izvršavaju u jezgru a mogu da operišu iznad bilo kakvog drugog fajl sistema, pri čemu nije potrebno pokretati nikakve dodatne korisničke procese.

Primjeri ovakvih složenih sistema datoteka su Cryptfs, NCryptfs i eCryptfs, pri čemu su Cryptfs, NCryptfs nastali na osnovu FIST-a (File System Translator Language) (Zadok and Nieh, 2000), koji dozvoljava programerima da opišu sistem na višem nivou i da potom generišu kod pomoću generatora koda (Zadok, 2001). Uslov za interakciju sa bilo kojim fajl sistemom je postojanje dobro definisanog interfejsa sa virtuelnim fajl sistemom (VFS). FIST može da generiše sistemske module za različite platforme, poput Solaris, Linux i FreeBSD operativnih sistema.

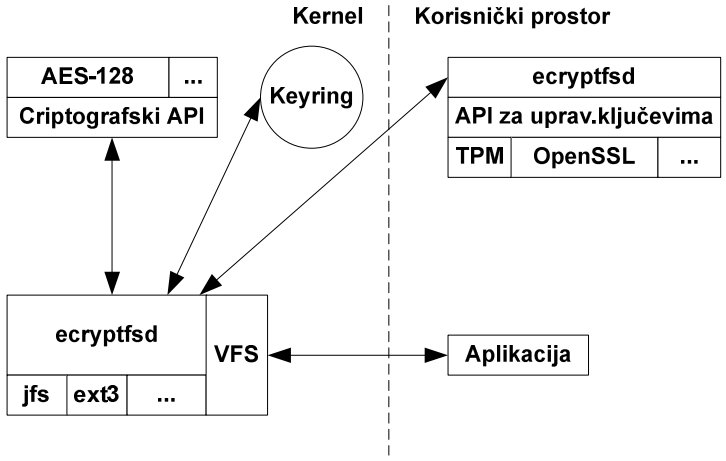
Cryptfs (Zadok et al. 1998) je složeni kriptografski sistem datoteka koji je nastao na osnovu FiST programskog alata. On je dizajniran više sa namjerom da dokaže FiST koncept nego da bude bezbijedan fajl sistem. Cryptfs koristi Virtuelne čvorove (VNode) koji u operativnim sistemima zasnovanim na UNIX-u predstavljaju otvorenu datoteku, drektorijum, uređaj ili drugi entitet za uključenje vlastitog modula u kernel, kao što je prikazano na slici 5.



Slika 12: Cryptfs: složeni kriptografski sistem datoteka (Zadok and Badulescu, 1999)

NCryptfs (Zadok and Badulescu, 1999) je složeni sistem datoteka koji je kreiran sa namjerom da obezbjedi bezbjednost, povjerljivost i performanse. NCryptfs je direktni nasljednik Cryptfs sistema, pri čemu je u mnogim oblastima unaprijedio rješenja starijeg sistema. NCryptfs podržava više korisnika, ključeva, načina autentikacije i algoritama šifrovanja. Osnovni cilj NCryptFS sistema bio je da obezbjedi transparentnu i lako portabilnu enkripciju datoteka bez većeg gubitka performansi.

eCryptfs (Halcrow, 2007) je složeni kriptografski sistem datoteka koji je kreiran za operativni sistem Linux. Ovakav sistem je složen iznad postojećih montiranih (mounted) sistema datoteka koji se nazivaju niži fajl sistemi. I eCryptfs i niži fajl sistem (ext3, JFS, NFS...) su registrovani u kernelovom virtuelnom sistemu datoteka (VFS). eCryptfs presreće pozive nižem fajl sistemu pa šifruje i dešifruje sadržaje koji se upisuju ili očitavaju iz njega. On dobija ključeve iz *keyringa* (vlastite baze podataka o ključevima) i koristi kernelov kriptografski API da bi izvršio šifrovanje i dešifrovanje sadržaja datoteka. eCryptfs za sve zahtjeve vezane za upravljanje ključevima koristi ecryptfsd daemon, kako je prikazano na slici 6.

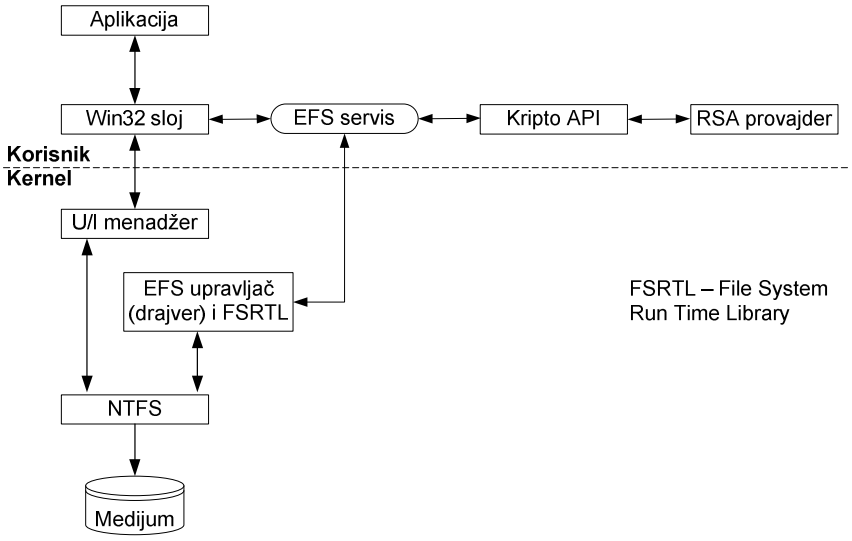


Slika 13: Način funkcionisanja eCryptfs sistema (Halcrow, 2007)

3.1.4 *Disk based* (*File based*) sistemi za enkripciju datoteka

Sistemi datoteka orijentisani ka disku (*disk based*) (Wright, 2004) obezbjeđuju strukturu nizu bita koji se nalazi na mediju, pružaju usluge imenovanja datoteka i direktorijuma te pristup datotekama i sektorima. Ovi sistemi šifruju podatke na višem nivou apstrakcije nego sistemi orijentisani ka blokovima (*block based*). S druge strane *disk based* fajl sistemi se izvršavaju na nižem nivou apstrakcije nego složeni (*stackable*) sistemi datoteka, softverski programi za enkripciju datoteka ili lokalni sistemi koji koriste NFS mehanizme. Ovakvi sistemi imaju pristup svim podacima i atributima karakterističnim za jednu datoteku ili direktorijum tako da mogu da koriste mnogo kompleksniju autorizaciju i autentikaciju nego block-based sistemi datoteka. Disk based sistemi datoteka imaju punu kontrolu nad metapodacima i operacijama direktorijuma i datoteka koje obrađuju.

Primjer jednog takvog sistema je Microsoft-ov Encryption File System (EFS) (Microsoft Corp. 2005). EFS proširuje dnevnik (*journaling*) a koristi i Windows autentikaciju i liste za kontrolu pristupa (ACL) (Microsoft Corp. 2008). EFS je podržan na operativnim sistemima koji su zasnovani na Microsoft NT jezgru, poput Windows 2000, XP, Vista i 7. Slika 7. demonstrira kako EFS proširuje NTFS unutar i van prostora jezgra.



Slika 14: EFS komponenate u korisničkom prostoru i u prostoru jezgra (Microsoft Corp. 2008)

EFS koristi i simetričnu i asimetričnu kriptografiju. Kada se datoteka šifruje po prvi put, kreira se za tu datoteku jedinstven simetrični FEK ključ (*File Encryption Key*), pa se pomoću njega šifruje datoteka. Taj ključ se ugrađuje u datoteku šifrovan korisnikovim javnim ključem. Proces dešifrovanja počinje dešifrovanjem FEK ključa korisnikovim privatnim ključem, koji se nalazi unutar njegovog Windows profila. Do Windows 2000 EFS je koristio DESX algoritam, a kasnije verzije operativnih sistema dodaju i AES i Triple DES.

Z sistem datoteka (*Z File System*) (Moffat, 2012) je sistem datoteka kreiran od strane kompanije SUN Microsystems. ZFS je implementiran kao open source softver, a sada je u vlasništvu kompanije Oracle. Oracle Sloaris 11 je Z sistemu datoteka dodao mogućnost transparentnog šifrovanja. Svi podaci i metapodaci fajl sistema

(poput vlasništva, lista za kontrolu pristupa, informacija o kvotama itd.) su sakriveni kada im se zaštiti privatnost u okviru ZFS pool-a. ZFS pool može da sadrži i mješavinu šifrovanih i otvorenih (nešifrovanih) skupova podataka. Pri ovome je šifrovanje aplikacijama i ostalim Oracle Solaris servisima, poput NFS i CIFS, potpuno transparentno.

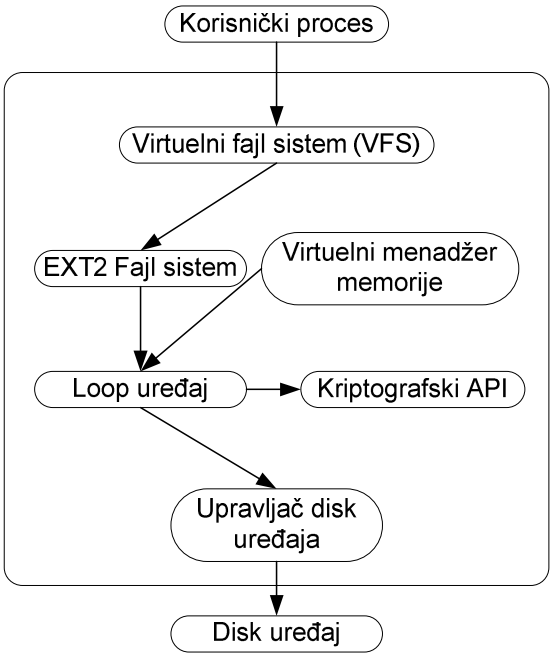
3.1.5 *Block based* sistemi za enkripciju datoteka

Sistemi datoteka orijentisani ka bloku (*block based*) vrše šifrovanje na nižem nivou apstrakcije od fajl sistema. Ovi sistemi funkcionišu transparentno ispod fajl sistema i šifruju podatke na nivou blokova podataka hard diska ili drugog medijuma.

BitLocker (Ferguson, 2006) dodatak operativnom sistemu je dostupan u Enterprise i Ultimate izdanjima Windows Vista i Windows 7 operativnih sistema. On je takođe dostupan u Pro i Enterprise izdanjima Windows 8, 8.1 i 10 operativnih sistema te u Windows Server 2008 R2 i Windows Server 2012 operativnim sistemima. Bitlocker™ Drive Encryption može da šifruje sve podatke i na sistemskom disku. BitLocker je nametnuo neke nove bezbjednosne zahtjeve koji nisu bili prisutni kod uobičajenih režima rada i algoritama. Zbog toga je kombinovan dobro poznat i istestiran algoritam AES u CBC režimu rada sa jednom novom komponentom koja je dobila ime Elefant raspršivač (*Elephant diffuser*), zbog čega se navedeni algoritam ponaša kao algoritam modifikovan javnim parametrom (Tweakable Block Cipher). On može da u radu koristi tri načina rada: transparentni režim uz upotrebu TPM (*Trusted Platform Module*) 1.2 *čipa* (IBM Blueprints, 2009), pomoću korisničke autentikacije kada zahtijeva da korisnik unese PIN ili priključi USB token.

Linux Cryptoloop je modul koji je dodat Linux kernelu da bi se olakšala integracija enkripcije ispod nivoa fajl sistema (Saarinen, 2005). Ova vrsta enkripcije funkcioniše bez obzira o kojem se sistemu datoteka radi. Sa Cryptoloop modulom, fizički disk ili neka logička particija kao i proizvoljna datoteka se predaje na obradu tzv. loop uređaju (*loop device*).

Linux Loopback upravljač (*device driver*) omogućava predstavljanje neke datoteke kao blok uređaja, pri čemu opciono transformiše podatke prije nego što ih očita ili upiše na medijum obično koristeći enkripciju. Trenutno IPSEC i Cryptoloop driver koriste ovu mogućnost. Nakon što se jezgru Linux sistema da ključ pomoću losetup uslužnog programa, *loop device driver* dalje transparentno vrši šifrovanje i dešifrovanje kad god se pristupa loop uređaju. Loop uređaj se može inicijalizovati i montirati (*mount*) da koristi bilo koji sistem datoteka. Način funkcionisanja Cryptoloop modula prikazan je na slici 8.



Slika 15: Cryptoloop modul (Saarinen, 2005)

Kao zamjena za Cryptoloop modul od verzije 2.6 operativnog sistema Linux koristi se dm-crypt plugin (podsistem). Dm-crypt funkcioniše tako što koristi *Device-Mapper* (Paranjape, 2005), infrastrukturu koja je predstavljena u verziji 2.6 operativnog sistema Linux, a koja je namijenjena da obezbijedi generički način za kreiranje virtuelnih slojeva blok uređaja iznad stvarnih blok uređaja koji komuniciraju sa hardverom. Dm-crypt koristi Linuxov kripto API i omogućava šifrovanje blok uređaja kao što su diskovi, particije ili RAID nizovi. Dm-crypt podržava kriptografske algoritme i režime rada koji su prisutni u Linux-ovom kripto API-ju što uključuje AES, DES, Serpent, Twofish, Blowfish, ECB i CBC režim rada, pri čemu je inicijalizacioni vektor baziran na broju sektora.

Enkripcija diska na Android operativnom sistemu je zasnovana na Dm-crypt podsistemu, koji je u stvari funkcija jezgra koja funkcioniše nad slojem blok uređaja (Android Open Source Project, 2016). Zbog toga ne može da funkcioniše sa YAFFS sistemom datoteka, koji se obraća direktno raw nand fleš čipu, ali može da radi sa Emmc i sličnim fleš uređajima koji se kernelu predstavljaju kao blok uređaji.

Iako je Dm-crypt pružio solidan interfejs za enkripciju diska, alati poput dmsetup uslužnog programa koji su služili za konfigurisanje Device Mappera nisu bili previše pristupačni za rad. Zbog toga je razvijen uslužni program koji se fokusirao na pojednostavljenje konfigurisanja Dm-crypt uređaja, a koji je dobio naziv Cryptsetup (Petullo, 2007). LUKS (*Linux Unified Key Setup*) je specifikacija za enkripciju diskova (Fruhwirth, 2008). Za razliku od drugih implementacija za enkripciju diska koje su često bila međusobno različite i nekompatibilne, LUKS navodi standardan format podataka na disku koji je nezavistan od platforme. Prve LUKS implementacije su nastale na proširenim verzijama programa cryptsetup.

FileValut (Apple Corp., 2012) je metod za enkripciju diska na Mac računarima. Iako je u verziji 1 koja je predstavljena sa operativnim sistemom Mac OS X Panther mogao da šifruje samo home direktorijum, u verziji 2 je pretrpio značajna unapređenja. Mac OS X Lion i noviji koriste FileValut 2 koji šifruje individualne blokove diska, tako da je enkripcija nevidljiva sistemu datoteka.

3.2 Mrežno orjentisani načini enkripcije

U slučaju mrežno orjentisanih načina enkripcije, trenutno se koriste tri najvažnije biblioteke koje aplikacije koriste za različite kriptografske operacije i desetine manjih biblioteka. OpenSSL je dobro poznata kriptografska biblioteka koju koriste brojni komercijalni i open source proizvodi, a koja je ponudila prvu poznatu open source implementaciju SSL protokola, dok je NSS (MDN NSS, 2016) kriptografska biblioteka koja je nastala u kompaniji Netscape u toku rada na kreiranju SSL protokola. GnuTLS (Mavrogiannopoulos and Josefsson, 2013) predstavljaja implementaciju TLS protokola iznad libgcrypt biblioteke.

SSL (*Secure Socket Layer*) i TLS (*Transport Layer Security*) su kriptografski protokoli koji osiguravaju prenos podataka širom Interneta (Lee et al. 2007). Ovi protokoli se koriste za obezbjeđenje autentikacije, privatnosti i integriteta podataka na Internetu. Nastali su na temeljima SSL specifikacije kompanije Netscape iz 1994. godine da bi 1999. godine TLS bio definisan kao IETF standard.

OpenSSL (Li et al. 2010) je open source implementacija SSL i TLS protokola. OpenSSL je dobro poznata kriptografska biblioteka koju koriste brojni komercijalni i open source proizvodi, a njene implementacije dostupne su za operativne sisteme Windows, Linux, Android, Solaris, Mac OS X i OpenBSD.

NSS (*Network Security Services*) je najstarija i najkompletnija kriptografska biblioteka koja je 4 puta ovjerena FIPS 140 dokumentom (Lim et al. 2002), (Microsoft Corp. 2014). NSS (MDN NSS, 2016) sadrži kompletnu open source implementaciju kriptografskih biblioteka koje podržavaju SSL/TLS i S/MIME sa opcionom podrškom za hardversku akceleraciju na strani servera i podrškom za smart kartice na strani klijenta.

GnuTLS (Mavrogiannopoulos and Josefsson, 2013) je biblioteka koja nam pruža API za pristup bezbjednosnim komunikacionim protokolima. Tehnički, GnuTLS je portabilna Ansi C biblioteka koja implementira protokole u rangu od SSL 3.0 do TLS 1.2.

3.3 Podrška kriptografskim akceleratorima u okvirima operativnih sistema

Kriptografski akceleratori predstavljaju koprocesore koji na sebe preuzimaju računarski zahtjevne procese šifrovanja i dešifrovanja te na taj način oslobađaju CPU da obavlja druge zadatke i poboljšavaju ukupne performanse sistema. Različiti operatvni sistemi pružaju veću ili manju podršku različitim hardverskim kriptografskim sklopovima.

Operativni sistem OpenBSD oslanja se na OpenBSD Cryptographic Framework (Keromytis et al. 2006) za pristup kriptografskim akceleratorima. OCF je sloj za virtuelizaciju koji je implementiran u okviru kernela operativnog sistema koji pruža uniforman pristup akceleraciji hardvera tako što sakriva hardverski specifične detalje od korisnika. Prijedlog za unapređenje OCF frejmvorka koji koristi GPU procesore (Harrison et al. 2010) efektivno integriše GPU i postiže 3.4% bolje rezutate AES enkripcije u odnosu na običnu implementaciju na jednom jezgru CPU procesora.

U Linux svijetu postoji problem da veliki broj različitih aplikacija koristi vlastite ideje o tome gdje treba smjestiti ključeve i sertifikate, te kad i kako koristiti pojedine kriptografske algoritme, što na kraju dovodi do redundanse i konfuzije. Trenutno postoje dvije inicijative za rješenje ovog problema. Projekat pod nazivom Fedora

Crypto Consolidation (Red Hat Inc, 2016) standardizuje upotrebu NSS servisa (Network Security Services) pri čemu se sav softver prilagođava da bi koristio kriptografske usluge navedene biblioteke. Postoji i prijedlog koji se pojavio na konferenciji Desktop Summit (Edge, 2011), kada je predstavljena ideja da se iskoristi PKCS#11 (Delaune, et al. 2008) koji definiše 'Cryptoki' API koji je dizajniran da bude interfejs između aplikacija i kriptografskih uređaja, poput pametnih kartica, HSM modula (*Hardware Security Modules*), te PCMCIA ili USB tokena.

Operativni sistem Oracle Solaris 11 koristi Solaris Cryptographic Framework (SCF) za pristup kriptografskim akceleratorima. Prema istraživanjima provedenim u Australiji (Christopher, 2010), pravilnom upotrebom akceleratora moguće je postići značajna poboljšanja kriptografskih performansi sistema.

Microsoftov CryptoAPI pruža interfejs ka kriptografskoj funkcionalnosti unutar Windows operativnih sistema pomoću CSP (*Cryptographic Service Provider*) modula (Microsoft Corp. 2016). CSP moduli su nezavisni moduli koji mogu da se koriste u različitim aplikacijama. Neki korisnički program poziva CryptoAPI funkcije, a njegovi pozivi se preusmjeravaju na CSP funkcije. CSP moduli koji se danas isporučuju sa operatvnim sistemom Windows više nisu ograničeni izvoznim zakonima SAD. CSP moduli se najčešće koriste za pristup HSM modulima (*Hardware Security Modules*) ili pametnim karticama.

3.4 Studije adaptivnosti softverskih resursa hardverskim mogućnostima radi poboljšanja performansi

Ideje o prilagodljivosti operativnog sistema raspoloživim resursima se istražuju već nekoliko decenija. Još davne 1976. godine Blevins and Ramamoorthy (1976) istražuju *real time* performanse velikih kompjuterskih sistema i mreža poput sistema za praćenje kontrole leta. U toku istraživanja adaptibilne kontrole i upravljanja dostupnim resursima sistema autori su razvili model za dijeljenje procesorskog vremena među pojedinim procesima.

U studiji slučaja koja je provedena na Georgija Institute of Technology (Mukherjee and Schwan, 1994) godine ispituje se na koji način se operativni sistem može konfigurisati tako da omogući rad različitih hardverskih konfiguracija (prvenstveno multiprocesorskih) i na koji način navedene konfigurabilne apstrakcije kernela mogu da se prilagode runtime promjenama da bi se poboljšale performanse sistema.

Takođe, u publikaciji (Im et al. 2013) autori predstavljaju mogućnost i prednosti dinamičke rekonfiguracije operativnog sistema u višejezgrenom sistemu, gdje se određenom procesu dodjeljuje veća ili manja porcija memorije i veći ili manji broj jezgara procesora.

Tokom istraživanja elastičnosti u oblaku, autori u (Gupta et al. 2013) predstavljaju koncept elastičnog operativnog sistema koji dozvoljava jednom procesu ili čak jednoj niti da izađe iz okvira jedne mašine i da koristi resurse koji su alocirani u više različitih mašina. U svom istraživanju autori su se bazirali na elastičnom korišćenju distrubuirane memorije sa implementacijom baziranom na Linux 3.2 operativnom sistemu prateći izvršenje (*execution trace*) upita na MySQL serveru.

Ukazujući na slabosti monolitnih operativnih sistema u (Youseff et al. 2012) autori predlažu upotrebu elastičnih servisa operatvnih sistema i prikazuju njihovu upotrebljivost u ispunjavanju različitih zahtjeva. Oni takođe predstavljaju i prototip elastičnog fajl sistem servisa pod nazivom fos. Prema ovom rješenju, na svakom

jezgru mikroprocesora pokrenut je po jedan softverski mikrokernel, na takav način da istovremeno sa paralelizacijom koristi i distribuirane resurse.

Ideje o prilagodljivosti nisu bile uvijek vezane samo za operativne sisteme. Adaptivnost kao kategorija istraživana je i u studiji prilagođenja IPSec arhitekture (Dandalisand and Prasanna, 2000) koja predstavlja ACE (*Adaptive Cryptographic Engine*) mehanizam baziran na FPGA a koji može dinamički da se prilagodi kriptografskim zahtjevima radi postizanja optimalnih performansi pri čemu su testirani kandidati za algoritam AES (MARS, RC6, Rijndael, Serpent i Twofish).

U toku analize upotrebe Linux-a kao platforme za rekonfigurabilni SOC (System On Chip) dizajn (Williams and Bergmann, 2004) autori integrišu podršku za Xilinx FPGA samo-rekonfiguraciju u kernel uClinux operativnog sistema na Microblaze procesoru.

U još jednoj studiji slučaja (Lagger et al. 2006), autori ispituju sistem koji se bazira na ROPES (*Reconfigurable Object for Pervasive Systems*) platformi a sastoji se od Xilinx FPGA sa Microblaze softcore procesorom, sa uClinux operativnim sistemom i opcionim koprocesorom koji može da preuzme dio kriptografske funkcionalnosti na sebe. U navednoj studiji istražuju se benefiti koji se mogu postići u slučajevima kada procesor dinamički rekonfiguriše koprocesor i upoređuju rezultati softverske i koprocesorski bazirane enkripcije algoritmima RC4, DES i 3DES.

Kombinovanje FPGA (*Field Programmable Gate Array*) kola sa GPP (*General Purpose Processor*) procesorima dovelo je do pojave RSoC (*Reconfigurable System on Chip*) sistema. Mogućnost prilagođenja operativnog sistema limitiranim resursima jednog ovakvog sistema dovela je do ideja o razvoju posebno dizajniranih servisa operativnog sistema i biološki inspirisanog algoritma (Samara et al. 2009) koji izračunava najbolju kombinaciju kritičnih resursa ovakvog sistema.

3.5 Komparativna analiza kriptografskih rješenja u okviru operativnih sistema

Kriptografska rješenja koja se koriste u okviru različitih operativnih sistema su veoma raznorodna i često koriste veoma različite principe za postizanje privatnosti podataka. Kada govorimo o načinima orijentisanim ka medijumu, prema mjestu na kojem se dešava enkripcija i dekripcija podataka, razmatrali smo block based i disk based sisteme.

Ako posmatramo block based sisteme u okviru operativnog sistema Windows se izdvaja BitLocker koji je podržan u novijim edicijama poput Pro i Enterprise izdanja Windows 8 operativnog sitema. U slučaju Linux operativnog sistema, vrlo je aktivan projekat u okviru kojeg se razvija Dm-crypt (zajedno sa Cryptsetup i Luks projektima) u kom se dnevno objavljuju nove verzije, dok je za Mac OS X aktuelna verzija FileValut 2.

U slučaju disk based sistema u Windows 8, 8.1 i 10 Pro i Enterprise verzijama dostupan je EFS, što potvrđuje da Microsoft ne odustaje od ove tehnologije. U slučaju operativnog sistema Oracle Solaris 11 Z File System prestaje da bude open source, dok od ostalih operativnih sistema treba spomenuti portove za OSX i FreeBsd operative sisteme. Kada su u pitanju složeni (stackable) sistemi datoteka pod operativnim sistemom Linux aktivno se razvija projekat eCryptfs, koji je zabilježio zadnje izdanje 18.avgusta 2015. godine.

Kada govorimo o mrežno orijentisanim rješenjima, izdvajaju se TLS i njegov prethodnik SSL kao de-facto standardni protokoli koje podržava većina današnjih

web čitača (*browser*). Takođe, mora se spomenuti i NSS kao skup open source biblioteka koje su nastale na osnovu rada na razvoju SSL protokola koji je inicijalno započet u kompaniji Netscape.

Istraživanja koja se vode unutar OpenBSD Cryptographic Framework projekta zahvaljujući konceptu virtuelizacije hardvera pružaju jedinstven način pristupa kriptografskim akceleratorima. Sličnu namjenu imaju i Solaris Cryptographic Framework (SCF) okruženje i Microsoftov CryptoAPI, dok projekat pod nazivom Fedora Crypto Consolidation pokušava da objedini veliki broj rješenja koja se koriste za čuvanje ključeva i sertifikata za upotrebu pojedinih kriptografskih algoritama na način da standardizuje upotrebu NSS servisa.

Do danas je objavljen određen broj istraživanja u kojima se ispituju mogućnosti prilagođenja i rekonfigurabilnosti linux operativnog sistema hardverskim resursima koja se baziraju uglavnom na FPGA kolima. Ovakva istraživanja pokrivaju oblast dinamičkog prilagođenja hardverskim resursima uopšte ali i hardverskim resursima zaduženim za kriptografiju.

Činjenica koja se može uočiti pregledom predstavljenih rješenja jeste da ona ostaju ili usko specijalizovana rješenja koja se mogu konfigurisati i koristiti u nekom operativnom sistemu ili u najboljem slučaju pokušavaju da pomoću jednog apstraktnog sloja ujednače pristup kriptografskim resursima. Međutim, navedena rješenja ne pružaju nikakvu podršku u smislu donošenja odluke o tome koji kriptografski resurs treba upotrebiti za dobijanje optimalnih performansi sistema, pri čemu se kao kriptografski resurs mogu posmatrati hardverski resursi, poput kriptografskih akceleratora ili postojanja više procesora/jezgara u sistemu, ali i softverski resursi poput različitih kriptografskih biblioteka. Kako se danas za potrebe zaštite privatnosti dominantno koristi algoritam AES, osnovni cilj je izgradnja modela koji će biti u stanju da izvrši odabir ili donese odluku o tome koji kriptografski resurs ili njihovu kombinaciju treba korisititi da bi se dobile najbolje performanse sistema pri korišćenju algoritma AES.

U pregledu su prikazana i neka rješenja čiji razvoj već neko vrijeme stagnira. Kao primjer lokalnih sistema datoteka koji koriste NFS mehanizme za šifrovanje i dešifrovanje navedeni su projekti CFS i TCFS, koji su predstavljeni još 1993. i 1997. godine, koji koriste zastarjeli DES algoritam. Kao primjer sistema datoteka u korisničkom prostoru navedeni su CryptoFS kod koga se već dugo ne dešavaju nikakve promjene i EncFS projekat koji je nekoliko godina mirovao da bi od skoro ponovo postao aktivan. Ipak, deskripcija ovih sistema je bila potrebna ne samo u svrhu klasifikacije, već i zbog činjenice da je u ovom istraživanju kao potvrda koncepta korišćen virtuelni fajl sistem Apache Commons VFS, koji se u potpunosti izvršava u korisničkom prostoru, ali ne koristi usluge FUSE modula.

Tabela 13: Uporedni prikaz pristupa za softversku enkripciju podataka na medijumu

Vrsta	Disk	Block				Stackable
Naziv	EFS	BitLocker	DM-crypt + Luks	TrueCrypt	File Valut 2	ECryptFS
Šifruje...	Datoteke	kompletne blok uređaje				Datoteke
Kontejner za šifrovane podatke može da bude:	Datoteka	Disk ili particija Datoteka koja se ponaša kao virtuelna particija				Datoteka
Odnos sa fajl sistemom:	Iznad fajl sistema	Operiše ispod fajl sistema Nije od značaja da li je blok uređaj koji šifruje fajl sistem, particiona tabela ili bilo šta drugo				Dodaje novi sloj postojećem fajl sistemu; automatski šifruje i dešifruje
Enkripcija je implementirana u...	U korisničkom prostoru	I u prostoru jezgra i u korisničkom prostoru	Prostoru jezgra		Prostoru jezgra	Prostoru jezgra
Kriptografski metapodaci su smješteni u...	-	U metadata sekcijama diska	U LUKS zaglavlju	Početku ili kraju uređaja	U CoreStorage metapodacima diska	Zaglavlju svake šifrovane datoteke
Ključ za šifrovanje može bit smješten	U FEK zaglavlju u okviru datoteke	U metadata sekcijama diska, vanjskom uređaju ili TPM čipu.	U LUKS zaglavlju	Početku ili kraju uređaja	U CoreStorage metapodacima diska	Datoteka sa ključevima koja može biti smještena bilo gdje.

Štiti metapodatke fajl sistema (broj datoteka, strukturu direktorijuma)	NE	DA	DA	DA	DA	NE
Može da šifruje cijeli hard disk;	NE	DA	DA	DA	DA	NE
Može da šifruje Swap prostor	NE	DA	DA	DA	DA	NE
Ne mora da alocira fiksnu porciju prostora prije šifrovanja	DA, ne mora	NE, mora da alocira prostor	NE, mora da alocira prostor	NE, mora da alocira prostor	?	DA, ne mora
Podrška za algoritme	DESX, TripleDES, AES, SHA, ECC - zavisno od verzije operativnog sistema	AES	AES, Anubis, CAST5/6, Twofish, Serpent, Camellia, Blowfish, …	AES, Twofish, Serpent	XTS-AES	AES, Blowfish, Twofish...
Podržava multithreading	?	?	DA	DA	?	?
Podržava AES-NI skup instrukcija	?	DA	DA	DA	?	DA

Prema (Intel Corporation, 2013), (Microsoft Corp., 2012), (Apple Corp., 2012) i (Arch Linux, 2016) kreirana je tabela 13 u kojoj je dat uporedni prikaz pristupa za softversku enkripciju podataka na medijumu.

Iz tabele je vidljivo da se mnogo više podataka može prikupiti o open source rješenjima nego o vlasničkim (closed source) rješenjima. Iako su Microsoftovi proizvodi u velikoj mjeri dobro dokumentovani, u momentu pisanja ovoga rada u Microsoftovoj dokumentaciji nije bilo moguće naći podatak da li BitLocker koristi AES-NI skup instrukcija, već je taj podatak dobijen na posredan način, zahvaljujući Intelovom AES-NI ekosistem (Intel, 2013) dokumentu.

Treba primijetiti i da navedeni Intelov dokument (Intel, 2013) ne navodi da eCryptFS koristi AES-NI skup instrukcija, dok je u dokumentaciji za eCryptFS navedeno da ovaj proizvod koristi AES-NI skup instrukciija.

4 Metode i tehnologije primijenjene u razvoju rješenja

4.1 Tehnologije za kreiranje kriptografskih modula

U svim istraživanjima vezanim za algoritam AES kao osnova se mora uzeti NIST-ova publikacija Federal Information Processing Standards Publication 197 (FIPS197, 2001). Pored ovoga, u procesu implementacije i modifikacije ovoga algoritma veliki uticaj su imali tesktovi (Daemen and Rijmen, 2002), (Bertoni et al. 2002) i (Gladman, 2007). U tom kontekstu moraju se spomenuti i neke *open-source* implementacije ovog algoritma od kojih izdvajamo nekoliko implementacija dr Brian Gladmana u programskom jeziku C/C++, zatim open source Java/C# implementaciju u okviru softverskog paketa Bouncy Castle, kao i vlastite implementacije razvijene na Fakultetu organizacionih nauka u toku izrade master rada (Damjanović, 2010) u programskim jezicima Java/Delphi/C kao i one predstavljene u (Damjanović i Simić, InfoM46, 2013) i (Damjanović and Simić, 2011) u programskom jeziku Java. U (Damjanović and Simić, 2013) ispituju se performanse različitih Java kriptografskih biblioteka i njihova prilagođenost pojedinim verzijama JDK. Pored ovoga, kada su u pitanju performanse ovoga algoritma, mora se spomenuti i potencijalna upotreba specifičnih rješenja, poput npr. asemblera ili jezika C/C++ za AES-NI skup instrukcija. AES-NI skup instrukcija bio je predmet brojnih radova od kojih treba izdvojiti (Gueron, 2009), (Gueron, 2012), (Intel Corporation, 2013), (Xu, 2010), (Akdemir et al. 2010), (Basu and Bhargav-Spantzel, 2012). U smislu poboljšanja performansi javljaju se i rješenja koja koriste GPU za paralelizaciju i poboljšanje performansi poput (Di Biagio et al. 2009), (Iwai, 2010) ili (Manavski, 2007).

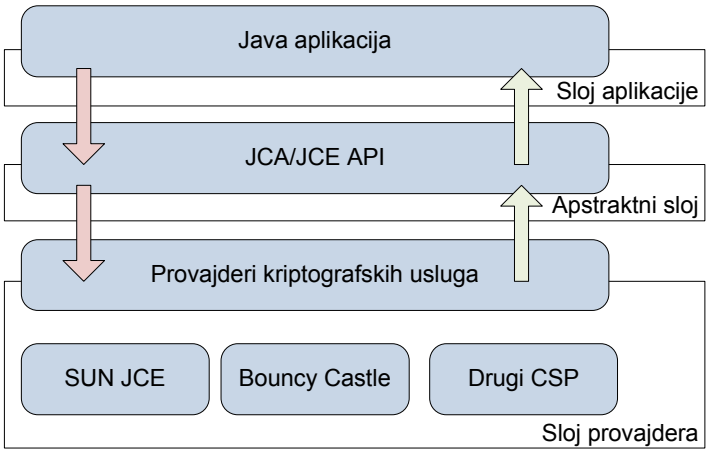
4.1.1 Java i potrebne kriptografske biblioteke i tehnologije

Java je konkurentan, moderan objektno orjentisan i zasnovan na klasama programski jezik koji spada u jezike višeg nivoa koji je specifično kreiran tako da ima što je moguće manje hardverskih zavisnosti pri implementaciji. Jedna od najčešće spominjanih osobina Jave je upravo njena nezavisnost od platforme, jer je Java programe moguće pokrenuti nezavisno od operativnog sistema i hardverske arhitekture. Pri izradi kriptografskih aplikacija u Javi koriste se dva dijela koja su smještena u različitim okruženjima. Prvi je JDK, koji u sebi sadrži klase potrebne za autentikaciju, a drugi je JCE koji u sebi objedinjuje veći broj kriptografskih algoritama.

JDK (*Java Development Kit*) predstavlja oblik SDK (*Software Development Kit*) jer u sebi uključuje alate za izradu, dibagiranje i monitoring Java aplikacija. JDK je implementacija Java SE, Java EE ili Java ME platformi koje su kreirane od strane Oracle korporacije. Od verzije 1.4 moduli za autentikaciju i autorizaciju su u formi JAAS (*Java Authentication and Authorization Service*) modula uključeni u standardnu Java SE ediciju.

Oracle (nekada Sun) Java platforma definiše skup API-ja koji pokrivaju najvažnije bezbjednosne oblasti koje uključuju kriptografiju, infrastrukturu javnih ključeva, autentikaciju, sigurne komunikacije i kontrolu pristupa. *Java Cryptography* Architecture (JCA RFG, 2016) i njoj pridružena *Provider* arhitektura (JCA PROV, 2016) su osnovni bezbjednosni koncepti koji su od verzije 1.1 uključeni u JDK. Dakle JDK 1.1 predstavio je po prvi put *Provider* arhitekturu sa prvim provajderom koji je nosio naziv SUN. Pošto su izvozni zakoni SAD u to doba zabranjivali izvoz kriptografije, paket *Java Cryptographic Extension* (JCE) je objavljen kao odvojen,

opcioni paket. Pod nazivom JCE predstavljen je istovremeno interfejs ali i implementacije različitih algoritama za šifrovanje, generisanje i dogovaranje ključeva. Kriptografske implementacije u okviru Oracle/SUN JDK se distribuiraju u okviru nekoliko provajdera koji u svom nazivu još koriste ime SUN ("Sun", "SunJSSE", "SunJCE", "SunRsaSign"). Brojne današnje implementacije poput Bouncy Castle, Cryptix, Flexi Provider ili sama Oracle-ova JCA biblioteka koriste JCE kao interfejs putem kojeg je moguće pozivati pojedine provajdere.



Slika 16: Struktura Java Cryptography Architecture

4.1.2 AES NI skup instrukcija

Advanced Encryption Standard New Instructions (AES-NI) skup instrukcija je ekstenzija x86 skupa instrukcija za mikroprocesore proizvođača INTEL i AMD (Gueron, 2009). Ova ekstenzija koja je predložena od strane kompanije Intel u martu 2008. godine namijenjena je za ubrzanje šifrovanja i dešifrovanja pomoću algoritma AES. Kompanija Intel je ovaj skup instrukcija predstavila početkom 2010. godine u svim 32 nm Intel *Core* procesorima koji su bazirani na *Westmare* mikroarhitekturi.

Kako se navodi u (Xu, 2010), Intel AES-NI je skup od sedam novih instrukcija posvećenih algoritmu AES. Četiri instrukcije ubrzavaju enkripciju i dekripciju, dvije instrukcije pomažu generisanje ključeva i manipulaciju matricama a sedma instrukcija dodaje *carry-less* množenje:

AESENC	Vrši šifrovanje jedne AES-ove runde tako što u jednoj instrukciji vrši transformacije ShiftRows, SubBytes, MixColumns i AddRoundKey.
AESENCLAST	Vrši šifrovanje zadnje AES-ove runde tako što kombinuje ShiftRows, SubBytes i AddRoundKey transformacije.
AESDEC	Vrši dešifrovanje jedne AES-ove runde tako što u jednoj instrukciji vrši transformacije InvShiftRows, InvSubBytes, InvMixColumns i AddRoundKey.
AESDECLAST	Vrši dešifrovanje zadnje AES-ove runde tako što kombinuje InvShiftRows, InvSubBytes i AddRoundKey transformacije.
AESKEYGENASSIST	Pomaže pri generisanju ključa runde.
AESIMC	Pomaže pri inverznoj MixColumns rutini, koja se koristi za

	konverziju ključeva rundi u formu pogodnu za dekripciju.
PCLMULQDQ	Vrši CLMUL operaciju (*Carryless multiply*). Ova instrukcija vrši množenje u konačnim (Galoa) poljima.

Listing 17: Skup AES-Ni instrukcija

Mikroprocesori koji podržavaju ovaj skup instrukcija su *Intel Westmere* procesori (osim *Core i3* i pojedinih *Core i5* procesora), zatim *Sandy Bridge* procesori (osim *Pentium*, *Celeron i Core i3* procesora), *Ivy Bridge* procesori (svi *Core i5*, *Core i7* i *Xeon* procesori). Kompletan spisak svih procesora koji podržavaju AES-NI skup instrukcija može se naći na Intelovoj Web stranici. Od AMD-ovih procesora, ovaj skup instrukcija podržavaju procesori bazirani na *Bulldozer*, *Piledriver, Steamroller, Puma* i *Jaguar* arhitekturama.

Standardne biblioteke koje koriste AES-NI skup instrukcija su OpenSSL od verzije 1.0 (od verzije 0.9.8 kao *patch*), zatim *Intel Integrated Performance Primitives* (IPP) kriptografske biblioteke od verzije 6.1 i *Microsoft Cryptography API: Next Generation* koji se isporučuje sa Windows 7 i novijim operativnim sistemima.

Kada se radi o kompajlerima, Intelov C/C++ kompajler podržava ovaj skup instrukcija od verzije 11, Visual Studio od verzije 2008 (SP1), a GCC od verzije 4.4. AES-NI instrukcije mogu trenutno da se pozovu ili pomoću *inline* asembler funkcija ili korišćenjem *intrinsic* funkcija.

U teoriji kompajliranja, *intrinsic* funkcija je funkcija nekog programskog jezika čija implementacija se od strane kompajlera tretira na poseban način. Tipično, ona mijenja neki niz instrukcija slično *inline* funkcijama. Za razliku od *inline* funkcija, kompajler *intrinsic* funkcije može bolje da optimizuje. Visual Studio i Intelov C/C++ kompajler dozvoljavaju upotrebu *intrinsic* funkcija uključenjem wmmintrin.h datoteke nakon čega je moguće pozivati AES-NI instrukcije korišćenjem standardne C/C++ sintakse umjesto asemblera.

Microsoft Visual C++ trenutno se isporučuje sa 6 ovakvih funkcija:

_mm_aesenc_si128	Vrši šifrovanje jedne AES-ove runde, poput AESENC asembleske funkcije.
_mm_aesenclast_si128	Vrši šifrovanje zadnje AES-ove runde, poput AESENCLAST.
_mm_aesdec_si128	Vrši dešifrovanje jedne AES-ove runde, poput AESDEC asembleske funkcije.
_mm_aesdeclast_si128	Vrši dešifrovanje zadnje AES-ove runde, poput AESDECLAST asembleske funkcije.
_mm_aesimc_si128	Pomaže pri inverznoj Mix Columns rutini, poput AESIMC.
_mm_aeskeygenassist_si128	Pomaže pri generisanju ključa runde, poput AESKEYGENASSIST asembleske funkcije.

Listing 18: Skup AES-Ni intrinsic instrukcija

4.1.2.1 Prethodni radovi koji obrađuju AES-NI skup instrukcija

U Intelovom dokumentu (Gueron, 2012) predstavljen je izvorni kod programa koji vrši generisanje ključeva, te 128-bitno, 192-bitno i 256-bitno šifrovanje i dešifrovanje u ECB, CBC i CTR kriptografskim režimima rada. Ovdje je prikazan i izvorni kod za istovremenu (paralelnu) obradu 4 bloka podataka u ECB, i CTR režimu rada i za dešifrovanje u CBC režimu rada. Svi primjeri se mogu prevesti pomoću Intelov-og C/C++ kompajlera v11 ili novijeg.

Botan (Lloyd, 2013) je *open source* kriptografska C++ biblioteka koja se isporučuje sa većim brojem Linux distribucija, a koja je objavljena pod BSD 2 (ili FreeBSD) licencom. Jezgro biblioteke Botan je napisano u C++11 verziji ISO standarda ovog programskog jezika, a u njenom izvornom kodu se nalazi i implementacija algoritma AES koja koristi intrinsic funkcije za pozivanje AES-NI skupa instrukcija.

Kako se navodi u OpenSSL dokumentaciji, (OpenSSL Software Foundation, 2015) u slučaju x86-64 arhitektura OpenSSL biblioteka može da koristi tri nivoa optimizacije - čisti C, bez bilo kakvih optimizacija, SSSE3 bazirane optimizacije i AES-NI + PCLMULQDQ + SSSE3 opotimizacije. U okviru OpenSSL paketa AES-NI skup instrukcija je implementiran u okviru EVP biblioteke (crypto/evp/e_aes.c).

Kako se navodi u dokumentu *NSS Release Notes* 3.21 (MDN NSS RN, 2016) u NSS biblioteku je, uz pomoć Shay Guerona iz kompanije Intel, od verzije 3.14.2 ugrađena podrška za AES-NI skup instrukcija. Izvorni kod za podršku je napisan u assembleru. Zadnja objavljena verzija ove biblioteke nosi oznaku NSS 3.21.

TrueCrypt (TrueCrypt, 2009) je *open source* kriptografska aplikacija koja može da koristi AES-NI skup instrukcija za hardversko ubrzanje AES algoritma na kompjuterima koji posjeduju odgovarajući procesor. TrueCrypt koristi AES-NI instrukcije koje izvode AES runde, ali trenutno ne koristi AES-NI instrukcije za generisanje ključeva.

AES-NI skup instrukcija donio je i nove mogućnosti vezane za paralelizaciju izvršavanja ovoga algoritma. U svom istraživanju (Gueron, 2009) prikazuje rješenje za paralelizaciju dekripcije algoritmom AES u CBC kriptografskom režimu rada. U Intelovom dokumentu (Gueron, 2012) prikazuje rješenja za paralelizaciju izvršavanja algoritma AES u CBC i CTR režimu rada.

Takođe i (Hoban et al. 2013) u istraživanju mogućnosti unapređenja metoda za enkripciju u okviru operativnog sistema Linux pružaju rješenje za paralelizaciju izvršenja ovog algoritma pomoću XTS kriptografskog režima rada.

4.1.3 CUDA i paralelno programiranje kao tehnologija za poboljšanje performansi algoritma AES

CUDA je paralelna kompjuterska arhitektura opšte namjene sa paralelnim programskim modelom i skupom instrukcija koji koriste mogućnosti NVIDIA grafičkih procesora za rješavanje kompleksnih problema na efikasniji način nego što ih je moguće riješiti pomoću CPU-a (NVIDIA, 2015).

Prvi DirectX 10 grafički procesor pod nazivom NVIDIA GeForce 8800 GTX koji je predstavljen 2006. godine bio je istovremeno i prvi GPU koji je bio zasnovan na NVIDIA CUDA arhitekturi. Za razliku od prethodnih generacija grafičkih procesora, CUDA arhitektura je dozvoljavala da se programski pristupa svakoj pojedinoj aritmetičko logičkoj jedinici (ALU) i da se na njima izvode izračunavanja opšte namjene (Sanders and Kandrot, 2011). Svega nekoliko mjeseci nakon predstavljanja

NVIDIA GeForce 8800 GTX procesora NVIDIA je objavila i poseban CUDA C kompajler namijenjen za programiranje CUDA uređaja.

Do 2013 godine (Wilt, 2013) su se pojavile tri arhitekture CUDA mikroprocesora:

Tesla, koja je predstavljena 2006. godine u GeForce 8800 GTX (G80) procesoru,

Fermi, koja je predstavljena 2010. godine u GeForce GTX 480 (GF100) procesoru i

Kepler, koja je predstavljena 2012. godine u GeForce GTX 680 (GK104) procesoru

U februaru 2014. godine predstavljena je i najnovija, četvrta po redu, **Maxwell** arhitektura.

Današnji grafički procesori su evoluirali u visoko paralelizovane, višenitne i višejezgrene procesore sa ogromnom računarskom snagom i sa ogromnim memorijskim propustnim opsegom. Prema (NVIDIA Corporation, 2012), osnovna razlika između običnog procesora opšte namjene (CPU) i grafičkih procesora (GPU) je u tome što su grafički specijalizovani za intenzivna, visoko paralelizovana izračunavanja kakva su grafička izračunavanja, te su zbog toga dizajnirani tako da je više njihovih tranzistora posvećeno obradi podataka nego keširanju i kontroli toka, kako je vidljivo sa sljedeće slike:

Slika 17: Razlike između CPU i GPU (NVIDIA Corporation, 2012)

Sve navedeno dovodi do povećanja interesovanja za GPU procesore u istraživanjima koja se bave poboljšanjem performansi algoritma AES.

4.1.3.1 Prethodni radovi vezani za poboljšanje performansi pomoću paralelizacije

U svom istraživanju, (Manavski, 2007) razmatra mogućnosti implementacije algoritma AES pomoću, u to doba tradicionalnog, pristupa koji je zasnovan na OpenGL biblioteci kao i na implementaciji pomoću NVIDIA CUDA platforme, te utvrđuje benefite koji se postižu korišćenjem nove arhitekture.

Postojanje različitih kripotgrafskih načina rada je već poslužilo kao ideja za paralelizaciju izvršavanja pojedinih algoritama. Prema (Lipmaa et al. 2000) blokove C1, C2,... moguće je šifrovati u isto vrijeme pa je zbog toga CTR način rada moguće paralelizovati.

Brzi razvoj GPU (Graphic Processing Unit) jedinica kao i korisničkih okruženja poput CUDA kompanije NVidia ili OpenCL doveo je do različitih pokušaja paralelizacije AES algoritma pomoću CTR načina rada. Tako prema (Tran et al. 2011) autori najprije povećavaju veličinu bloka u odnosu na standardom definisani AES algoritam, a zatim koriste grubu (*coarse grained*) granularnost za paralelizovanje algoritma.

S druge strane autori u rješenju prikazanom u (Di Biagio et al. 2009) koriste finu (*fine grained*) granularnost i interni paralelizam svake runde tako što nezavisno manipulišu sa 4 32-bitne riječi (T-riječi) koje se javljaju u svakoj AES-ovoj rundi. U ovom istraživanju implementacije AES-CTR algoritma na CUDA platformi autori tvrde da postižu ubrzanje od oko 14 puta u odnosu na OpenSSL implementaciju koja se oslanjala na upotrebu običnog centralnog procesora (CPU) računara.

U (Jacquin, 2010) autori modifikuju već postojeće open source rješenje koje je ponudio Can Berk Güder (Berk Guder, 2009) da bi ubrzali izvršenje pomoću grube (*coarse grained*) granularnosti i CTR načina rada. U (Zola, 2012) autori koriste CUDA arhitekturu i CTR način rada sa WAES bibliotekom. Prema (Zola, 2012) WAES biblioteka je prva biblioteka koja je koristila predprocesiranje ili prethodnu obradu (Lipmaa et al. 2000) za povećanje brzine obrade podataka. Kako je kriptografska obrada u šifrovanju poruke M kod CTR načina rada nezavisna od M (Lipmaa et al. 2000) jer se šifruje brojač (*counter*), može se koristiti i predprocesiranje ili prethodna obrada za povećanje brzine. Ovu osobinu WAES biblioteke autori nazivaju špekulativna enkripcija (Zola, 2012).

Istraživanje uticaja alocirane GPU memorije i granularnosti programskih niti (*threads*) na performanse izvršavanja AES algoritma u ECB režimu rada koje su obavili (Iwai et al. 2010) pokazalo je da najbolje performanse pokazuju implementacije koje obrađuju 16 bajta u jednoj niti, dok implementacije koje obrađuju 8 bajta u dvije ili 4 bajta u 4 niti pokazuju lošije performanse zbog potrebe sinhronizacije niti, a pored toga im je potrebna i dijeljena memorija za rad.

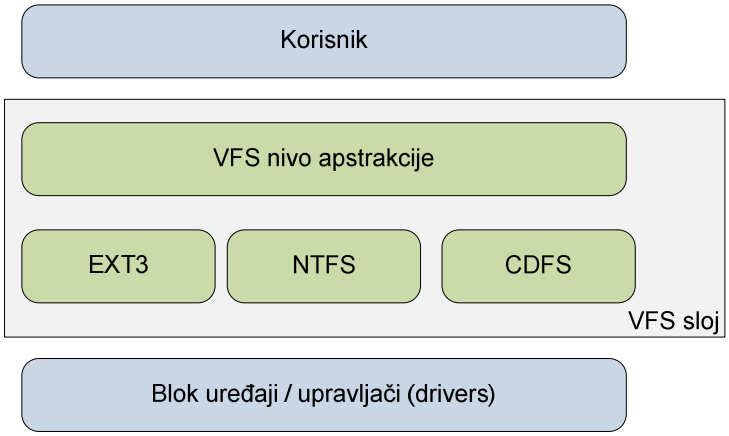
U toku istraživanja koje je predstavljeno u (Mei et al. 2010), autori istražuju mogućnost poboljšanja paralelizacije algoritma AES u ECB načinu rada pažljivim aranžiranjem podataka u memoriji grafičke karte. Prema njihovim navodima, oni uspijevaju da postignu određena ubrzanja u odnosu na jednonitnu sekvencijalnu implementaciju AES algoritma, kao i u odnosu na implementaciju koja na CPU paralelno obrađuje podatke pomoću 4 niti.

U (Nishikawa et al. 2011) autori su implementirali četiri različita blokovska algoritma, a zatim su procjenjivali njihovu efikasnost na dva CUDA GPGPU koji se oslanjaju na Fermi i GT200 arhitekturu.

Autori u (Ortega et al. 2011) koriste OpenMP (*Open Multiprocessing*) API da bi uporedili mogućnosti paralelizacije izvršenja na različitim testnim platformama sa različitim CPU i GPU jedinicama. Prema prikazanim rezultatima, moguće je postići značajna poboljšanja performansi kada se koristi paralelna obrada na GPU jedinicama.

4.2 Virtuelni fajl sistem Apache Commons

Virtuelni fajl sistem (VFS) je apstraktan sloj koji se nalazi iznad nekog konkretnog fajl sistema. Namjena virtuelnih fajl sistema je da dozvoli klijentskim aplikacijama da pristupaju različitim tipovima konkretnih fajl sistema na jedinstven način. Dakle, VFS nije niti fajl sistem na disku, ni mrežni fajl sistem, niti neka posebna struktura podataka. Ovakvi sistemi se koriste da od aplikacije sakriju interfejse niskog nivoa koji pripadaju različitim implementacijama (FAT, EXT3 itd).



Slika 18: Struktura virtuelnog fajl sistema

Apache Commons je projekat Apache fondacije koji je fokusiran na razvoj različitih Java komponenata. Jedna od komponenata koja se razvija u okviru ovoga projekta je i Commons VFS (Apache Fondation, 2014) virtuelni fajl sistem koji tretira različite tipove datoteka i protokola kao kompletne fajl sisteme. Commons VFS pruža jedan API za pristup različitim fajl sistemima, pri čemu uniformno predstavlja datoteke koje dolaze iz različitih izvora, kao što su ZIP arhiva, HTTP server ili datoteka na lokalnom disku.

Neke od osnovnih karakteristika virtuelnog fajl sistema Apache Commons su (Apache Fondation, 2014):

- Jedinstven i konzistentan API za pristup datotekama različitog tipa;
- Podrška velikom broju tipova datoteka;
- Keširanje informacija o datotekama, koje se obavlja u JVM. Opciono, može da kešira Informacije iz lokalnog fajl sistema;
- Podrška lokalnim fajl sistemima koji su sačinjeni od datoteka iz različitih fajl sistema;
- Alati za integraciju Commons virtuelnog fajl sistema u aplikacije, poput ClasLoader-a i klase URLStreamHandlerFactory;
- Postojanje skupa Ant zadataka (Ant tasks) koji su namijenjeni za kopiranje, pomjeranje i brisanje datoteka, kreiranje direktorijuma, te sinhronizaciju sadržaja nekog odredišnog foldera sa skupom izvornih foldera i datoteka.

U okviru ovog rada će umjesto kreiranja kompletnog operativnog sistema biti korišćen Apache Commons VFS kao fajl sistem koji će služiti kao kontejner za aplikaciju koja demonstrira u radu navedene koncepte.

4.3 Rudarenje podataka i mašinsko učenje

Sa napretkom računarskih tehnologija svakodnevno se uvećavaju mogućnosti smještanja i obrade velikih količina podataka. Zahvaljujući sveprisutnim računarima, moguće je sačuvati mnoge podatke koji bi ranije bili bačeni. Jeftini diskovi i *online* prostor omogućavaju da jednostavno odložimo odluku o tome šta da se uradi sa nekim podacima. Na svakom koraku elektronika bilježi ljudske odluke, izbore u supermarketu, finansijske navike. WWW je preplavljen informacijama, a svaka odluka nekog korisnika se bilježi. Činjenica je da postoji disparitet između količine

informacija koja se danas generiše i mogućnosti razumijevanja tolike količine informacija (Witten et al. 2011).

U oblasti rudarenja i analize podataka (*Data Mining*), podaci su smješteni na elekronski način a sve pretrage su automatizovane, odnosno poboljšane od strane kompjutera. *Data mining* se kao oblast bavi analiziranjem podataka koji su već prisutni u bazama podataka. Zbog toga se ova oblast definiše kao proces otkrivanja obrazaca (*pattern*) u podacima. Obrasci koji se otkriju treba da nose neko značenje da bi donijeli neku korist (Witten et al. 2011).

Disciplina koja se u velikoj mjeri podudara sa rudarenjem podataka je mašinsko učenje. Zahvaljujući pomenutom tehnološkom napretku mašinsko učenje se danas koristi u sve većem obimu kao tehnologija koja podacima daje smisao. Mašinsko učenje leži na presjeku računarskih nauka i inženjerstva pri čemu se u velikoj mjeri oslanja na statistiku. Najveći dio umjetnosti mašinskog učenja sastoji se u redukovanju ranga velikog broja mogućnosti na mali broj prototipa (Smola and Vishwanathan, 2008).

Obrasci omogućavaju davanje netrivijalnih predviđanja za nove podatke. Obrasci koji omogućavaju da se ispituje njihova struktura, o kojoj se može rasuđivati i koristiti za donošenje budućih odluka nazivaju se strukturni obrasci. U pronalaženju strukture i analizi podataka koriste se algoritmi mašinskog učenja (Witten et al. 2011).

Postoji veliki broj problema kod kojih rješenje nije determinističko, jer za njih ili nema dovoljno podataka ili nema dovoljno procesorske snage za ispravno modeliranje problema. Npr. broj pokrenutih aplikacija i zauzeće računarskih resursa predstavljaju problem koji je veoma teško modelirati bez pomoći statistike. Međutim, čak i kada nije moguće proces potpuno modelirati i identifikovati, moguće je konstruisati upotrebljivu aproksimaciju i otkriti određene obrasce i pravila koja bolje objašnjavaju neki proces.

Mašinsko učenje se može klasifikovati na nadzirano (*supervised*) i nenadzirano (*unsupervised*) učenje. Zadaci nadziranog mašinskog učenja su klasifikacija i regresija dok su zadaci nenadziranog mašinskog učenja klastering (grupisanje sličnih stavki) i procjena gustoće, koja predstavlja pronalaženje statističkih vrijednosti koje opisuju podatke (Harrington, 2012).

Rudarenje podataka (*data mining*) je multidisciplinarna oblast koja se još naziva i otkrivanje znanja u podacima (*Knowledge Discovery from Data*, KDD). Ona predstavlja automatizovano izdvajanje obrazaca koji predstavljaju znanje koje je implicitno smješteno ili zarobljeno u velikim bazama podataka, skladištima podataka (*data warehouses*), na Webu ili u drugim masovnim repozitorijumima informacija (Han and Kamber, 2006).

Danas u svijetu postoji veliki broj komercijalnih softverskih paketa koji služe za rudarenje podataka: Angoss KnowledgeSTUDIO, Clarabridge, HP Vertica Analytics Platform, IBM SPSS Modeler, Microsoft Analysis Services, Oracle Data Mining, STATISTICA Data Miner

Pored komercijalnih programa, postoji i veliki broj *open source* rješenja poput: KNIME, R, ML-Flex, Databionic ESOM Tools, Orange, Natural Language Toolkit, SenticNet API , ELKI , RapidMiner, SCaViS, UIMA, Weka, Chemicalize.org, Vowpal Wabbit, GNU Octave, Mlpy, MALLET, Shogun, Scikit-learn, LIBSVM, LIBLINEAR, Lattice Miner, Rattle GUI, Dlib, Apache Mahout, Jubatus, KEEL, Gnome-datamine-tools.

U dostupnoj literaturi nisu pronađeni tragovi da postoje bilo kakva rješenja koja koriste metode mašinskog učenja i rudarenja podataka (*data mining*) u procesu prilagođenja operativnih sistema raspoloživim resursima radi poboljšanja performansi bilo kojeg kriptografskog algoritma.

U ovom istraživanju je korišćeno mašinsko učenje radi optimizacije performansi računara uz upotrebu trening podataka i na osnovu njih generianih prediktivnh modela za izbor najefikasnijeg kriptografskog resursa za izvršavanje kriptografskog algoritma AES. U tu svrhu je korišćen softverski paket Weka (*Waikato Environment for Knowledge Analysis*) pisan u programskom jeziku Java.

Kako se navodi u (Lantz, 2013), tokom procesa reprezentacije znanja, računar sumarizuje sirove podatke u ***model***, kao eksplicitan opis strukturiranih obrazaca unutar podataka. Primjeri modela su jednačine, dijagrami, logička If/else pravila, grupisanje podataka u klastere itd. Proces uklapanja podataka u određeni model naziva se ***trening***. U nastavku će sumarizovani podaci koji su pomoću metoda mašinskog učenja predstavljeni jednačinama i dijagramima biti navođeni pod nazivom model, dok će navedeni proces uklapanja podataka u model biti referenciran pod nazivom trening.

4.3.1 Weka - radna površina za rudarenje podataka

Weka (*Waikato Environment for Knowledge Analysis*) je popularna biblioteka programa za rudarenje podataka i mašinsko učenje koja je pisana programskom jeziku u Java. Biblioteka Weka je razvijena na Univerzitetu Waikato u Novom Zelandu.

Weka *workbench* je kolekcija *state-of-the-art* algoritama za mašinsko učenje i alata za predprocesovanje podataka. Ova biblioteka je dizajnirana tako da olakšava ispitivanje postojećih metoda nad novim skupovima podataka na fleksibilan način. Weka pruža veliku podršku cijelom procesu mašinskog učenja i rudarenja podataka, uključujući i pripremu ulaznih podataka, statističko izračunavanje obrazaca podataka, analizu podataka te izradu i snimanje prognostičkih (prediktivnih) modela (Hall et al. 2009), vizuelizaciju ulaznih podataka i rezultata učenja (Witten et al. 2011).

Prva verzija Weka biblioteke nosila je naziv TCL/TK a bila je pisana kao *front-end* ka različitim algoritmima koji su bili implementirani u drugim programskim jezicima. Najnovija verzija ove biblioteke čiji razvoj je počeo još 1997. godine napisana je potpuno u Java programskom jeziku a nosi naziv Weka 3. Ova biblioteka se danas koristi u brojnim oblastima, a posebno u oblasti obrazovanja i u različitim istraživanjima.

Kako je Weka pisana u programskom jeziku Java, ona se može koristiti na gotovo svim današnjim platformama. Testirana je na Linux, Windows i Macintosh operativnim sistemima, pa čak i na PDA uređajima. Ona pruža jedinstven interfejs za pristup velikom broju algoritama iz oblasti mašinskog učenja, zajedno sa metodama za pred i post procesiranje podataka (Witten et al. 2011).

Prednosti koje Weka nudi su:

- Besplatna dostupnost u okviru GNU GPL licence.
- Portabilnost, jer se može pokrenuti na gotovo svakoj modernoj platformi.
- Zavidna kolekcija algoritama i tehnika za obradu podataka i modeliranje.

- Jednostavnost upotrebe zahvaljujući grafičkom radnom okruženju.

Weka *workbench* sadrži metode za rješavanje najvažnijih problema iz oblasti rudarenja podataka: regresiju, klasifikaciju, klastering, traženje pravilnosti u vezama i selekciju atributa. Pored ovoga, u okviru Weke se nalaze i mnogi alati za vizuelizaciju i predprocesovanje podataka. Svi algoritmi uzimaju podatke na ulazu u jedinstvenom formatu pod nazivom ARFF, koji može biti očitan iz datoteke ili iz rezultata upita nad nekom bazom podataka (Witten et al. 2011).

Weki se pomoću grafičkog interfejsa pristupa kroz komponentu pod nazivom Explorer. On omogućava učitavanje podataka iz ARFF datoteke, nakon čega prikazuje samo algoritme koji su odgovarajući za navedene podatke. Sličnu funkcionalnost pružaju i interfejs Knowledge Flow i komandna linija. U Weki postoji i interfejs koji se naziva Experimenter, koji omogućava sistematsko poređenje različitih algoritama mašinskog učenja na različitim kolekcijama podataka.

Veoma važan dio Weka *workbencha* je i *online* dokumentacija koja se automatski generiše iz *open source* izvornog koda i koja odražava njegovu strukturu. *Online* dokumentacija pruža najbolji uvid u kompletnu listu trenutno dostupnih algoritama. Kako Weka neprekidno raste, dokumentacija koja se generiše na osnovu izvornog koda obezbjeđuje da je ona uviek ažurna.

U mnogim *data mining* aplikacijama komponenta zadužena za mašinsko učenje je samo mali dio mnogo većeg softverskog sistema. Weka biblioteka omogućava programerima pristup velikom broju algoritama iz svog izvornog koda. Na ovaj način, programeri mogu da rješavaju probleme vezane za mašinsko učenje direktno iz svojih aplikacija, uz minimum dodatnog kodiranja.

Grana mašinskog učenja pod nazivom empirijsko učenje skoncentrisana je na izgradnju ili reviziju modela u svjetlu velikih brojeva, pri čemu uzima u obzir probleme poput nedostajućih podataka ili šuma. Mnogi takvi modeli uključuju klasifikaciju i algoritme učenja koji generišu stabla odlučivanja (*decision trees*) i koji su efikasni, robustni i relativno jednostavni. Međutim, postoje zadaci koji zahtijevaju da model predviđa numeričku vrijednost a ne klasu kao u slučaju stabala odlučivanja. Postoje modeli prema kojima se u svrhu predviđanja grade regresiona stabla koja se razlikuju od stabala odlučivanja samo po tome što u listovima imaju vrijednosti umjesto stabala. Mnoge klasične metode poput višestruke linearne regresije obrađuju isti zadatak na takav način da najprije formiraju jednostavan model, a zatim pronalaze parametre koji se maksimalno uklapaju u trening podatke (Quinlan, 1992). U nastavku će biti opisane M5 i M5prim metode koje se zasnivaju na kombinaciji stabala odlučivanja i regresionih metoda.

4.3.2 Regresiona analiza

Regresiona analiza je metoda koja se koristi za opisivanje odnosa između dvije ili više varijabli. Ona istražuje uticaj jedne illi više varijabli na neku drugu varijablu. Regresiona analiza procjenjuje uslovna očekivanja vrijednosti zavisne varijable u odnosu na promjene jedne ili više nezavisnih varijabli. U svojoj najjednostavnijoj formi koristi se linearna regresija za modeliranje skupa tačaka kako slijedi:

$$\overline{Y} = \overline{B}_0 + \overline{B}_1 x \tag{60}$$

U okviru regresione analize koristi se procjena standardnih grešaka da bi se kreirali intervali povjerenja. Standardna greška je mjera disperzije koeficienata B0 i B1.

Interval povjerenja predstavlja procjenjen rang vrijednosti koji će vjerovatno da uključi nepoznati parametar populacije (Lovrić et al. 2008).

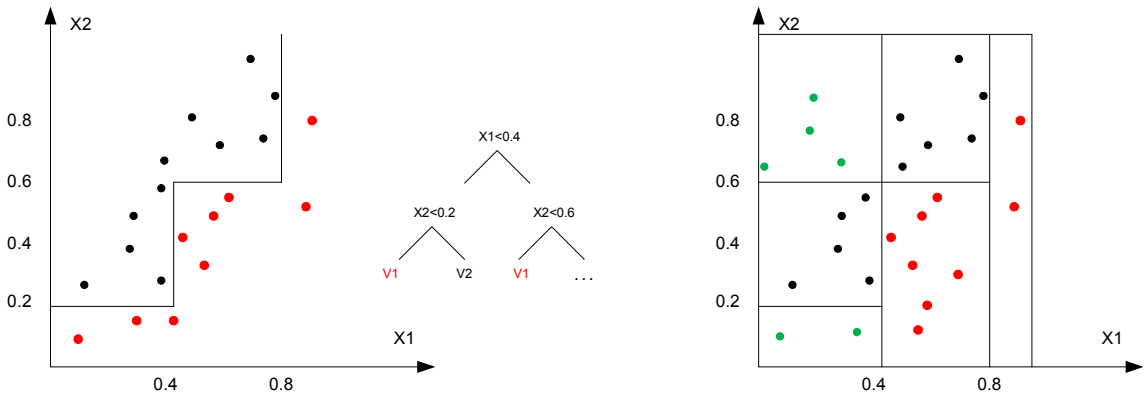
4.3.3 M5 i M5' metode

M5 je sistem za kreiranje modela za mašinsko učenje koji predviđaju vrijednosti. Kao i CART (Classification And Regression Trees) metod, M5 gradi modele bazirane na stablima. Međutim, dok regresiona stabla imaju vrijednosti u svojim stablima, stablo koje je konstruisano pomoću M5 algoritma sadrži višestruke (multivarijantne) linearne modele. Ovi linearni modeli su zbog toga analogni hibridnim (*piecewise*) funkcijama koje se sastoje od mnogo linearnih podfunkcija. Zbog toga M5 metoda može efikasno da obrađuje zadatke veoma velikih dimenzija koje se mjere u stotinama atributa (Quinlan, 1992).

Indukcija predstavlja izvođenje opštih zakonitosti na osnovu uvida u konkretne pojave, odnosno slučajeve (Alpaydın, 2010). U praksi se vrlo često javlja situacija kada treba predvidjeti ponašanje "klase" koja uzima kontinuirane numeričke vrijednosti. Klasične metode poput stabla odlučivanja su razvijene za slučaj kada su vrijednosti klasa i atributa diskretni. U teoriji je vrlo čest slučaj i da se proširuju tehnike indukcije koje funkcionišu sa numeričkim atributima na takav način da odaberu prag na svakom čvoru stabla, a zatim testiraju vrijednost u odnosu na taj prag. Ipak, stabla odlučivanja i pravila odlučivanja (*decision trees*, *decision rules*) nisu upotrebljivi ako su vrijednosti klasa numeričke (Wang and Witten, 1996).

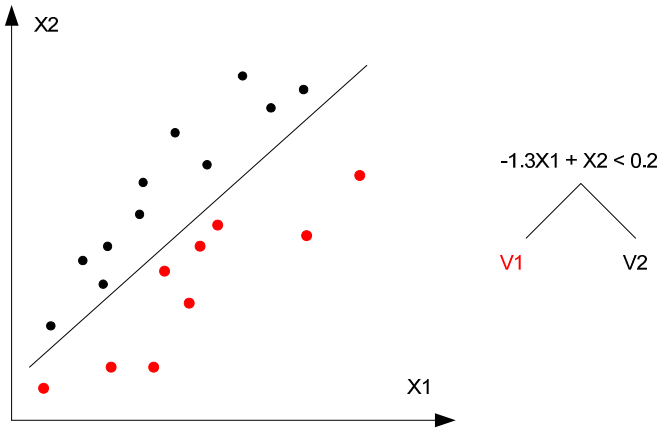
Postoje mnoge druge tehnike učenja koje predviđaju numeričke vrijednosti. Ove tehnike uključuju standardnu regresiju, neuronske mreže, metode zasnovane na instancama (*instance based*), regresiona stabla i predviđanje pomoću diskretizacije vrijednosti klase unaprijed. Međutim, sve navedene metode imaju određene slabosti (Wang and Witten, 1996). Prvi korak ka prevazilaženju tih slabosti napravljen je kreiranjem tehnike za obradu problema sa kontinuiranim vrijednostima klasa, odnosno "stablo modela" koji se u (Quinlan, 1992) još naziva i M5 metoda. Kombinovanjem navedene metode sa CART metodom (Breiman et al. 1984), za rad sa pobrojanim atributima i metode za tretiranje nedostajućih vrijednosti rezultiralo je metodom koja indukuje stabla modela pod nazivom M5 prime ili M5'.

Ova metoda se razlikuje od metode regresionog stabla koja se koristi kada se predviđeni izlaz iz algoritma može posmatrati kao realan broj. U slučaju regresionog stabla, sve linije koje vrše particionisanje su paralelne osama, jer svaki pojedini čvor provjerava da li je neka varijabla veća ili manja od date vrijednosti (Shalizi, 2009).



Slika 19: Primjeri regresionih stabala (Shalizi, 2009).

Stabla modela (*model trees*) su veoma slična regresionim stablima, ali svaki čvor koristi funkciju linearne regresije za odlučivanje.



Slika 20: Primjer stabala modela (Shalizi, 2009).

Za kreiranje stabla koristi standardnu redukciju devijacije (*Standard Deviation Reduction, SDR*):

$$SDR = sd(T) - \sum_i \frac{|T_i|}{T} \times sd(T_i) \tag{61}$$

gdje su T[1], T[2] itd. skupovi podijeljenih podataka na čvoru.

Za orezivanje stabla koristi se traženje najmanje procijenjene apsolutne greške:

$$\frac{n+v}{n-1} \times aps.procijenjena.greška \tag{62}$$

gdje je n broj trening instanci koje dosegnu čvor, a v je broj parametara u linearnom modelu.

Finalna faza u M5 prim metodi je glačanje (*smooting*). Ovaj proces se koristi da bi se kompenzovali oštri prekidi koji se javljaju na prelasku između susjednih linearnih modela na listovima orezanog drveta. Smooting je tehnika koja uzima izlaznu vrijednost modela linearne regresije i koristi je za odgovarajući čvor:

$$p' = \frac{np + kq}{n + k} \tag{63}$$

gdje je :

p' - predviđanje proslijeđeno roditeljskom čvoru,

p - predviđanje koje je primljeno iz dječijeg čvora,

q - vrijednost koja je predviđena od modela u ovom čvoru,

n - broj instanci koje dosegnu čvor,

k - smooting konstanta.

Opisani metod predstavlja javnu metodu za indukovanje modela iz numeričkih podataka koji uključuju kontinuirane klase. Određena implementacija nekog kriptografskog algoritma se, u određenim uslovima, izvršava brže ili sporije od neke druge implementacije. Atributi koji utiču na vrijeme izvršavanja poput veličine internog bafera, veličine datoteke, kriptografskog algoritma, dužine ključa, broja niti ili načina rada (*mode of operation*) prema ovoj metodi mogu da se najprije klasifikuju, na osnovu čega može da se vrši predviđanje vremena izvršenja, kao još jednog atributa po pojedinoj klasi. Ova metoda je implementirana u Weka biblioteci i može se jednostavno uključiti u izvorni kod neke aplikacije.

4.3.4 Tehnika penjanja (*Hill-climbing*)

Tehnika penjanja (*Hill-climbing*) je tehnika za matematičku optimizaciju koja pripada familiji lokalnih pretraga. Ovo je iterativni algoritam koji počinje sa jednim proizvoljnim rješenjem problema, a zatim traži bolje rješenje tako što iterativno mijenja jedan element. Ako promjena proizvede bolje rješenje, iterativna pretraga se nastavlja od boljeg rješenja, i nastavlja se sve dok se mogu postići poboljšanja (Russell and Norvig, 2003). *Hill-climbing* algoritam je prikazan u listingu 19 u *steepest-ascent verziji*.

```
Function hill-climbing (problem) returns a state that is local maximum
 current = MakeNode(problem, InitialState)
 Loop do
   neighbor  a highest valued successor of current
   If (neighbor.Value < current.Value) then Return current.State
   current =  neighbor
 End Loop
End Function
```

Listing 19: Hill-climbing algoritam (Russell and Norvig, 2003)

Ovaj algoritam se kontinuirano pomiče u pravcu rastuće vrijednosti. Pri svakom slijedećem koraku, trenutni čvor je zamjenjen najboljim susjedom, odnosno susjedom sa najvišom vrijednošću.

Hill-climbing algoritam se još naziva i gramziva lokalna pretraga (*greedy local search*) jer uzima prvog susjeda bez pokušaja da predvidi slijedeći korak. Nedostaci ovog algoritma su problemi lokalnog maksimuma i platoa, koje ovaj algoritam teže rješava.

Random restart Hill-climbing algoritam pokušava da riješi navedeni problem tako što će izvršiti seriju *Hill-climbing* pretraga iz slučajno odabranih inicijalnih stanja.

4.4 Sistem za upravljanje bazama podataka Firebird

Firebird je open source relacioni sistem za upravljanje bazama podataka koji je moguće instalirati na Linux, Windows i brojne Unix platforme. Ovaj RDBMS (*Relational Database Management System*) je nastao iz Borlandovog InterBase-a 2000. godine, kada je Borland objavio izvorni kod InterBase-a. Iste sedmice kada je Borland objavio izvorni kod SUBP InterBase, 25. jula 2000. godine na Web lokaciji SourceForge je kreiran projekt Firebird.

U februaru 2004. godine, objavljena je verzija 1.5 ovog servera, koja je počela da se sve više odvaja od Borlandovog Interbase-a. Trenutno stabilna verzija (početak 2016.) Firebird servera je 2.5.5.

Ovaj SUBP zasnovan je na klijent - server arhitekturi, a podržava većinu naprednih koncepata SQL standarda, uključujući podršku stored procedurama i trigerima u vlastitom PSQL jeziku. Ovaj SUBP od 2006 godine dodaje podršku 64-bitnoj arhitekturi, a od 2013. godine na konferenciji *Google Summer Code* počeli su radovi na integraciji SUBP Firebird kao zamjene za HSQLDB u okviru programskog paketa Libre Office.

4.5 Java Database Connectivity (JDBC)

Kako se navodi u (Iyer et al. 2010) *Java Database Connectivity* (JDBC) je Java standard koji daje interfejs za povezivanje Java programa sa relacionim bazama podataka. JDBC standard je definisan od strane kompanije Sun Microsistems i implementiran je preko *java.sql* interfejsa. On omogućava individualnim dobavljačima da implementiraju i proširuju standard svojim vlastitim JDBC upravljačima (*drivers*).

JDBC dozvoljava postojanje višestrukih implementacija u okviru iste aplikacije. Ovaj API pruža mehanizam za dinamičko učitavanje korektnih Java paketa i njihovo registrovanje pomoću JDBC Driver Manager upravljača. JDBC konekcije omogućavaju kreiranje i izvršavanje SQL iskaza, bilo da se radi o ažuriranju (*Create*, *Insert*, *Update* ili *Delete*) ili o *Select* iskazima za pribavljanje podataka.

Biblioteka programa za mašinsko učenje Weka se može povezati sa bilo kojim sistemom za upravljanje bazama podataka koji posjeduje JDBC upravljač.

4.6 Mehanizmi interprocesne komunikacije

U literaturi se mogu naći različite definicije i različite metode interprocesne komunikacije. Tako se u (Silberschatz et al. 2011) navodi da je za kooperaciju procesa neophodno obezbjediti mehanizme inteprocesne komunikacije koji će im omogućiti razmjenu podataka i informacija. Tu se takođe navodi i da postoje dva osnovna modela interprocesne komunikacije – dijeljena memorija (*shared memory*) i proslijeđivanje poruka (*message passing*), dok se za komunikaciju u klijent server okruženjima navode soketi (*sockets*), daljinsko pozivanje procedura (*remote procedure call - RPC*) i cijevi (*pipes*). U (Tanenbaum and Bos, 2014) autori navode semafore (*semaphores*), mutekse (*mutexes*), monitore (*monitors*) i i proslijeđivanje poruka (*message passing*) kao metode interprocesne komunikacije. U Microsoftovoj dokumentaciji (Microsoft Corp. IPC, 2016) navodi se da su u operativnom sistemu Windows podržani slijedeći mehanizmi interprocesne komunikacije: Clipboard, COM, Data Copy, Dynamic Data Exchange (DDE), File Mapping, Mailslots, Pipes, RPC i Windows Sockets.

Kako su u primijenjenom dijelu rada korišćeni mehanizmi soketa i cijevi, u nastavku će biti detajnije predstavljena ova dva načina interprocesne komunikacije.

4.7 Tehnologija soketa

Soket (*socket*) se definiše kao krajnja tačka u komunikaciji. Par procesa koji komuniciraju preko mreže će najčešće uposliti par soketa - po jedan soket za svaki proces. Soket se identifikuje pomoću IP adrese nakon koje slijedi broj porta. Generalno, soketi koriste klijent server arhitekturu. Server očekuje zahtjev klijenta osluškivanjem određenog porta. Jednom kada je zahtjev primljen, server prihvata konekciju od klijentskog soketa da bi kompletirao konekciju. Serveri koji

implementiraju specifične servise (poput telnet, FTP i HTTP) osluškuju unaprijed određene portove. Npr. telet osluškuje port 23, FTP server osluškuje port 21, a Web ili HTTP server osluškuje port 80 (Silberschatz et al. 2011). Inetrnet soketi su bazirani na Berkley Sockets standardu.

Za implementaciju adaptibilnog virtuelnog kriptografskog fajl sistema i za implementaciju pojedinih kriptografskih modula korišćeni su programski jezici Java i Microsoft Visual C/C++. U okviru programskog jezika Java korišćene su klase Socket i ServerSocket, koje implementiraju serverske i klijentske sokete. Serverski soketi osluškuju zahtjeve na nekom portu, izvršavaju neku operaciju koja zavisi od zahtjeva i potencijalno vraćaju rezultat pošiljaocu. Klasa Socket implementira klijentske sokete (ili samo sokete) koji predstavljaju krajnju tačku u komunikaciji između mašina.

U okviru Microsoft .NET 4.5 radnog okruženja nalazi se klase TcpListener i TcpClient koje služe sa kreiranje serverskog soketa i prihvatanje konekcije odnsono za kreiranje klijentskog soketa. Međutim, u slučaju CUDA aplikacije koja koristi CUDA C kompajler bilo je potrebno implementirati klijentski i serverski soket u okviru Microsoft Winsock2 API-ja pomoću objekta SOCKET koji služi i kao serverski i kao klijentski soket.

4.8 Tehnologija imenovanih i neimenovanih cijevi (Named Pipes - Unnamed, Anonymous Pipes)

Neimenovane cijevi predstavljaju pojednostavljeni (*simplex*) FIFO kanal komunikacije koji može da se koristi za jednosmjernu međuprocesnu komunikaciju. Imenovane cijevi (FIFO) predstavljaju ekstenziju tradicionalnog koncepta imenovanih cijevi iz operativnog sistema UNIX koji se koristi za međuprocesnu komunikaciju.

Kada je programski jezik Java u pitanju, neimenovane cijevi mogu da se koriste na takav način da glavni program prije kreiranja dječijeg procesa kreira cijev. Navedena cijev se zatim obično pomoću ulazno-izlaznih tokova povezuje sa glavnim programom.

Imenovane cijevi se u UNIX i Linux operativnim sistemima kreiraju pomoću ključnih riječi *mkfifo*, dok se u Windows operativnim sistemima kreiraju i konfigurišu pomoću API funkcija koje su specifične za ulazno izlazne mogućnosti (Wells, 2009).

U ovom istraživanju su korišćene neimenovane cijevi i Java tehnologija ulazno-izlaznih tokova da bi se ostvarila jednosmjerna komunikacija od nekog kriptografskog modula ka adaptibilnom virtuelnom kriptografskom fajl sistemu. Da bi se kreirao proces pomoću programskog jezika Java korišćena je klasa ProcessBuilder u sastavu paketa *java.lang.ProcessBuilder*. Prema (Gosling et al. 2013), svaki ProcessBuilder objekat upravlja većim brojem atributa, od kojih je ovdje od interesa standardan ulaz (*input*), izlaz (*output*) i greška (*error*). Podrazumjevano, podproces šalje standardan izlaz i grešku pomoću cijevi, koje Java aplikacija može da preuzme pomoću metode Process.getInputStream() .

Navedena tehnologija je korišćena za prijem jedne poruke iz pokrenutog kriptografskog modula. Za slučajeve razmjene više poruka sa više podataka korišćena je ranije opisana tehnologija soketa.

4.9 XML standard

Extensible Markup Language (XML) je alat za skladištenje podataka koji je moguće konfigurisati tako da primi informaciju bilo koje vrste, ali takođe i otvoren standard koji se i dalje razvija i koji je prihvata sve zahtjeve, bez obzira da li oni dolaze od strane bankara ili od strane webmastera (Ray, 2001).

XML je kontejner koji služi za čuvanje, oblikovanje, označavanje, struktuiranje i zaštitu informacija. XML sve nabrojano postiže zahvaljujući simbolima koji su ugrađeni u tekst koji se naziva *markup* (oznake). XML *markup* ili oznaka proširuje značenje informacije na više načina jer identifikuje njene dijelove i utvrđuje vezu koja postoji među tim dijelovima informacije (Ray, 2003).

XML se, na jednom nivou može posmatrati kao protokol za smještanje i upravljanje informacijama. Na drugom nivou, on predstavlja porodicu tehnologija koja je u stanju da radi gotovo sve – od formatiranja dokumenata do filtriranja podataka. A na najvišem nivou, XML je filozofija pomoću koje upravljamo informacijama (Ray, 2001).

Extensible Markup Language (XML) 1.0 (XML W3C 1.0, 2008) je opisan u istoimenom W3C dokumentu. Navedena specifikacija je osnova iz koje raste XML stablo različitih tehnologija. Ova specifikacija se nadovezuje na UNICODE standard da bi obezbjedila striktna pravila za formatiranje teksta kao i na Document Type Definition (DTD) jezik za validaciju. Specifikacija XML 1.1 (XML W3C 1.1, 2006) je prva revizija koja mijenja definiciju dobro formatiranog dokumenta. Primarna izmjena je revizija tretmana karaktera u XML specifikaciji da bi se specifikacija što prirodnije prilagodila promjenama u Unicode specifikacijama, kao i da bi olakšao normalizaciju karaktera širom različitih Unicode verzija u skladu sa preporukama u specifikaciji Character Model for the World Wide Web 1.0 (CM W3C, 2005). XML je zasnovan na Standard Generalized Markup Language (SGML) jeziku, koji je definisan kao ISO 8879:1986 standard. XML s jedne strane predstavlja značajno pojednostavljenje SGML-a, a s druge strane uključuje prilagođenja zbog kojih postaje mnogo bolje prilagođen Web okruženju (Ogbuji, 2004).

XML standard se u istraživanju koristi kao primarno sredstvo za definisanje načina i formatiranje poruka koje treba da se izmjenjuju između adaptibinog virtuelnog kriptografskog fajl sistema i pojedinh kriptografskih modula.

4.10 Kriptografski načini rada koji su uticali na paralelizaciju izvršenja

Danas postoje dva osnovna mehanizma koje simetrični algoritmi (Mollin, 2010) (Chakraborty and Rodriguez-Henriquez, 2009) koriste za uzimanje podataka u procesu šifrovanja. Podaci se mogu uzimati blok po blok i kao jedan tok. Blokovski algoritmi (Mollin, 2010) (Chakraborty and Rodriguez-Henriquez, 2009) uzimaju blok podataka, šifruju ga pomoću tebele ključeva, pa zatim uzimaju slijedeći blok i tako u krug. Zahvaljujući svojoj velikoj brzini i efikasnosti, simetrični blokovski algoritmi koriste se u aplikacijama u kojima je potrebna enkripcija velike količine podataka velikom brzinom.

Međutim, osobina simetričnih blok algoritama je da će dva identična bloka koja se nalaze u okviru otvorenog teksta kojeg šifrujemo istim ključem kao rezultat dati identične blokove šifrata. Ovaj nedostatak blokovskih algoritama doveo je do pojave brojnih različitih kriptografskih načina rada (Mollin, 2010) (Rogaway, 2011) (Ferguson et al, 2010) (Dworkin, 2001) (modes of operation). Iako se u literaturi najčešće spominje 5 osnovnih (Mollin, 2010) (Dent and Mitchell, 2005) ili tradicionalnih

(Chakraborty and Rodriguez-Henriquez, 2009) načina rada, Phillip Rogaway (Rogaway, 2011) opisuje čak 17 ovakvih načina rada.

Blokovski algoritmi se intenzivno koriste još od 1971. godine, kada je grupa istraživača pod vođstvom Horsta Feistela u IBM-u razvila familiju algoritama pod imenom Lucifer (Feistel, 1971). Rane verzije Lucifera su radile nad blokovima podataka koji su bili dugački 24 bita. Nešto jača varijanta koja je objavljena 1973. godine radila je nad blokovima dužine 128 bita i sa ključevima dužine 128 bita. Izmjenjena verzija ovog algoritma postala je širom svijeta poznata kao DES (*Data Encryption Standard*). Osnova oba navedena algoritma, ali i brojnih kasnijih algoritama bila je konstrukcija koja se naziva Feistelova mreža. Feistelova mreža je simetrična struktura koja se koristi u brojnim blokovskim algoritmima. (DES, Triple DES, BlowFish, TwoFish, RC5 itd.). Feistelova mreža u stvari nije algoritam, već način dizajna koji se koristi u velikom broju algoritama. Feistelova mreža je veoma slična SP mreži jer je karakterišu dva primitivna kriptografska elementa: S-kutija (supstitucija) i P-kutija (permutacija). Ova dva elementa svoje korjene vuku iz radova velikog američkog matematičara Kloda Šenona (Claude Shannon). Šenon je 1949. godine predstavio ideju o supstituciono-permutacionim mrežama (*S-P networks*), koje se baziraju na ova dva primitivna kriptografska elementa. Najpoznatiji algoritam koji je zasnovan na SP mreži je algoritam AES (*Advanced Encryption Standard*). Velika razlika između Feistelove i SP mreže leži i u činjenici da algoritmi koji se baziraju na Feistelu šifruju samo polovinu bloka u jednoj rundi, tako da im je potrebno najmanje dvije runde da bi šifrovali cijeli podatak.

Većina blokovskih algoritrama ima iterativan dizajn – svaki blok podataka će biti obrađen više puta ponavljanjem odgovarajućih transformacija tokom određenog broja rundi. Zbog toga je najčešće potrebno inicijalni ključ proširiti pomoću funkcije ekspanzije ključeva (*key schedule*) tako da u svakoj rundi algoritam ima na raspolaganju novu porciju ključa. Zbog toga bi mogli reći da se svaki ovakav algoritam sastoji od tri dijela ili algoritma – algoritma šifrovanja, dešifrovanja i ekspanzije ključeva.

Međutim, osobina blokovskih algoritama je da će dva identična bloka koja se nalaze u okviru otvorenog teksta kojeg šifrujemo istim ključem kao rezultat dati identične blokove šifrata. Ovaj nedostatak blokovskih algoritama doveo je do pojave brojnih različitih kriptografskih načina rada (*modes of operation*) - Phillip Rogaway opisuje čak 17 ovakvih načina rada.

Sam termin način rada (*mode of operation*) se najčešće koristi bez termina blokovski algoritam, jer se podrazumjeva da se svaki od njih bazira na blokovskom algoritmu. Pored nekog podređenog blokovskog algoritma koji u datom momentu koristi, neki od načina rada (*modes of operation*) može koristiti jednostavne kriptografske alate poput manipulacije bitima, XOR operacije, dodavanja dopune (*padding*) porukama ili aritmetike nad konačnim poljima.

S druge strane, kako se navodi u (Menezes et al. 1996), protočni algoritmi primjenjuju jednostavnu transformaciju šifrovanja koja zavisi od toka ključeva koji se koristi. Tok ključeva može da bude generisan na slučajan način ili pomoću algoritma koji generiše tok ključeva na osnovu inicijalnog toka (koji se naziva sjeme, *seed*) ili na osnovu takvog sjemena i prethodno dobijenog šifrata. Takav algoritam se naziva generator toka ključeva.

CFB, OFB i CTR imaju dvije zanimljive zajedničke karakteristike. Oni od nekog blokovskog algoritma prave protočni algoritam a za njihovo funkcionisanje nije potrebna funkcija dešifrovanja.

4.10.1 CTR (Counter) ili SIC (Segmented Integer Counter) način rada

Counter ili CTR način rada je jedan od najjednostavnijih i najelegantnijih načina rada među modovima koji obezbjeđuju isključivo povjerljivost. Ovaj način rada predložili su Diiffie i Hellman (Diffie, 1979), kao jedan od osnovnih (bazičnih) načina rada (Rogaway, 2011).

CTR način rada uzima na ulazu jedan inicijalizacioni vektor IV (brojač), a zatim se tokom svake iteracije vrijednost inicijalizacionog vektora (brojača) inkrementira a zatim i šifruje. Šifrat se proizvodi tako što se rezultati šifrovanja prethodno opisanih IV-a odnosno brojača pomoću XOR operacije spajaju sa blokovima otvorenog teksta (Chakraborty and Rodriguez-Henriquez, 2009). Sekvenca brojača mora da bude takva da je svaki blok u sekvenci različit od svakog drugog bloka u sekvenci. Ovaj uslov nije ograničen na samo jednu poruku, jer tokom svih poruka koje se šifruju jednim istim ključem, svi brojači moraju da budu različiti. Ako imamo sekvencu brojača T1, T2, ..., Tn, CTR mod se definiše kako slijedi (Dworkin, 2001):

CTR šifrovanje
$O_J = CIPH_K(T_J)$ Za j = 1, 2,...,n;
$C_J = P_J$ XOR O_J Za j = 1, 2,...,n-1;
Zadnji blok:
$C^*_N = P^*_N$ XOR $MSB_U(O_N)$

CTR dešifrovanje
$O_J = CIPH_K(T_J)$ Za j = 1, 2,...,n;
$P_J = C_J$ XOR O_J Za j = 1, 2,...,n-1;
Zadnji blok:
$P^*_N = C^*_N$ XOR $MSB_U(O_N)$

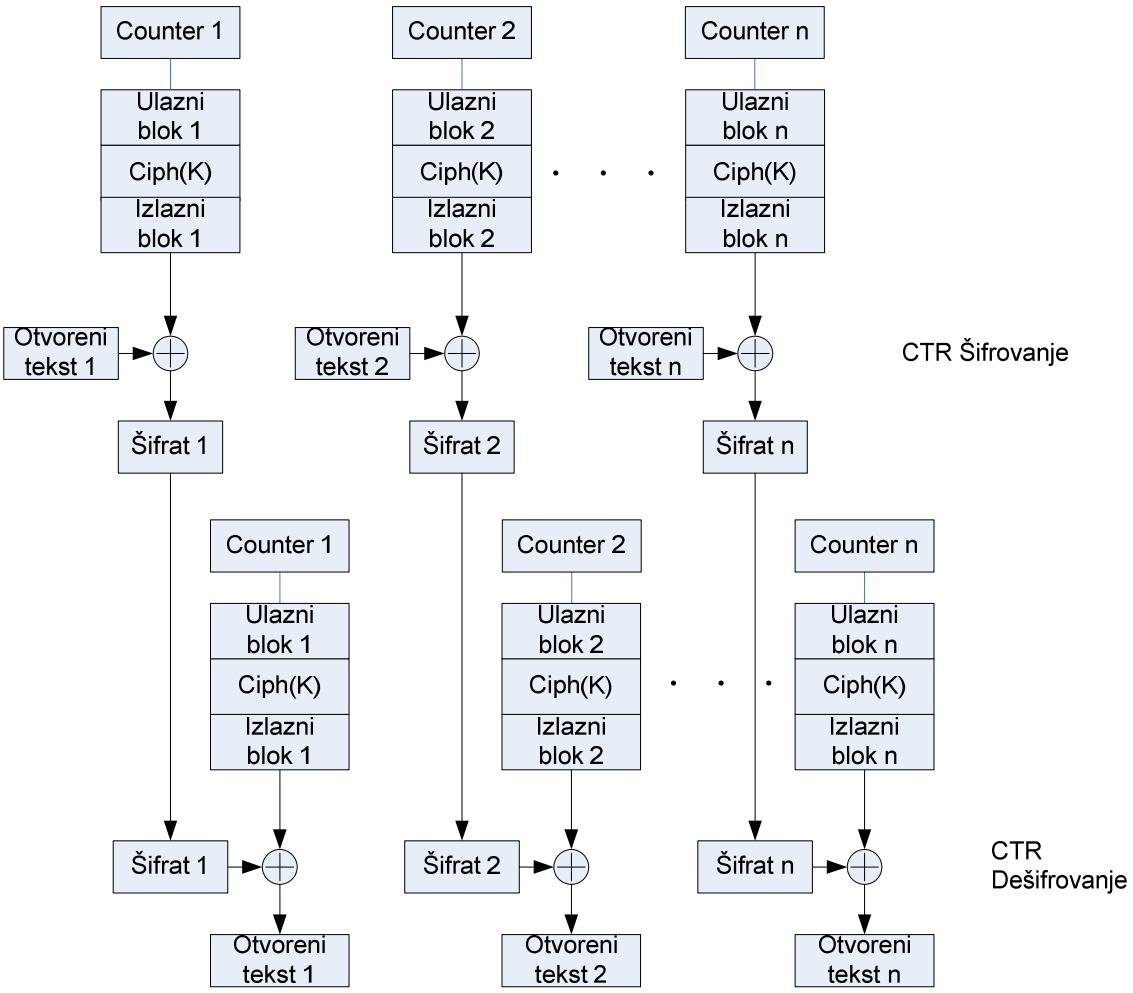
Listing 20: Pseudo kod za CTR šifrovanje i dešifrovanje

Prema (Dworkin, 2001), pri CTR šifrovanju, funkcija enkripcije se poziva za svaki blok brojača, a zatim se rezultirajući izlazni blokovi pomoću XOR operacije kombinuju sa otvorenim tekstom da bi se dobili blokovi šifrata. Pri šifrovanju zadnjeg bloka otvorenog teksta može se desiti da se pojavi samo dio ovog bloka dužine *u* bita. U tom slučaju se samo najznačajnijih *u* bita bloka kombinuje pomoću XOR operacije, a preostalih *(b-u)* bita se odbacuje.

U slučaju CTR dešifrovanja, funkcija šifrovanja (CIPH) podređenog algoritma se ponovo poziva za svaki blok brojača, a rezultirajući izlazni blokovi se pomoću operacije XOR spajaju sa odgovarajućim blokovima šifrata da bi se dobili odgovarajući blokovi otvorenog teksta. Za zadnji blok, koji može biti djelimični blok dužine *u* bita, najznačajnijih *u* bita zadnjeg izlaznog bloka se koriste u XOR operaciji, a preostalih (*b*-*u*) bita se odbacuju.

I u slučaju CTR šifrovanja i u slučaju CTR dešifrovanja, funkcija šifrovanja podređenog algoritma se može izvršavati paralelno. Blokovi otvorenog teksta koji odgovaraju određenom bloku šifrata mogu biti restaurirani nezavisno od drugih blokova otvorenog teksta ako može da se utvrdi vrijednost odgovarajućeg bloka brojača. Osim toga, funkcija šifrovanja podređenog algoritma se može primjenjivati na brojače prije nego što postane dostupan otvoreni tekst ili šifrat.

Treba primjetiti da je za CTR potreban samo algoritam šifrovanja, tako da nije potrebno implementirati transformacije dešifrovanja (InvCIPH) podređenog algoritma.



Slika 21: CTR način rada

4.10.2 OFB (Output Feedback) način rada

Output Feedback (OFB) način rada je režim koji obezbjeđuje povjerljivost poruke i koji se uzastopno izvršava nad inicijalizacionim vektorom da bi generisao sekvencu izlaznih blokova koji se nakon toga pomoću operacije XOR spajaju sa otvorenim tekstom da bi proizveli šifrat i obrnuto, spajaju se sa šifratom da bi proizveli otvoreni tekst. Output Feedback (OFB) način rada zahtjeva da inicijalizacioni vektor bude samo jednokratni (*nonce*) za svako izvršenje ovog načina rada za dati ključ (Dworkin, 2001).

U OFB načinu rada inicijalizacioni vektor (IV) se uzastopno šifruje da bi se dobio neprekidni tok slučajnih bajta. U ovom načinu rada niti jedan dio otvorenog teksta se nikada ne daje kao ulaz u podređeni algoritam. Zbog toga je ovaj režim rada vrlo sličan protočnim algoritmima, gdje protočni algoritam proizvodi tok nasumičnih bajta, a takav tok se pomoću operacije XOR stapa sa otvorenim tekstom da bi se formirao šifrat (Chakraborty and Rodriguez-Henriquez, 2009).

Output Feedback način rada se u NIST-ovom dokumentu Special Publication 800-38A definiše kako slijedi (Dworkin, 2001):

OFB šifrovanje
$I_1 = IV$

$I_J=O_{J-1}$ Za j = 2,...,n;
$O_J = CIPH_K(I_J)$ Za j = 1, 2,...,n;
$C_J = P_J$ XOR O_J Za j = 1, 2,...,n-1;
$C^*_N = P^*_N$ XOR $MSB_U(O_N)$

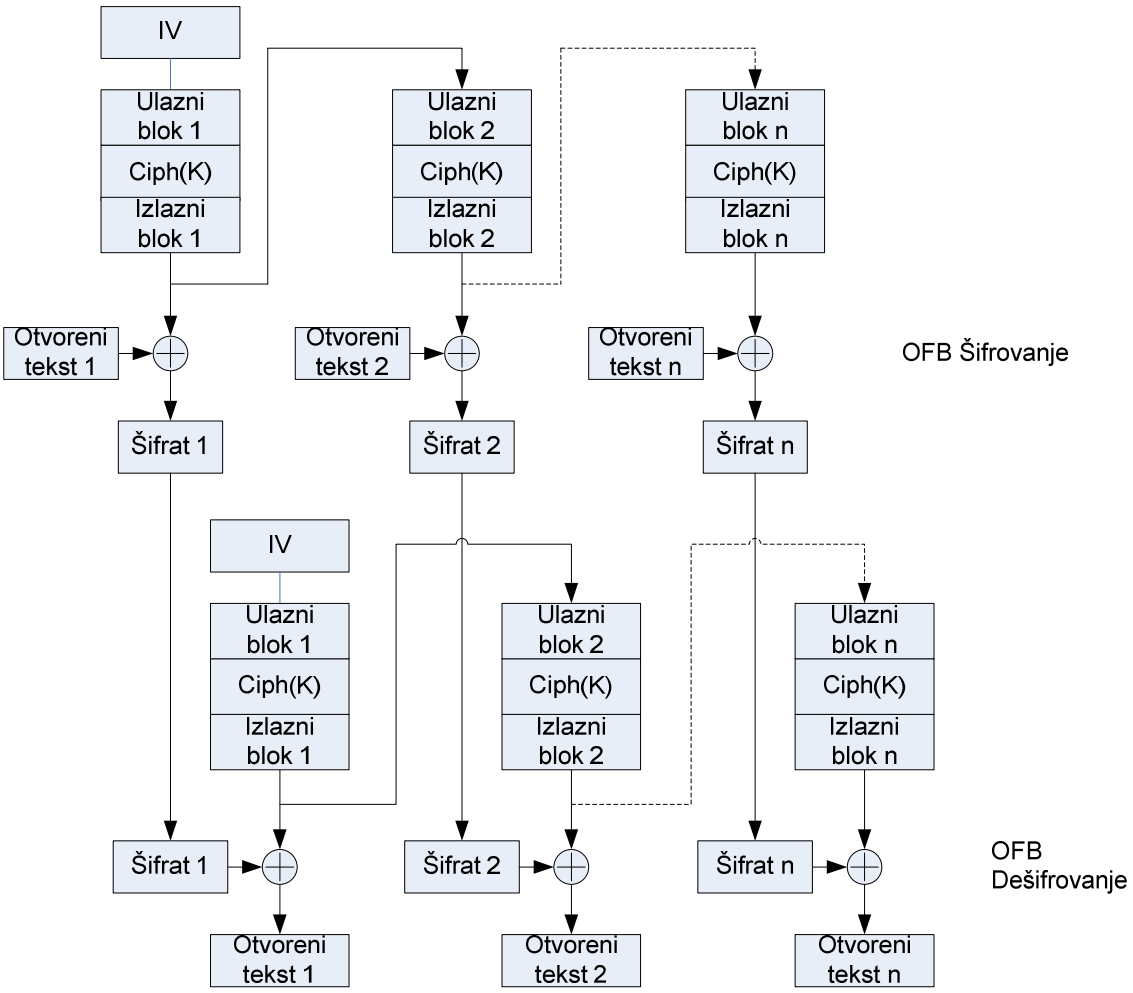
OFB dešifrovanje
$I_1 = IV$
$I_J=O_{J-1}$ Za j = 2,...,n;
$O_J = CIPH_K(I_J)$ Za j = 1, 2,...,n;
$P_J = C_J$ XOR O_J Za j = 1, 2,...,n-1;
$P^*_N = C^*_N$ XOR $MSB_U(O_N)$

Listing 21: Pseudo kod za OFB šifrovanje i dešifrovanje

Kod OFB načina rada, inicijalizacioni vektor se transformiše pomoću funkcije šifrovanja $CIPH_K$ podređenog algoritma da bi se proizveo prvi izlazni blok (O) iz podređenog algoritma. Takav prvi izlazni blok se zatim pomoću operacije XOR stapa sa prvim blokom otvorenog teksta da bi proizveli prvi blok šifrata. Slijedeća operacija šifrovanja podređenog algoritma se tada izvršava nad prethodnim izlaznim blokom (O) i na taj način formira drugi izlazni blok. Ovaj drugi izlazni blok se ponovo pomoću operacije XOR kombinuje sa drugim blokom otvorenog teksta da bi se proizveo drugi blok šifrata, a slijedeća operacija šifrovanja se izvršava nad drugim izlaznim blokom (O). Na ovaj način, sukcesivni izlazni blokovi se proizvode tako da se funkcija šifrovanja podređenog algoritma primjenjuje na prethodnim izlaznim blokovima, dok se svaki takav blok pomoću XOR operacije kombinuje sa odgovarajućim blokovima otvorenog teksta da bi proizveli blokove šifrata. Za zadnji blok, koji može biti parcijalni blok od *u* bita, najznačajnijih *u* bita od zadnjeg izlaznog bloka se koriste u XOR operaciji dok se preostalih *b*-*u* bita odbacuje (Dworkin, 2001).

Tokom OFB dešifrovanja, *IV* se ponovo transformiše pomoću funkcije šifrovanja $CIPH_K$ podređenog algoritma da bi se proizveo prvi izlazni blok (O) iz podređenog algoritma. Takav prvi izlazni blok se zatim kombinuje pomoću XOR operacije, ali sada sa prvim blokom šifrata da bi proizveo prvi blok otvorenog teksta.

OFB način rada zahtjeva da postoji jedinstven *IV* za svaku poruku koja je ikada šifrovana jednim ključem. Ako bi, uprkos ovom zahtjevu, isti inicijalizacioni vektor bio korišten za šifrovanje više od jedne poruke, tada bi bila kompromitovana povjerljivost ovakvih poruka. Verzija Output Feedback načina rada prema specifikaciji datoj u ISO standardu (ISO/IEC 10116, 2006) koja proizilazi iz dokumenta FIPS 81 (FIPS PUB 81, 1980), ima jedan dodatni patametar - parametar *j*. Samo *j* lijevih bita svakog bloka *Yi* se koristi da bi se maskirao *P* da bi se dobio šifrat *C* (Dworkin, 2001).



Slika 22: OFB način rada

4.10.3 CFB (Cipher Feedback) način rada

Cipher Feedback (CFB) način rada je režim koji obezbjeđuje povjerljivost poruke i kod kojega se proces šifrovanja zasniva na povratnoj sprezi uzastopnih segmenata šifrata. CFB mod zahtjeva da postoji inicijalizacioni vektor (IV) kao ulazni parametar. On ne mora biti sakriven, ali mora biti nepredvidljiv (Dworkin, 2001).

Ono što je interesantno za Cipher Feedback način rada je da su procesi šifrovanja i dešifrovanja vrlo slični te da se tokom procesa dešifrovanja ne koristi funkcija inverzna šifrovanju (InvCipher ili EK^{-1}) podređenog algoritma.

Treba primjetiti da i Cipher Feedback način rada prema specifikaciji datoj u ISO standardu (ISO/IEC 10116, 2006) koja proizilazi iz dokumenta 81 (FIPS PUB 81, 1980), može po potrebi da koristi i samo dio navedenog prethodnog bloka. Cipher Feedback način rada tada zahtjeva još jedan cjelobrojni parametar, koji se označava sa *s*, takav da je *1* <= *s* <= *b*, gdje je *b* veličina bloka koju koristi podređeni algoritam izražena u bitima. Vrijednost *s* je ponekad pripojena u naziv načina rada tako da imamo 1-bitni CFB mod, 8-bitni CFB mod, 64-bitni CFB ili 128-bitni CFB način rada.

CFB način rada se definiše kako slijedi:

CFB šifrovanje

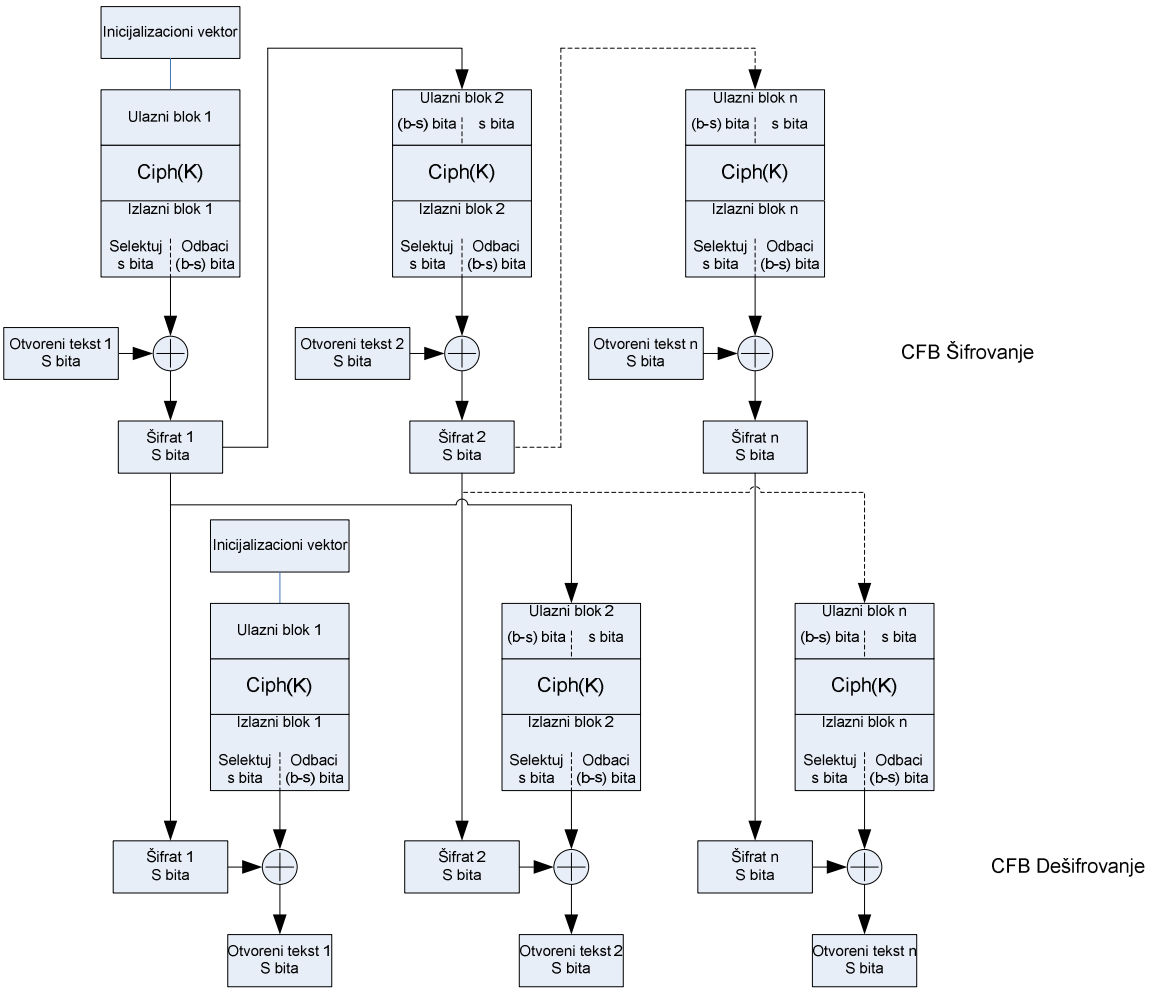
$I_1 = IV$
$I_J = LSB_{B-S}(I_{J-1}) \mid C^{\#}_{J-1}$ Za j = 2,...,n;
$O_J = CIPH_K(I_J)$ Za j = 1, 2,...,n;
$C^{\#}_J = P^{\#}_J$ XOR $MSB_S(O_J)$ Za j = 1, 2,...,n;

CFB dešifrovanje
$I_1 = IV$
$I_J = LSB_{B-S}(I_{J-1}) \mid C^{\#}_{J-1}$ Za j = 2,...,n;
$O_J = CIPH_K(I_J)$ Za j = 1, 2,...,n;
$P^{\#}_J = C^{\#}_J$ XOR $MSB_S(O_J)$ Za j = 1, 2,...,n;

Listing 22: Pseudo kod za CFB šifrovanje i dešifrovanje

U CFB šifrovanju, prvi ulazni blok je *IV*, tako da se operacija šifrovanja primjenjuje na *IV* da bi proizvela prvi izlazni blok. Dalje se prvi segment šifrata dobija tako da se prvi segment otvorenog teksta pomoću XOR operacije stapa sa *s* najznačajnijih bita prvog izlaznog bloka. Ostalih *(b-s)* bita izlaznog bloka se odbacuju. Preostalih *(b-s)* bita inicijalizacionog vektora *IV* se zatim sastavljaju sa s bita prvog segmenta šifrata da bi na taj način formirali drugi izlazni blok (Dworkin, 2001). U trećoj ediciji ISO/IEC 10116 standarda (ISO/IEC 10116, 2006) ovakav proces šifrovanja se opisuje pomoću *feedback buffer-a* FB, čiji biti se šiftaju ulijevo i dopunjavaju bitima šifrata. Prema (Dworkin, 2001), alternativni opis formiranja drugog izlaznog bloka je da se biti prvog ulaznog bloka cirkularno pomjeraju (šiftaju) s pozicija u lijevo, a da nakon toga prethodni segment šifrata dopunjava bite sa desne strane.

Kako se navodi u (Dworkin, 2001), proces se dalje ponavlja nad uzastopnim ulaznim blokovima sve dok se ne kreira segment šifrata na osnovu svih ulaznih blokova. Generalno, svaki uzastopni ulazni blok se šifruje da bi proizveo izlazni blok. Najznačajnijih *s* bita svakog ulaznog bloka se XOR operacijom stapaju sa odgovarajućim segmentima otvorenog teksta da bi formirali segment šifrata. Svaki segment šifrata (osim zadnjeg) je povratno spregnut (*fed back*) sa prethodnim ulaznim blokom (kako je već opisano) da bi formirao novi ulazni blok. Povratna sprega može biti opisana individualnim bitima u nizu kako slijedi: ako biti $i_1, i_2,..., i_b$ predstavljaju j-ti ulazni blok i ako $c_1, c_2,...,c_s$ predstavljaju j-ti segemnt šifrata, onda je (j+1) ulazni blok $i_{s+1}, i_{s+2},..., i_b, c_1, c_2,...,c_s$.



Slika 23: CFB načina rada

4.11 XEX, XE konstrukcije i XTS-AES način rada

Liskov, Rivest i Wagner (Liskov et al. 2002) definisali su pojam izmjenjivim parametrom modifikovanog (tweakable) blokovskog kriptografskog algoritma. Rogaway (Rogaway, 2004) je formalno definisao *tweakable* blokovski kriptografski algoritm kao preslikavanje:

$$\widetilde{E} : Kx\mathcal{T}x\{0,1\}^n \to \{0,1\}^n \qquad (64)$$

gdje je

$K \in \{0,1\}^k$ ključ;

$\mathcal{T} \in \{0,1\}^t$ skup izmjenjivih parametara (*tweak*);

$P \in \{0,1\}^n$ skup poruka (*plaintext*);

$C \in \{0,1\}^n$ skup proizvedenih šifrata (*ciphertext*);

i gdje je svako

$$\widetilde{E}_T^K(\cdot) = \widetilde{E}(K,T,\cdot) \tag{65}$$

permutacija nad skupom ključeva K i skupom izmjenjivih parametara T.

XEX (*XOR Encrypt XOR*) mod je predložio Rogaway (Rogaway, 2004) a instanciran je od organizacije IEEE u okviru standarda (IEEE, 2008) i NIST-ove publikacije (Dworkin, 2010) koje opisuju XTS-AES način rada. XTS-AES mod koristi dva ključa za šifrovanje. Prvi ključ se koristi da šifruje adresu sektora na disku i da generiše izmjenjivi parametar (*tweak*), a drugi se koristi da se pomoću njega šifruju podaci.

XEX konstrukcija (Rogaway, 2004) se definiše pomoću

$$\widetilde{E}_K^{N_{I_1..I_k}}(M) = E(M \oplus \Delta) \oplus \Delta \tag{66}$$

gdje je

$$\Delta = \alpha^{i_1}\alpha^{i_2},...,\alpha^{i_K}N$$

i

$$N = E_K(N)$$

Kako se navodi u (IEEE, 2008) XEX konstrukcija najprije izračunava vrijednost izmjenjivog parametra T koja se koristi za maskiranje

$$T = Enc(K2,s) \otimes \alpha^t \tag{67}$$

gdje je

- množenje definiano u GF(2^n), pri čemu je n veličina bloka podređenog algoritma,
- dok je α primitivni element u GF(2^n).

Za dati otvoreni tekst P i šifrat C, funkcije šifrovanja i dešifrovanja su opisane slijedećim jednačinama:

$$C = Enc(K1, P \oplus T) \oplus T \tag{68}$$

$$P = Dec(K1, C \oplus T) \oplus T$$

U slučaju XTS-AES moda, logička adresa sektora na disku se koristi kao izmjenjivi ili javni parametar (tweak) koja se koristi za modifikaciju procesa enkripcije. Specifikacija za XTS-AES način rada predstavlja posebnu vrstu mutiplikacije koja se izvodi pomoću slijedeće procedure:

Ulaz: j je eksponent od a

niz bajta $a_0[k], k = 0,1,...,15$

Izlaz: niz bajta $a_j[k], k = 0,1,...,15$

Izlazni niz je definisan rekurzivno pomoću slijedećih formula, pri čemu se i iterira od 0 do j:

$$a_{i+1}[0] \longleftarrow\!\!\!- (2(a_i[0] \bmod 128)) \oplus (135 \lfloor a_{15}[0]/128 \rfloor) \quad (69)$$

$$a_{i+1}[k] \longleftarrow\!\!\!- (2(a_i[k] \bmod 128)) \oplus \lfloor a_{15}[k-1]/128 \rfloor, K = 1,2,...15$$

4.12 Testne platforme

Kao testne platfrome korišćeni su računari Dell Inspiron N5110 sa Core i7-2630QM procesorom koji ima 4 fizička i 8 logičkih jezgara i sa 6 GB operativne memorije, te Dell Precision T5400 računar sa dva Xeon E5410 procesora sa po četiri fizička jezgra svaki i sa 4 GB operativne memorije. Na prvoj mašini bio je instaliran 64-bitni Windows 7 Ultimate operativni sistem na kojem je bio instaliran 64-bitni Java SE7u51 Developement Kit, a na drugoj 32-bitni Windows 7 Ultimate operativni sistem sa 32-bitnim Java SE7u51 Developement Kitom. Prva testna platforma će u daljem tekstu biti navođena nazivom N5110, a druga testna platforma pomoću naziva T5400. Pored njih, u jednom slučaju je, zbog specifičnih potreba istraživanja koja su bila vezana za brzinu operativne memorije, korišćena je Toshiba Satellite L50-A-1D5 mašina sa Core i7-4700MQ procesorom i 8GB operativne memorije koja se sastojala od DDR3-1600 memorijskih modula. Na ovoj mašini, koja će u nastavku biti referencirana kao L50A bio je instaliran Windows 8.1 operativni sistem, te Java(TM) SE Runtime Environment (build 1.8.0_25-b18) i Java HotSpot(TM) 64-bit server (build 25.25-b02).

5 Model i osnovne karakteristike adaptibilne primjene AES algoritma u savremenim operativnim sistemima

5.1 Model adaptibilnog virtuelnog kriptografskog fajl sistema

Već je navedeno da se u ovom radu kao centralni problem razmatra mogućnost adaptibilne primjene algoritma AES u okviru operativnog sistema u zavisnosti od dostupnih resursa. Zbog tog je bilo potrebno razviti model pomoću kojega se na najbolji način mogu iskoristiti hardverski i softverski resursi računara radi prilagođenja i izbora najefikasnije mogućnosti za enkripciju i dekripciju algoritmom AES. Neke od osnovnih karakteristika takvog modela morale su da budu konfigurabilnost, proširivost i modularnost, mogućnost donošenja odluka te *run-time* prilagodljivost postojećim hardverskim i softverskim resursima radi efikasnijeg korišćenja algoritma AES.

Najvažniji uslov za uspješan razvoj navedenog modela je bilo projektovanje modula za koordinaciju i u okviru njega sistema za trening i kategoriztaciju, te sistema za selekciju sa podsistemima za klasifikaciju i nominaciju. Kreiran je modul za koordinaciju koji je sposoban:

- za mjerenje i evidentiranje performansi svakog pojedinog kriptografskog resursa radi stvaranja trening podataka,
- za provođenje treninga putem odabrane metode mašinskog učenja radi stvaranja prediktivnog (prognostičkog) modela koji predviđa ponašanje datog kriptografskog resursa (modula),
- te za izvođenje analize u svrhu donošenja odluke o angažovanju najefikasnijeg kriptografskog resursa.

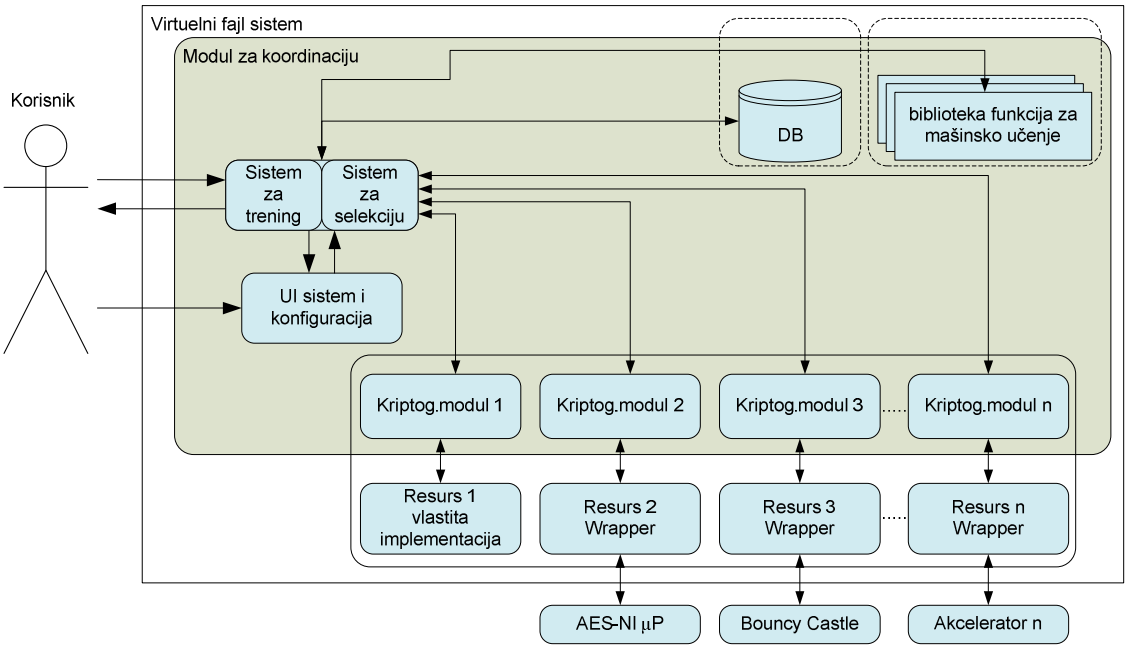
Uslijed heterogenosti kriptografskih resursa koji se mogu naći u nekom sistemu, načini komunikacije sa navedenim resursima mogu se međusobno bitno razlikovati. Zbog toga je provedeno i istraživanje mogućnosti komunikacije između adaptibinog virtuelnog kriptografskog fajl sistema i pojedinih kriptografskih resursa radi utvrđivanja minimalnog skupa poruka koje imaju jednoobrazan oblik a koje se u toku komunikacije moraju razmjenjivati. Na ovaj način je kreiran model komunikacije koji obuhvata minimalan i maksimalan skup poruka i parametara koji su potrebni za pokretanje bilo kog kriptografskog resursa, pri čemu se ostavlja mogućnost za specifična podešavanja svakog pojedinog resursa na strani samog resursa, kao i za proširenja ovakvog modela u narednim verzijama.

U toku istraživanja implementiran je određeni broj kriptografskih resursa koji su bili uklljučeni u adaptibilni virtuelni kriptografski fajl sistem. Da bi bile pokrivene različite vrste softverskih i hardverskih kriptografskih resursa (originalna implementacija, omotač, hardverski akcelerator, jedna ili više niti), u programskim jezicima Java i C/C++ je kreirano više rješenja koja su u stanju da koriste jedno ili više jezgara mikroprocesora, za AES specifičan skup instrukcija AES-NI, CUDA grafički kartu te integraciju pomoću omotača sa rješenjima nezavisnih proizvođača softvera. Mjerenjem njihovih performansi su prikupljani empirijski podaci koji su služili kao trening podaci za modul za koordinaciju. Na osnovu ovih podataka, modul za koordinaciju je pomoću metoda mašinskog učenja generisao prediktivni (prognostički) model za svaki kriptografski resurs, pomoću kojeg se može vršiti predviđanje njegovog ponašanja. Navedeni prediktivni modeli se snimaju u

tekstualne datoteke ili u nekoj relaciji (tabeli) u bazi podataka radi ubrzanja njihovog korišćenja. Svako naredno mjerenje performansi nekog kriptografskog modula korišćeno je za evaluaciju i korekciju ranije generisanih prediktivnih modela na nivou pojedinih kriptografskih resursa i na nivou cijelog sistema.

Nezavisnost upotrebljenog modela od operativnog sistema predstavlja važan zahtjev na koji se morala obraćati pažnja pri njegovoj izradi, da bi model mogao da bude primjenjen na različitim softverskim i hardverskim platformama.

Modul za koordinaciju na osnovu korisničkih podešavanja, kreiranih prediktivnh modela za predviđanje ponašanja pojedinih kriptografskih resursa, te na osnovu izvršavanja podsistema za selekciju i u okviru njega podsistema za klasifikaciju i nominaciju, bira najefikasniji način šifrovanja ili dešifrovanja algoritmom AES. Svi prethodno navedeni koncepti integrisani su u okviru adaptibilog virtuelnog kriptografskog fajl sistema, kao reprezenta adaptibilog operativnog sistema.



Slika 24. Konceptualni model kriptografski adaptibilnog operativnog sistema

Postoji nekoliko načina na koje je moguće integrisati module koji pristupaju različitim hardverskim i softverskim resursima u jednu funkcionalnu cjelinu. Jedan od načina bio bi da se svi moduli (i njihove implementacije) integrišu u okviru fajl sistema, odnosno operativnog sistema. Iako autori operativnog sistema (i fajl sistema) u ovom slučaju imaju potpunu kontrolu nad sistemom, ovakav sistem ima i određene nedostatke. Nedostatak ovakvog rješenja ogleda se u potrebi neprekidnog kodiranja i objavljivanja zakrpa pri pojavi svakog novog hardverskog ili softverskog kriptografskog resursa.

Zbog toga je u istraživanju primjenjen nešto drugačiji pristup, u okviru kojega su određeni dijelovi sistema koji logički pripadaju adaptibinom virtuelnom kriptografskom fajl sistemu i njegovom modulu za koordinaciju, kao što su sistem za upravljanje bazama podataka, kriptografski (AES) moduli, biblioteka funkcija za mašinsko učenje, fizički izmješteni van njega, u obliku kriptografskih i drugih modula. Ovaj cilj je postignut kreiranjem ranije pomenutog jednoobraznog modela za razmjenu poruka. U ovom slučaju je različitim hardverskim i softverskim resursima za enkripciju i

dekripciju moguće iz adaptibinog virtuelnog kriptografskog fajl sistema pristupati na uniforman način, tako da se nekoj eksternoj aplikaciji pruži svega nekoliko parametara (datoteka koju treba šifrovati ili dešifrovati i parametri enkripcije). Eksterni programi koje bi izrađivali nezavisni proizvođači hardvera i softvera morali bi da u sebi integrišu najmanje jednu od navedenih mogućnosti:

- funkcije softverske enkripcije i dekripcije algoritmom AES;
- mogućnost pozivanja drugih hardverskih resursa (npr. akceleratora, AES-NI skupa instrukcija) direktno ili u formi omotača (wrapper);
- mogućnost pozivanja drugih softverskih resursa direktno ili u formi omotača, poput posrednika koji poziva enkripciju ili dekripciju pomoću Oracle Sun JCA/JCE ili Bouncy Castle kriptografskog paketa.

Zbog ovakvog pristupa sistem se lako može dograđivati novim modulima odmah po uključenju bilo kog novog kriptografskog resursa, i to ne samo od strane autora operativnog sistema, već i od nezavisnih proizvođača softvera. Na ovaj način se vlastiti moduli stavljaju u isti položaj kao i moduli nezavisnih proizvođača, jer se u oba slučaja jednostavno mogu uključiti u sistem. Nezavisni proizvođači na ovaj način dobijaju priliku da razvijaju vlastite implementacije ali i zadatak da poštuju model za razmjenu poruka koji je na uniforman način usklađen sa ostalim komponentama sistema. Na ovaj način se u sistem po potrebi mogu uključivati ponekad i različite implementacije koje se u radu oslanjaju na jedan isti resurs. Ako korisnik nije eksplicitno zahtjevao da se enkripcija/dekripcija vrši pomoću određenog modula, zadatak modula za koordinaciju je da predvidi koji kriptografski modul će dati najbolje vrijeme izvršenja i da ga upotrebi za obradu podataka. Predviđanje vremena potrebnog za izvršenje se vrši na takav način da se na osnovu trening podataka pomoću metoda mašinskog učenja izgrade prediktivni (prognostički) modeli koji predstavljaju pojedine kriptografske resurse (module). Unaprijed pripremljeni i snimljeni modeli se zajedno sa nezavisnim parametrima šalju biblioteci funkcija za mašinsko učenje gdje se koriste za brzo izračunavanje zavisnog parametra - vremena potrebnog za obradu podataka.

5.2 Definisanje ograničenja i pretpostavki istraživanja

U ovom istraživanju će biti pretpostavljeno da postoji siguran način čuvanja ključeva u okviru operativnog sistema, bez ulaženja u njegovo objašnjavanje. Takođe će u istraživanju biti pretpostavljeno da postoji način za zaštićeni prenos poruka između adaptibinog virtuelnog kriptografskog fajl sistema i kriptografskih resursa. Zbog uniformnog oblika i dužine svih poruka, biće pretpostavljeno da njihova zaštita pomoću npr. asimetrične kriptografije uvijek jednako traje, tako da se njen uticaj na performanse manifestuje na isti način na sve kriptografske module. Takođe će biti pretpostavljeno je moguće da se u sistemu nađu i umrežene mašine, te da je takva mreža povjerljiva i u stanju da sačuva povjerljivost i integritet podataka. Takve umrežene mašine predstavljaju samo dio sistema i da njihovi resursi mogu da se kao kriptografski moduli uključe u adaptibilni virtuelni kriptografski fajl sistem.

5.3 Kriptografski moduli

Iako kriptografski moduli logički pripadaju modulu za koordinaciju, prema ideji prikazanoj u istraživanju treba da budu fizički izmješteni van njega. Prema predloženom rješenju, oni treba da budu implementirani kao samostalni programi koji

sa operativnim sistemom (adaptibilnim virtuelnim kriptografskim fajl sistemom) komuniciraju putem neke od metoda interprocesne komunikacije. Oni mogu da služe i kao omotač (*wrapper*) oko različitih provajdera sa implementacijama algoritma AES. Na ovaj način je postignuta modularnost sistema, čime je izbjegnuta potreba da se mjenja cijeli sistem po pojavi nekog novog resursa. Istovremeno su u ravnopravan položaj dovedeni resursi nezavisnih proizvođača sa vlastitim kriptografskim resursima, pa je arhitektura operativnog sistema učinjena otvorenom i pogodnom za nadmetanje među različitim proizvođačima.

U okviru istraživanja su radi obezbjeđenja empirijskih podataka implementiani i testirani različiti kriptografski moduli, koji su zasnovani na različitim idejama - od paralelizacije izvršenja algoritma AES do upotrebe AES-NI skupa instrukcija. Pored ovoga, kreirani su i moduli - omotači (*wrapper*) za povezivanje sa kriptografskim bibliotekama Oracle Sun JCA/JCE i Bouncy Castle.

5.3.1 Zajedničke karakteristike implementiranih kriptografskih modula

U toku istraživanja adaptibilne primjene AES algoritma u okvirima modernih operativnih sistema testiran je određen broj kriptografskih modula. Neka od prikazanih rješenja su korišćena za prikupljanje empirijskih trening podataka, dok neka rješenja predstavljaju eksperimente kojima se ispituju mogućnosti proširenja algoritma AES i mogućnosti paralelizacije njegovog izvršenja.

Važna zajednička karakteristika svih kriptografskih modula koja je potrebna da bi se oni mogli koristiti kao kriptografski resursi uključeni u adaptibilni virtuelni kriptografski fajl sistem je postojanje standardizivanog modela za uniformnu komunikaciju pomoću razmjene poruka koje su formatirane na uniforman način i koje se razmjenjuju odgovarajućim redoslijedom.

Za povezivanje modula i sistema moguće je koristiti više od jednog načina interprocesne komunikacije, pri čemu je ostavljena mogućnost da adaptibilni virtuelni kriptografski fajl sistem odredi koji je način najbrži za upotrebu. U primijenjenom dijelu rada su radi potvrde koncepta implementirana s dva načina. Kao prvi implementirana je kombinacija soketa (*socket*) i XML standarda. Drugi način realizovan je pomoću neimenovanih cijevi, uz ponovnu upotrebu XML standarda.

Kao tehnologija za formatiranje podataka odabran je XML (*Extensible Markup Language*) standard koji navodi skup pravila za slaganje dokumenata na način koji je čitljiv i ljudima i mašinama. Izbor XML standarda i način kreiranja dokumenata je takav da predviđeni format dokumenata nije cementiran, već da se može prilagođavati iz verzije u verziju. Ovakav pristup omogućava da se iz verzije u verziju unapređuju mogućnosti komunikacije, a da pri tome fajl sistem zadrži kompatibilnost (*backward compatibility*) sa prethodnim, starijim verzijama kriptografskih modula.

Svaki kriptografski modul, pored standardizovanog interfejsa za uniformnu komunikaciju mora da bude u stanju da dekodira primljenu poruku i da u skladu sa sadržajem poruke šifruje određenu datoteku pomoću algoritma koji je naveden u sastavu zahtjeva, pomoću navedenog ključa i po potrebi inicijalizacionog vektora, te da datoteku koja nastane kao rezultat šifrovanja pohrani na lokaciju specificiranu u zahtjevu.

5.4 Koncept modula za koordinaciju

U okviru istraživanja, u adaptibilnom virtuelnom kriptografskom fajl sistemu je implementiran modul za koordinaciju koji je namjenjen da na osnovu korisničkih podešavanja i rezultata ranijih izvršenja odabere najefikasniji resurs za šifrovanje ili dešifrovanje algoritmom AES. Njegovo funkcionianje se može opisati pomoću slijedećih koraka:

- po pozivu korisnika ili prilikom instalacije modul za koordinaciju treba da izvrši inicijalna testiranja i mjerenja brzine izvršavanja pojedinih kriptografskih resursa radi dobijanja trening podataka. Dobijene podatke treba da upiše u posebnu bazu podataka. Modul za koordinaciju zatim vrši kategorizaciju podataka i pomoću biblioteke funkcija za mašinsko učenje kreira jedan ili više prediktivnih (prognotičkih) modela za svaki kriptografski resurs;
- dinamički, prije otpočinjanja enkripcije ili dekripcije treba da, na osnovu prediktivnih modela ranije kreiranih pomoću metoda mašinskog učenja, izvrši predviđanje korišćenjem postupka klasifikacije i nominacije radi selekcije najefikasnijeg kriptografskog resursa (modula);
- na osnovu dobijenog rezultata treba da pokrene modul za koji prethodna analiza pokaže najbolje performanse;
- ako je tako konfigurisan, treba da vrši mjerenja tokom izvršavanja pokrenutog modula i rezultat izvršavanja ponovo upiše u ranije pomenutu bazu podataka.

5.5 Neke posljedice izbora tehnologije i modela nezavisnih od platforme

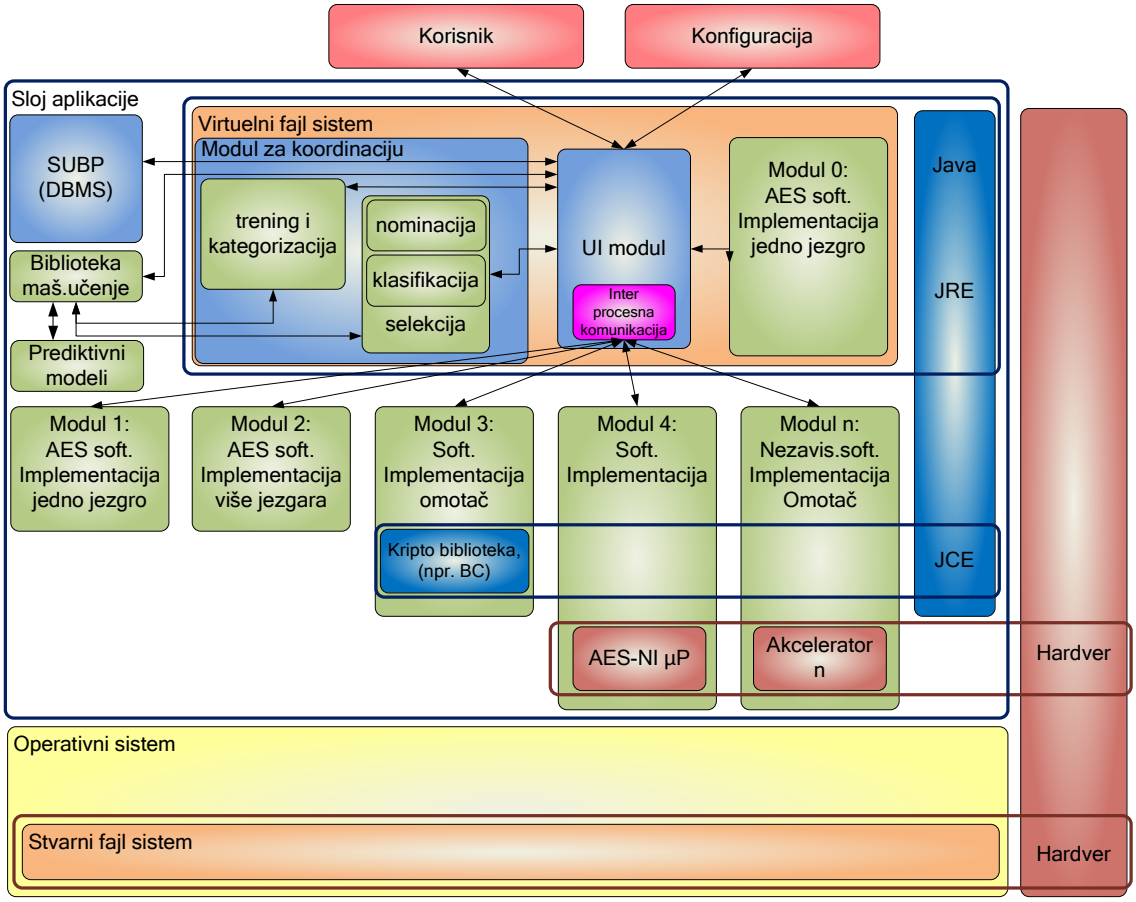
Karakteristika prikazanog modela sistema je da on nije oslonjen niti na jednu hardversku ni softversku platformu. Kako su pojedini kriptografski resursi, iako logički pripadaju sistemu, fizički izmješteni iz njega, oni mogu da budu implementirani pomoću različitih tehnologija i na različitim hadrverskim i softverskim platformama. Pri tome treba imati u vidu da se nakon pojave nekog novog kriptografskog akceleratora ili drugog resursa ne moraju objavljivati zakrpe operativnog sistema, već je dovoljno obezbjediti da navedeni akcelerator bude u stanju da se uključuje u operativni sistem bilo direktno bilo putem softverskog omotača.

Pošto je istraživanje koncentrisano na performanse, jedno od definisanih ograničenja u kojima se ono provodi jeste da su i umrežene mašine samo dio sistema, a da je takva mreža povjerljiva i da je u stanju da sačuva povjerljivost i cjelovitost podataka. Zbog toga kriptografski resursi takvih umreženih mašina mogu da se kao kriptografski moduli uključe u adaptibilni virtuelni kriptografski fajl sistem.

Sa stanovišta performansi ove činjenice dovode do vrlo interesantnih mogućnosti. Nezavisnost od operativnog sistema i hardvera omogućava da se ovakav sistem instalira npr. na nekom Android uređaju, a da se pri šifrovanju i dešifrovanju koristi neka umrežena mašina koja ima višestruko brži procesor ili neki hardverski akcelerator.

6 Arhitektura rješenja za adaptibilnu primjenu AES algoritma na nivou operativnog sistema

Na slijedećoj slici je prikazana detaljna arhitektura rješenja operativnog sistema sa adaptibilnom primjenom AES algoritma:



Slika 25. Detaljna arhitektura rješenja

Sa slike je vidljivo da je kompletno rješenje sa svim pripadajućim modulima smješteno u okviru virtuelnog fajl sistema na sloju aplikacije. Važan dio sistema predstavlja modul za koordinaciju koji se može podijeliti na nekoliko dijelova: sistem za trening i kategorizaciju, sistem za selekciju sa podsistemima za klasifikaciju i nominaciju. Pored njega, važnu ulogu igra i ulazno-izlazni modul sa pripadajućim vezama sa konfiguracionim datotekama, sa korisnikom, vezama sa SUBP, sa bibliotekom programa za mašinsko učenje i vezama sa pojedinim kriptografskim resursima (modulima). Veze sa pojedinim kriptografskim resursima se mogu realizovati pomoću različitih metoda interprocesne komunikacije, pri čemu sistem ima mogućnost da bira najefikasniju metodu. Važan dio rješenja čine takođe i različiti kriptografski moduli (resursi) i njihov model komunikacije sa sistemom, koji omogućava da se, bez obzira na međusobne razlike, oni mogu na modularan način uključiti u sistem.

Treba još jednom primjetiti da iako je ideja primjenjena u sloju aplikacije, nema prepreka da se ideja realizuje na bilo kom drugom sloju, bilo da se radi o sistemu

datoteka u korisničkom prostoru, složenom (*stackable*) sistemu, sistemu orjentisanom ka disku (*disk based*) ili sistemu orjenitisanom ka bloku (*block based*).

6.1 Struktura modula za koordinaciju

Zadatak modula za koordinaciju je da najprje vrši mjerenja radi stvaranja trening podataka i prediktivnih modela, a potom da na osnovu korisničkih podešavanja i rezultata ranije kreiranih prediktvnih modela odabere najefikasniji resurs (modul) za šifrovanje ili dešifrovanje algoritmom AES. On po pozivu korisnika ili prilikom instalacije sistema vrši testiranja i mjerenja brzine šifrovanja i dešifrovanja radi dobijanja trening podataka i njihove kategorizacije. Testiranje se obavlja korišćenjem datoteka koje se automatski generišu od strane sistema i nakon obavljenih testova brišu. Dobijene podatke ovaj modul upisuje u posebnu bazu podataka. Ovi podaci se dalje koriste da se pomoću metoda mašinskog učenja kreiraju prediktivni (prognostički) modeli. Ovi modeli se zapisuju u odgovarajuće datoteke i/ili u relaciju baze podataka, a dalje se koriste pri izboru optimalnog resursa. Ovaj modul će prije započinjanja procesa šifrovanja ili dešifrovanja, na osnovu ranije kreiranih prediktivnih (prognostičkih) modela pomoću metoda mašinskog učenja izvršiti predviđanje da bi pretpostavio brzinu izvršavanja u navedenim uslovima te će odabrati najefikasniji kriptografski resurs. Na osnovu dobijenog rezultata on treba da pokrene kriptografski resurs (modul) za koji prethodna analiza pokaže najbolje performanse. Poslije pokretanja optimalnog resursa (ako je sistem tako konfigurisan), modul za koordinaciju evidentira brzinu izvršavanja nekog kriptografskog resursa (modula) i rezultat izvršavanja ponovo upisuje u ranije pomenutu bazu podataka, čime dodatno povećava skup trening podataka i dodatno poboljšava prognostičke modele dobijene pomoću metoda mašinskog učenja kao i rezultate njihovog predviđanja.

Modul za koordinaciju se logički sastoji od nekoliko podsistema koji su fizički realizovani kao posebni moduli.

6.1.1 Modul za trening i kategorizaciju

Modul za treniranje namjenjen je za prikupljanje trening podataka na osnovu kojih se kreira prognostički model (3D funkcija) koji predviđa brzinu (opisuje ponašanje) pojedinog kriptografskog resursa u odgovarajućim uslovima.

Proces treniranja realizovan je u dva koraka. Prvi korak se sastoji od iterativnog pokretanja pojedinih modula i mjerenja trajanja obrade podataka nakon čega slijedi njihovo evidentiranje u odgovarajućoj bazi podataka. U toku procesa prikupljanja trening podataka mora se voditi računa o mnogim ulaznim parametrima:

- jedinstveni naziv modula,

- broj rundi mjerenja,

- modul je aktivan ili neaktivan,

- lista datoteka koje će se obrađivati,

- lista veličina kriptografskih bafera,

- lista ključeva odnosno njihovih dužina koje utiču na vrstu šifrovanja (npr. AES128 ili AES256),

- broj jezgara, kod modula koji mogu da koriste više jezgara,

- kriptografski način rada (*mode of operation*),

- inicijalizacioni vektor (po potrebi),

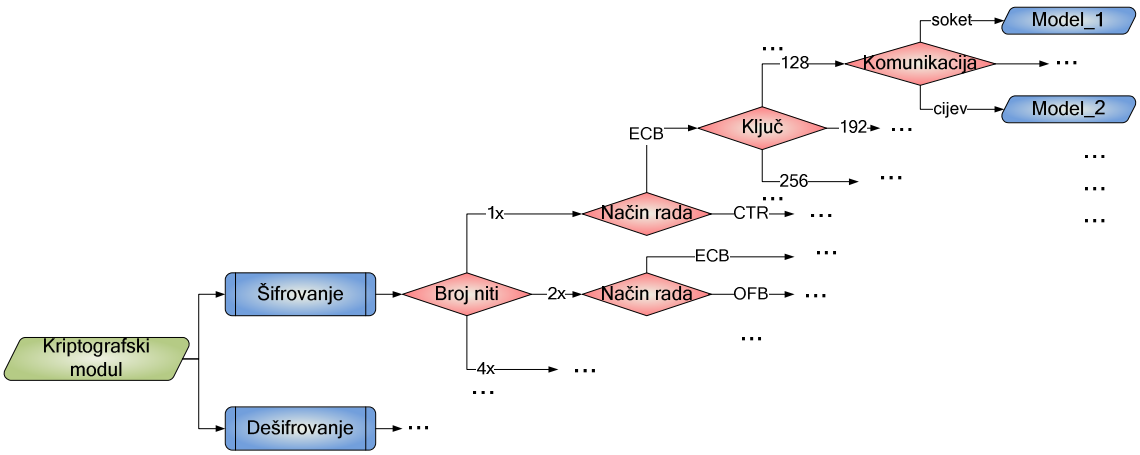
- vrsta komunikacije između virtuelnog fajl sistema i modula (npr. pomoću soketa, neimenovane cijevi),

- šifrovanje ili dešifrovanje.

Veliki broj ulaznih parametara se nalazi u okviru inicijalizacione datoteke ili u okviru baze podataka. Primjer jedne stavke navedene inicijalizacione datoteke dat je u poglavlju 6.2.3. Navedeni ulazni parametri utiču kako na izvršavanje mjerenja tako i na kreiranje modela. Sam proces se u prvom koraku provodi se na takav način da se podaci o kriptografskim resursima (modulima) i načinima njihovog pokretanja očitaju iz inicijalizacionih tijela (relacija u bazi podataka ili inicijalizacionih datoteka) a da se nakon toga iterativno pokreću kriptografski resursi (moduli) jedan za drugim, pri čemu se dalje u obzir uzimaju i ostali parametri koji utiču na kreiranje prediktivnih modela. Pri ovome se dobijeni rezultati, odnosno vremena potrebna za šifrovanje/dešifrovanje najprije kategorišu, a zatim zapisuju u odgovarajućoj relaciji baze podataka.

U drugom koraku, kreiranje modela provodi se postupkom treniranja odabranom metodom mašinskog učenja, kada se dobijeni rezultati (vremena potrebna za šifrovanje) očitavaju iz odgovarajuće relacije baze podataka. Na taj način se gradi prediktivni (prognostički) model koji se potom zapisuje u posebnu datoteku ili relaciju u bazi podataka.

Na proces kreiranja modela utiče veliki broj faktora. Šifrovanje različitim dužinama ključeva u osnovi predstavlja obradu podataka različitim algoritmima, tako da dobijeni rezultati pripadaju različitim kategorijama i samim tim proizvode različite prediktivne modele. Odabrana tehnologija interprocesne komunikacije takođe utiče na kreiranje različitih prediktivnih modela. Sve navedeno važi i za dešifrovanje - dakle dešifrovanje različitim dužinama ključa, korišćenjem različitih vrsta komunikacije itd. proizvodi rezultate koji pripadaju različitim kategorijama za koje je potrebno proizvesti različite prediktivne modele. U slučajevima kada je primjenjivo, kod prikupljanja trening podataka moraju se uzeti u obzir druge karakteristike modula, kao što je kriptografski način rada (*mode of operation*) ili broj niti koji će učestvovati u obradi podataka, koje takođe treba predstaviti različitim prediktivnim modelima.



Slika 26. Veliki broj parametara koji utiču na kreiranje modela

Uticaj velikog broja parametara iskazuje se kreiranjem odgovarajućeg broja klasa podataka koje se potom pretvaraju u pojedinačne prediktivne modele, koji se potom snimaju u datoteke ili u odgovrajućoj relaciji baze podataka. Na ovaj način se modul za koordinaciju oslobađa velikog broja izračunavanja u prisustvu velikog broja

parametara, jer se, u najtežem slučaju, proračunavanje svodi na traženje minimuma nekoliko klasa. Zahvaljujući planiranoj strukturi inicijalizacione datoteke koja je prikazana u listingu 31, postupak kategorizacije se programski može jednostavno riješiti bez velikog utroška procesorske snage.

Važan korak u kreiranju trening podataka je davanje naziva kategoriji podataka koja će proizvesti prediktivni (prognostički) model. Ovaj naziv se dalje koristi kako za pristupanje podacima tokom kreiranja modela tako i za pristupanje ranije kreiranim prediktivnim modelima. Naziv modela kreira se u okviru modula za treniranje na osnovu više faktora. U naziv ulazi najprije opisni identifikator resursa na koji se dodaje kriptografski način rada, slovo koje označava šifrovanje ili dešifrovanje, korišćeni mod, dužina ključa za koju se vrši testiranje, broj niti koji je korišćen i vrsta komunikacije. U slučaju stavke koja je prikazana u listingu 31, naziv modela bi izgledao ka u tabeli 14.

```
OraSun_ + C + OFB + 128 + 1 + s = OraSun_COFB1281smx10
```

Tabela 14: Naziv modela

Formirani naziv se sastoji od slijedećih dijelova:

- opisni identifikator: OraSun_ (Oracle/SUN AES),

- šifrovanje ili dešifrovanje: c (šifrovanje),

- način rada: OFB

- dužina ključa: 128,

- broj jezgara: 1

- komunikacija: smx10 (putem soketa, maksimalni skup poruka, verzija 1.0).

Prilikom kreiranja trening podataka moguće je odrediti i broj mjerenja (rundi) koja će biti obavljena po svakoj odabranoj tački ulaznih podataka. Na taj način je moguće po jednoj tački na XY ravni (datoteka, bafer) u kojoj se vrši mjerenje dobiti po nekoliko rezultata koji se nalaze jedan iznad drugog po Z osi (vrijeme). Ovakav skup podataka se može metodama mašinskog učenja obrađivati neizmjenjen ili se može najprije grupisano izvući srednja vrijednost rezultata koji se nalaze jedan iznad drugog po Z osi po svakoj tački na XY ravni, nakon čega se odabranoj metodi mašinskog učenja daju takve srednje vrijednosti na dalju obradu. Pribavljanje podataka iz odgovarajuće relacije u bazi podataka koji se koriste u metodama mašinskog učenja vrši se odgovarajućim SQL upitima čiji rezultati se koriste kao trening podaci za kreiranje modela. Kada se govori o SQL upitima i tu je postignuta modularnost na način da je ostavljena mogućnost da se oni mogu izmjeniti pomoću konfiguracionih datoteka, s tim da je moguće odabrati jedan od u listingu 23 prikazana dva oblika iskaza.

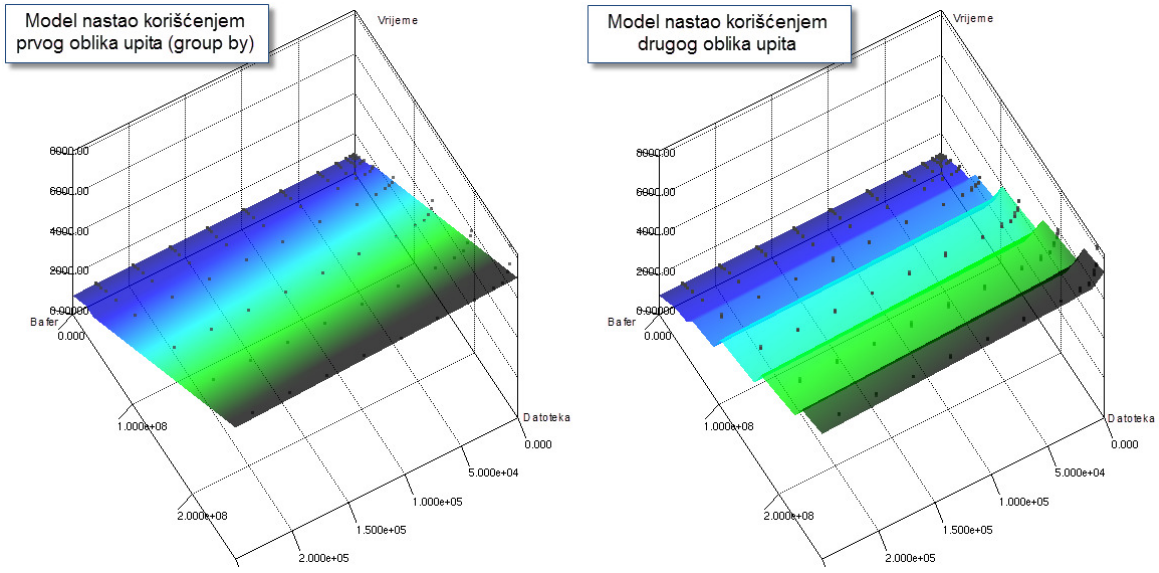
```
// Prvi oblik
"select BUFF, FILESIZE, avg(POLJE) from TESTDATACIPHER
where Klasa = KLASA and Komunikacija = KOMUNIKACIJA group by BUFF,
FILESIZE"

// Drugi oblik
"select BUFF, FILESIZE, POLJE from TESTDATACIPHER
where Klasa = KLASA and Komunikacija = KOMUNIKACIJA"
```

Listing 23: Sintaksa SQL upita

Za uključenje one metode mašinskog učenja koja će se koristiti u procesu adaptacije virtuelnog fajl sistema raspoloživim resursima ponovo je odabran modularan pristup. Ostavljena je mogućnost zamjene inicijalno odabrane metode mašinskog učenja

nekom drugom koja može pokazati bolje rezultate u specifičnim uslovima. U (Witten et al. 2011) se kao metode za numeričko predviđanje navode metode M5P (stabla modela), M5Rules (pravila iz stabla modela), Multilayer perceptron (vrsta neuronske mreže) i LWL (*locally weighted learning*, lokalno ponderisano učenje). Generisani modeli se mogu prikazati matematičkim modelima (kao na listingu 24), ali i kao površi u prostoru.

Kada se koriste prikupljeni trening podaci koji su nastali šifrovanjem bibliotekom BouncyCastle, 128 bitnom enkripcijom, ECB režimim rada i korišćenjem jednog jezgra, u slučaju kada se koriste dva navedena oblika SQL upita za očitavanje ovih podataka radi generisanja modela pomoću metode M5', dobijaju se površi koje su prikazane na slici 27. Zbog relativno malog skupa inicijalnih trening podataka, model generisan pomoću metode M5' se bolje uklapa u skup podataka kada se koristi prvi oblik SQL upita kako je prikazano na slici 27.



Slika 27. Modeli nastali korišćenjem M5' metode različitim oblicima upita

Među navedenim metodama, najveći stepen korelacije i najmanje greške pokazvali su modeli koji su nastali pomoću M5' metode u slučaju kada se koristio prvi oblik upita sa grupisanim podacima kada je od SUBP dobijena srednja vrijednost. U tabeli 16 prikazan je odnos koeficienata korelacije i grešaka matematičkih modela koji su kreirani različitim metodama mašinskog učenja koje su primjenjene na podatke koji su dobijeni od različitih kriptografskih modula.

M5'	**M5 Rules**	**Multilayer perceptron**
Bouncy Castle 128 Correlation coefficient 0.9945 Mean absolute error 83.8665 Root mean squared error 189.1539 Relative absolute error 5.9459 % Root relative squared error 10.4909 % Total Number of Instances 168	Bouncy Castle 128 Correlation coefficient 0.9945 Mean absolute error 85.6463 Root mean squared error 188.6227 Relative absolute error 6.0721 % Root relative squared error 10.4615 % Total Number of Instances 168	Bouncy Castle 128 Correlation coefficient 0.9929 Mean absolute error 400.748 Root mean squared error 452.8177 Relative absolute error 28.4121 % Root relative squared error 25.1143 % Total Number of Instances 168
Oracle-Sun 128 Correlation coefficient	Oracle-Sun 128 Correlation coefficient	Oracle-Sun 128 Correlation coefficient

0.9955 Mean absolute error 76.2062 Root mean squared error 169.8342 Relative absolute error 5.4214 % Root relative squared error 9.47 % Total Number of Instances 168	0.9955 Mean absolute error 77.7956 Root mean squared error 169.3916 Relative absolute error 5.5345 % Root relative squared error 9.4453 % Total Number of Instances 168	0.9941 Mean absolute error 268.2979 Root mean squared error 322.6751 Relative absolute error 19.087 % Root relative squared error 17.9924 % Total Number of Instances 168
Cuda AES 128 Correlation coefficient 0.9108 Mean absolute error 1151.4918 Root mean squared error 3207.2547 Relative absolute error 38.4934 % Root relative squared error 48.9132 % Total Number of Instances 168	Cuda AES 128 Correlation coefficient 0.9601 Mean absolute error 949.1043 Root mean squared error 1923.7328 Relative absolute error 31.7277 % Root relative squared error 29.3385 % Total Number of Instances 168	Cuda AES 128 Correlation coefficient 0.8382 Mean absolute error 1769.5201 Root mean squared error 3686.8224 Relative absolute error 59.1535 % Root relative squared error 56.227 % Total Number of Instances 168

Tabela 15: Poređenje koeficienta korelacije i grešaka za modele dobijene pomoću M5' i M5 rules i Multilayer perceptron metoda

Dodatna evaluacija generisanog matematičkog modela može se vršiti nad istim skupom podataka nad kojim je vršen trening, kada se utvrđuje koliki je stepen korelacije, srednja apsolutna greška, korijen iz srednje kvardratne greške, relativna apsolutna greška i korijen iz relativne kvadratne greške izgrađenog modela sa trening podacima ili sa novim skupom podataka, kada se utvrđuje stepen slaganja izgrađenog prediktivnog modela sa novim podacima. Primjer jednog generisanog prediktivnog (prognostičkog) modela sa stepenom korelacije i izračunatim greškama dat je u listingu 24.

```
M5 pruned model tree:
(using smoothed linear models)

FILESIZE <= 24576000 :
|   FILESIZE <= 2304000 :
|   |   FILESIZE <= 20480 : LM1 (36/4.778%)
|   |   FILESIZE >  20480 : LM2 (36/4.845%)
|   FILESIZE >  2304000 :
|   |   BUFF <= 6144 : LM3 (9/7.581%)
|   |   BUFF >  6144 : LM4 (27/3.863%)
FILESIZE >  24576000 :
|   BUFF <= 6144 : LM5 (15/94.008%)
|   BUFF >  6144 :
|   |   BUFF <= 24576 : LM6 (10/9.686%)
|   |   BUFF >  24576 :
|   |   |   FILESIZE <= 98304000 :
|   |   |   |   FILESIZE <= 49152000 : LM7 (7/3.319%)
|   |   |   |   FILESIZE >  49152000 : LM8 (7/0.951%)
|   |   |   FILESIZE >  98304000 :
|   |   |   |   FILESIZE <= 229376000 : LM9 (14/4.737%)
|   |   |   |   FILESIZE >  229376000 : LM10 (7/3.537%)

LM num: 1
AVG =
      -0.0023 * BUFF
      + 0.0002 * FILESIZE
```

```
      + 906.8617

LM num: 2
AVG =
      -0.0023 * BUFF
      + 0.0002 * FILESIZE
      + 1140.0487

LM num: 3
AVG =
      -0.1261 * BUFF
      + 0 * FILESIZE
      + 1897.7393

LM num: 4
AVG =
      -0.003 * BUFF
      + 0 * FILESIZE
      + 1459.9705

LM num: 5
AVG =
      -2.5288 * BUFF
      + 0.0001 * FILESIZE
      + 9156.17

LM num: 6
AVG =
      -0.0664 * BUFF
      + 0 * FILESIZE
      + 3286.6054

LM num: 7
AVG =
      -0.0153 * BUFF
      + 0 * FILESIZE
      + 2354.1881

LM num: 8
AVG =
      -0.0156 * BUFF
      + 0 * FILESIZE
      + 2427.6144

LM num: 9
AVG =
      -0.0158 * BUFF
      + 0 * FILESIZE
      + 2601.1068

LM num: 10
AVG =
      -0.0158 * BUFF
      + 0 * FILESIZE
      + 2646.1666

Number of Rules : 10
1.10
Results
======
Correlation coefficient                  0.9108
Mean absolute error                   1151.4918
Root mean squared error               3207.2547
Relative absolute error                 38.4934 %
```

```
Root relative squared error              48.9132 %
Total Number of Instances              168
```

Listing 24: Primjer modela kreiranog pomoću biblioteke Weka korišćenjem M5' metode sa koef.korelacije i greškama

Za izračunavanje svakog ovakvog modela potrebno je određeno vrijeme koje zavisi od količine podataka koje on mora da obradi. Zahvaljujući činjenici da se neki prediktivni model može zapisati u posebnu datoteku ili relaciju u bazi podataka, jednom pripremljeni i zapisani model se može ponovo korisiti bez gubljenja vremena za ponovno izračunavanje.

Zavisno od konfiguracije, sistem može da nastavi da prikuplja trening podatke pri svakom pokretanju kriptografskog resursa. Ako je neki kriptografski resurs kreiran na standardan način, kako je to opisano u poglavljima (4.7) i (6.2), sistem može da ga pokreće na predviđen način i da po završetku njegovog rada snimi podatke o njegovom izvršenju da bi bio obogaćen skup trening podataka. Ova mogućnost se može po potrebi ukinuti jednostavnim postavljanjem prekidača u okviru konfiguracione datoteke. Ako sistem konfigurisan tako da zapisuje podatke o svakom novom izvršenju, potrebno je povremeno ponovo izvršiti kreiranja modela. Na ovaj način se obezbjeđuje elastičnost sistema koji na ovaj način samog sebe popravlja novim podacima.

Testovi se pokreću prilikom instalacije sistema ali i po zahtjevu korisnika, kada se mogu testirati svi kriptografski moduli ili samo određeni modul koji je npr. tek ugrađen u računarski sistem. U slučaju dodavanja novog kriptografskog resursa (novog modula), on se navodi kao stavka u inicijalizacionoj datoteci, a potom se pristupa njegovom testiranju. Ako se ne provede postupak njegovog testiranja, sistem neće moći da odabere ovakav modul jer bez trening podataka neće moći da generiše model niti da utvrdi njegovu efikasnost.

6.1.2 Sistem za selekciju (Modul selektor)

Zadatak sistema za selekciju je da odabere najefikasniji kriptografski resurs (modul) koji će uz minimalan utrošak resursa sistema najbrže izvršiti šifrovanje odnosno dešifrovanje podataka. On se sastoji od dva dijela: podsistem za klasifikaciju i podsistem za nominaciju.

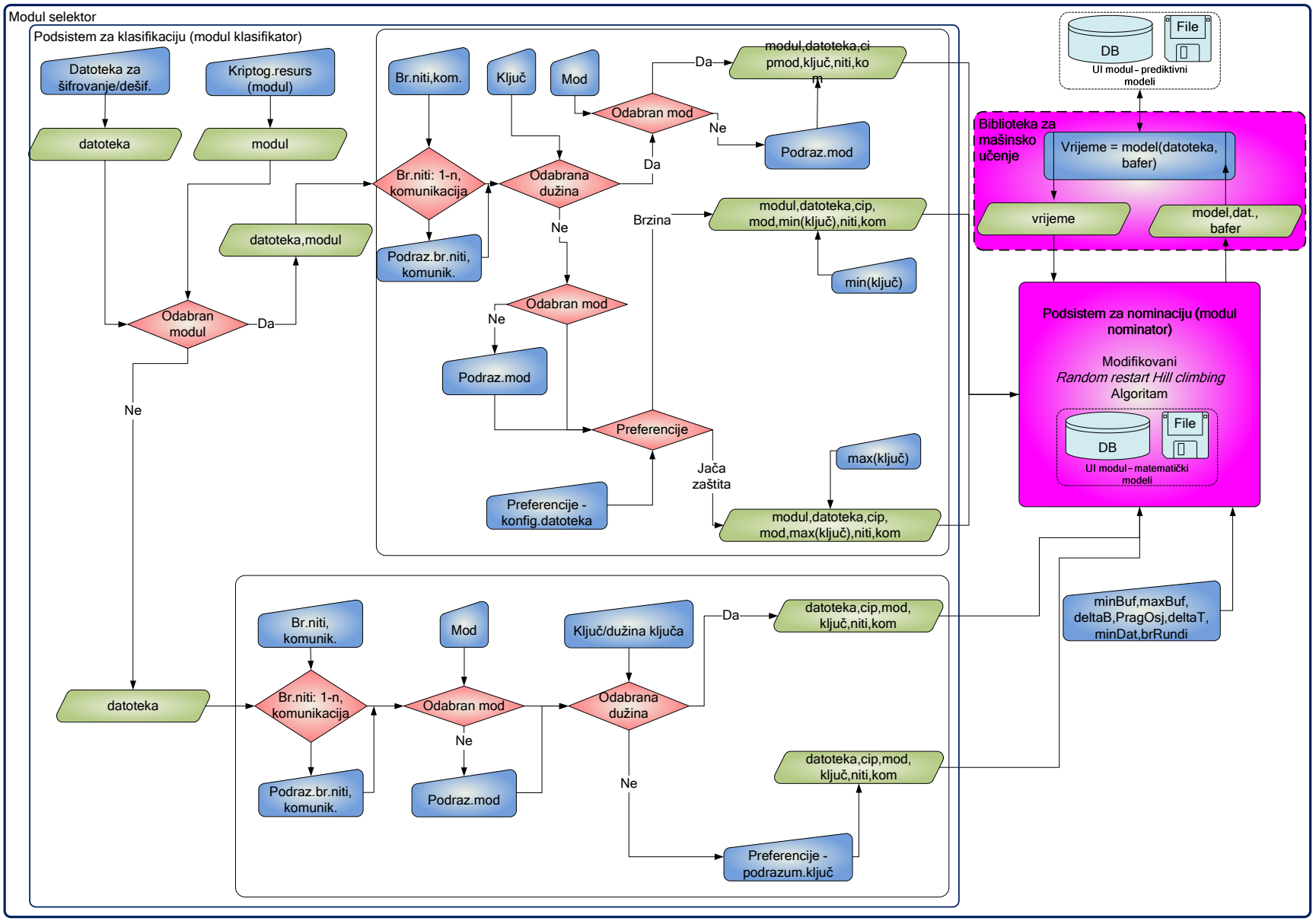
6.1.2.1 Podsistem za klasifikaciju (modul za klasifikaciju)

Podsistem za klasifikaciju se u svom radu oslanja na ranije izvršena mjerenja čiji su rezultati predstavljali trening podatke za proces mašinskog učenja kada su kreirani prognostički modeli koji predviđaju brzinu izvršavanja pojedinih kriptografskih resursa u datim uslovima. U okviru modula za trening izvršena je kategorizacija trening podataka, što predstavlja veoma važan korak od kojeg zavisi rad podsistema za klasifikaciju. Ovaj podsistem svoje ponašanje treba da prilagodi kako zahtjevima korisnika koje on zadaje bilo direktno, npr. putem komandne linije ili indirektno, putem sistema inicijalizacionih datoteka. Na slici 28 je prikazana blok šema ovog podsistema. Sa slike je vidljivo da je prvi podatak koji ovaj modul uzima na svom ulazu datoteka (preciznije dužina datoteke) koju treba šifrovati a da je opcioni parametar kriptografski resurs (modul) kojim treba vršiti šifrovanje.

Ako je zadan kriptografski resurs (modul) kojim treba vršiti šifrovanje, ovaj podsistem provjerava da li je zadan ključ (odnosno dužina ključa), kriptografski način rada (*mod of operation*) i način komunikacije. Ukoliko je primjenljivo, sistem koristi zadani broj

niti koje se mogu korisititi za šifrovanje ili podrazumjevanu jednu nit. Ukoliko kriptografski način rada (mod) nije zadan, sistem provjerava preferencije korisnika. Ukoliko ključ (odnosno dužina ključa) nisu zadati, sistem ponovo provjerava preferencije korisnika. Preferencije korisnika zapisane su u konfiguracionoj datoteci kao uređena petorka: (jača zaštita ili brzina, minimalna dužina ključa, maksimalna dužina ključa, podrazumjevana dužina ključa, kriptografski način rada). U slučaju da je korisnik u konfiguraciji naveo da želi veću brzinu, iz konfiguracione datoteke se uzima predefinisana minimalna dužina ključa, a zatim klasifikator prepušta podsistemu za nominaciju da izabere najefikasniji kriptografski resurs. U slučaju da korisnik u konfiguraciji navede da želi jaču zaštitu, iz konfiguracije se bira podrazumjevana maksimalna dužina ključa, pa se podaci ponovo prosliijeđuju modulu nominatoru. Kako je u poglavlju 5.2 navedeno, pretpostavlja se da postoji siguran način čuvanja ključeva u okviru operativnog sistema odakle će podsistem za klasifikaciju odabrati ključ željene dužine da bi inicijalizovao odabrani kriptografski resurs.

Ako je zadana datoteka, ali ne i kriptografski resurs pomoću kojega treba vršiti šifrovanje, sistem najprije provjerava da li je zadan kriptografski način rada (*mode of operation*) i način komunikacije. Ukoliko kriptografski način rada (mod) nije zadan, sistem ponovo provjerava preferencije korisnika. Ukoliko je primjenljivo, sistem koristi zadani broj niti koje se mogu korisititi za šifrovanje ili podrazumjevanu jednu nit. Nakon toga, klasifikator provjerava da li je zadan ključ (odnosno dužina ključa). Ako je zadana dužina ključa, selektor šalje ove podatke (datoteka, ključ, mod) modulu nominatoru na dalju obradu. Ako nije zadan ključ, na osnovu preferencija korisnika iz konfiguracione datoteke uzima se podrazumjevana dužina ključa, pa se podaci (datoteka, ključ, mod) uz podatke o načinu komunikacije i o operaciji (šifrovanje ili dešifrovanje - cip) ponovo proslijeđuju podsistemu za nominaciju.



Slika 28. Blok šema modula selektora

Na prethodnoj slici su korišćene sliijedeće skraćenice:

Cip – Logička konstanta, označava da li se radi o šifrovanju ili dešifrovanju;

Datoteka – dužina datoteke u bajtima, brojčani podatak;

Modul – Identifikator kriptografskog resursa, može da bude tekstulni ili numerički podatak;

Br.niti – broj niti koje će se koristiti u šifrovanju ili podrazumjevana jedna nit, brojčani podatak;

Kom. – željeni način interprocesne komunkacije, tekstualni podatak; moguće je navesti nekoliko načina komunikacije, od kojih će svaki rezultirati određenim brojem prediktivnih modela;

Ključ – dužina ključa, brojčani podatak;

Mod – kriptografski način rada (*mode of operation*), tekstualni podatak;

minBuf, maxBuf – minimalna i maksimalna dužina bafera koja će se koristiti pri testiranju brzine

deltaB – vrijednost za koju će se bafer uvećavati,

deltaT – prag osjetljivosti; podsistem za nominaciju prekida traganje za dati resurs ako je dobit u brzini šifrovanja (u milisekundama) pala ispod ovog praga;

brRundi – broj rundi koji se koristi za modifikovani *Hill-climbing* algoritam;

minDat – dužina datoteke ispod koje se neće vršiti testovi. Ova vrijednost će detaljnije biti objašnjena kasnije.

6.1.2.2 Podsistem za nominaciju (modul nominator)

Zadatak ovog podsistema je da na osnovu podataka koje dobija od klasifikatora izabere najefikasniji kriptografski resurs. Algoritam na osnovu kojeg će modul nominator donijeti odluku zavisi od broja podataka koje dobija. Podaci se ovom modulu dobavljaju u obliku uređene šestorke ili sedmorke [kriptografski resurs/modul, datoteka, šifrovanje/dešifrovanje, način rada (mod), dužina ključa, broj niti, način komunikacije], zavisno od toga da li je korisnik odabrao kripotografski resurs. Ostale podatke potrebne za rad ovaj modul dobija iz konfiguracionih datoteka, kao uređenu šestorku [minimalna dužina bafera, maksimalna dužina bafera, dužina fragmenta bafera deltaB, prag osjetljivosti deltaT, minimalna dužina datoteke, broj rundi]. U zavisnosti od ulaznih podataka ovaj modul može da funkcioniše na dva načina.

Kada na ulazu dobije podatak o kriptografskom resursu koji će se korisiti (ako je korisnik odabrao kriptografski modul), modul nominator će izračunati kolika je dužina najmanjeg internog bafera pri kojem se postiže najveća brzina šifrovanja. Ako na ulazu ne dobije podatke o željenom kriptografskom resursu, nominator će predvidjeti koji je najbrži kriptografski resurs koji će uz najmanju veličinu kriptografskog bafera na najbrži način izvršiti šifrovanje. U poglavlju 4.3.4 predstavljen je *Hill-climbing* algoritam pomoću

kojega je moguće pronaći lokalne minimume i maksimume i njegova nešto naprednija verzija *Random restart Hill-climbing*. Ovaj algoritam je djelimično modifikovan na takav način da se umjesto slučajno odabranih stanja bira n (podrazumjevano 4) ravnomjerno raspoređenih stanja (dužina bafera), a zatim se traži onaj pri kojem se postiže najveća brzina. Zatim se u okolini najboljeg dobijenog rezultata odabere novih n (podrazumjevano 4) dužina u čijim krajnjim tačkama se vrši predviđanje, pri čemu se odustaje od pretrage ako je postignuto poboljšanje manje od nekog zadanog praga delta (podrazumjevano 50ms). Na ovaj način se postiže, imajući u vidu prag osjetljivosti, najveća brzina šifrovanja ili dešifrovanja uz optimalan utrošak vremena.

U oba slučaja koristi se algoritam koji je predstavljen pseudo kodom na listingu 25. Jedina razlika je u broju pozvanih modela koji predstavljaju kriptografske resurse u momentu kada se traži minimum. U prvom slučaju kriptografski resurs je već određen, tako da se samo u drugom slučaju traži minimum koji u datim uslovima predviđaju pojedini moduli.

```
INPUT (CipInv, dat, K, mode, threads, Comm, M) or
      (CipInv, dat, K, mode, threads, Comm)

while (Mn in [M1..MN])
  minB <- config(min_buf)
  oldB = minB
  maxB <- config(max_buf)
  minD <- config(min_dat)
  rounds <- config(max_round)
  round = 1
  if (dat < minD) exit
  delta <- config(prag_osjetljivosti)

  while (round < rounds)

    while (newB < maxB)
      newT = Mn(newB, dat)
      if (newT < oldT)
        oldT = newT
        bestB = newB
        Mb = Mn
        if ((newT - oldT) < delta)
          results(oldT, bestB, Mb)
          quit // dobit je manja od delta - kraj
        end if
        oldB = newB
        inc(newB, deltaB)
      else
        break     // nema dobiti - kraj unutrašnje petlje
      end if

    end while
  inc(round, 1)
  maxB = bestB + (maxB-minB)/(rounds/2) // nastavi u okolini najboljeg
  minB = bestB - (maxB-minB)/(rounds/2)
  end while

end while
```

```
results(oldT, bestB, Mb)
quit

gdje je:
INPUT (CipInv, M, dat, K, mode, threads, Comm) or
      (CipInv, dat, K, mode, threads, Comm) - ulazni podatak:
Mn – odabrani modul,
Mb – modul za koji je dobijeno do sada najbolje vrijeme,
config() - očitavanje podataka iz konfiguracione datoteke,
min_buf - minimalna dužina bafera koja će se koristiti,
max_buf, maxB - maksimalna dužina bafera koja će se koristiti,
max_round, rounds - maksimalan broj iteracija kroz koje algoritam
prolazi,
round – trenutna runda modifikovanog Hill Climbing algoritma
min_dat, minD - minimalna dužina datoteke. Manje dtoteke od ove se ne
testiraju,
oldT – najbolje vrijeme izvršenja,
newT – novo vrijeme izvršenja,
oldB – stara dužina bafera,
newB – nova dužina bafera,
bestB –dužina bafera pri kojoj je postignuto do sada najbolje vrijeme
dat – dužina datoteke,
K – dužina ključa
M1(newB, dat),..., MN(newB, dat) - rezultati predviđanja koje daju
ranije generisani modeli.
```

Listing 25: Pseudo kod - modul nominator

Modul nominator se u svom radu oslanja na pozive odabranoj biblioteci funkcija za mašinsko učenje. Ranije generisani i u bazi podataka ili datotekama zapisani prediktivni modeli se prema zahtjevima algoritma očitavaju i zajedno sa ulaznim parametrima predaju na izračunavanje biblioteci algoritama za mašinsko učenje, koja vraća pretpostavljenu brzinu izvršavanja za date ulazne parametre i model. Pretpostavljena brzina izvršavanja dalje učestvuje u donošenju odluke o izboru optimalnog kriptografskog resursa.

Skupovi rezultata izvršavanja pojedinih resursa (modula) koji vrše obradu podataka pomoću kriptografskog algoritma AES najprije su podijeljeni u kategorije podataka (klase) koje zavise od velikog broja parametara (enkripcija ili dekripcija, modul, broj niti, način rada, ključ, komunikacija) kako je to prikazano na slici 26. Podaci koji pripadaju jednoj klasi dalje se mogu predstaviti kao odnos dvije nezavisne varijable (dužina datoteke i bafera). Oni se prema brzini šifrovanja u grupišu u oblike koji se intuitivno mogu predstaviti kao površi, pri čemu svaka površ predstavlja kontinuiranu klasu. Ove klase trening podataka se pomoću metoda mašinskog učenja pretvaraju u prediktivne modele koji služe za predviđanje brzine izvršenja pojedine klase.

Dakle, dio atributa koji utiču na vrijeme izvršavanja prema ovom algoritmu najprije se kategorizuje, pa se na osnovu trening podataka vrši kreiranje prediktivnih modela i predviđanje vremena izvršenja.

Radi potvrde koncepta primjenjena je *open source* biblioteka Weka koja pruža pristup velikom broju algoritama iz oblasti rudarenja podataka i mašinskog učenja, uključujući i metodu M5'. Ova metoda omogućava kreiranje prediktivnih (prognostičkih) modela iz numeričkih podataka. Biblioteka Weka ima mogućnost snimanja prediktivnih modela u

tekstualnu datoteku, a ovakvi tekstualni podaci se jednostavno mogu snimiti i u nekoj bazi podataka. Obje ove mogućnosti su takođe implementirane u adaptibilnom virtuelnom kriptografskom fajl sistemu.

Podsistem za nominaciju u primjenjenom rješenju ima integrisanu podršku za povezivanje sa SUBP, koja je u ovom slučaju realizovana pomoću JDBC tehnologije, zbog čega se primjenjeni SUBP Firebird može po potrebi lako zamjeniti nekim drugim sistemom.

6.1.3 Ulazno-izlazni modul

Modul za koordinaciju sa svojom okolinom komunicira uz pomoć ulazno-izlaznog modula. Ulazno izlazni modul posjeduje podsistem koji je zadužen za očitavanje inicijalizacionih datoteka. U njima se nalaze osnovna podešavanja sistema te lokacije i podešavanja kriptografskih resursa (modula) uključenih u sistem.

Na osnovu konfiguracione datoteke u kojoj se nalaze odgovarjući podaci se odlučuje da li će neki kriptografski modul biti korišćen ili ne, na koji način će biti korišćen, kako će biti vršeno treniranje i kako će u bazi podataka odgovarajuća relacija biti popunjena podacima o izvršenim mjerenjima.

Veza sa korisnikom je realizovana putem komandne linije, pri čemu se korisnik obraća adaptibilnom virtuelnom kriptografskom fajl sistemu, a navedene ulazne podatke preuzima ulazno-izlazni modul. Dobijeni podaci se dalje obrađuju pomoću parsera komandne linije.

Komande koje korisnik na ovaj način može da zada obuhvataju kreiranje, kopiranje, pomjeranje i brisanje datoteka i direktorijuma, zatim šifrovanje podataka navođenjem izvorne i odredišne datoteke ili direktorijuma te opciono ključa i željenog kriptografskog resursa. U slučaju da korisnik ne navede željeni kriptografski resurs ni ključ pomoću kojega želi da izvrši šifrovanje/dešifrovanje, AKVFS je dužan da odabere resurs pomoću kojega može najoptimalnije da izvrši šifrovanje ili dešifrovanje, pri čemu će se po potrebi konsultovati sa konfiguracionom datotekom u kojoj su zapisane preferencije korisnika.

U okviru ulazno izlaznog modula nalazi se i veza sa omotačem za pristup sistemu za upravljanje bazama podataka. Za ovaj omotač treba odabrati tehnologiju koja će omogućiti da se neki sistem za upravljanje bazama podataka po potrebi može lako zamjeniti drugim.

U ulazno-izlaznom modulu je implementiran i podsistem za mjerenje koji mjeri vrijeme izvršenja svakog kriptografskog resursa pokrenutog iz adaptibilnog virtuelnog kriptografskog fajl sistema. Tako dobijeni podaci se, ako je tako konfigurisano, zapisuju u bazu podataka. Ovi podaci dalje ponovo obrađuju metodama mašinskog učenja i utiču na poboljšanje procjene brzine obrade podataka.

U primjenjenom rješenju, veza sa konfiguracionim datotekama ostvarena je pomoću Java biblioteka iz *org.apache.commons.configuration* paketa. Parser komandne linije je realizovan oko Java biblioteke iz *org.apache.commons.cli* paketa.

Kao sistem za upravljanje bazama podataka (SUBP) u koji će za potrebe testiranja biti zapisivana mjerenja brzine šifrovanja pojedinih implementacija, u primijenjenom dijeu rada odabran je open source SUBP Firebird. Ovo je SUBP lagane (*lightweight*) kategorije koji je implementiran na Windows, Linux i Unix platformama. Ovaj SUBP se jednostavno povezuje sa bilo kojim programom pomoću JDBC tehnologije. Upotreba JDBC tehnologije u adaptibilnom kriptografskom virtuelnom fajl sistemu omogućava da se odabrani SUBP može tretirati samo kao još jedan modul koji je po potrebi moguće lako zamjeniti nekim drugim rješenjem. Ovakav pristup otvara mogućnost da se na nekoj drugoj platformi upotrebi sistem koji je na takvoj platformi dostupan, a koji se pomoću JDBC tehnologije može u formi modula uključiti u sistem.

Rad ulazno-izlaznog modula karakteriše i podsistem (modul) koji je zadužen za interprocesnu komunikaciju. Ovaj modul dozvoljava korišćenje različitih metoda za interprocesnu komunikaciju, a zadatak sistema za nominaciju je da odabere najefikasniji metod. U primjenjenom dijelu su radi potvrde koncepta implementirani načini komunikacije pomoću tehnologije neimenovanih cijevi i soketa.

Ulazno izlazni modul je u velikoj mjeri zaslužan za mogućnost uključivanja originalnih i *third party* aplikacija kao *plug-in*-ova u adaptabilni virtuelni kriptografski fajl sistem. Standardizacija poruka koja je opisana u okviru poglavlja (6.2) i mogućnost unapređenja poruka iz verzije u verziju je osnovni preduslov za uključenje različitih modula u ovakav sistem.

6.1.4 Uloga sistema za upravljanje bazama podataka u okviru ulazno-izlaznog podsistema

Pristup sistemu za upravljanje bazama podataka neophodan je radi smještanja trening podataka i na osnovu njih kreiranja modela. U bazi podataka se opciono mogu smjestiti konfiguracioni podaci i generisani modeli.

Za funkcionisanje sistema dovoljna je jedna baza podataka. Najvažnija relacija u navedenoj bazi podataka je relacija pod nazivom TestDataCipher. U ovoj relaciji se nalaze različiti rezultati mjerenja performansi, odnosno trening podaci. Ova relacija ima slijedeću strukturu:

<ID, BUFF, FILESIZE, TIMELEN, RES, CLIENTENCDECTIME, CLIENTWHOLETIME, COMM, VFS_OKTIME, NUMBEROFTHREADS, KEYSIZE, CIPINV>.

Pojedini atributi (kolone) u relaciji (tabeli) imaju značenje kako slijedi:

- BUFF - veličina bafera koji se koristi u procesu šifrovanja ili dešifrovanja.
- FILESIZE - veličina datoteke koja je štifrovana;
- TIMELEN - ukupno vrijeme šifrovanja ili dešifrovanja, mjereno u AKVFS-u;
- RES - naziv resursa koji se koristi za šifrovanje ili dešifrovanje. Naziv se formira kako je objašnjeno u poglavlju 6.1.1;
- CLIENTENCDECTIME - vrijeme obrade koje vraća kriptografski resurs, a u koje je uključeno samo vrijeme obrade podataka (EncDec) bez inicijalizacije i UI

operacija, ako je primjenljivo. Ako ova mogućnost nije implementirana na strani klijenta, upisuje se nula;

- CLIENTWHOLETIME - ukupno vrijeme obrade koje je izmjereno na strani kriptografskog resursa u koje je uključena inicijalizacija i UI operacije, ako je primjenljivo. Ako ova mogućnost nije implementirana na strani klijenta, upisuje se nula;
- COMM - navodi korišćenu vrstu komunikacije (u primijenjenom dijelu rada korišćeni su soketi ili neimenovane cijevi) te verzija protokola (minimalni ili maksimalni skup poruka sa verzijom);
- VFS_OKTIME - vrijeme koje protekne od pokretanja nekog kriptografskog modula dok adaptibilni virtuelni kriptografski fajl sistem ne dobije OK odgovor, ako je primjenljivo. Ako ova mogućnost nije implementirana na strani klijenta, upisuje se nula;
- NUMBEROFTHREADS - za module koji koriste više niti/jezgara. Za odule koji koriste jedno jezgro, upisuje se broj 1;
- KEYSIZE - podaci o dužini ključa;
- CIPINV - navodi da li se radi o rezultatima šifrovanja ili dešifrovanja;

Slijedeća relacija koja se (ako je konfigurisano) koristi je relacija pod nazivom Genmodels. U ovoj relaciji se nalaze generisani modeli nastali na osnovu trening podataka. Ova relacija ima jednostavnu strukturu:

<ID, MODELNAME, MODELDATA >.

Pojedini atributi u relaciji imaju značenje kako slijedi:

- MODELNAME – jedinstven (UNIQUE) naziv modela;
- MODELDATA - binarni (BLOB) podaci koji opisuju model.

Sadržaj ove relacije se u slučaju drugačije konfiguracije može naći i u različitim datotekama, gdje svaka datoteka nosi ime jednog prediktivnog modela i pripadajuće podatke koji opisuju model.

U bazi podataka se nalazi i relacija pod imenom Preferencije sa strukturom:

<ID, ZASTITA_BRZINA, MINK, MAXK, DEFK, MINDAT>

Pojedini atributi (kolone) u relaciji (tabeli) imaju značenje kako slijedi:

- ZASTITA_BRZINA - 0 - korisnik preferira jaču zaštitu, odnosno 1 - korisnik preferira brzinu, 2 – podrazumjevan nivo zaštite;
- MINK - minimalna dužina ključa koja će se koristiti ako korisnik preferira brzinu;
- MAXK - maksimalna dužina ključa koja će se koristiti ako korisnik preferira zaštitu;
- DEFK - podrazumjevana dužina ključa koja će se koristiti ako korisnik ne preferira ni brzinu ni jaču zaštitu (oznaka 2 u polju ZASTITA_BRZINA);
- MINDAT – minimalna dužina dateke ispod koje će se neće tražiti najbrži kriptografski resurs (modul) već će se koristiti interni modul.

Kako je u poglavlju 5.2 navedeno, u istraživanju je pretpostavljeno da postoji siguran način čuvanja ključeva u okviru operativnog sistema, zbog čega u relaciji Preferencije navodimo samo dužine ključeva.

U bazi podataka se nalazi i relacija pod imenom DefNominator koja drži podatke neophodne za rad modula nominatora, sa strukturom:

<ID, PRAG, MINB, MAXB, BRRUNDI>

Pojedini atributi u relaciji imaju značenje kako slijedi:

- PRAG - prag osjetljivosti nakon kojeg se prekida rad nominatora;
- MINB - minimalna dužina bafera od koje se počinje postupak traženja minimuma;
- MINK - maksimalna dužina bafera kada se prekida postupak traženja minimuma;
- BRRUNDI - broj rundi tokom kojih se vrši postupak traženja minimuma;
- MIN_DAT, MAX_DAT – minimalna i maksimalna veličina datoteke koje se koriste pri traženju praga iskoristivosti internog modula;
- DELTAB – veličina koja se koristi pri traženju praga iskoristivosti.

U bazi podataka se nalazi i relacija pod imenom ML koja drži podatke neophodne za pozivanje biblioteke za mašinsko učenje, sa strukturom:

<ID, SQL, METODA>

Pojedini atributi u relaciji imaju značenje kako slijedi:

- SQL – vrsta upita koja će se koristiti pri kreiranju prediktivnih modela, kako je to prikazano u listingu 23;
- METODA – odabrana metoda mašinskog učenja, u primjenjenom dijelu realizovane su M5P, M5Rules, MultilayerPerceptron i LWL;

Podaci iz četiri od prikazanih relacija (Genmodels, Preferencije, DefNominator, ML) mogu biti smješteni u istoimenim datotekama, kada se navedene relacije ostavljaju bez podataka. Navedene četiri relacije sadrže samo po jednu N-torku (jedan red). Jedino podaci koji se nalaze u relaciji TestDataCipher moraju biti smješteni u nekoj bazi podataka.

U primjeni, omotač za komunikaciju sa odabranim sistemom za upravljanje bazama podataka je implementiran pomoću JDBC tehnologije, zbog čega se odabrani SUBP Firebird može tretirati samo kao još jedan modul i po potrebi lako zamjeniti nekim drugim rješenjem.

6.2 Modeli komunikacije kriptografskih resursa i adaptibilnog virtuelnog kriptografskog fajl sistema

U toku razvoja generalnog modela za prikaz adaptibline primjene kriptografskog algoritma AES u modernim operativnim sistemima, pojavila se potreba za kreiranjem modela za uniformnu komunikaciju različitih kriptografskih modula sa sistemom, te za klasifikacijom i deskripcijom tih načina komunikacije koji se koriste u rješenju.

Kako je prikazano na slici 25, arhitektura rješenja je takva da su vlastiti kriptografski moduli i moduli nezavisnih proizvođača stavljeni u istu ravan, jer se svi oni fizički nalaze van adaptibilnog virtuelnog kriptografskog fajl sistema i modula za koordinaciju. Jedan od ciljeva istraživanja je bio i postizanje nezavisnosti modela adaptibilnog virtuelnog kriptografskog fajl sistema od operativnog sistema u kojem će biti primjenjen. Zbog toga je ostavljena mogućnost da se koriste različita rješenja za interprocesnu komunikaciju, a u radu su primijenjena i testirana dva načina povezivanja: pomoću neimenovanih cijevi i pomoću soketa.

Jedan od najvažnijih problema koji je morao biti riješen da bi model adaptibilnog virtuelnog kriptografskog fajl sistema postao upotrebljiv je bio problem heterogenosti korišćenih kriptografsih resursa. Ideja da se vlastiti moduli i moduli nezavisnih proizvođača stave u istu ravan podrazumjevano je sa sobom donijela mogućnost da će se u sistemu pojaviti različite vrste tehnoloških heterogenosti.

Zbog toga je za formatiranje podataka korišćen XML (Extensible Markup Language) standard koji navodi skup pravila za enkodiranje dokumenata na način koji je čitljiv i ljudima i mašinama. Za razliku od CORBA, IDL i Java interfejsa, XML nije vezan ni za jednu posebnu tehnologiju ni middleware standard, pa se danas često koristi kao *ad-hoc* format za obradu podataka između različitih a često nekompatibilnih *middleware* platformi. Kao dodatak svemu ovome, XML je vrlo fleksibilan, a ta fleksibilnost ga pozicionira kao najpogodniji standard za rješavanje problema heterogenosti aplikacija (Roshen 2009). Na izbor XML standarda za kreiranje dokumenata je uticala je i odluka da predviđeni format dokumenata ne treba da bude cementiran, već da se može prilagođavati prema potrebama iz verzije u verziju.

Iako je izbor XML-a važan za rješavanje problema heterogenosti, XML sam po sebi nije dovoljan da bi dvije strane (kriptografski fajl sistem i resurs) mogle ispravno da komuniciraju. Da bi komunikacije bile efikasne, učesnici u komunikaciji moraju da budu u stanju da razmjenjuju poruke prema unaprijed utvrđenom formatu odnosno prema nekom protokolu. Zbog toga se radi rješenja ovog problema pošlo od utvrđivanja minimalnog i maksimalnog skupa poruka koje su potrebne da bi komunikacija uspjela.

Minimalni skup poruka obuhvata odgovor kriptografskg resursa u kom on obavještava fajl sistem o tome koja vrsta paketa (verzija protokola) je u pitanju, da li je enkripcija ili dekripcija obavlljena uspješno i u kom on šalje rezultate nekih mjerenja koja se obavljaju na strani kriptografskog resursa (modula). Ovdje se pretpostavlja da kriptografskom resursu (modulu) nisu potrebni ulazni argumenti ili da su dostavljeni pomoću komandne linije, inicijalizacione datoteke ili na neki drugi način. Takođe je ostavljena mogućnost da se u slijedećoj verziji paketa nađu neka druga polja koja će omogućiti neki novi način komunikacije. U nastavku je dat izgled paketa u prvoj verziji protokola (1.0) kojim kriptografski modul šalje odgovor adaptibilnom virtuelnom kriptografskom fajl sistemu.

```
<?xml version="1.0"?>
<packet_receive>
    <packet_version>packet_version</packet_version>
    <id>id_data</id>
    <packet_type>type_data</packet_type>
    <millisecondsED>numerical data</millisecondsED>
```

```
    <milliseconds>numerical_data</milliseconds>
    <packet_type>packet_ type</packet_type>
    <error_no>errnumber</error_no>
    <error_msg>error_message</error_msg>
    <additional_data>for_further_use</additional_data>
</packet_receive >

Gdje je:
packet_version - verzija paketa
packet_type - šifrovanje ili dešifrovanje,
millisecondsED - vrijeme koje modul utroši samo na šifrovanje (bez
komunikacije sa tvrdim diskom)
milliseconds - ukupno vrijeme obrade u modulu
packet_type - tip paketa, string "results"
error_no - broj greške
error_msg - poruka o grešci
additional_data - dodatni podaci - rezervisano za buduću upotrebu
```

Listing 26: Format paketa poslanog iz modula ka VFS

Maksimalni skup poruka obuhvata razmjenu više poruka. Najprije, kriptografski resurs mora imati mogućnost da javi fajl sistemu da je pokrenut i koju verziju protokola može da koristi u okviru OK poruke. OK poruka kojom kriptografski modul potvrđuje da je ispravno pokrenut je jednostavna - u njenom sastavu se nalazi verzija paketa (protokola) i identifikator modula. Izgled paketa kojim kriptografski modul potvrđuje da je ispravno pokrenut je:

```
<?xml version="1.0"?>
<packet_ok>
    <packet_version>packet_version</packet_version>
    <id>id_data</id>
</packet_ok >
```

Listing 27: Format paketa potvrde (OK paketa) poslanog iz modula ka VFS

Po dobijanju ove poruke, adaptibilni virtuelni kriptografski fajl sistem šalje kriptografskom resursu (modulu) poruku u kom navodi parametre enkripcije ili dekripcije u skladu sa verzijom protokola. Oblik ove poruke u verziji 1.0 prikazan je na slijedećem listingu.

```
<?xml version="1.0"?>
<packet_send>
    <packet_version>packet_version</packet_version>
    <id>id_data</id>
    <packet_type>cip_invcip</packet_type>
    <key>key_data</key>
    <key_size>key_size</key_size>
    <buffer_size>buffer_size</buffer_size>
    <source_file>source_file_name</source_file>
    <dest_file>dest_file_name</dest_file>
    < PThrds>PThrds</PThrds>
    <mode_of_op>mode</mode_of_op>
    <iv>iv</iv>
    <additional_data>for_further_use</additional_data>
</packet_send>
```

```
Gdje je:
packet_version - verzija paketa
packet_type - šifrovanje ili dešifrovanje,
key - ključ za šifrovanje ili dešifrovanje
key_size - dužina ključa
buffer_size - veličina bafera
source_file - ulazna datoteka; u slučaju šifrovanja otvoreni tekst
dest_file - izlazna datoteka; u slučaju šifrovanja šifrat
PThrds - broj niti/jezgara
mode - kriptografski način rada
iv - IV kao string heksadekadnih brojeva
```

Listing 28: Format paketa poslanog iz VFS ka modulu

Izgled paketa kojim kriptografski modul šalje odgovor adaptibilnom virtuelnom kriptografskom fajl sistemu je identičan kao u kod minimalnog skupa poruka, kako je to prikazano u listingu 26.

6.2.1 Neimenovane cijevi u funkciji povezivanja modula i adaptibilnog virtuelnog kriptografskog fajl sistema

U primijenjenom dijelu istraživanja, minimalni način komunikacije je korišćen u kombinaciji sa neimenovanim cijevima. Operativni sistemi i programski jezici već dugo koriste imenovane i neimenovane cijevi za međuprocesnu komunikaciju. Kako se navodi u (Bidulock, 2008) cijevi imaju bogatu istoriju u operativnom sistemu UNIX. Predstavljene su u ranim verzijama operativnog sistema UNIX koji je proizveden u Belovim laboratorijama (*Bell Laboratories*). Cijevi su korišćene i u BSD i u System V izdanjima. U BSD-ovom izdanju 4.4 implementirane su pomoću soketa, a u System V izdanju 4 implementirane su pomoću tokova (*streams*). U (IBM Corporation, 2015) se navodi da ako neka z/OS C/C++ aplikacija pokreće procese sa kojima treba da komunicira, tada se mogu iskoristiti POSIX.1 kompatibilne neimenovane cijevi. Ako aplikacija treba da komunicira sa procesima koje nije ona pokrenula, tada mogu da se koriste imenovane cijevi. Kako se navodi u (Oracle Corporation 2016) programski jezik Java može da pokreće podprocese sa kojima može da komunicira korišćenjem cijevi, jer podproces podrazumjevano čita ulaz iz cijevi i podproces podrazumjevano ispisuje *standard output* i *standard error* u cijevi. Java može da preusmjeri ove cijevi na odgovarajuće tokove i da ih potom očita.

U ovom istraživanju su radi potvrde koncepta korišćene neimenovane cijevi i Java tehnologija ulazno-izlaznih tokova da bi se ostvarila komunikacija između kriptografskog modula i fajl sistema.

Kada adaptibilni virtuelni kriptografski fajl sistem kao glavni proces prenosi komande kriptografskom modulu kao podprocesu, on putem komandne linije prenosi komande pomoću dva formata:

```
Pgm Ci KeySize HexKey InFile OutFile BufSize hexIV PThrds mode
Ili
Java -jar Pgm Ci KeySize HexKey InFile OutFile BufSize hexIV
PThrds mode

Gdje je:
```

```
Pgm - naziv (sa putanjom) programa koji se pokreće
Ci - da li se pokreće šifrovanje ili dešifrovanje
KeySize - dužina ključa
HexKey - ključ kao string heksadekadnih brojeva
InFile - ulazna datoteka; u slučaju šifrovanja otvoreni tekst
OutFile - izlazna datoteka; u slučaju šifrovanja šifrat
BufSize - dužina bafera koji učestvuje u enkripciji
hexIV - IV kao string heksadekadnih brojeva
PThrds - broj niti/jezgara
mode - kriptografski način rada

prefiks java -jar se koristi u slučaju pokretanja Java programa
```

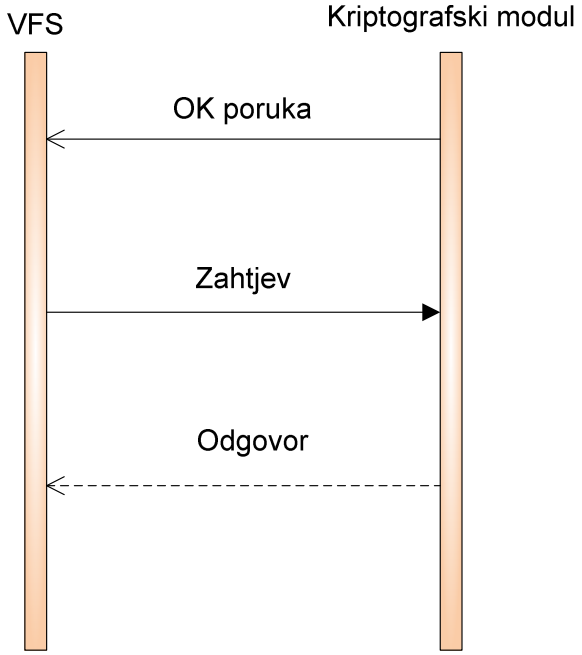
Listing 29: Format komande poslane iz VFS ka modulu

Odgovor kriptografskog modula se vraća putem neimenovane cijevi. Kriptografski modul šalje odgovor fajl sistemu u već opisanoj formi koja je data u listingu 26.

6.2.2 Soketi (*sockets*)u funkciji povezivanja modula i adaptibilnog virtuelnog kriptografskog fajl sistema

Kada je u pitanju maksimalni način komunikacije, on je u primijenjenom dijelu istraživanja korišćen u kombinaciji sa soketima. Mrežni soketi (network socket) su krajnje tačke u međuprocesnoj komunikaciji u računarskim mrežama.

Kada je u pitanju maksimalni način rada, odmah po pokretanju koje (na bilo koji način) inicira VFS, svaki kriptografski modul šalje nazad, prema fajl sistemu, OK poruku kojom potvrđuje da je u stanju da razmjenjuje podatke prema traženoj verziji protokola. Na ovaj način kriptografski modul obavještava VFS da je uspješno pokrenut.



Slika 29. Redoslijed razmjene poruka između VFS i pojedinih modula

Izgled sve tri poruke koje se razmjenjuju između VFS i pojedinih kriptografskih modula dat je u listinzima 26, 27 i 28.

6.2.3 Ostali podsistemi za povezivajne adaptibilnog virtuelnog kriptografskog fajl sistema sa okolinom

Adaptibilni virtuleni kriptografski fajl sistem sa korisnikom komunicira pomoću komandne linije i pomoću *stdout* interfejsa. Moguće je kreirati i grafičko okruženje koje služi kao omotač oko navedene arhitekture. Konfiguracija sistema se, osim putem baze podataka, obavlja pomoću sistema konfiguracionih datoteka koje imaju formu sekcija-ključ-vrijednost, kako je to prikazano u listingu 5:

```
 [section]
key = value
```

Listing 30: Format konfiguracione datoteke

Uticaj korisnika na izbor kriptografskih resursa koje operativni sistem može da koristi ostvarivaće se putem konfiguracione datoteke pod nazivom resursi.ini, u kojoj mogu da budu nabrojani svi resursi koje korisnik ima na raspolaganju i koje želi da koristi. Navedeni resursi u konfiguracionoj datoteci moraju da budu jedinstveno identifikovani po nazivu sekcije i numeričkom identifikatoru, nakon čega slijedi lista ključnih atributa i odgovarajućih vrijednosti. Dio navedene liste može da ima zajedničku formu za sve resurse (npr. aktivan/neaktivan, putanja do programa) dok drugi dio može da bude karakterističan za pojedini resurs (npr. veličina bafera).

Izgled jedne sekcije ini datoteke na osnovu koje će se vršiti testiranje dat je u slijedećem listingu:

```
...
...
[OraSunDoc]
id=OraSunDoc
numId=0
active = true
path = "java"
path1 = "-jar"
path2 = "C:\\Users\\bori\\Documents\\NetBeansProjects\\_SunDoc\\dist\\_SunDoc.jar"
path3 = "C:\\Users\\bori\\Documents\\NetBeansProjects\\_SunDoc\\dist\\"
ks=128,192,256
tNums=1
modes=ECB,CBC,CTR,OFB,CFB
loop=5
...
...
```

Listing 31: Sadržaj jedne sekcije inicijalizacone datoteke Resursi.ini

Podaci potrebni za pokretanje pojedinih modula se nalaze u sekcijama koje počinju jedinstvenim nazivom sekcije u okviru srednjih zagrada. Svaka sekcija počinje sa opisnim i numeričkim identifikatorom pojedinog modula, koji moraju biti jedinstveni na nivou sistema. Nakon toga slijedi podatak o tome da li je modul aktivan ili ne, a zatim podaci koji služe za pokretanje različitih procesa. Za pokretanje izvršnih datoteka obično je dovoljna jedna linija, dok je za pokretanje Java programa na pojedinim sistemima potrebno više linija sa podacima. Nakon ovoga slijedi podatak o korišćenom

kriptografskom načinu rada, a potom se navode dužine ključeva koje su implementirane u navedenom modulu. Ako modul može da koristi više niti, izlistavaju se i podaci o brojevima niti ili se navodi broj jedan ako se radi o jednonitnom rješenju. Nakon toga slijede podaci o tome da li je navedenom modulu potreban inicijalizacioni vektor za rad i koliko testova će (ako je pokrenuto testiranje) biti obavljeno u petlji nad ovim modulom.

Kako je već rečeno, podaci iz četiri relacije (Genmodels, Preferencije, DefNominator, ML) mogu biti smješteni u istoimenim konfiguracionim datotekama, kada se koristi forma ključ-vrijednost za zapisivanje podataka.

Razvoj ovakvog modela koji kombinuje jedinstven i jednostavan interfejs za pristup resursima i njihovo mapiranje u velikoj mjeri olakšava particionisanje sistema na module koji se zatim po potrebi mogu uključivati u sistem bez potrebe za njegovim korjenitim izmjenama.

6.3 Apache Commons VFS kao kontejner za adaptibilni virtuelni kriptografski fajl sistem

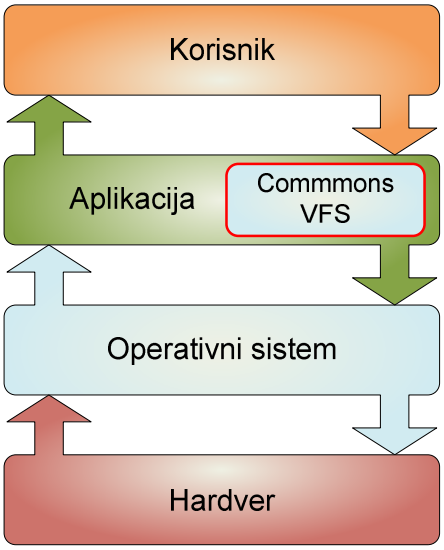
Virtuelni fajl sistem (VFS) predstavlja sloj apstrakcije iznad fajl sistema koji je implementiran u okviru operativnog sistema. Ovakvi fajl sistemi mogu da upravljaju sistemskim pozivima i da dozvole klijentskim aplikacijama da pristupaju različitim nižim fajl sistemima na uniforman način. U većini slučajeva, virtuelni fajl sistemi imaju slojevitu strukturu, pri čemu su nekada smješteni na višem nivou, u korisničkom prostoru kakav je GnomeVFS, a nekada na nižem nivou u prostoru jezgra, kakav je LinuxVFS.

Virtuelni fajl sistem Apache Commons VFS koji je korišćen radi potvrde koncepta u istraživanju nalazi se u korisničkom prostoru i na aplikacionom sloju u okviru operativnog sistema, kao na slici 30.

Ovaj virtuelni fajl sistem namjenjen je da pruži uniforman API (*Application Programming Interface*) za pristup resursima bez obzira da li se oni nalaze na lokalnim diskovima, na FTP serverima ili datotekama sa sadržajem, kao što je ZIP datoteka.

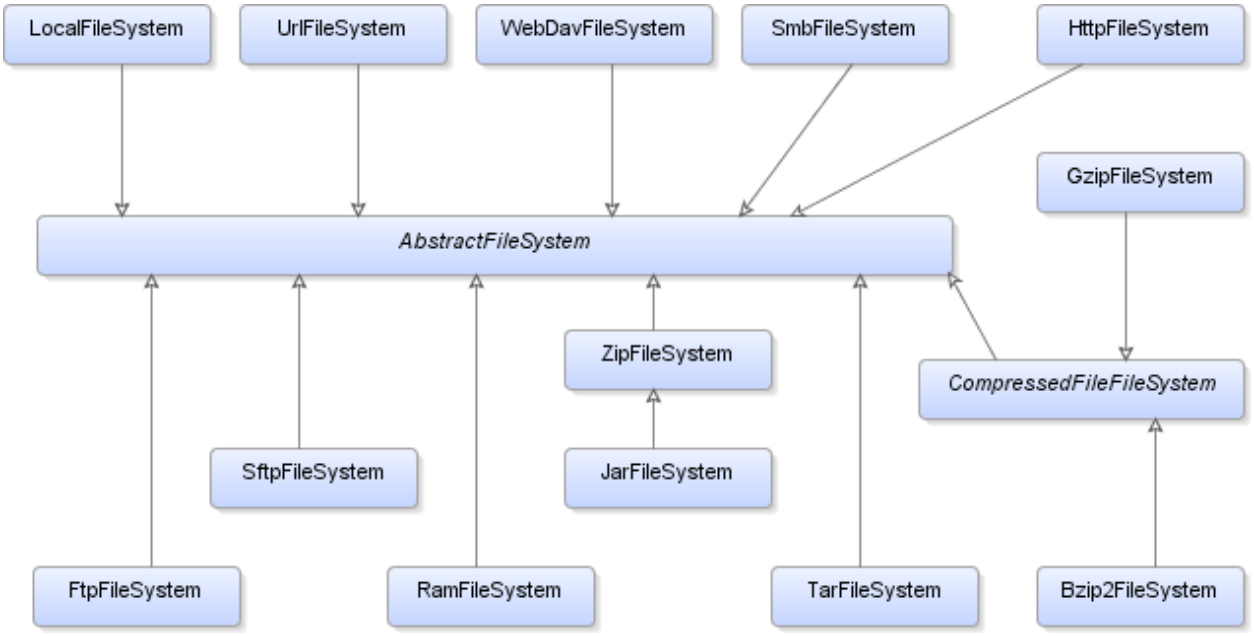
Treba voditi računa o činjenici da pozicija virtuelnog fajl sistema i u okviru njega kriptografskih resursa ne utiče na u istraživanju primjenjeno rješenje, jer se ideja lako može implementirati na bilo kom drugom mjestu, bilo da se radi o sistemu datoteka u korisničkom prostoru, složenom (*stackable*) sistemu, sistemu orjentisanom ka disku (*disk based*) ili sistemu orjenitisanom ka bloku (*block based*).

Commons VFS je *open source* projekat koji se razvija u programskom jeziku Java i koji pruža jedinstven API za pristup različitim fajl sistemima, kao što su ZIP arhiva, HTTP server ili datoteka na lokalnom disku. Ovaj VFS je iskorišćen kao kontejner kojem su dodati brojni moduli koji ga zajedno pretvaraju u adaptibilni virtuelni kriptografski fajl sistem. Ovakav sistem ima mogućnost prilagođenja postojećim kriptografskim resursima u smislu izbora najboljeg i najoptimalnijeg resursa koji mu u trenutnom okruženju stoji na raspolaganju.



Slika 30. Pozicija *Apache Commons* virtuelnog fajl sistema u okviru operativnog sistema

Commons VFS pruža jedinstven Java API za manipulisanje datotekama koji je nadograđen različitim modulima da bi mogao da pruži željenu kriptogafsku adaptibilnost.



Slika 31. Pozicija Commons virtuelnog fajl sistema u odnosu na druge fajl sisteme

U adaptiblni kriptografski virtuelni fajl sistem su sa jedne strane ugrađeni brojni moduli koji proširuju funcionalnost Apache Commons VFS-a, dok je sa druge strane omogućen uniforman način da se u takav sistem kao priključci (*plug-ins*) uključuju posebni kriptografski moduli.

6.4 Kriptografski moduli korišćeni kao kriptografski resursi

Originalna rješenja kriptografskih modula za jednojezgrene i višejezgrene mikroprocesore oslanjaju se na standard FIPS 197 koji opisuje AES algoritam, na modifikovane ideje autora algoritma, originalna proširenja algoritma AES te na određene ideje o korišćenju različitih kriptografskih režima rada (*modes of operation*) radi paralelizacije njegovog izvršenja. Za realizaciju modula koji su učestvovali u eksperimentima korišćeni su Java SE7u51 Developement Kit, NetBeans 7.3, 7.4 i 8.1, Eclipse Indigo 3.7.2., Eclipse Kepler 4.3.1 i Microsoft Visual Studio 2012 C/C++ te NVIDIA Cuda C/C++ kompajleri.

U okviru ovog istraživanja prikazane su osnovne ideje i empirijski rezultati izvršavanja aplikacije EaesG zasnovane na idejama autora algoritma (T tabele). Navedena aplikacija koristi 128, 192 i 256-bitnu enkripciju i u potpunosti poštuje standard definisan dokumentom FIPS-197.

U istraživanju su realizovana i određena proširenja standardom definisanog algoritma AES. Ova proširenja su obuhvatila smanjenje dužina ključa na 64 i 32 bita uz odgovarajuće smanjenje broja rundi na 8 odnosno 7. Takođe su realizovana i proširenja vezana za ojačavanje ovog algoritma 320-bitnom, 384-bitnom, 448-bitnom i 512-bitnom enkripcijom i dekripcijom.

Omotač (*wrapper*) je izveden kao primjer korišćenja kriptografskih biblioteka nezavisnih proizvođača koje imaju implementiran AES algoritam. Prvi omotač je kreiran korišćenjem programskog jezika Java oko Oracle JCE kriptografskog paketa. Ova biblioteka koristi JCE (*Java Cryptography Extension*) arhitekturu koja omogućava da se sigurnosne usluge implementiraju pomoću provajdera koji se ugrađuju u Java sigurnosnu platformu.

Slijedeći kriptografski modul je kreiran korišćenjem Microsoftovog kompajlera Visual C++ 2012. Ova implementacija koristi ubrzanja koja su predstavili autori algoritma a koja se zasnivaju na postojanju T-tabela. I ova implementacija se može pokretati i koristiti kao samostalan program ali i kao program koji sa adaptibilnim kriptografskim virtuelnim fajl sistemom komunicira pomoću soketa i pomoću neimenovanih cijevi.

Slijedeća originalna implementacija realizovana je korišćenjem AES-NI skupa instrukcija, Microsoftovog kompajlera Visual C++ 2012 i specijalne grupe funkcija koje su poznate pod imenom *intrinsics*. Kombinacija *open source* rješenja (biblioteka Botan i Intel White Paper) koja su bila namjenjena za GNU C/C++ odnosno Intel C++ dovela su do realizacije rješenja koje je moguće kompajlirati u programskom jeziku C/C++ u okviru Visual Studija 2010 i Visual Studija 2012 a koje se izvršava u ECB režimu rada. Ova implementacija se u radu oslanja na postojanje hardverske akceleracije u okviru samog mikroprocesora.

Naredno originalno rješenje modula se oslanja na CUDA arhitekturu i kombinaciju CUDA C kompajlera te kompajlera Visual C++ 2012. Navedeni izvorni kod je napisan u skladu sa sintaksom i preveden uz pomoć CUDA C kompajlera tako da se paralelno izvršava na velikom broju jezgara pri čemu svako jezgro obrađuje po jedan blok podataka (Stanje). Ova implementacija se u radu oslanja na postojanje specifičnog hardverskog sklopa, odnosno CUDA grafičkog procesora.

Veoma važna je i Java implementacija koja demonstrira mogućnost paralelizacije izvršenja algoritma AES pomoću kriptografskih režima rada. Ova implementacija koristi razliku u broju rundi koja postoji u slučaju šifrovanja različitim dužinama ključeva i osobinu pojedinih kriptografskih načina rada, poput OFB moda, kojima je za funkcionisanje potreban samo generator ključeva, i kojima nije potrebna funkcija dešifrovanja za funkcionisanje. Ideja koja stoji u osnovi ove implementacije ima ogroman potencijal i sigurno će naći svoju primjenu u budućnosti sa razvojem hardvera kao i sa razvojem tehnika i mogućnosti paralelnog programiranja. Ova ideja je dovela i do objavljivanja rada (Damjanović and Simić, 2015).

U okviru istraživanja predviđen je i implementiran još jedan kriptografski modul koji se nalazi u sastavu adaptibilnog virtuelnog kriptografskog fajl sistema. Važnost ovog modula je veoma velika za slučaj obrade datoteka manje dužine kada, zbog trajanja međuprocesne komunikacije, modul za koordinaciju izračuna da se ne isplati pozivati ni jedan eksterni kriptografski modul.

6.5 Podsistem za određivanje praga iskoristivosti u odnosu na dužine datoteka

Iako pripada sistemu za trening i kategorizaciju, opis podsistema za određivanje praga iskoristivosti morao je biti opisan tek nakon predstavljanja ostalih podsistema od kojih se sastoji modul za koordinaciju.

Prema arhitekturi sistema koja je prikazana na slici 25, jedan kriptografski modul kojim se može vršiti šifrovanje i dešifrovanje se nalazi u sistemu, dok se ostatak kriptografskih resursa (modula) fizički nalazi van adaptibilnog virtuelnog kriptografskog fajl sistema. Za pokretanje navedenih vanjskih modula te za dobijanje njihovog odziva, troši se određeno vrijeme bez obzira na to koja tehnologija se koristi za komunikaciju. Takođe, određeno vrijeme se mora potrošiti na rad modula selektora, kada se obavlja postupak klasifikacije i nominacije. Vrijeme koje je potrebno modulu selektoru za obavljanje svog zadatka može da se izračuna tek nakon što se provede postupak kreiranja trening podataka, njihove kategorizacije i na osnovu toga postupak kreiranja modela podataka. Tek kada su modeli zapisani u nekoj bazi podataka ili u odgovarajućim datotekama, može se testirati brzina njihovog rada, odnosno brzina rada cijelog sistema za selekciju.

Namjena podsistema za određivanje praga iskoristivosti je određivanje dužine datoteke nakon koje je isplativo, sa aspekta utrošenog vremena, koristiti neki vanjski modul, odnosno obrnuto, da utvrdi koja je to minimalna dužina datoteke nakon koje ne treba koristiti vanjske module za obradu podataka.

Rezultati brzine izvršavanja podsistema za nominaciju, odnosno kompletnog podsistema za selekciju se nalaze u intervalu od deset milisekundi, tako da se efikasno mogu predstaviti pomoću srednje vrijednosti. Ukupno vrijeme koje je potrebno za šifrovanje nekim eksternim kriptografskim modulom može da se podijeli na vrijeme potrebno određivanje (selekciju) najefikasnijeg resursa, vrijeme potrebno za enkripciju i vrijeme potrebno za komunikaciju između modula i sistema. Ukupno vrijeme potrebno za šifrovanje internim modulom manje je za vrijeme potrebno za određivanje (selekciju)

najefikasnijeg resursa i za vrijeme potrebno za komunikaciju između modula i sistema, kao u jednačini 70.

ExtEncT = Ts + Tc + Te (70)

IntEncT = Te

Gdje je:

ExtEncT - ukupno vrijeme za enkripciju nekim eksternim resursom,

IntEncT - ukupno vrijeme za enkripciju internim resursom,

Ts - prosječno vrijeme potrebno za selekciju najefikasnijeg resursa,

Tc - vrijeme potrebno za komunikaciju između eksternog kriptog. modula i sistema,

Te - vrijeme potrebno za enkripciju.

U toku kreiranja trening podataka mjeri se vrijeme Tec, odnosno ukupno vrijeme koje je potrebno za obradu podataka i komunikaciju između sistema i vanjskog kriptografskog modula. Da bi dobili ukupno vrijeme potrebno za obradu nekim eksternim kriptografskim resursom, vrijeme Te se mora uvećati za prosječno vrijeme potrebno za selekciju najefikasnijeg resursa Ts i za vrijeme potrebno za komunikaciju Tc.

Funkcionisanje ovog podsistema moguće je opisati pomoću listinga 32.

```
dat <- config(min_dat)
delta <- config(delta)
maxdat <- config(max_dat)
key <- config(key)
selT = getSelectionTime()
while (dat < maxdat)
internalT = minM(M0(dat, buf1...bufN))
exernalT  = min( minM(M1, dat, b1,..bn)),..,
                 minM(MN, dat, b1,..bn)) )+selT
if (exernalT < internalT)
    break
inc(dat, delta)
end while

gdje je:
min_dat - dužina datoteke od koje će početi proces traženja,
dat – trenutna dužina datoteke
internalT – vrijeme dobijeno internim modulom M0
externalT – minimalno vrijeme obrade dobijeno kada se posmatraju svi eksterni
moduli (M1..MN)
minM(MN, dat, b1,..bn)) – funkcija koja traži minimum procjena vremena
izvršenja modela Mn kada se koriste sve dužine bafera koje su korišćene pri
treniranju modela. Funkcija vraća procijenjeno vrijeme koje je potrebno za
enkripciju/dekripciju
selT – vrijeme potrebno za selekciju
delta - vrijednost za koju će se uvećavati veličina datoteke
max_dat - veličina datoteke nakon koje treba prestati sa izračunavanjem ako
nije pronađena minimalna datoteka
key - dužina ključa koja se koristi u šifrovanju
```

Listing 32: Pseudo kod - modul za određivanje praga iskoristivosti

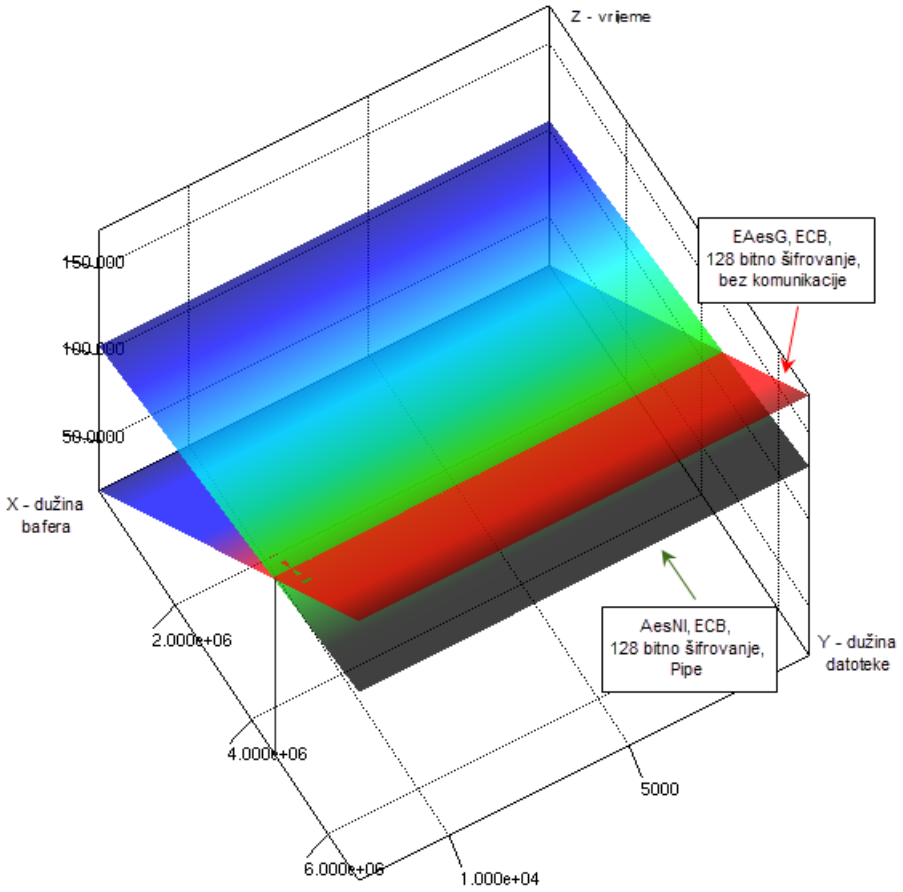
Algoritam funkcioniše tako što za određenu dužinu datoteke traži minimum predviđenih izvršenja svih eksternih modula i poredi ga sa predviđanjem izvršenja internog kriptografskog modula. Pomenuti minimum izvršenja eksternih modula se dobija predviđanjem pomoću generisanih modela nad navedenom dužinom datoteke i svim dužinama bafera koje su se koristile u kreiranju trening podataka. Izvršavanje ovog algoritma se završava kada se pronađe dužina datoteke za koju je sistem predviđa da će se brže izvršiti pomoću bilo kog eksternog modula nego pomoću internog modula.

Nakon određivanja ove datoteke, njena dužina se, zajedno sa odgovarajućom dužinom ključa, zapisuje u konfiguracionu datoteku ili relaciju baze podataka pod imenom Preferencije u formi koja je opisana u poglavlju 6.1.4. Pri slijedećem šifrovanju navedenom dužinom ključa, adaptibilni virtuelni kriptografski fajl sistem neće ni pokušavati da koristi eksterne module ako je datoteka kraća od izračunate, već će pretpostaviti da u datim uslovima najefikasniji interni modul. Tek ako je datoteka duža od navedene, sistem će pokrenuti mehanizam za selekciju najefikasnijeg kriptografskog resursa koji je predstavljen u poglavlju 6.1.2.

Prema prikazanom rješenju, kreirani prediktivni modeli mogu da budu smješteni u nekom sistemu za upravljanje bazama podataka i u fajl sistemu, jer je svaki prediktivni model u postupku klasifikacije jedinstveno određen svojim imenom.

Na testiranim sistemima, kada su se prediktivni (prognostički) modeli za predviđanje brzine smještali i pribavljali pomoću sistema za upravljanje bazama podataka, dobijene se nešto lošije performanse. Mjerenja koja su obavljena prilikom odabira kriptografskog resursa kada se na prvoj testnoj platformi koristio sistem za upravljanje bazama podataka (SUBP) Firebird mogu da se podijele prema nastanku. Povezivanje sa SUBP pomoću JDBC tehnologije u prosjeku je trošilo 115 milisekundi. Prvi poziv biblioteci Weka, odnosno izračunavanje brzine prvog modela trošilo je prosječno 94 milisekunde, a svako slijedeće izračunavanje brzine modela trošilo je prosječno 5 milisekundi (od 0 do 15 milisekundi). U ovom slučaju ukupnom vremenu potrebnom za obradu podataka dodavano je, zavisno od broja testiranih modula, više od 210 milisekundi.

Kada su modeli podataka koji predviđaju brzinu izvršavanja pojedinih resursa bili smješteni fajl sistemu, dobijeni su znatno bolji rezultati. Ovdje je ponovo prvi poziv biblioteci Weka, trošilo prosječno 94 milisekunde, a svako slijedeće izračunavanje brzine modela trošilo je prosječno 5 milisekundi (od 0 do 15 milisekundi), ali se nije trošilo vrijeme za povezivanje na sistem za upravljanje bazama podataka. Obzirom da je biblioteka Weka implementirana korišćenjem programskog jezika Java, navedena razlika u brzini izračunavanja prvog i svih ostalih rezultata vezana je za način izvršavanja Java programa (*Code warmup*, *Class loading* i *Mixed mode*) koji je pominjan u poglavlju 7.1.1.



Slika 32. Poređenje prediktivnog modela internog EaesG(128) i eksternog AesNi(128) modula prva testna platforma

Kada ne bi postojalo kašnjenje koje unosi veza sa SUBP, interni Java modul bi se, za slučaj 128 bitne enkripcije, isplatilo koristiti za datoteke čija je dužina manja od 4.3MB, jer ga nakon toga prestiže eksterni kriptiografski resurs koji se oslanja na AesNI skup instrukcija, kako je prikazano na slici 32. Međutim, kada se unese prosječnih 115ms kašnjenja koja vrši modul za vezu sa SUBP, eksterni modul koji se oslanja na AesNI skup instrukcija postaje brži za datoteke dužine 10MB, kako je prikazano u listingu 33.

```
FileLen:1MB
 interni:41.133543354077766
 externi:105.6062482035348
FileLen:2MB
interni:63.02862900507731
externi:109.2912096479822
...
...
FileLen:10MB
 interni:238.18931421307377
 externi:138.77090120356112
FileLen:11MB
 interni:260.0843998640733
 externi:142.45586264800846
```

Listing 33: Modul za određivanje praga efikasnosti u radu

7 Jednonitni moduli

U toku istraživanja adaptibilne primjene AES algoritma u okvirima modernih operativnih sistema morali su biti generisani realni trening podaci koje sistem može da koristi radi izrade prediktivnih modela koji opisuju ponašanje pojedinih kriptografskih resursa. Takođe su u praksi morali biti provjereni i modeli namjenjeni za međusobnu komunikaciju sistema sa pojedinim kriptografskim resusursima (modulima). Zbog toga je implementiran određen broj kriptografskih modula kakvi su opisani u poglavlju 5.3. Navedeni kriptografski moduli su pisani u programskim jezicima Java, C/C++ i CUDA C. Neka od prikazanih rješenja su implementirana kao potvrda koncepta, neka od njih predstavljaju eksperimente kojima se ispituju mogućnosti proširenja algoritma AES i mogućnosti paralelizacije njegovog izvršenja, dok neka predstavljaju potpuno funkcionalne aplikacije. U nastavku će biti prikazane najvažnije karakteristike pomenutih rješenja.

7.1.1 Jednonitni Java moduli za šifrovanje pomoću AES algoritma

Istraživanja koja su prikazana u ovom poglavlju su već dovela do objavljivanja niza naučnih radova. Jednonitne implementacije algoritma AES bazirane su na istraživanju koje je provedeno na Fakultetu organizacionih nauka u Beogradu tokom izrade Master rada (Damjanović 2010) kao i na istraživanjima koja su predstavljena u publikacijama (Damjanović i Simić, InfoM46, 2013), (Damjanović and Simić, 2011) i (Damjanović and Simić, 2013).

Jednonitni Java modul se bazira na idejama autora algoritma (Daemen and Rijmen, 2002) i implementacijama dr Briana Gladmana (Gladman, 2007), a u svom radu koristi osam tabela sa unaprijed pripremljenim međurezultatima, koje su poznate pod imemom T tabele. Navedena implementacija je predstavljena u (Damjanović and Simić, 2011), gdje su se njene performanse empirijski poredile sa performansama implementacija velikih i poznatih proizvođača softvera, kao što je Oracle, Bouncy Castle, Cryptix i FlexiProvider. Kada je u pitanju metodologija mjerenja performansi Java programa, postoje dvije kategorije mjerenja: mikro-mjerenja (*micro-benchmarks*) i makro-mjerenja (*macro-benchmarks*) (Boyer1, 2008) i (Boyer2, 2008). Mikro-mjerenja su fokusirana na jedan određeni aspekat sistema, odnosno performanse primitivnih operacija podržane od strane osnovne platforme. Pri izvještavanju se obično koristi aritmetička sredina. Pri pokretanju makro-mjerenja (*macro-benchmarks*) programu se daju ulazni podaci, a zatim se mjeri kompletno vrijeme izvršenja programa.

Prema (Georges et al. 2007) i (Boyer1, 2008), većina Java implementacija ima prilično komplikovan životni ciklus kada su u pitanju performanse. Generalno, inicijalne performanse su nešto sporije, a zatim se veoma brzo popravljaju dok ne dosegnu stabilan nivo. Postoji nekoliko faktora koji utiču na ovaj proces. Java virtuelna mašina obično učitava klase pri prvom izvršenju programa. Zbog toga, prvi poziv nekog programa obično uključuje učitavanje svih klasa koje on koristi, operacije se diskom, parsiranje i verifikaciju koda. Uticaj učitavanja klasa se može smanjiti izvršavanjem zadatka više puta, pod uslovom da zadatak ne uključuje komplikovane grane. Tipično,

Java virtuelna mašina automatski upravlja zauzećem dvije vrste resursa: sakupljanje smeća (*garbage collection*) i finalizacijom objekata. Iz perspektive programera, na ove dvije aktivnosti on ne može da ima nikakav uticaj.

Kako se navodi u (Georges et al. 2007), (Georges et al. 2008) i (Boyer1, 2008), moderne JVM najprije neko vrijeme interpretiraju kod (*warm up time*) prije nego što počnu sa JIT (*Just In Time*) prevođenjem. To znači da će se prava brzina nekog procesa pokazati tek nakon velikog broja poziva odgovarajućem bloku koda. Međutim, posebno je pitanje da li će neki proces ikada dostići veliki broj poziva nekom bloku koda.

Kako je navedeno u poglavlju 4.12, kao testne platfrome korišćeni su računari N5110 sa Core i7-2630QM procesorom i T5400 sa dva Xeon E5410 procesora. Na navedenim platformama izvršene su serije testova nad različitim implementacijama, gdje su vršena mjerenja brzine izvršavanja svake pojedine implementacije nad datotekama dužina 1KB, 2KB, 8KB, 32KB, 128KB, 512KB, 4MB, 8MB, 16MB, 32MB, 64MB, 128MB, 256MB. Svaka pojedina datoteka je po pet puta šifrovana baferima veličine 1024, 2048, 4096, 8192, 16384, 32768, 65536, 98304, 131072, 163840, 196608, 229376 sa različitim implementacijama i različitim dužinama ključeva. Sva mjerenja su bila organizovana isključivo kao makro-mjerenja (*macro-benchmarks*), na način da je svaki rezultat nastao nakon jednog kompletnog izvršenja programa u jednom pozivu virtuelne mašine.

Dobijeni rezultati se grupišu tako da se intuitivno mogu predstaviti kao površi u prostoru, pri čemu se na osama X i Y nalaze dužina datoteke i dužina bafera, a na Z osi se nalazi vrijeme izvršenja svake pojedine implementacije. Na ovaj način svaka površ predstavlja jedan od prediktivnih modela kojim se predviđa brzina izvršenja koja karakteriše neki kriptografski resurs u zavisnosti od većeg broja parametara. Ostali modeli koji nastaju na osnovu jednog resursa zavise od enkripcije ili dekripcije određenom dužinom ključa, načinom rada, brojem niti, načinom komunikacije itd.

7.1.1.1 Originalni modul zasnovan na T-tabelama - EaesG modul

Na osnovu implementacije koja je bazirana na idejama autora algoritma (Daemen and Rijmen, 2002) i implementacijama dr Briana Gladmana (Gladman, 2007) kreiran je kriptografski modul koji može da bude uključen u adaptibilni virtuelni kriptografski fajl sistem. Ovaj modul u svom radu koristi osam tabela sa unaprijed pripremljenim međurezultatima, koje se u literaturi navode pod nazivom T tabele. On može da funkcioniše samostalno, kada se pokreće iz komandne linije, a može i da se pokreće i iz adaptibilnog virtuelnog kriptografskog fajl sistema predviđenim načinima komunikacije koji su prikazani u (4.7), (4.8) i (6.2). Sva izvršena mjerenja su provedena radi prikupljanja trening podataka koji su bili potrebni za izradu prediktivnih modela podataka koji karakterišu ponašanje pojedinih kriptografskih resursa i koji predviđaju njihovo ponašanje u odnosu na date ulazne podatke.

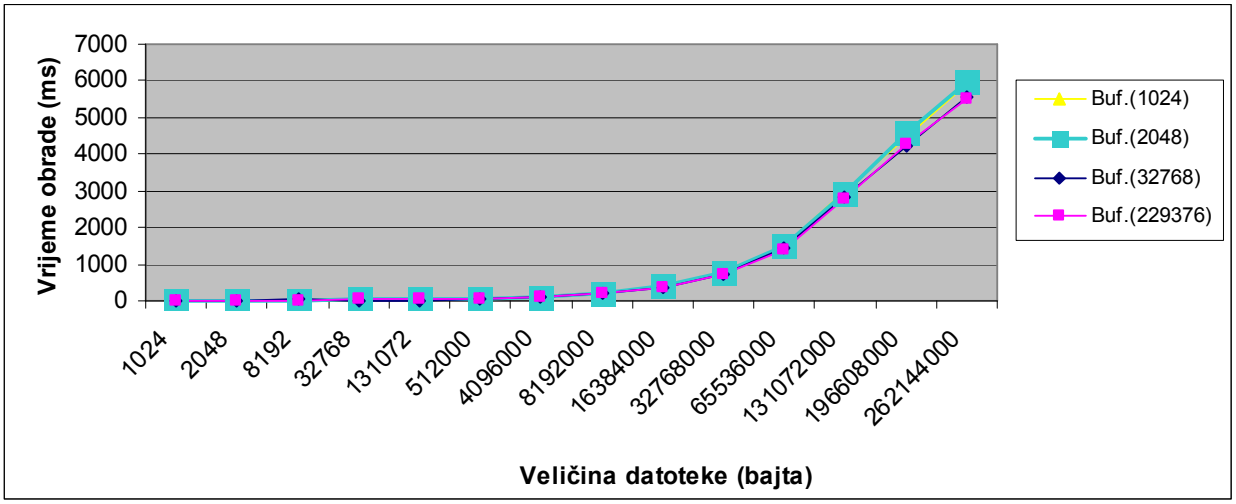
Istovremeno sa prikupljanjem trening podatkaka, vršena su dodatna mjerenja da bi se utvrdilo koliko kašnjenje unosi svaki od odabranih načina komunikacije. Osim mjerenja vremena izvršenja koja se obavljaju u okviru adaptibilnog virtuelnog kriptografskog fajl sistema, modul je mjerio i vraćao kao rezultat i ukupno vrijeme koje mu je potrebno za

šifrovanje/dešifrovanje određene datoteke bez uticaja komunikacije. Ovo vrijeme je korišćeno da se odredi veličina kašnjenja koju unosi odabrani način komunikacije između adaptibinog virtuelnog kriptografskog fajl sistema i kriptografskog modula.

Iako je proveden znatno veći broj mjerenja, u nastavku će biti prikazana dva skupa trening podataka. Podaci za prvi skup dobijeni su mjerenjem vremena potrebnog za šifrovanje kada se mjeri i komunikacija između adaptibinog virtuelnog kriptografskog fajl sistema i modula koja se obavlja putem soketa razmenom maksimalnog skupa poruka, za slučaj 128 bitne ECB enkripcije pomoću jedne niti. Drugi skup je formiran mjerenjima vremena potrebnog za šifrovanje u koje je uključeno i vrijeme potrebno za komunikaciju putem neimenovanih cijevi i minimalnog skupa poruka, za slučaj 128 bitne ECB enkripcije pomoću jedne niti.

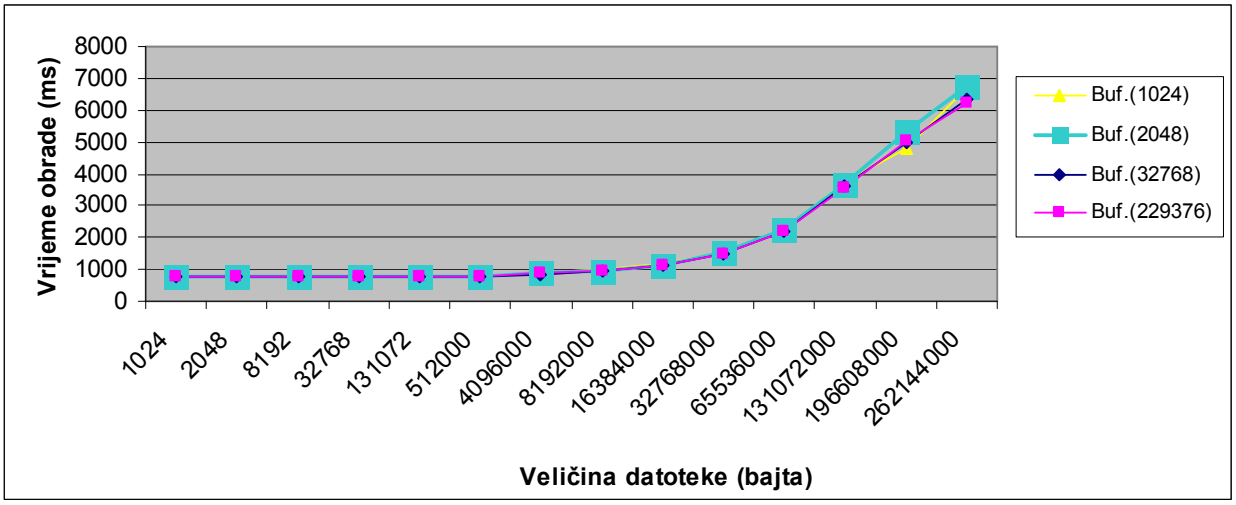
U oba slučaja u mjerenjima su korišćene datoteke dužina 1KB, 2KB, 8KB, 32KB, 128KB, 512KB, 4MB, 8MB, 16MB, 32MB, 64MB, 128MB, 256MB koje su šifrovane korišćenjem kriptografskih bafera dužine 1024, 2048, 4096, 8192, 16384, 32768, 65536, 98304, 131072, 163840, 196608, 229376. Zbog količine podataka, na slikama 33 i 34 prikazan je samo po dio podataka koji u 2D prostoru karakteriše dobijene rezultate mjerenja.

Ovaj modul je u tekstu naveden pod imenom EAesG, sa dodatkom dužine ključa, načina rada, navođenjem broja jezgara i načina komunikacije nakon naziva. Na slici 33 su prikazana izmjerena vremena šifrovanja datoteka različitih dužina za slučaj kada se iz rezultata isključi vrijeme potrebno za komunikaciju između adaptibinog virtuelnog kriptografskog fajl sistema i modula.



Slika 33: Uticaj dužina datoteka na vrijeme potrebno za šifrovanje (EAesGCECB1281, N5110), bez uticaja komunikacije

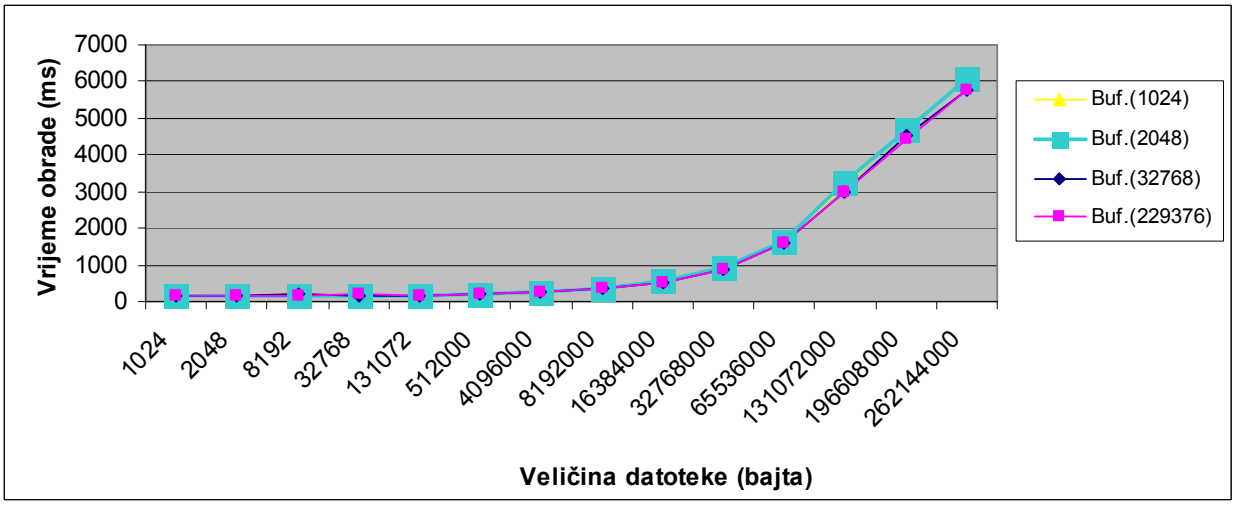
Kada se u adaptibilnom virtuelnom kriptografskom fajl sistemu mjeri ukupno vrijeme koje je potrebno da se dovrši proces šifrovanja, uključujući i pokretanje procesa i razmjenu poruka između VFS-a i kriptografskog modula putem soketa korišćenjem maksimalnog skupa poruka, dobijene vrijednosti su u prosjeku uvećane za 760 milisekundi kako je to prikazano na slici 34.



Slika 34: Uticaj dužina datoteka na vrijeme potrebno za šifrovanje (EAesGCECB1281s, N5110), kom.pomoću soketa uz maksimalan skup poruka

Radi poređenja različitih načina komunikacije izvršena je još jedna serija testova u kojima je mjereno vrijeme koje se potroši na pokretanje procesa i komunikaciju pomoću neimenovanih cijevi korišćenjem minimalnog skupa poruka. Navedeni rezultati, prikazani na slici 35, su primjetno bolji nego u slučaju komunikacije pomoću soketa.

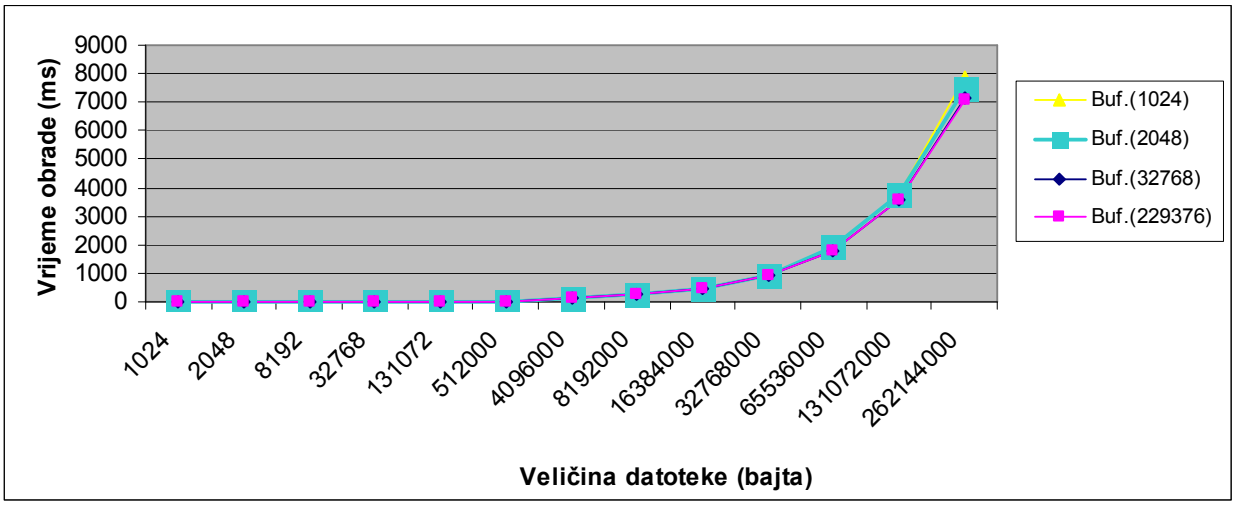
Rezultati mjerenja pokazuju da je šifrovanje uz komunikaciju putem neimenovanih cijevi bilo za u prosjeku 164 milisekunde sporije nego šifrovanje bez uticaja komunikacije. Treba primjetiti da su u toku komunikacije putem soketa nakon startovanja procesa razmjenjene 3 poruke sa različitim XML podacima dok je tokom komunikacije putem neimenovanih cijevi nakon startovanja procesa vraćena samo jedna poruka.



Slika 35: Uticaj dužina datoteka na vrijeme potrebno za šifrovanje (EAesGCECB1281p, N5110), kom.pomoću neimenovanih cjevi uz minimalan skup poruka

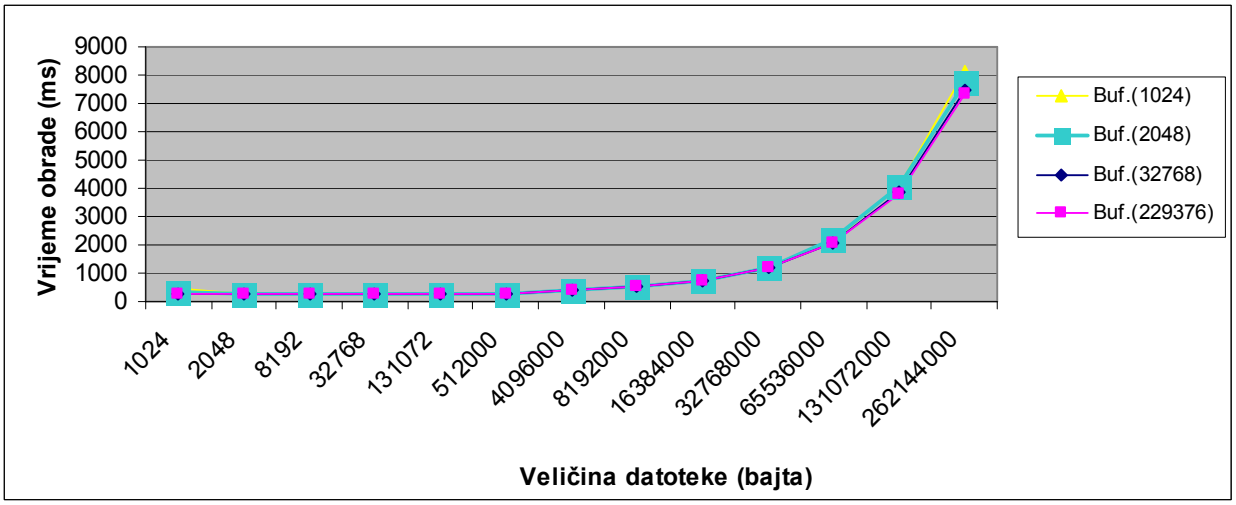
Kao druga testna platforma odabran je opisani računar Dell Precision T5400 sa dva procesora i ukupno 8 jezgara (oznaka T5400). I na ovoj platformi je najprije izmjereno

ukupno vrijeme koje je potrebno za šifrovanje/dešifrovanje određene datoteke bez uticaja komunikacije.



Slika 36: Uticaj dužina datoteka na vrijeme potrebno za šifrovanje (EAesGCECB1281, T5400), bez uticaja komunikacije

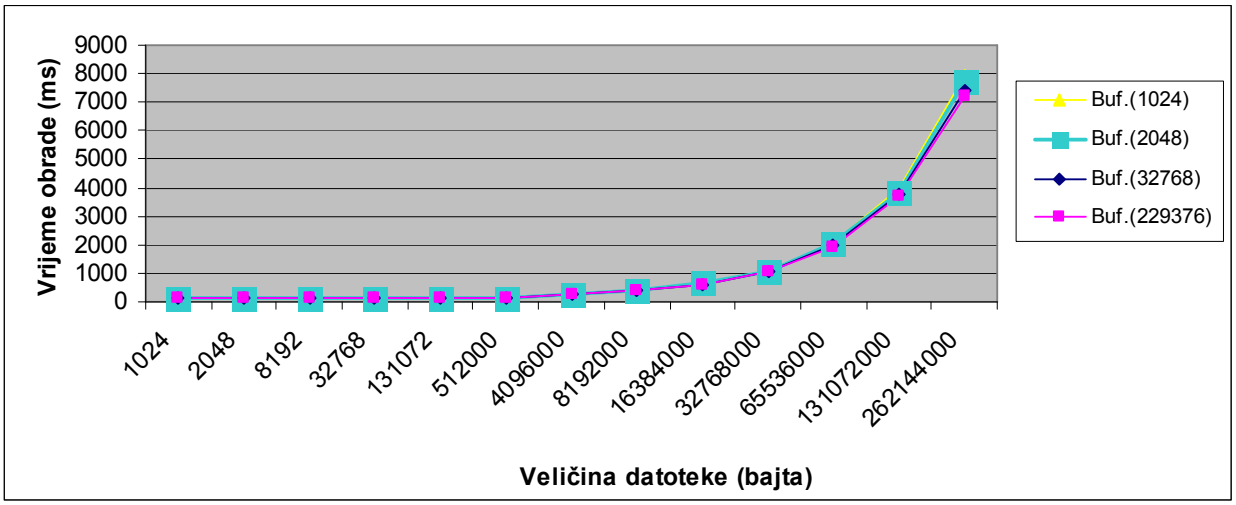
Kada se u adaptibilnom virtuelnom kriptografskom fajl sistemu mjeri ukupno vrijeme koje je potrebno da se dovrši proces šifrovanja, uključujući i pokretanje procesa i razmjenu poruka između VKFS-a i kriptografskog modula pomoću soketa korišćenjem maksimalnog skupa poruka, dobijene vrijednosti su u prosjeku uvećane za iznos od 273 milisekunde. U slučaju druge testne platforme, kašnjenje koje unosi komunikacija putem soketa je višestruko kraće nego u slučaju prve testne platforme, kako je vidljivo sa slike 37.



Slika 37: Uticaj dužina datoteka na vrijeme potrebno za šifrovanje (EAesGCECB1281s, T5400), kom.pomoću soketa uz maksimalan skup poruka

Pri mjerenju vremena koje je potrebno za šifrovanje kada se poruke razmjenjuju pomoću neimenovanih cijevi korišćenjem minimalnog skupa poruka, dobijeni rezultati su

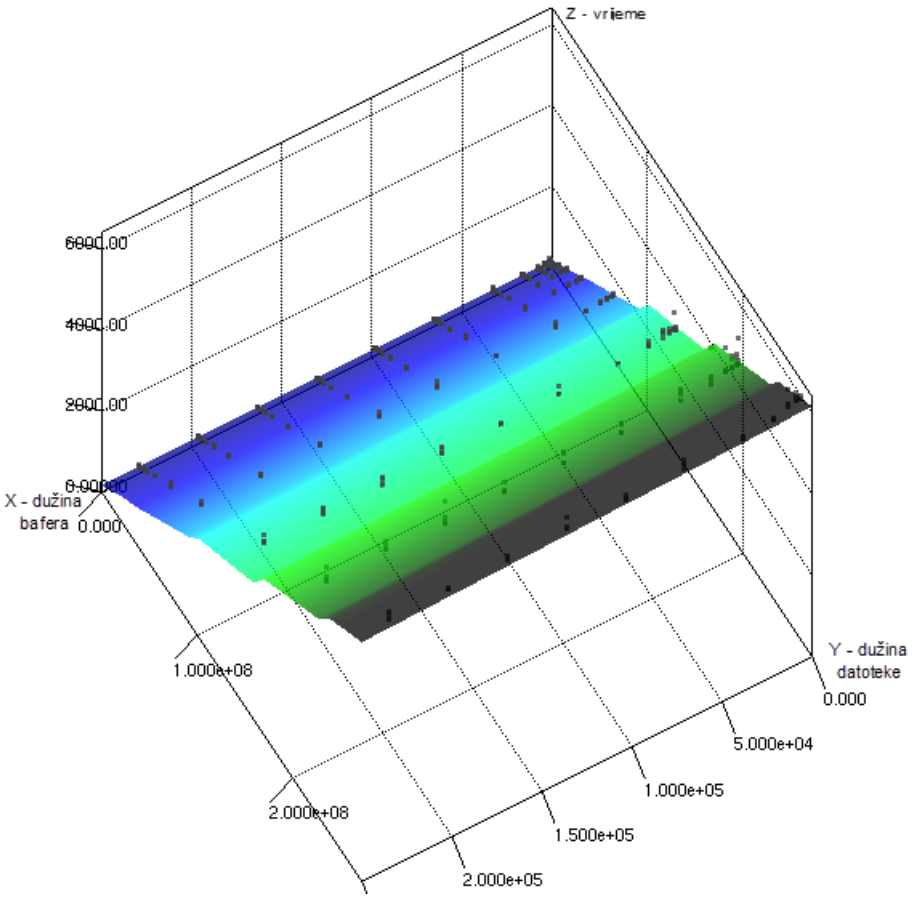
za u prosjeku 129 milisekundi slabiji nego postignuti rezultati bez uticaja komunikacije, kako je prikazano na slici 38.



Slika 38: Uticaj dužina datoteka na vrijeme potrebno za šifrovanje (EAesGCECB1281p, T5400), kom.pomoću neimenovanih cijevi uz minimalan skup poruka

Na prvoj testnoj platformi (N5110) postignuti su mnogo bolji rezultati nego na drugoj testnoj platformi (T5400) zahvaljujući novijem i bržem procesoru (Core i7-2630QM prema Xeon E5410) i RAM memoriji (DDR3 SDRAM PC3-10666 na 666MHz prema DDR2 SDRAM PC2-5300 na 333MHz). Međutim, na prvoj testnoj platformi se javlja znatno veće prosječno kašnjenje (760 milisekundi) ako se radi o komunikaciji putem soketa u odnosu na 273 milisekunde na drugoj testnoj platformi.

Na slici 39 dat je grafički prikaz iz kojega je moguće vidjeti kako se kreirani prediktivni (prognostički) model EAesGCECB1281p koji koristi neimenovane cijevi uklapa u trening podatke dobijene na prvoj testnoj platformi (N5110). Vrijeme izvršenja je prikazano na Z osi, veličina datoteke na Y osi, a dužine bafera su prikazane na X osi. Svaka interpolirana tačka predstavlja procjenu vrijednosti vremena koje će biti potrebno za šifrovanje datoteke određene dužine (Y osa) pomoću internog bafera određene dužine (X osa), onako kako to predviđa M5' metoda mašinskog učenja na osnovu kreiranog pediktivnog modela.



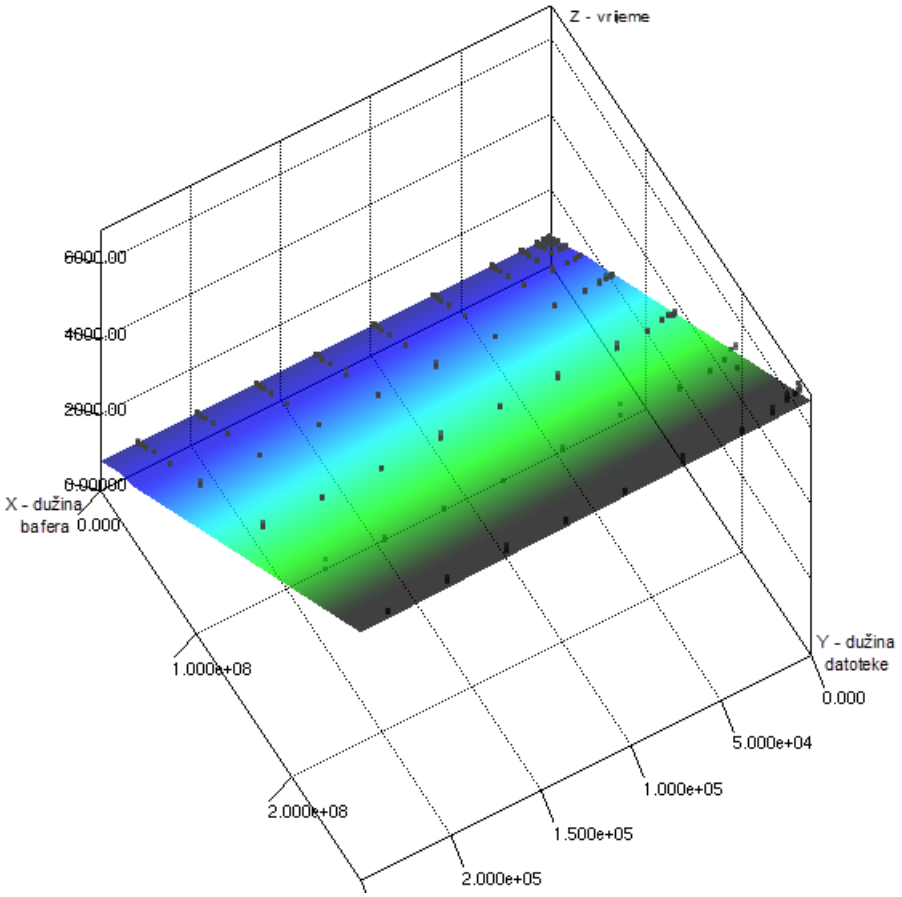
Slika 39: Vizuelni prikaz slaganja trening podataka i kreiranog EAesGCECB1281p prediktivnog modela (N5110), trening podaci

U listingu 34 prikazani su podaci koji govore da navedeni prediktivni (prognostički) model ima vrlo visok stepen slaganja sa pojavom koju opisuje (koef.korelacije = 0.9997) te da ima vrlo malu relativnu standardnu (1.6535%) i relativnu kvadratnu (2.3584%) grešku.

```
Koef.korelacije                    0.9997
Srednja apsolutna greška          23.3488
Srednja kvadratna greška          42.2753
Relativna apsolutna greška         1.6535%
Relativna kvadratna greška         2.3584%
Broj instanci                    168
```

Listing 34: Koef.korelacije, srednja apsolutna i srednja kvadratna greška, te relativna apsolutna i relativna kvadratna greška prediktivnog modela EAesGCECB1281p

Na slici 40 i listingu 35 predstavljen je prediktivni model koji koristi isti kriptografski resurs (EAesG) za šifrovanje (C) pomoću ECB moda (ECB), koristeći 128 bitni ključ (128), jedno jezgro (1) i komunikaciju putem soketa (s). I ovaj model ima veoma veliki koeficient korelacije (0.9995) sa malom relativnom standardnom (2.1781%) i relativnom kvadratnom (3.3064%) greškom.

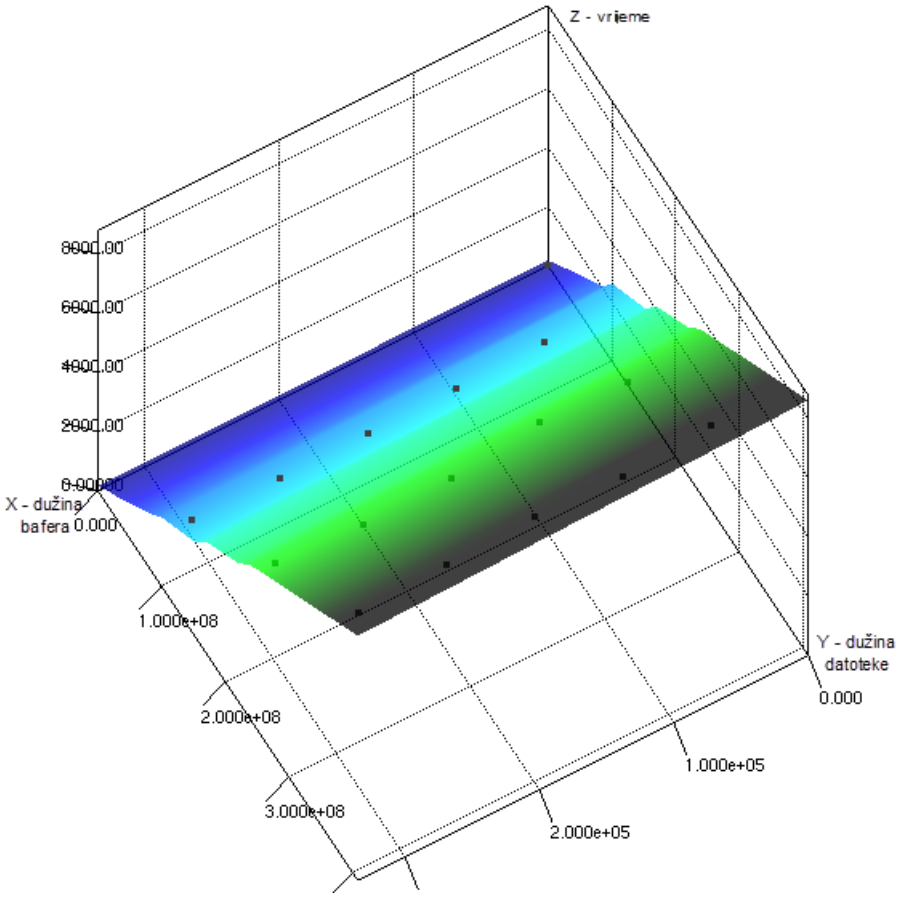


Slika 40: Vizuelni prikaz slaganja trening podataka i kreiranog EAesGCECB1281s prediktivnog modela (N5110)

```
Koef.korelacije                    0.9995
Srednja apsolutna greška          30.3076
Srednja kvadratna greška          58.4784
Relativna apsolutna greška         2.1781%
Relativna kvadratna greška         3.3064%
Broj instanci                    168
```

Listing 35: Koef.korelacije, srednja apsolutna i srednja kvadratna greška, te relativna apsolutna i relativna kvadratna greška prediktivnog modela EAesGCECB1281s, trening podaci

Da bi se mogla provjeriti tačnost predviđanja ovakvog prediktivnog modela, urađena je još jedna serija mjerenja sa probnim ulaznim podacima koji nisu učestvovali u ranijem kreiranju prediktivnog modela podataka EAesGCECB1281p. Dio mjerenja je obavljen za datoteku veću od najveće prethodno šifrovane (393216000 bajta) i kriptografski bafer veći od najvećeg prethodno korišćenog (327680 bajta).

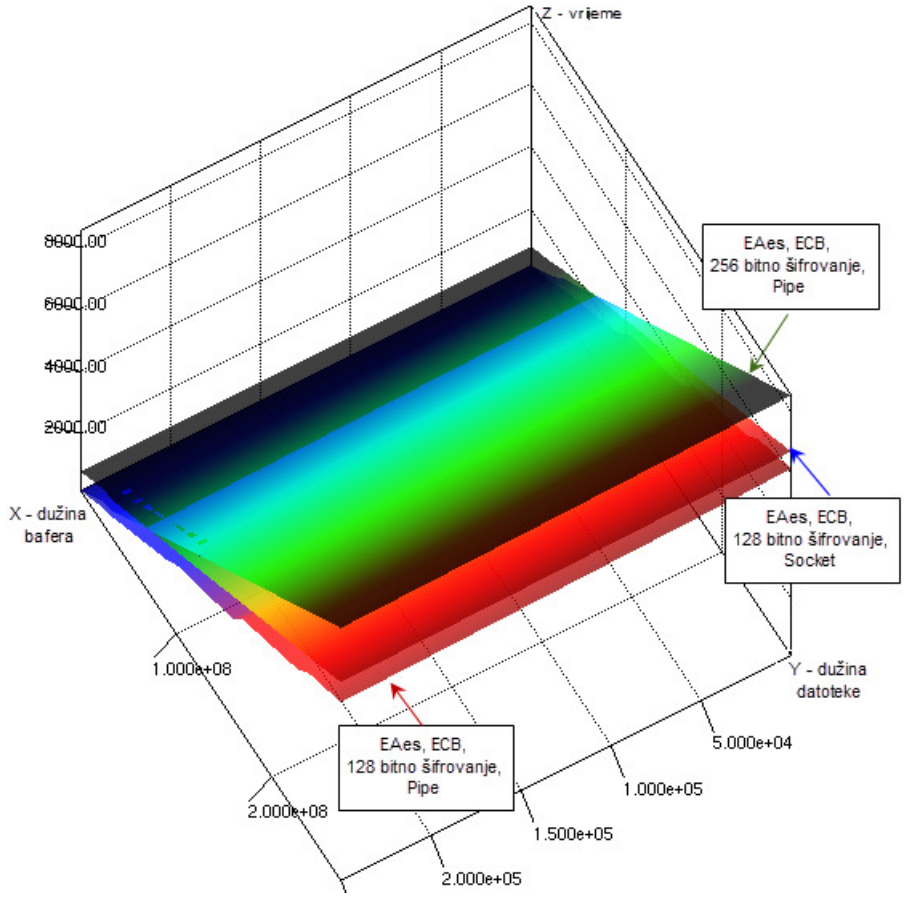


Slika 41: Vizuelni prikaz slaganja probnih podataka i kreiranog EAesGCECB1281p prediktivnog modela (N5110)

```
Koef.korelacije                  0.9984
Srednja apsolutna greška       223.5119
Srednja kvadratna greška       285.2728
Relativna apsolutna greška       4.9243%
Relativna kvadratna greška       5.6226%
Broj instanci                  15
```

Listing 36: Koef.korelacije, srednja apsolutna i srednja kvadratna greška, te relativna apsolutna i relativna kvadratna greška prediktivnog modela EAesGCECB1281p, probni podaci

Usprkos malom skupu trening podataka na osnovu kojih je nastao, prediktivni model podataka EAesGCECB1281p zadržava vrlo visok koeficient korelacije i malu relativnu apsolutnu i relativnu kvadratnu grešku u odnosu na probne podatke, kako je vidljivo sa slike 41 i listinga 36. Treba imati u vidu da je model kreiran na osnovu trening podataka od samo 168 tačaka, a da je predviđao brzinu šifrovanja za datoteke dužine do 393216000 bajta i kriptografske bafere dužine 327680 bajta.



Slika 42: Prediktivni (prognostički) modeli nastali od kriptografskog resursa (modula) EAesG (N5110) – vizuelni prikaz. Treba imati u vidu da je na slici prikazan samo dio modela koje je moguće proizvesti iz ovog modula.

U poglavlju 6.1.1 i na slici 26, prikazano je kako određeni broj parametara putem klasifikacije utiče na kreiranje prediktivnog modela. Zadnja dva ulazna parametra (datoteka i kriptografski bafer) zaokružuju proces kreiranja pojedinih prediktivnih modela na osnovu trening podataka u formi uređene trojke (datoteka, kriptografski bafer, vrijeme). Na ovaj način se iz jednog kriptografskog resursa (modula) proizvodi više prediktivnih modela. Konkretno, na osnovu prethodno prikazanih mjerenja (slike 25 i 35) za prvu testnu platformu nastaju dva modela: EAesGCECB1281s i EAesGCECB1281p. Treniranjem ovog kriptografskog resursa korišćenjem ključeva različitih dužina, različitih kriptografskih načina rada (*modes of operation*) dobija se još veći broj prediktivnih modela. Na slici 42 su prikazani prethodno opisani modeli EAesGCECB1281s i EAesGCECB1281p kojima je dodan model koji predviđa brzinu 256 bitnog šifrovanja jednom niti u ECB modu kada se komunikacija obavlja putem neimenovanih cijevi (EAesGCECB2561p). Ovdje je, osim uticaja dužine datoteke i bafera, grafički iskazan i uticaj dužine ključa i načina komunikacije na brzinu obrade podataka.

Na osnovu jednonitnog kriptografskog resursa EaesG korišćenjem samo jednog (ECB) moda je, uz pomoć klasifikacije, nastalo 12 modela:

$$N = cip\_inv * modes * keys * threads * comm = 2 * 1 * 3 * 1 * 2 = 12 \quad (71)$$

Prikazivanje i poređenje svih navedenih modela u ovom istraživanju nije značajno u smislu dobijanja relevantnih rezultata jer se, za jedan kriptografski resurs (modul), brzine šifrovanja npr. različitim dužinama ključa, različitim kriptografskim načinima rada (*modes of operation*) ili različitim brojem niti ne mogu porediti. Ono što može da se poredi jesu brzine šifrovanja koje pripadaju istim klasama (ista dužina ključa, način rada, broj niti itd.).

U istraživanju su prikupljeni podaci i kreirani odgovarajući modeli za veliki broj kriptografskih resursa na dvije testne platforme (N5110 i T5400). Od navedene dvije platforme, starija i sporija T5400 platforma pokazuje lošije rezultate u svim mjerenjima softverskih kriptografskih resursa. Interesantna karakteristika ove platforme je da je na njoj izmjerena brzina komunikacije putem soketa u prosjeku veća, odnosno de je kašnjenje (273 u odnosu na 760 milisekundi) bitno manje u odnosu na prvu N5110 testnu platformu. Neki od razloga zbog kojih se na različitim Windws sistemima dobijaju različite brzine soketa navedeni su u (Microsoft Corp. KB1, 2013). Osim ove činjenice, poređenje rezultata koji su dobijeni mjerenjima softverskih modula na navedene dvije platforme nije značajno za istraživanje, jer su rezultati dobijeni na prvoj testnoj platformi dovoljni za klasifikaciju i deskripciju kako pojedinih modela podataka koji nastaju na osnovu empirijskih mjerenja, tako i za opis ukupnog modela kojim se predstavlja struktura i način funkcionisanja adaptibilnog virtuelnog kriptografskog fajl sistema.

7.1.1.2 Java moduli koji se oslanjaju na third party kriptografske biblioteke

Danas u svijetu postoji veći broj kriptografskih biblioteka poput Nettle, Crypto++, GNU Crypto, OpenSSL, Oracle/Sun, Bouncy Castle, FlexiProvider, Cryptix i mnogih drugih. Ovakve biblioteke su dizajnirane tako da se pozivima odgovarajućih funkcija mogu jednostavno ugraditi u bilo koju aplikaciju. Da bi fajl sistem kao reprezent operativnog sistema mogao da pristupa ovim resursima, mora biti kreirana aplikacija koja kao posrednik ili omotač standardizovane zahtjeve sistema datoteka transformiše u odgovarajuće pozive funkcijama neke biblioteke i koja sistemu vraća standardizovane odgovore.

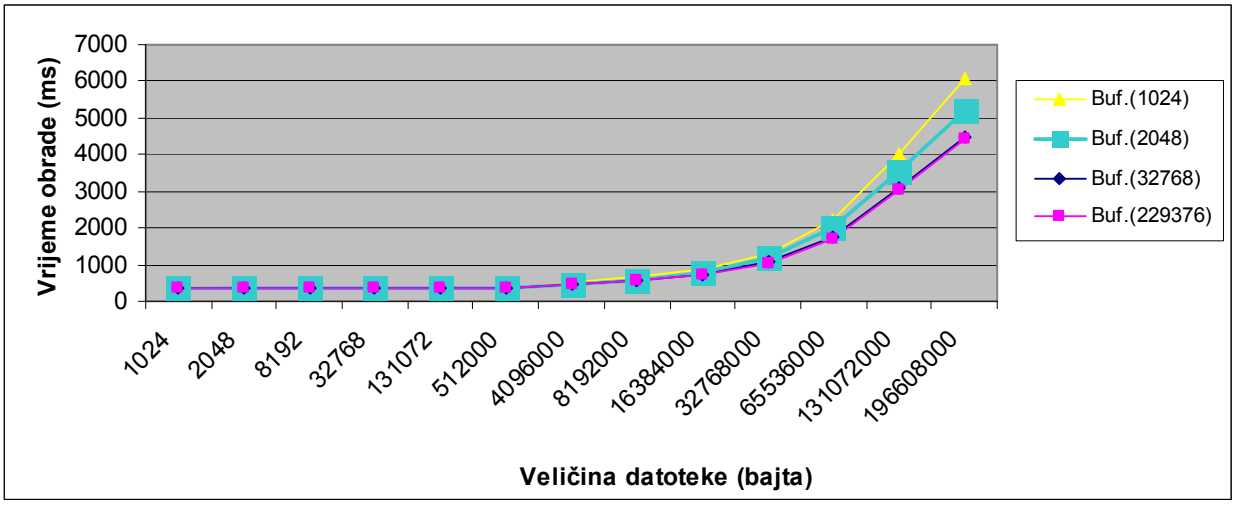
Navedene kriptografske biblioteke su rigorozno testirane u odnosu na ispravnost njihovog rada, dok se optimizaciji njihovog koda kada su u pitanju performanse i utrošak memorije poklanja najveća pažnja. Neke od navedenih biblioteka i njihove performanse su bile predmet istraživanja na osnovu kojih je objavljen dio naučnih radova. Tako je istraživanje performansi implementacija AES algoritma u okviru Java kriptografskih biblioteka pod Linux operativnim sistemom (Damjanović and Simić, 2013) obuhvatilo Oracle/Sun, Bouncy Castle, FlexiProvider i Cryptix kriptografske pakete. Svi navedeni paketi se u svom radu oslanjaju na Java Cryptography Extension (Hook, 2005) bez korišćenja AES-NI skupa instrukcija. U toku navedenog istraživanja performansi AES algoritma razvijene su aplikacije koje se u svom radu oslanjaju na kriptografske pakete Oracle/Sun, Bouncy Castle, FlexiProvider i Cryptix. Izvorni kod navedenih aplikacija se jednostavno može modifikovati tako da se one pretvore u kriptografske module virtuelnog operativnog sistema koji služe kao omotač (*wrapper*) oko različitih implementacija algoritma AES, kako je to navedeno u poglavlju 4.2.3. U nastavku je

predstavljen Java SunJCE provajder koji se standardno isporučuje u okviru instalacije Java JRE i Java JDK paketa.

7.1.1.3 Oracle SunJCE modul

U svrhu testiranja kreiran je modul koji je koristio Oracle-ov JCE provajder koji se identifikuje nazivom SunJCE 1.7. Ovaj kriptografski resurs (modul) je implementiran kao omotač (*wrapper*) koji će biti korišćen u okviru adaptibilnog virtuelnog kriptografskog fajl sistema. Ovaj modul je kreiran na takav način da može da se pokreće iz adaptibilnog virtuelnog kriptografskog fajl sistema predviđenim načinima komunikacije koji su prikazani u (4.7), (4.8) i (6.2), ali može da funkcioniše i samostalno, kada se pokreće iz komandne linije. Modul u toku izvršenja mjeri ukupno vrijeme koje mu je potrebno za šifrovanje određene datoteke, gdje je uključena inicijalizacija, UI operacije i vrijeme potrebno za obradu podataka. U samom fajl sistemu dalje se vrše mjerenja koja, pored vremena eventualno potrebnog za pokretanje i izvršenje nekog procesa, uključuju i vrijeme koje je potrebno da se razmjene poruke između adaptibilnog virtuelnog kriptografskog fajl sistema i modula. Ovaj modul, kao i svi ostali moduli, mora da bude u stanju da razmjenjuje poruke sa VKFS-om kako je to opisano u poglavljima (4.7), (4.8) i (6.2)

Ovaj modul je u daljem tekstu naveden pod imenom OraSun, sa dodatkom dužine ključa, načina rada, navođenjem broja jezgara i načina komunikacije nakon naziva. Na slici 43 su prikazana izmjerena vremena šifrovanja datoteka različitih dužina za slučaj kada se u fajl sistemu mjeri ukupno vrijeme koje je potrebno da se dovrši proces šifrovanja, uključujući i razmjenu poruka između VFS-a i kriptografskog modula putem neimenovanih cijevi.

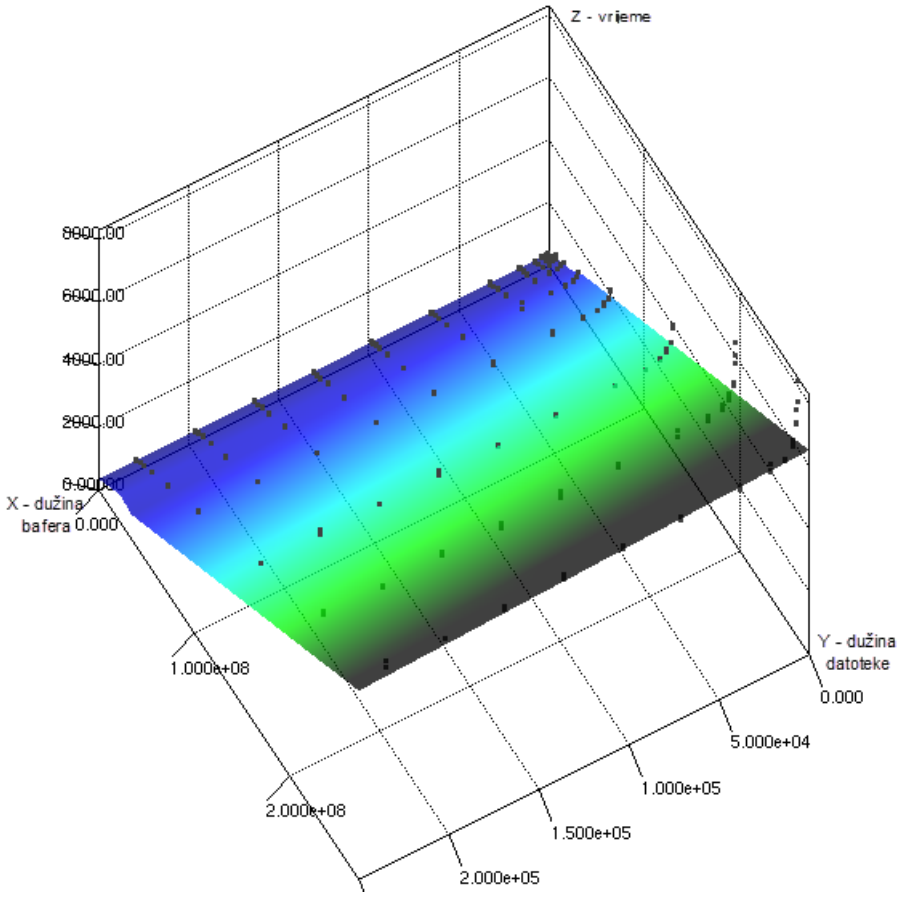


Slika 43: Uticaj dužina datoteka na vrijeme potrebno za šifrovanje (OraSunCECB1281p, N5110), kom.pomoću neimenovanih cjevi uz minimalan skup poruka

Rezultati mjerenja pokazuju da je šifrovanje uz komunikaciju putem neimenovanih cijevi sa minimalnim skupom poruka bilo u prosjeku za 579 milisekundi brže od šifrovanja koje se obavlja pomoću soketa sa maksimalnim skupom poruka, zbog čega ovdje neće

biti prikazani prediktivni modeli nastali kada se komunikacija obavljala putem soketa. Rezultati mjerenja pokazuju i da je šifrovanje uz komunikaciju putem neimenovanih cijevi bilo u prosjeku za 179 milisekundi sporije nego šifrovanje bez uticaja komunikacije i pokretanja procesa.

Na slici 44 dat je grafički prikaz prediktivnog modela koji je nastao na osnovu kriptografskog resursa (OraSun) šifrovanjem (C) pomoću ECB moda (ECB), koristeći 128 bitni ključ (128), jedno jezgro (1) i komunikaciju putem neimenovanih cijevi (p) sa minimalnim skupom poruka. Sa slike je moguće vidjeti kako se kreirani model OraSunCECB1281p koji koristi neimenovane cijevi uklapa u trening podatke. U listingu 37 prikazani su podaci koji govore da navedeni prediktivni model ima visok stepen slaganja sa pojavom koju opisuje (koef.korelacije = 0.9938) te da ima umjerenu relativnu standardnu (5.9798 %) i relativnu kvadratnu (11.0878%) grešku. Prikazani model OraSunCECB1281p zadržava veoma visok stepen slaganja sa trening podacima na osnovu kojih je nastao. Ipak, dobijene nešto veće stope za relativnu apsolutnu i relativnu kvadratnu grešku ukazuju i na mali skup trening podataka kojim je ovaj model kreiran i na veću raspršenost rezultata mjerenja.



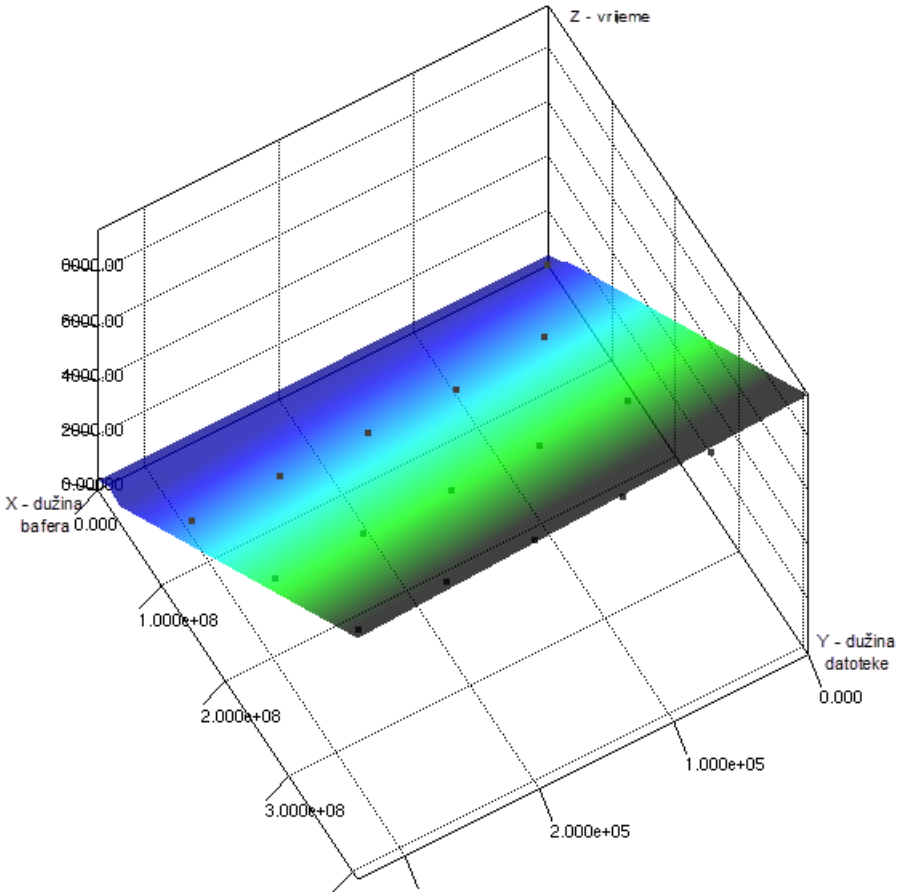
Slika 44: Vizuelni prikaz slaganja trening podataka i kreiranog OraSunCECB1281p prediktivnog modela

Koef.korelacije	0.9938

```
Srednja apsolutna greška                84.5211
Srednja kvadratna greška               200.7942
Relativna apsolutna greška               5.9798 %
Relativna kvadratna greška              11.0878 %
Broj instanci                          168
```

Listing 37: Koef.korelacije, srednja apsolutna i srednja kvadratna greška, te relativna apsolutna i relativna kvadratna greška prediktivnog modela OraSunCECB1281p, trening podaci

Tačnost predviđanja prediktivnog modela OraSunCECB1281p je provjerena dodatnom serijom mjerenja. U prikazanom slučaju, relativna apsolutna i relativna kvadratna greška su manje nego u slučaju evaluacije na osnovu trening podataka, zahvaljujući činjenici da su veća odstupanja od modela uglavnom nastala za male vrijednosti bafera, u blizini Y ose. Kako su probni podaci udaljeni i od X i od Y ose, dobijaju se bolji rezultati nego u slučaju izračunavanja grešaka za trening podatake i model.



Slika 45: Vizuelni prikaz slaganja probnih podataka i kreiranog OraSunCECB1281p prediktivnog modela

```
Koef.korelacije                         0.9967
Srednja apsolutna greška              259.7251
Srednja kvadratna greška              306.4047
Relativna apsolutna greška              5.9551%
```

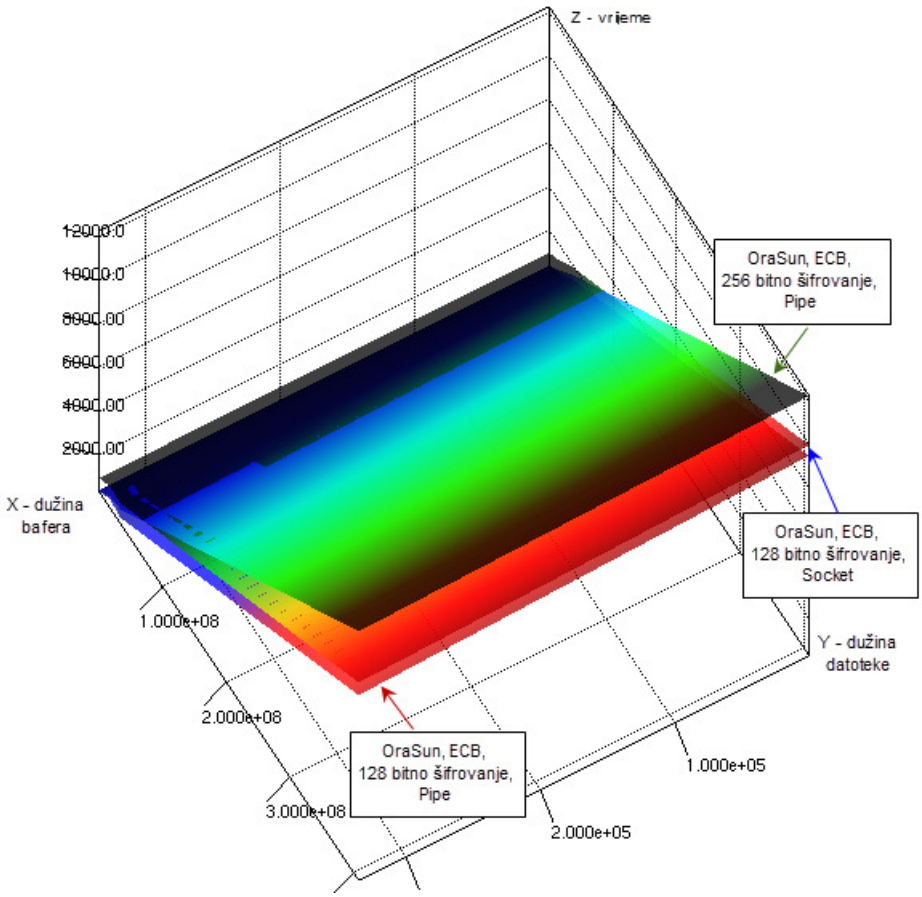
```
Relativna kvadratna greška                6.3207 %
Broj instanci                            15
```

Listing 38: Koef.korelacije, srednja apsolutna i srednja kvadratna greška, te relativna apsolutna i relativna kvadratna greška prediktivnog modela OraSunCECB1281p, probni podaci

Kako je već navedeno, uticaj različitih parametara na brzinu šifrovanja izražava se najprije njihovom klasifikacijom, a zatim kreiranjem prediktivnog modela metodama mašinskog učenja na osnovu trening podataka koji pripadaju takvim klasama.

N = cip_inv * modes * keys * threads * comm = 2 * 7 * 3 * 1 * 2 = 84 (72)

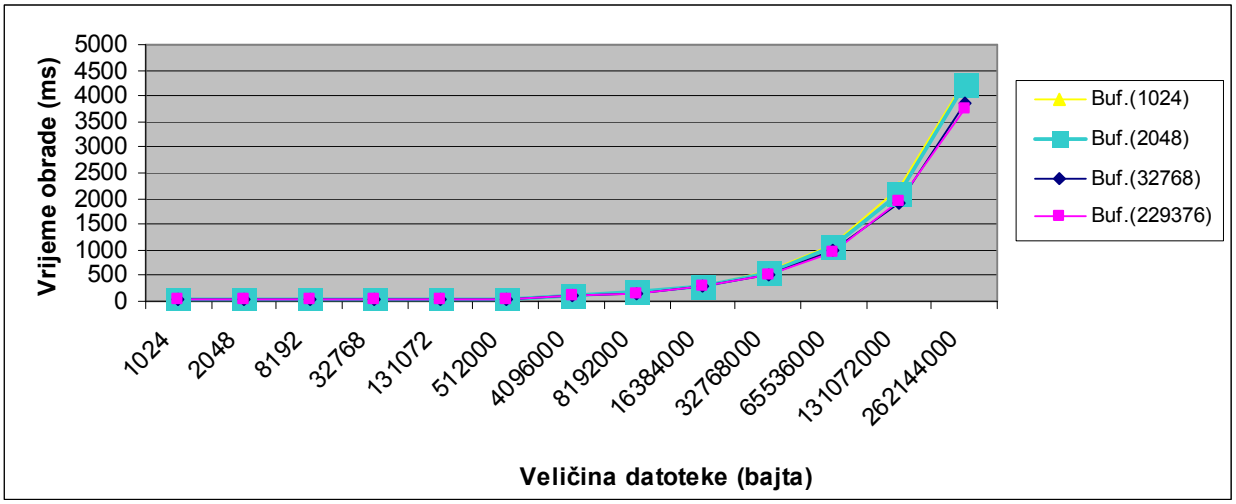
Provajder SunJCE kompanije Oracle za algoritam AES pruža 7 osnovnih odnosno 245 zbirnih načina rada: ECB, CBC, PCBC, CTR, CTS, CFB, CFB8..CFB128, OFB, OFB8..OFB128. Na osnovu njih bi mogli kreirati čak 2*245*3*1*2 =12*245=2940 modela. Na slici 46 prikazano je samo 3 modela od ukupno 245, odnosno 2940 koliko je moguće krirati na osnovu OraSun modula. Kada je u pitanju ovaj modul, treba napomenuti da su provedena mjerenja radi kreiranja trening podataka bila ograničena na četiri klasična kriptografska načina rada i da se kod CFB i OFB koristila samo osnovna varijanta koja za povratnu vezu koristi samo cijeli blok podataka Stanje.



Slika 46: Dio prediktivnih modela koji su nastali od kriptografskog resursa (modula) OraSun..

7.1.2 Modul zasnovan na jednonitnoj C/C++ implementaciji AES algoritma

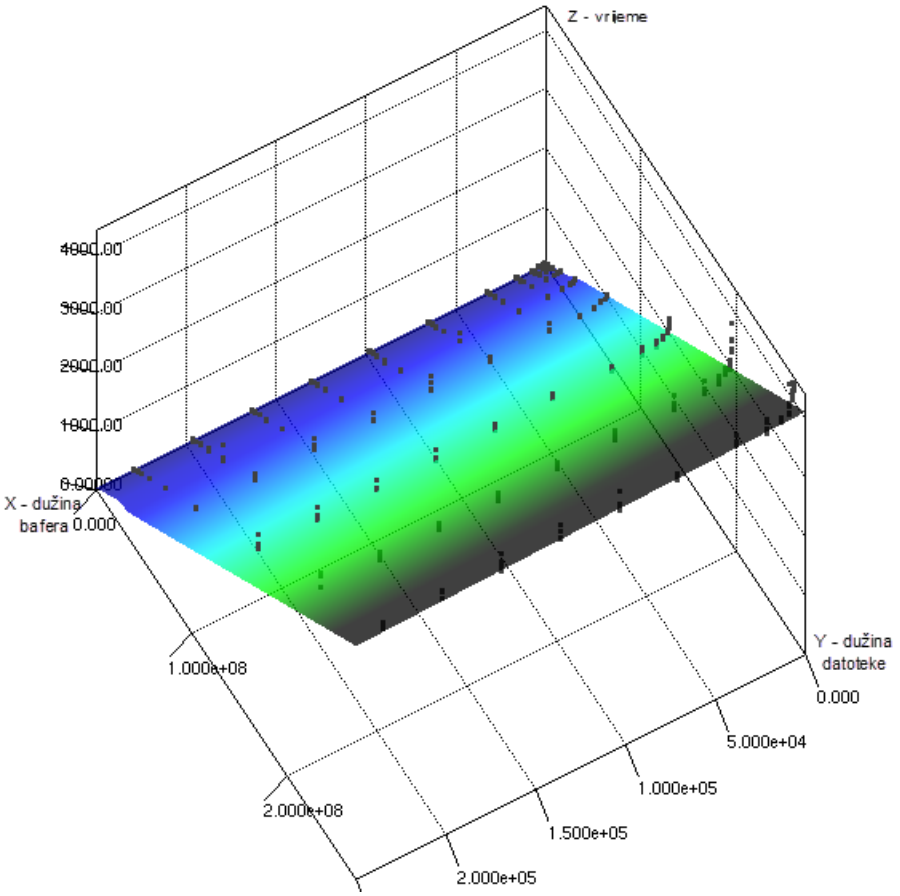
Ovaj modul je takođe baziran na idejama autora algoritma (Daemen and Rijmen, 2002) i implementacijama dr Briana Gladmana (Gladman, 2007). Kreiran je na takav način da može da bude uključen u adaptiblni virtuelni kriptografski fajl sistem. Osim što može da se pokreće iz adaptibilnog virtuelnog kriptografskog fajl sistema predviđenim načinima komunikacije koji su prikazani u (4.7), (4.8) i (6.2), ovaj modul može da se pokreće i iz komandne linije. Za njegovu izradu korišćen je Microsoft Visual C++ kompajler u sastavu Microsoft Visual Studija 2012. Ovaj modul će u daljem tekstu biti naveden pod imenom MSVCAes, sa dodatkom dužine ključa, načina rada, navođenjem broja jezgara i načina komunikacije nakon naziva. Na slici 47 prikazana su mjerenja vremena potrebnog za šifrovanje datoteka različitih dužina za slučaj kada se mjeri ukupno vrijeme koje je potrebno za šifrovanje, uključujući i komunikaciju između VFS-a i kriptografskog resursa MSVCAes putem neimenovanih cijevi.



Slika 47: Uticaj dužina datoteka na vrijeme potrebno za šifrovanje (MSVCAesCECB1281p, N5110), kom.pomoću neimenovanih cjevi uz minimalan skup poruka

Rezultati mjerenja pokazuju i da je šifrovanje uz komunikaciju putem neimenovanih cijevi uz minimalan skup poruka bilo u prosjeku za svega 33 milisekunde sporije nego šifrovanje bez uticaja komunikacije.

Kada se koristi modul MSVCAes za šifrovanje 128 bitnim ključem uz komunikaciju putem neimenovanih cijevi uz minimalan skup poruka, u ECB režimu rada pomoću jedne niti, generisani prediktivni model je prikazan na slici 48. Ova slika prikazuje da postoji veoma visok stepen slaganja modela i pojave koju opisuje, što je potvrđeno i veoma visokim koeficientom korelacije (0.9985) u listingu 39. U listingu je takođe prikazana mala relativna standardna (3.1297%) i relativna kvadratna (5.3858%) greška. Visok stepen slaganja ukazuje na činjenicu da je model dobro uklopljen u podatke. Mali iznosi grešaka istovremeno ukazuju na to da su rezultati mjerenja grupisani oko prediktivnog modela.



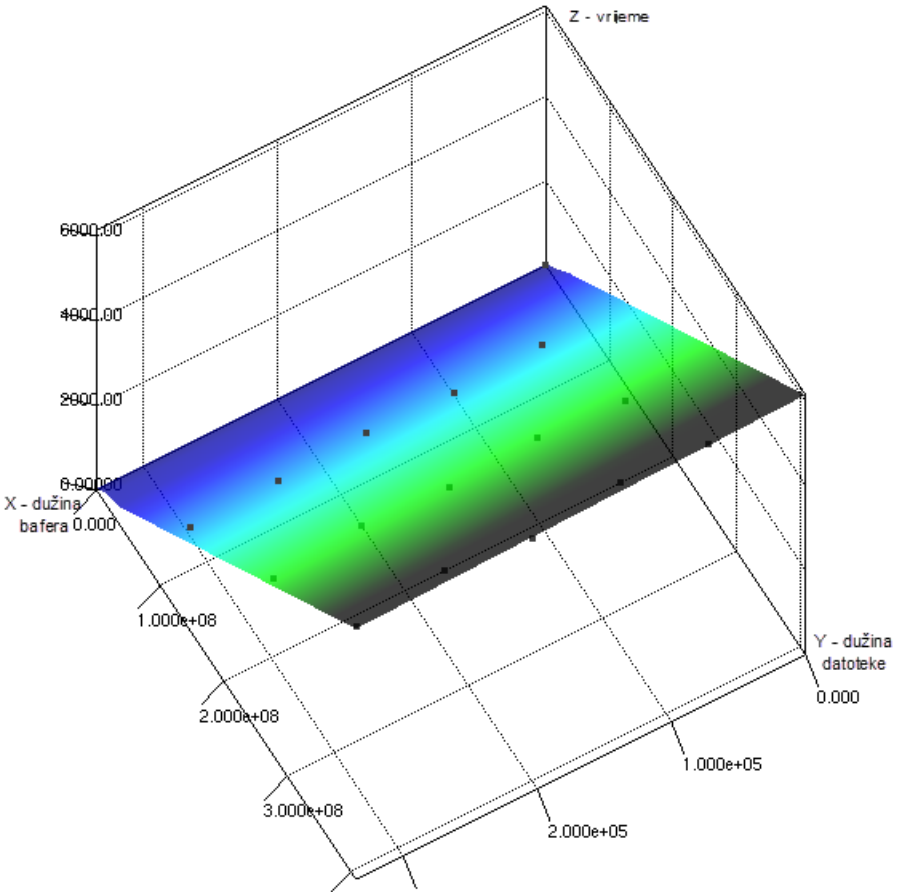
Slika 48: Vizuelni prikaz slaganja trening podataka i kreiranog MSVCAesCECB1281p prediktivnog modela

```
Koef.korelacije                  0.9985
Srednja apsolutna greška        29.8684
Srednja kvadratna greška        65.3686
Relativna apsolutna greška       3.1297%
Relativna kvadratna greška       5.3858%
Broj instanci                  168
```

Listing 39: Koef.korelacije, srednja apsolutna i srednja kvadratna greška, te relativna apsolutna i relativna kvadratna greška modela MSVCAesCECB1281p (N5110), trening podaci

Provjera prikazanog modela ponovo je obavljena pomoću novog skupa mjerenja, koja su različita od trening podataka na osnovu kojih je nastao model. Zbog toga su poređeni rezultati mjerenja modula MSVCAes sa 128 bitnim ECB šifrovanjem uz komunikaciju putem neimenovanih cijevi sa ranije kreiranim modelom MSVCAesCECB1281p.

Dobijeni rezultati prikazani na slici 49 i listingu 40 ukazuju na dobro kreiran model, sa visokim koeficientom korelacije i malim iznosima za relativnu apsolutnu i kvadratnu grešku.



Slika 49: Vizuelni prikaz slaganja probnih podataka i kreiranog MSVCAesCECB1281p prediktivnog modela (N5110)

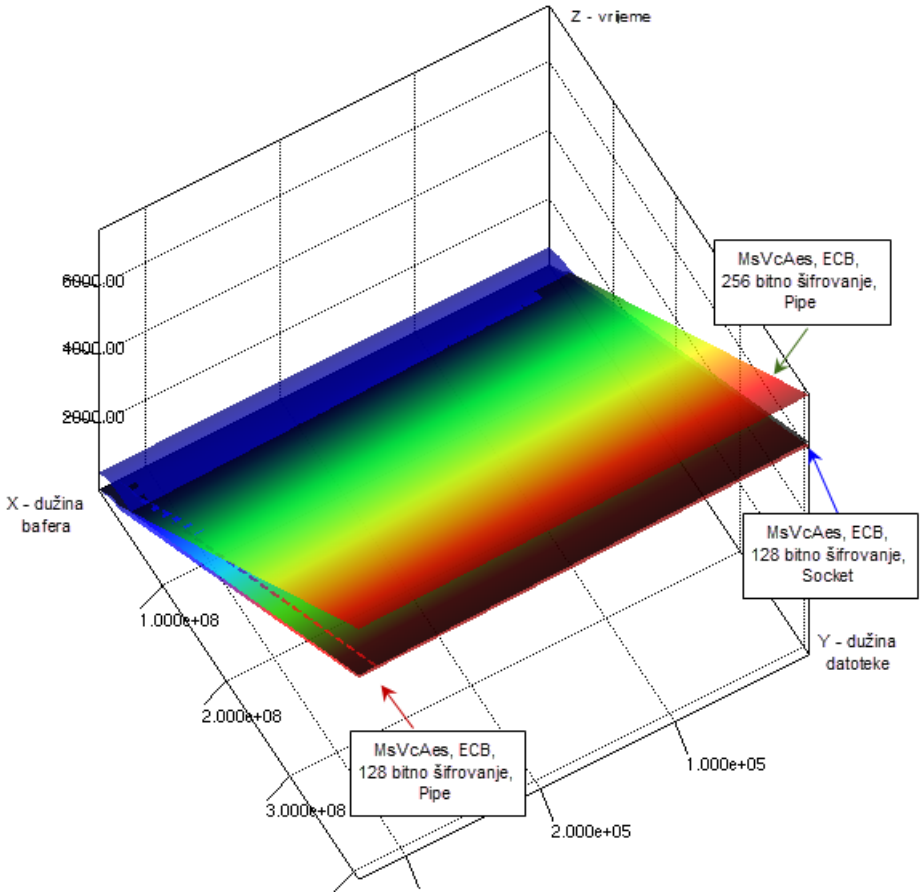
Koef.korelacije	0.998
Srednja apsolutna greška	118.2341
Srednja kvadratna greška	137.5148
Relativna apsolutna greška	3.8859%
Relativna kvadratna greška	4.0598%
Broj instanci	15

Listing 40: Koef.korelacije, srednja apsolutna i srednja kvadratna greška, te relativna apsolutna i relativna kvadratna greška prediktivnog modela MSVCAesCECB1281p, probni podaci

Kako je u ovom modulu implementiran samo ECB način rada, na osnovu njega je moguće kreirati 12 prediktivnih modela:

N = cip_inv * modes * keys * threads * comm = 2 * 1 * 3 * 1 * 2 = 12 (73)

Na slici 50 prikazana su tri od ukupno dvanaest kreiranih prediktivnih modela koji su nastali iz modula MSVCAes.



Slika 50: Prediktivni modeli nastali od kriptografskog resursa (modula) MSVCAes (N5110) – grafički prikaz. Treba imati u vidu da je na slici prikazan samo dio modela koje je moguće proizvesti iz ovog modula.

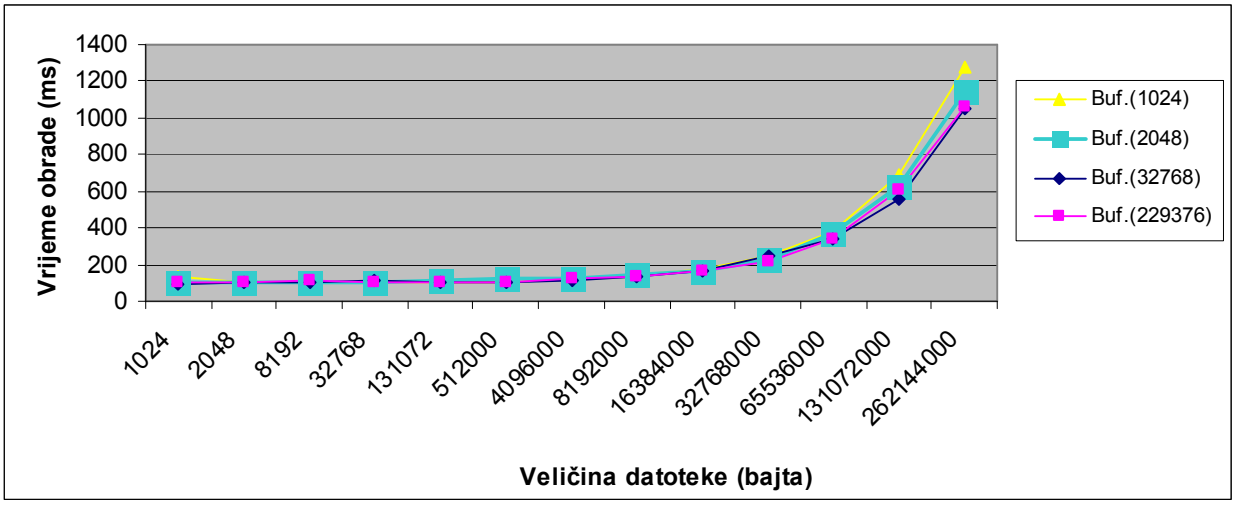
7.1.3 Modul koji koristi AES-NI skup instrukcija

Kriptografski modul koji vrši šifrovanje pomoću AES algoritma koji je zasnovan na AES New Instructiuons (AES-NI) skupu instrukcija razvijen je na osnovu kombinacije prethodno pomenutog rada na razvoju originalnih modula, teksta (Gueron 2009) i Intelovog White-Paper uputstva (Gueron 2012), te na osnovu izvornog koda Intelove biblioteke Intel® AES sample library (Rott 2011) i na osnovu izvornog koda biblioteke Botan (Lloyd et al. 2013). Intelovo White-Paper uputstvo pored opisa AES algoritma i struktura podataka sadrži i primjere izvornog koda koji se, kako je rečeno, može posebno preuzeti, a koji je pisan prvenstveno za Intelov C++ kompajler. Botan je kriptografska C++ biblioteka koja je objavljena pod BSD 2 (ili FreeBSD) licencom, a koja se danas isporučuje sa većim brojem poznatih Linux distribucija (Fedora, EPEL, Debian, Ubuntu, Gentoo, Arch Linux, Slackbuild, FreeBSD, NetBSD itd). Kombinacija navedenih open source rješenja, koja su bila namijenjena za GNU C/C++ odnosno Intel C++, dovela je do realizacije rješenja koje je moguće kompajlirati u programskom jeziku C/C++ u okviru Visual Studija 2012, a koje se izvršava u ECB režimu rada. Korišćenjem

Microsoft Visual C++ *intrinsic* funkcija kreirana je aplikacija koja kao kriptografski modul može da bude uključena u adaptibilni virtuelni kriptografski fajl sistem.

Ovaj kriptografski modul koristi identičan način komunikacije koji je već prikazan u slučaju Java i C/C++ aplikacija. On mjeri vlastito vrijeme izvršenja i šalje podatke o tome okruženju. Takođe, on posebno mjeri vrijeme koje mu je potrebno za šifrovanje i dešifrovanje podataka, bez vremena potrebnog za inicijalizaciju i UI operacije. U adaptibilnom virtuelnom kriptografskom fajl sistemu se, kao i u prethodnim slučajevima mjeri i ukupno vrijeme koje uključuje trajanje komunikacije koja se korišćenjem soketa ili neimenovanih cijevi obavlja između adaptibilnog virtuelnog kriptografskog fajl sistema i aplikacije. Ovaj modul će u daljem tekstu biti navođen pod imenom AesNI. Modul koristi samo jednonitno šifrovanje i dešifrovanje ključevima dužine 128, 192 i 256 bita u ECB načinu rada.

Kada se koristi modul AesNI za 128 bitno ECB šifrovanje uz komunikaciju putem neimenovanih cijevi uz minimalan skup poruka, trening podaci se grupišu kako je to prikazano na slici 51.

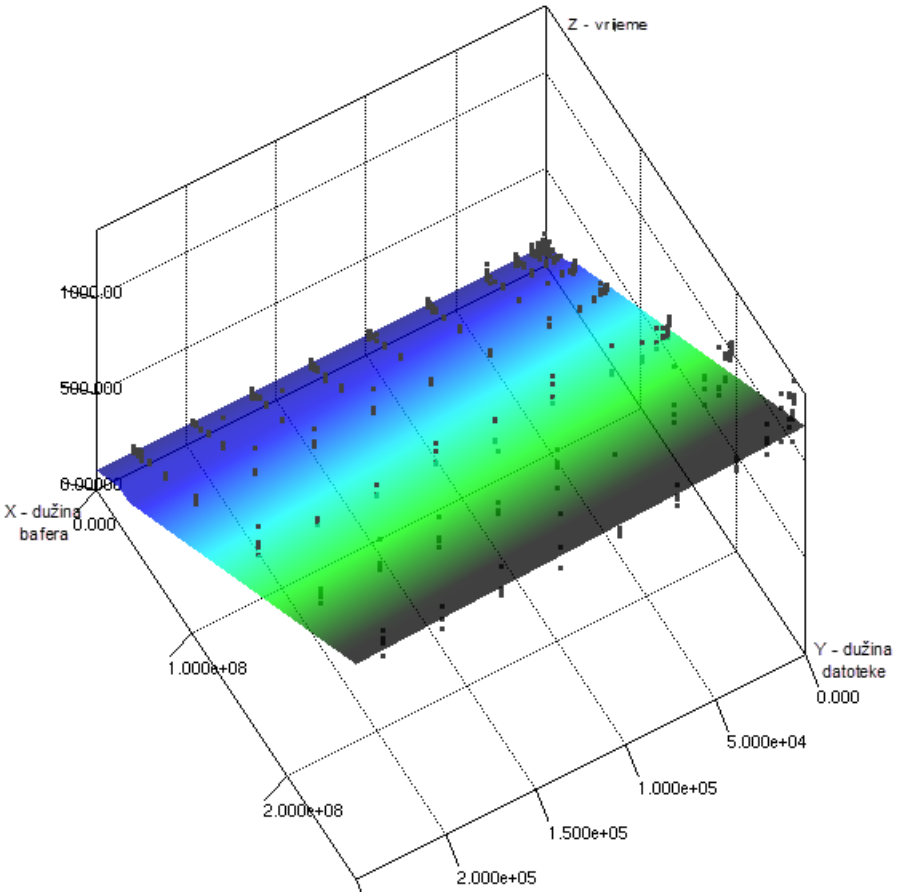


Slika 51: Uticaj dužina datoteka na vrijeme potrebno za šifrovanje (AesNICECB1281p, N5110), kom.pomoću neimenovanih cjevi uz minimalan skup poruka

Već na osnovu prikazanog dijela trening podataka može se uočiti veliki dobitak u brzini koji se postiže hardverskom akceleracijom, jer je ovaj modul za više od dvije sekunde brži pri šifrovanju datoteke dužine 256MB od najbržeg prikazanog C/C++ modula.

Rezultati mjerenja pokazuju da je šifrovanje uz komunikaciju putem neimenovanih cijevi bilo u prosjeku za 101 milisekundu sporije nego šifrovanje bez uticaja komunikacije.

Na slici 52 dat je grafički prikaz iz kojeg je moguće vidjeti kako se prediktivni model AesNICECB1281p uklapa u trening podatke. U listingu 41 prikazani su podaci koji potvrđuju da model ima veliki stepen slaganja (koef.korelacije = 0.9962) sa trening podacima. Model zahvaljujući velikoj brzini obrade podataka pokazuje veću srednju apsolutnu (14.6167) i srednju kvadratnu grešku (27.7379) od svih do sada prikazanih modela.

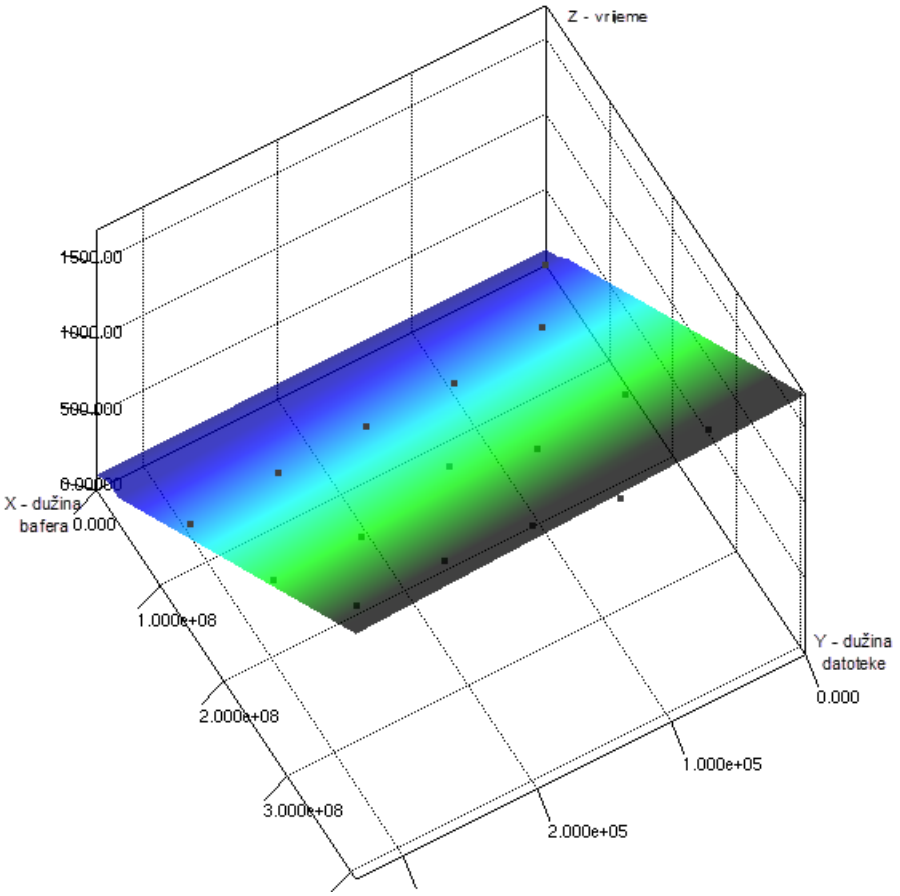


Slika 52: Vizuelni prikaz slaganja trening podataka i kreiranog AesNICECB1281p prediktivnog modela (N5110), trening podaci

Koef.korelacije	0.9962
Srednja apsolutna greška	14.6167
Srednja kvadratna greška	27.7379
Relativna apsolutna greška	5.8407%
Relativna kvadratna greška	8.6775%
Broj instanci	168

Listing 41: Koef.korelacije, srednja apsolutna i srednja kvadratna greška, te relativna apsolutna i relativna kvadratna greška prediktivnog modela AesNICECB1281p (N5110)

Radi provjere tačnosti predviđanja prediktivnog modela AesNICECB1281p, urađena je još jedna serija mjerenja sa probnim ulaznim podacima koji nisu bili dio trening skupa podataka.



Slika 53: Vizuelni prikaz slaganja probnih podataka i kreiranog AesNICECB1281p prediktivnog modela (N5110)

Koef.korelacije	0.9871
Srednja apsolutna greška	62.1682
Srednja kvadratna greška	69.8848
Relativna apsolutna greška	7.693%
Relativna kvadratna greška	7.6843%
Broj instanci	15

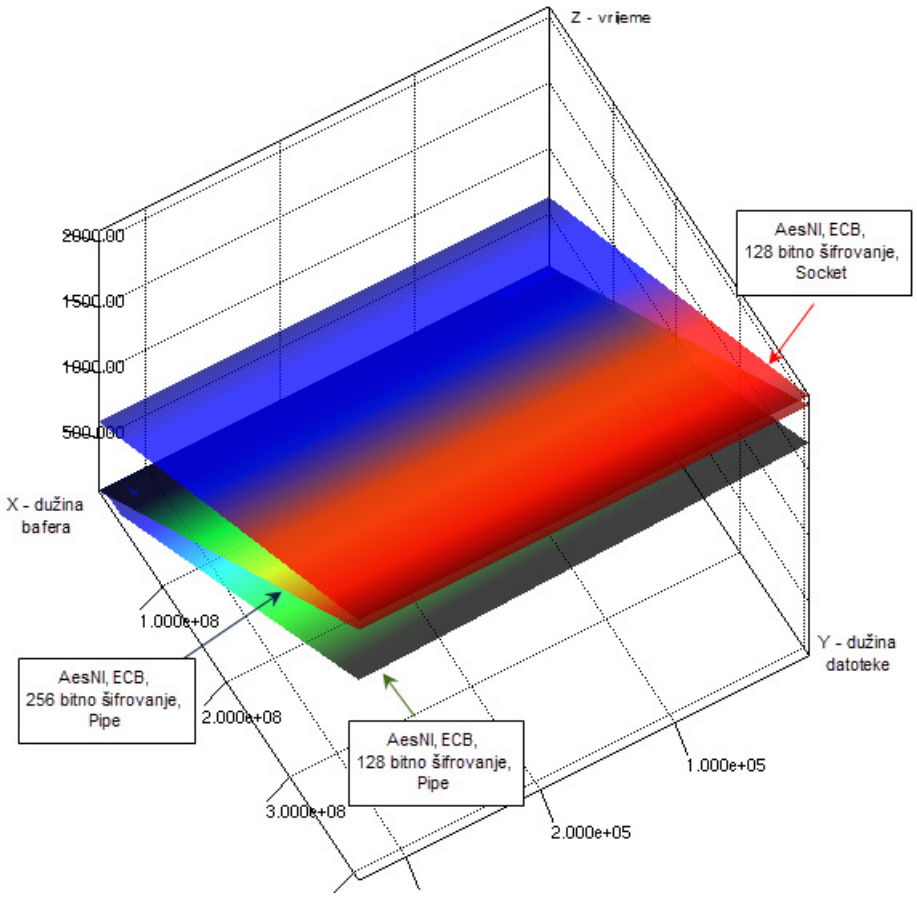
Listing 42: Koef.korelacije, srednja apsolutna i srednja kvadratna greška, te relativna apsolutna i relativna kvadratna greška prediktivnog modela AesNICECB1281p (N5110), probni podaci

Kao i kod ranije prikazanih kriptografskih modula, uticaj načina rada, dužine ključa i broja niti na predviđanje brzine ispoljava se kroz klasifikaciju, da bi se na kraju na osnovu dužine datoteke i kriptografskog bafera kreirao skup trening podataka na osnovu kojeg nastaje prediktivni model. Kako je modul AesNI imlementiran tako da koristi samo ECB mod, 3 dužine ključa, dva načina komunikacije i samo jednu nit, broj modela koje mogu da nastanu od njega je

$$N = cip\_inv * modes * keys * threads * comm = 2 * 1 * 3 * 1 * 2 = 12 \quad (74)$$

Na slici 54 prikazan je odnos AesNi prediktivnih modela od kojih dva prikazuju šifrovanje uz komunikaciju putem neimenovanih cijevi uz minimalan skup poruka, a jedan šifrovanje uz komunikaciju putem soketa i maksimalan skup poruka. Izmjereni

trening podaci i na osnovu njih nastali modeli predviđaju da će, za datoteke čija je dužina manja od 256MB, 128 bitno šifrovanje koje se obavlja uz komunikaciju putem soketa uz maksimalan skup poruka biti sporije od 256 bitnog šifrovanja koje se obavlja uz komunikaciju putem neimenovanih cijevi uz minimalan skup poruka.



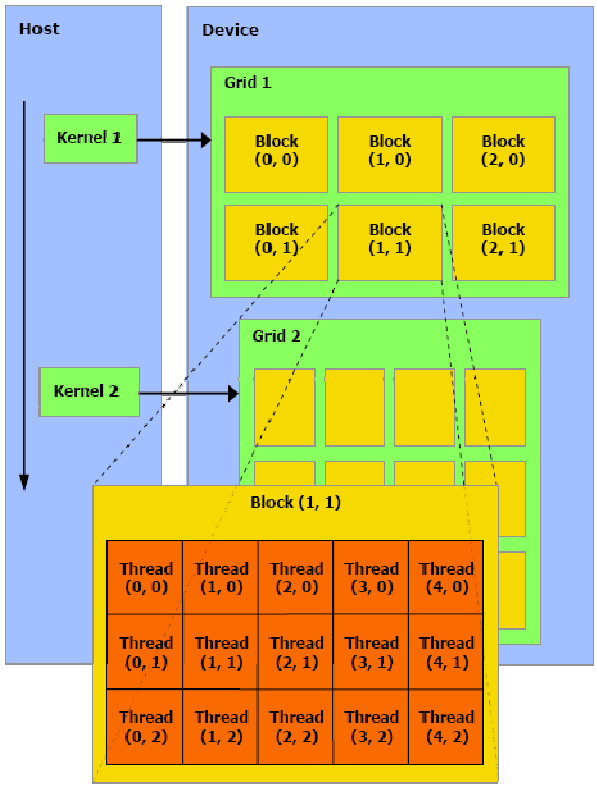
Slika 54: Prediktivni modeli podataka nastali od kriptografskog resursa (modula) AesNI (N5110). Na slici je prikazan samo dio modela koje je moguće proizvesti iz ovog modula.

8 Paralelni CUDA modul za šifrovanje i dešifrovanje AES algoritmom kao kriptografski resurs operativnog sistema

Modul za šifrovanje i dešifrovanje pomoću algoritma AES koji je zasnovan na CUDA arhitekturi i na CUDA kompajleru oslanja se na prethodno razvijene jednonitne Java i C++ module na kojima su primijenjena ubrzanja koja koriste T-tabele u skladu s idejama autora algoritma i na odgovarajući izvorni kod jednononitnih aplikacija.

Izvorni kod ovog modula je napisan u skladu sa sintaksom i preveden uz pomoć CUDA C kompajlera tako da se paralelno izvršava na velikom broju jezgara, pri čemu svako jezgro obrađuje po jedan blok Stanje. Navedeni modul je implementiran tako da se paralelno izvršava u ECB režimu rada.

CUDA je platforma za paralelnu obradu podataka i razvojno okruženje koje je kreirano u kompaniji NVIDIA (NVIDIA, 2015). Svaki CUDA grafički procesor ima N multiprocesora (MP) i globalnu memoriju. Svaki multiprocesor ima N skalarnih procesora (SP), dijeljenu memoriju i skup 32-bitnih registara. Svaki multiprocesor je dizajniran pomoću posebne arhitekture koja se naziva SIMT (Single Instruction, Multiple-Thread) koja mu omogućava da izvršava hiljade niti istovremeno.



Slika 55. CUDA grafički procesor: Grid-Block-Thread prikaz (NVIDIA, 2015)

Paralelno izvršavanje se u CUDA C implementira posebnim funkcijama koje se nazivaju kernel funkcije (*kernels*) koje se, kada su pozvane, izvršavaju paralelno N puta od strane N niti. Svaka nit ima vlastiti ID pomoću kojeg se referencira. Radi referenciranja, niti se grupišu u blokove, a blokovi u *gridove*. Multiprocesor kreira i izvršava niti u grupama od po 32 paralelne niti koje se nazivaju *warpovi*. Ovakva arhitektura se u

(NVIDIA, 2015) naziva SIMT (*Single Instruction Multiple Threads*) arhitektura. Broj niti po bloku treba odabrati tako da bude umnožak veličine *warpa*. Za one blokove kod kojih broj niti nije umnožak broja 32, neki *warpovi* neće biti do kraja popunjeni aktivnim nitima (Wilt, 2013).

Na performanse nekog CUDA programa u velikoj mjeri utiče ukupan broj niti i blokova koji se paralelno izvršavaju kao i raspoloživi broj registara po jednoj niti. Broj blokova i *warpova* koji su alocirani u svakom multiprocesoru za pojedini poziv kernela zavisi od broja *gridova* i blokova koji se navode pri pozivu kernela (*Execution Configuration*), memorijskih resursa multiprocesora i količine resursa koji su kernelu potrebni za rad. Da bi programeri mogli lakše da odaberu optimalnu veličinu bloka, u sastavu CUDA SDK se isporučuje CUDA Occupancy Calculator.

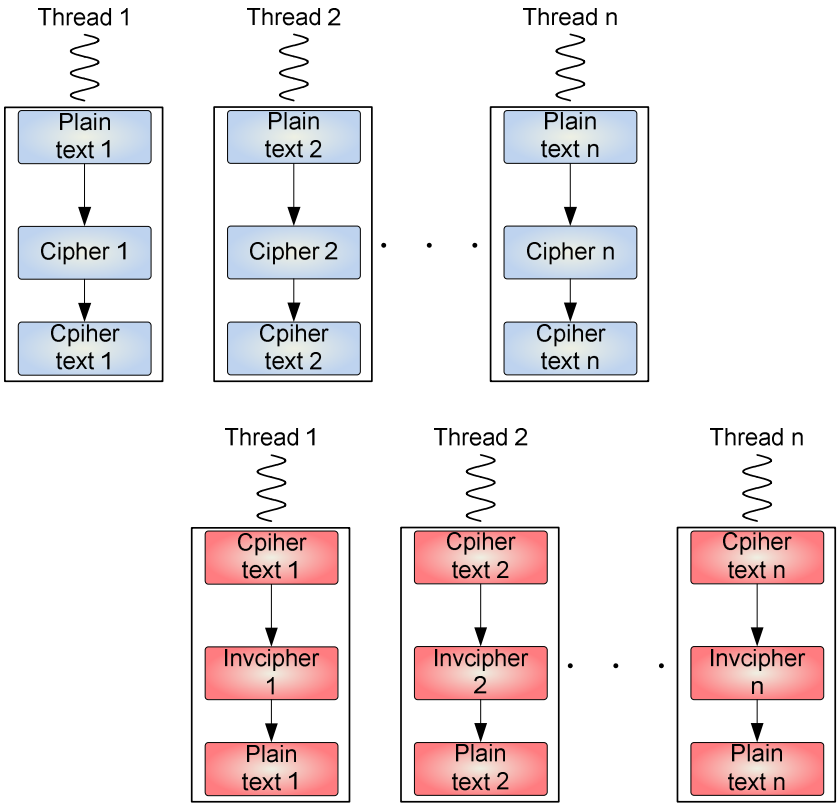
CUDA programi moraju da prenesu sve varijable i podatke koje će obrađivati iz memorije opšte namjene računara u memoriju uređaja. Rezultati obrade se nakon toga moraju vraćati nazad u memoriju računara. Svaki prenos podataka u memoriju uređaja škodi performansama, tako da je sa aspekta performansi mnogo bolji jedan veliki transfer nego više malih koji prenose istu količinu podataka.

Kao testna platforma korišćen je Dell Inspiron N5110 računar sa procesorom i7-2630QM i 6 GB RAM memorije, instaliranim 64-bitnim Windows 7 Ultimate operativnim sistemom, na kojem je kao druga grafička karta bila instalirana NVidia GeForce GT 525M.

U implementaciji opisane paralelizacije korišćen je Visual Studio 2012 C/C++ kompajler kao i C kompajler koji se isporučuje sa Nvidia Cuda v6.0 toolkitom. Navedena implementacija zahtijeva da se najprije S-Box, četiri T-Boxa, IV, ključ i bafer sa otvorenim (*plain*) tekstom odnosno šifratom prenesu u globalnu i dijeljenu memoriju uređaja (*device*), a zatim se pomoću poziva kernel funkcija vrši paralelno šifrovanje pojedinih blokova otvorenog teksta. Implementirane su dvije kernel funkcije, jedna pod nazivom processBlock za šifrovanje i druga pod nazivom invProcessBlock dešifrovanje algoritmom AES.

CUDA arhitektura (i upošte arhitektura GPU-a) je slična arhitekturi superkompjutera. Sam SDK je kreiran sa fokusom na paralelizacju. Najvažnija funkcija u CUDA SDK je Kernel, koja može da pokreće hiljade niti paralelno. Svaka kernel funkcija se pokreće sa dodatnim parametrima, od kojih su najvažniji broj blokova i broj niti po jednom bloku. Npr. poziv kernel funkcije cipherBytes<<<16, 256>>> pokreće 16x256 = 4096 niti paralelno, pri čemu će svaka nit u slučaju AES algoritma obraditi isti kod ali sa drugim podacima (npr. sa drugim otvorenim tekstom i ključem). Treba primijetiti da je CUDA blok u stvari blok pokrenutih niti i da ga treba razdvojiti od kriptografskog bloka Stanje.

Implementirano rješenje funkcioniše u skladu s CUDA/GPU arhitekturom, tako da kada se pokrene jedna kernel funkcija, u stvari se paralelno obrađuju podaci pomoću hiljada niti (Slika 56), pri čemu svaka nit obradi jedan AES-ov blok podataka (jedno Stanje/*State*). Samo šifrovanje je ovdje munjevito brzo, ali se određena količina vremena potroši na kreiranje i inicijalizaciju kernel funkcije te na transfer podataka do memorije CUDA uređaja.

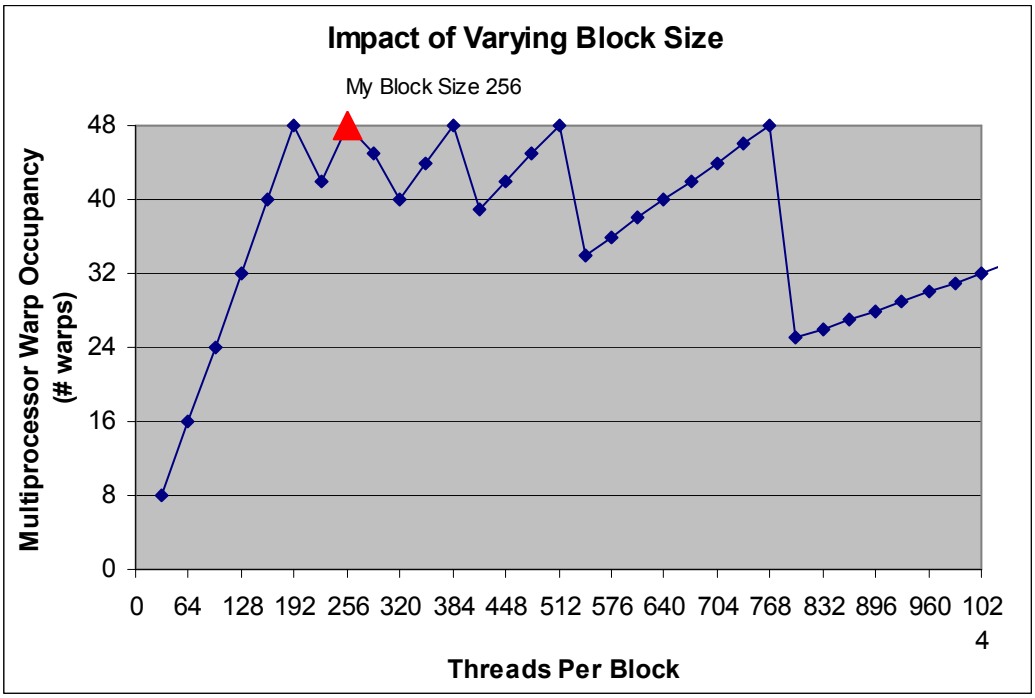


Slika 56. Blok šema paralelne obrade podataka pomoću n-niti CUDA procesora

Ovakav *coarse grain* dizajn može se implementirati na grafičkim procesorima na takav način da se S-Box, četiri T-Boxa, IV, ključ i otvoreni (plain) tekst odnosno šifrat prenesu u globalnu i dijeljenu memoriju uređaja (device), a zatim se pomoću poziva kernel funkcija vrši paralelno šifrovanje pojedinih blokova otvorenog teksta. U (Gervasi et al. 2010) je prikazano slično rješenje koje koristi OpenCL okruženje (framework) i koje u memoriju uređaja prenosi samo S-Box.

Još jedna karakteristika CUDA platforme je da neće svaka kombinacija broja blokova i niti dati isti rezultat. Kako broj registara koji se koristi u jednoj kernel funkciji, veličina dijeljene memorije i verzija grafičkog procesora u saradnji sa veličinom bloka utiču na brzinu obrade, kompanija NVidia je objavila CUDA *Occupancy Calculator* koji se može koristiti za optimizaciju izvršenja.

Da bi pomoću CUDA *Occupancy Calculatora* mogao biti odabran optimalan broj niti i blokova koji obezbjeđuje najbolje performanse, mora biti utvrđen broj registara koji će biti korišćen od strane jednog izvršenja funkcije kernela u okviru jedne niti. Ovu informaciju je pomoću Nvidia Cuda v6.0 moguće dobiti jedino u toku kompajliranja programa pomoću prekidača „--ptxas-options=-v“. Kako navedena implementacija za svoja jezgra zahtjeva po 18 registara, za NVidia GeForce GT 525M grafičku kartu koja zadovoljava 2.1 *Compute Capability* sa 49152 bajta dijeljene memorije, kalkulator je ponudio vrijednosti od 192, 256, 384, 512 i 768 niti po bloku, za koje će iskorišćenost svakog multiprocesora biti maksimalna i pri kojima se postiže 48 aktivnih *warpova* po multiprocesoru.



Slika 57. CUDA Occupancy Calculator rezultati za NVidia GeForce GT 525M

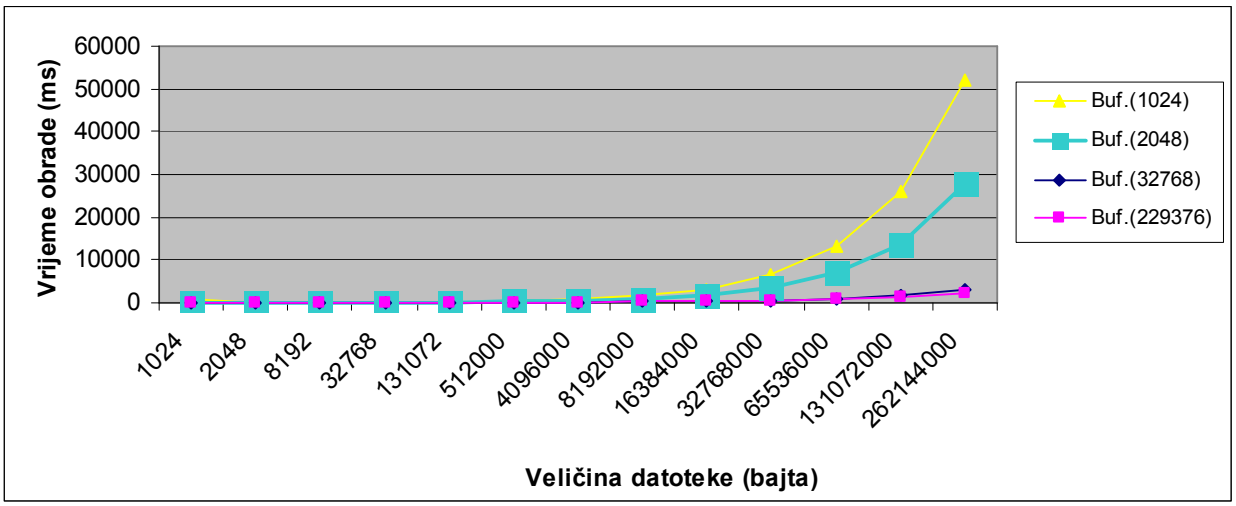
CUDA Occupancy Calculator daje još dosta dodatnih podataka o CUDA arhitekturi. Tako se za GPU koji je kompatibilan sa verzijom 2.1 mogućnosti obrade podataka (*CUDA compute capability*) po jednom multiprocesoru može koristiti maksimalno 8 blokova.

Physical Limits for GPU Compute Capability:	**2.1**
Threads per Warp	32
Warps per Multiprocessor	48
Threads per Multiprocessor	1536
Thread Blocks per Multiprocessor	8
Total # of 32-bit registers per Multiprocessor	32768
Register allocation unit size	64
Register allocation granularity	warp
Registers per Thread	63
Shared Memory per Multiprocessor (bytes)	49152
Shared Memory Allocation unit size	128
Warp allocation granularity	2
Maximum Thread Block Size	1024

Tabela 16: Isječak podataka CUDA Occupancy Calculatora za GPU koji je kompatibilan sa verzijom 2.1 mogućnosti obrade podataka

Kako je u testovima korišćen NVidia GeForce GT 525M GPU koji ima 2 multiprocesora, u implementaciji je korišćeno 16 blokova veličine po 256 niti svaki.

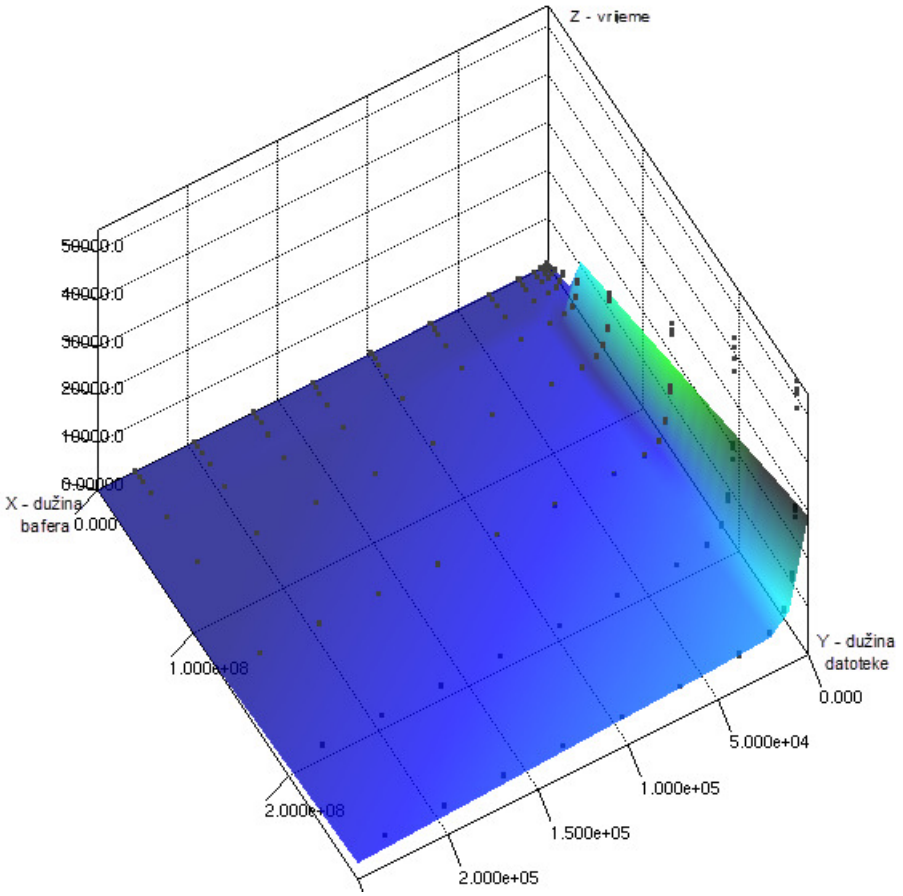
Rezultati testiranja ovoga modula pokazuju da postoji ogromna razlika u brzini u zavisnosti od veličine internog bafera koja se koristi u procesu obrade podataka, kako je prikazano na slici 58.



Slika 58: Uticaj dužina datoteka na vrijeme potrebno za šifrovanje (CudaCECB128np, N5110), kom.pomoću neimenovanih cijevi uz minimalan skup poruka

Prikazana razlika u brzini je u skladu sa datim objašnjenjima i slikom 56 koji opisuju način implementacije. Za interni bafer dužine 512 bajta u ovoj implementaciji je moguće uposliti samo 32. niti pri jednom pozivu kernela. Istovremeno se dešava veliki broj prenosa podataka u memoriju uređaja koji takođe škodi performansama. Sa povećanjem bafera (ali i datoteke) povećavaju se mogućnosti upošljavanja niti, zbog čega je moguće uposliti hiljade niti u jednom pozivu kernela, a da se pri tome smanjuje negativan uticaj prenosa podataka u memoriju uređaja. Ovdje je prikazan rezultat 128-bitnog ECB šifrovanja, ali su sva razmatranja primjenljiva i na enkripciju većim dužinama ključeva, uz uslov da treba imati na umu da njihovo izvršenje, zbog većeg broja rundi, traje nešto duže.

Na slici 59 grafički je prikazano kako se kreirani prediktivni model CudaCECB128np uklapa u trening podatke. Model CudaCECB128np ima visok stepen slaganja sa trening podacima (koef.korelacije = 0.9082) ali i nešto višu relativnu standardnu (37.2064%) i relativnu kvadratnu (47.9832%) grešku.



Slika 59: Vizuelni prikaz slaganja trening podataka i kreiranog CudaCECB128np prediktivnog modela (N5110)

Koef.korelacije	0.9082
Srednja apsolutna greška	1050.0652
Srednja kvadratna greška	3001.4239
Relativna apsolutna greška	37.2068%
Relativna kvadratna greška	47.9832%
Broj instanci	168

Listing 43: Koef.korelacije, srednja apsolutna i srednja kvadratna greška, te relativna apsolutna i relativna kvadratna greška prediktivnog modela CudaCECB128np (N5110), trening podaci

Iako opisane pojave imaju visok stepen slaganja, javljaju se nešto više vrijednosti relativne asolutne i kvadratne greške. Navedene greške su nastale na osnovu trening podataka koji su skoncentrisani na veoma malom prostoru uz Y osu.

Zbog prikazanih grešaka, nad trening podacima (sa uključenim vremenom za komunikaciju putem neimenovanih cijevi) su probane još dvije metode mašinskog učenja. Tako su nad trening podacima kreirani i modeli pomoću metoda Multilayer perceptron i M5Rules. Metoda Multilayer perceptron je proizvela niži koeficient korelacije (0.8425) i višu relativnu standardnu (64.2092%) i relativnu kvadratnu (56.7113%) grešku. Metoda M5Rules je proizvela nešto viši stepen slaganja (koef.korelacije 0.9344) i nešto manju relativnu standardnu (27.6093%) i relativnu kvadratnu (41.5549%) grešku. Ovo je jedini kriptografski resurs (modul) na čijim

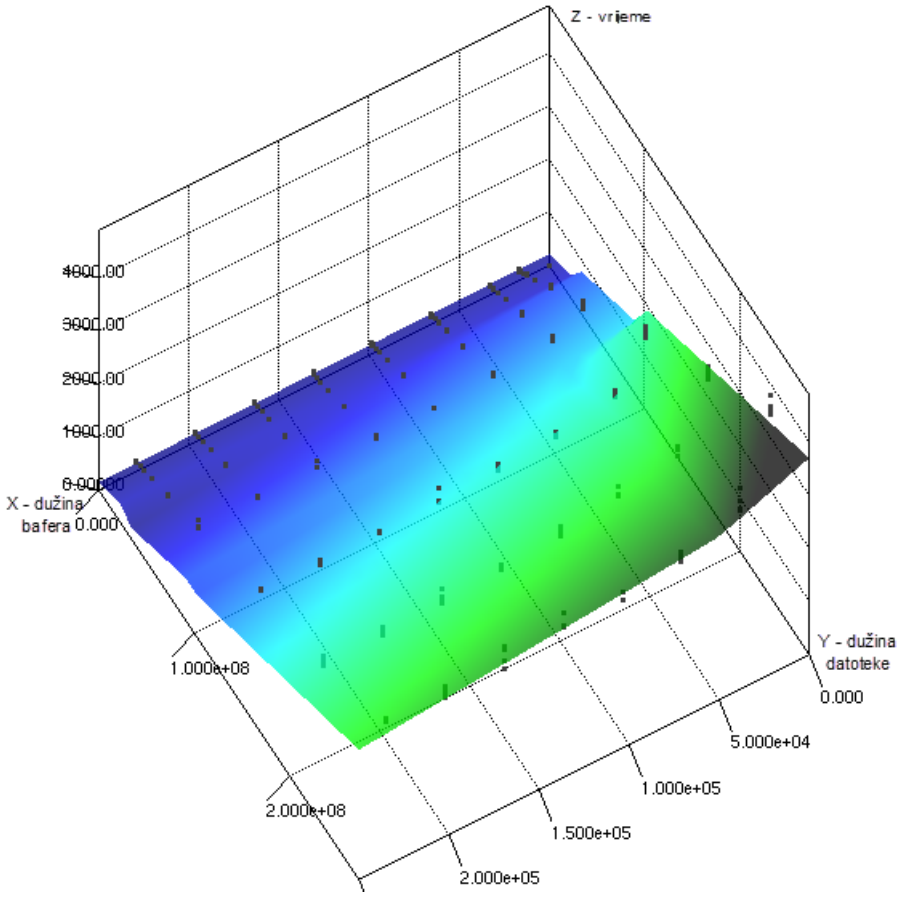
podacima je neka od testiranih metoda mašinskog učenja proizvela bolje rezultate od metode M5'. Kako je metoda M5Rules na svim ostalim resursima pokazivala mnogo lošije rezultate, u ovom istraživanju smo nastavili da koristimo metodu M5'.

Značajna karakteristika modula zasnovanog na CUDA arhitekturi koja je vidljiva iz prikazanih trening podataka i modela je da se sa povećanjem dužine internog bafera vrijeme potrebno za obradu podataka veoma brzo smanjuje. Ova činjenica bi se pri izradi modela mogla uzeti u obzir kroz konstrukciju SQL iskaza kojim se pribavljaju trening podaci, tako da se u WHERE klauzulu doda uslov da je dužina bafera veća od neke vrijednosti, kao što je prikazanu u listingu 44.

```
"select BUFF, FILESIZE, POLJE from TESTDATACIPHER
where Klasa = KLASA and USINGSOCKET = KOMUNIKACIJA and Buff > 8192"
```

Listing 44: Sintaksa SQL upita

Kada se na osnovu ovakvog smanjenog skupa trening podataka generiše model (kom.pomoću neim.cijevi), on se znatno bolje uklapa u trening podatke, kako je prikazano na slici 60 i u listingu 45.



Slika 60: Vizuelni prikaz slaganja probnih podataka i kreiranog CudaCECB128np prediktivnog modela (N5110)

Koef.korelacije	0.9795
Srednja apsolutna greška	76.5707
Srednja kvadratna greška	176.9616

```
Relativna apsolutna greška                11.6658%
Relativna kvadratna greška                20.244%
Broj instanci                            112
```

Listing 45: Koef.korelacije, srednja apsolutna i srednja kvadratna greška, te relativna apsolutna i relativna kvadratna greška prediktivnog modela CudaCECB128np (N5110)

Iz prikazanih podataka može da se zaključi da modul zasnovan na CUDA arhitekturi može da bude veoma efikasan ako se pomoću njega obrađuju veće količine podataka, odnosno veoma neefikasan za obradu malih količina podataka, što treba uvijek imati u vidu pri njegovoj upotrebi.

Testirane metode mašinskog učenja uspijevaju da postignu dovoljno visok koeficient korelacije, koji je posebno visok ako se modul koristi samo za obradu većih količina podataka.

9 Jednonitni eksperimenti sa mogućim načinima modifikacije AES algoritma

U literaturi postoji određen broj evidentiranih pokušaja modifikacije AES algoritma, uglavnom u školske svrhe i radi njegove kriptoanalize. Tako mini verzija AES algoritma koju je razvio profesor Chung-Wei Phan (Chung-Wei Phan, 2002) u radu koristi niblove (eng. *nibble*, 4 bita) za razliku od pune verzije algoritma koja koristi bajte. Zbog ovoga su u mini verziji AES algoritma morale biti ponovo redefinisane operacije množenja i sabiranja tako da koriste GF(2^4). Ovakva verzija je koristila blok podataka (*State*) veličine 16 bita, za razliku od pune verzije koja je koristila blok veličine 16 bajta. Kako se navodi u (Gordon, 2013) i ovaj autor u nastavi sa studentima koristi pojednostavljene verziije AES algoritma. Pored ovoga zabilježeni su i različiti drugi pokušaji pojednostavljivanja kriptografskih algoritama za različite svrhe. Tako autori u (Manangi et al. 2010) predlažu pojednostavljenu verziju AES algoritma za upotrebu u embedded sistemima. Treba reći da je tokom 2009. godine, u toku istraživanja u okviru izrade Master rada (Damjanović, 2010) na Fakultetu organizacionih nauka u Beogradu nezavisno od rada prikazanog u (Manangi et al. 2010) nastalo nekoliko redukovanih verzija ovog algoritma.

Kada govorimo o kriptoanalizi, treba reći da je veliki broj redukovanih i pojednostavljenih verzija različitih algoritama našao primjenu u kriptoanalizi. U (Demirci et al. 2008) autori predstavljaju redukovanu verziju AES algoritma radi izvođenja *meet-in-the-middle* napada. U (Biryukov et al. 2009) autori opisuju mogućnost *Related Key* napada na pune 192 i 256 bitne verzije algoritma AES. U (Daemen and Rijmen, 2012) autori algoritma navode da navedeni napad ima samo akademsku vrijednost. U (Biryukov et al. 2010) autori opisuju mogućnost *Key Recovery* napada na 256-bitnu verziju AES algoritma koja je redukovana na 9 rundi, kao i *Chosen-plaintext* i *Chosen-ciphertext* napad na 256-bitnu verziju AES algoritma koja je redukovana na 10 rundi.

U toku ovog istraživanja proveden je niz eksperimenata koji su se bavili mogućim modifikacijama algoritma AES, u smislu njegovog redukovanja u školske svrhe i u smislu njegovog ojačanja sa ključevima dužine veće od 256 bita. Namjena prve, redukovane implementacije (Damjanović i Simić, InfoM46, 2013) je da studentima olakša shvatanje načina funkcionisanja ovog algoritma i da ih pripremi za savladavanje standardom definisanog algoritma AES. Redukovana verzija algoritma je kasnije upotrebljena u provođenju eksperimenata sa paralelizacijom modifikovane verzije algoritma AES na višejezgrenim procesorima.

Namjena ojačane implementacije je da napravi prve eksperimentalne korake koji nas vode ka poboljšanju AES-ove otpornosti na kriptoanalizu. U dostupnoj literaturi nema tragova da je bilo pokušaja ojačanja ovoga algoritma.

9.1 Standardom definisani algoritam i pravila generisanja ključeva

Algoritam AES kako je opisano u (FIPS197, 2001), operiše nad 128-bitnim blokovima podataka (16 bajta) korišćenjem ključeva dužine 128, 192 i 256 bita. Ulaznih 16 bajta su

organizovani u 4x4 matricu pod nazivom Stanje (*State*). Operacija šifrovanja se izvodi cikličnim ponavljanjem transformacija pod nazivima *SubBytes(), ShiftRows(), MixColumns()* i *AddRoundKey()* u okviru 10, 12 ili 14 rundi sa različitim dužinama ključeva. U procesu dešifrovanja koriste se inverzne transformacije *InvSubBytes(), InvShiftRows()* i *InvMixColumns(),* dok se transformacija *AddRoundKey()* zadržava u neizmijenjenom obliku.

Za sve navedene implementacije veoma važna rutina koja se mora realizovati je rutina za ekspanziju ključeva. U standardnom algoritmu AES, dužine inicijalnog ključa su 128, 192 i 256 bita. Da bi AES algoritam bio funkcionalan, inicijalni ključ se mora u funkciji ekspanzije ključeva razviti do određenog broja bajta, koji zavisi od tipa algoritma (AES-128, AES-192 ili AES-256). Broj rundi u algoritmu zavisi od dužine ključa i obrnuto. Ako dužinu inicijalnog ključa u bitima predstavimo sa Nbk, a broj rundi sa Nr, imamo:

- Nr = 10 kada je Nbk = 128,
- Nr = 12 kada je Nbk = 192,
- Nr = 14 kada je Nbk = 256.

Rutina za ekspanziju ključeva na osnovu inicijalnog ključa dužine 128, 192 ili 256 bita generiše listu ključeva dužine Nk.

$$Nk = Nb(Nr+1) \tag{75}$$

gdje je Nb broj bajta Stanja (za AES uvjek 16), a Nr broj rundi algoritma (10, 12 ili 14). Prema ovome, inicijalni ključ mora biti proširen do odgovarajućeg broja bajta da bi algoritam bio funkcionalan.

U standardnom algoritmu promjena broja rundi za 2 povezana je sa promjenom broja bita inicijalnog ključa za 64, kao u slijedećoj tabeli:

Tabela 17: Odnosi dužina ključeva i broja rundi standardnog AES algoritma

Br.rundi AES	Inicijalni ključ (bita)	Inicijalni ključ (bajta)	Prošireni ključ (bajta)
14	256	32	240
12	192	24	208
10	128	16	176

9.1.1 Modifikacija algoritma ispod granice od 128 bita

U toku istraživanja vezanih za izradu Master rada (Damjanović, 2010) proveden je niz eksperimenata tokom kojih su implementirana proširenja algoritma AES na 32, 64, 320, 384, 448 i 512-bitnu enkripciju/dekripciju bez bilo kakvih optimizacija koda. U sljedećem nizu eksperimenata (Damjanović i Simić, InfoM46, 2013) implementirana su proširenja AES algoritma na 32 i 64-bitnu enkripciju/dekripciju sa optimizacijama zasnovanim na modifikovanim idejama autora algoritma kao i sa optimizacijama zasnovanim na modifikovanim Bertonijevim idejama. U nastavku će biti predstavljene modifikovane

verzije AES algoritma koje su redukovane na 32 i 64-bitnu enkripciju/dekripciju te verzije koje su proširene do 512 bitne enkripcije.

Kako je već pomenuto, u standardnom algoritmu promjena broja rundi za 2 povezana je sa promjenom broja bita inicijalnog ključa za 64, kao u sljedećoj tabeli:

Tabela 18: Odnosi dužina ključeva i broja rundi standardnog i modifikovanog AES algoritma

Br.rundi AES	Br.rundi pojednostavljeni AES	Inicijalni ključ (bita)	Inicijalni ključ (bajta)	Prošireni ključ (bajta)
14		256	32	240
12		192	24	208
10		128	16	176
	8	64	8	144
	7	32	4	128

Na osnovu svega prethodno rečenog, u (Damjanović, 2010) su formirana sljedeća pravila proširenja:

1. Kako je u AES algoritmu broj kolona stanja Nk uvijek jednak 4, a kolona se sastoji od 4 bajta, za dužinu proširenog ključa korišćena je ista formulu kao i u standardnom algoritmu:

   Nk(Nr+1), kolona odnosno (76)

   16(Nr+1) izraženo u bajtima

2. Kao i u standardnom AES algoritmu, koristićemo slijedeću formulu za utvrđivanje broja rundi:

   Nr=Nw+6, odnosno (77)

   Nr=Nbk/4+6, izraženo u bajtima

   gdje je

   Nw – broj riječi u inicijalnom ključu

   Nbk – broj bajta u inicijalnom ključu

3. U skladu sa poglavljem 2.1.2.3.6, usvojićemo da je minimalna veličina inicijalnog ključa 4 bajta, odnosno jedna 32-bitna riječ.

Tokom implementacije ovako modifikovanog algoritma, morale su biti izmjenjene rutine za ekspanziju ključeva, dok su funkcije za šifrovanje i dešifrovanje pretrpile tek neznatne izmjene. Zbog toga su u kod dodate još dvije nove rutine za proširenje ključeva pod nazivom KeyExpansion64 i KeyExpansion32. Pseudo kod obje rutine prikazan je u listingu 2.

```
keyExpBase(byte i)
begin
byte a = 0
Call RotWord
while a <= 3
  tmpkey[a] = sbox[ tmpkey[a] ]
  tmpkey[0] = tmpkey[0] XOR Rcon(i)
  a = a + 1
end while
end
keyExpansion64
begin
byte c = 8
byte i = 1
byte a
while(c < 144)
   a = 0
   while(a <= 3)
      tmpkey[a] = key[a + c - 4]
      a = a + 1
   end while
   if (c mod 8 = 0)
      Call keyExpBase(i)
      i = i + 1
   end if
   a = 0
   while(a <= 3)
      key[c] = key[c - 16] XOR tmpkey[a]
      c = c + 1
      a = a + 1
   end while
end while
end
keyExpansion32
begin
byte c = 4
byte i = 1
byte a
while(c < 128)
   a = 0
   while(a <= 3)
      tmpkey[a] = key[a + c - 4]
      a = a + 1
   end while
   Call keyExpBase(i)
   i = i + 1
   a = 0
   while(a <= 3)
      key[c] = key[c - 16] XOR tmpkey[a]
      c = c + 1
      a = a + 1
   end while
end while
end
```

Listing 46: Pseudo kod rutina za proširenje ključa za slučajeve 32-bitnih i 64-bitnih ključeva

Verzija AES algoritma prikazana u (Damjanović i Simić, InfoM46, 2013) nije u znatnijoj mjeri izmjenjena u odnosu na standardom definisani AES algoritam. Sve transformacije standardom definisanog algoritma (*SubBytes*, *MixColumns*, *ShiftRows*, *AddRoundKey*) su u redukovanoj verziji algoritma zadržane u neizmjenjenoj formi.

Jedina rutina koja je pretrpila određene izmjene je rutina za ekspanziju ključeva. U okviru ove transformacije se kod standardom definisanog AES 256, 192 i 128-bitnog algoritma svakih 32, 24 i 16 bajta poziva rutina *keyExpBase*. Zadatak rutine keyExpBase je da na trenutna četiri bajta ključa primijeni AES S-Box te da se prvi bajt pomoću xor operacije izmješa sa nizom RCon. Kod najmanje standardom definisane verzije (AES128) rutina za ekspanziju ključeva se poziva na svakih četiri puta četiri (16) bajta ključa. U slučaju modifikovane verzije, rutina za proširenje 64-bitnog inicijalnog ključa poziva funkciju *keyExpBase* na svakih 8 bajta (odnosno dva puta po četiri bajta), dok je rutina za proširenje 32-bitnog inicijalnog ključa poziva za svaka 4 bajta, kako je to prikazano u isječku međurezultata (Listing 47):

```
A) Ključ 128 bita 000102030405060708090a0b0c0d0e0f
 |          | After    | After    |          | After    |           | key[i]=temp
 | temp     | RotWord  | SubWord  | Rcon     | xor Rcon | key[i-16] | xor key[i-16]
 |----------|----------|----------|----------|----------|-----------|-------------
 | 0C0D0E0F | 0D0E0F0C | D7AB76FE | 01000000 | D6AB76FE | 00010203  | D6AA74FD
 | D6AA74FD |          |          |          |          | 04050607  | D2AF72FA
 | D2AF72FA |          |          |          |          | 08090A0B  | DAA678F1
 | DAA678F1 |          |          |          |          | 0C0D0E0F  | D6AB76FE
 | D6AB76FE | AB76FED6 | 6238BBF6 | 02000000 | 6038BBF6 | D6AA74FD  | B692CF0B
 | B692CF0B |          |          |          |          | D2AF72FA  | 643DBDF1
 | 643DBDF1 |          |          |          |          | DAA678F1  | BE9BC500
 | BE9BC500 |          |          |          |          | D6AB76FE  | 6830B3FE
 | 6830B3FE | 30B3FE68 | 046DBB45 | 04000000 | 006DBB45 | B692CF0B  | B6FF744E
 | B6FF744E |          |          |          |          | 643DBDF1  | D2C2C9BF
 | D2C2C9BF |          |          |          |          | BE9BC500  | 6C590CBF
 | 6C590CBF |          |          |          |          | 6830B3FE  | 0469BF41
 | 0469BF41 | 69BF4104 | F90883F2 | 08000000 | F10883F2 | B6FF744E  | 47F7F7BC
 | 47F7F7BC |          |          |          |          | D2C2C9BF  | 95353E03
 | 95353E03 |          |          |          |          | 6C590CBF  | F96C32BC
 | F96C32BC |          |          |          |          | 0469BF41  | FD058DFD...
...
...
...


B) Ključ 64 bita 0001020304050607
 |          | After    | After    |          | After    |           | key[i]=temp
 | temp     | RotWord  | SubWord  | Rcon     | xor Rcon | key[i- 8] | xor key[i- 8]
 |----------|----------|----------|----------|----------|-----------|-------------
 | 04050607 | 05060704 | 6B6FC5F2 | 01000000 | 6A6FC5F2 | F0FF1102  | 9A90D4F0
 | 9A90D4F0 |          |          |          |          | 9A90D4F0  | 00000000
 | 00000000 | 00000000 | 63636363 | 02000000 | 61636363 | 00010203  | 61626160
 | 61626160 |          |          |          |          | 04050607  | 65676767
 | 65676767 | 67676765 | 8585854D | 04000000 | 8185854D | 9A90D4F0  | 1B1551BD
 | 1B1551BD |          |          |          |          | 00000000  | 1B1551BD
 | 1B1551BD | 1551BD1B | 59D17AAF | 08000000 | 51D17AAF | 61626160  | 30B31BCF
 | 30B31BCF |          |          |          |          | 65676767  | 55D47CA8
 | 55D47CA8 | D47CA855 | 4810C2FC | 10000000 | 5810C2FC | 1B1551BD  | 43059341...
...
...
...
```

```
C) Ključ 32 bita 00010203
 |          | After    | After    |          | After    |           | key[i]=temp
 | temp     | RotWord  | SubWord  | Rcon     | xor Rcon | key[i- 4] | xor key[i- 4]
 |----------|----------|----------|----------|----------|-----------|-------------
 | 00010203 | 01020300 | 7C777B63 | 01000000 | 7D777B63 | AC801102  | D1F76A61
 | D1F76A61 | F76A61D1 | 6802EF3E | 02000000 | 6A02EF3E | F0FF1102  | 9AFDFE3C
 | 9AFDFE3C | FDFE3C9A | 54BBEBB8 | 04000000 | 50BBEBB8 | 50BBEBB8  | 00000000
 | 00000000 | 00000000 | 63636363 | 08000000 | 6B636363 | 00010203  | 6B626160
 | 6B626160 | 6261606B | AAEFD07F | 10000000 | BAEFD07F | D1F76A61  | 6B18BA1E...
...
...
...
```

Listing 47: Dio međurezultata proizveden pri ekspanziji 128, 64 i 32-bitnog inicijalnog ključa

Na listingu 3 su prikazane tri sekcije međurezultata koje su proizvedene pri proširenju 128-bitnog, 64-bitnog i 32-bitnog inicijalnog ključa. Međurezultati su prikazani na identičan način na koji su navedeni u dokumentu FIPS-197 koji definiše algoritam AES. Da bi rješenje ostalo što je više moguće u skladu sa standardom definisanim algoritmom, u slučaju 32-bitne enkripcije, funkcija proširenja *keyExpBase* se poziva za svaki blok od po četiri bajta inicijalnog ključa. Zbog ovoga, prema prikazanoj ideji nije moguće smanjivati broj bita ključa ispod ove vrijednosti.

9.1.2 Modifikacija algoritma iznad granice od 256 bita

U toku istraživanja koja su prikazana u (Damjanović, 2010) u eksperimentalne svrhe su implementirana proširenja algoritma AES na 320, 384, 448 i 512-bitnu enkripciju/dekripciju koja su zasnovana na modifikovanim idejama autora algoritma.

U skladu sa jednačinama 76 i 77, dužinu proširenog ključa možemo povećavati u zavisnosti od broja rundi Nr prema veličinama datim u sljedećoj tabeli:

Nr - standardni AES	Nr - Prošireni (Extended) AES	16(Nr+1) izraženo u bajtima
10		176
12		208
14		240
	15	256
	16	272
	17	288
	18	304
	19	320
	20	336
	21	352
	22	368
...	...	...

Tabela 19: Odnos broja rundi i proširenog ključa

Na osnovu formula 68 i 69 možemo uspostavilti i odnos između broja rundi i broja bita inicijalnog ključa, kao u slijedećoj tabeli:

Nr - standardni AES	Nr - Prošireni (Extended) AES	Nbk= 4*(Nr-6) Inicijalni ključ izraženo u bajtima	Inicijalni ključ izraženo u bitima
10		**16**	**128**
12		**24**	**192**
14		**32**	**256**
	15	36	288
	16	**40**	**320**
	17	44	352
	18	**48**	**384**
	19	52	416
	20	**56**	**448**
	21	60	480
	22	**64**	**512**
...	...	...	...

Tabela 20: Odnos broja rundi prema inicijalnom ključu

Najvažnija rutina AES algoritma koja je bitna za njegovo proširenje je njegova rutina za ekspanziju ključeva. Ulazni podatak za ovu transformaciju je inicijalni ključ odgovarajuće dužine koji se u njoj proširuje na potreban broj bajta. Pošto navedena rutina ni na koji način ne zavisi ni od otvorenog teksta ni od šifrata, prošireni ključ se može unaprijed izračunati.

Pseudo kod rutina za proširenje inicijalnih ključeva većih od 256 bita dat je u sljedećem listingu:

```
keyExpBase(byte i)
begin
byte a = 0
Call RotWord
while a <= 3
  tmpkey[a] = sbox[ tmpkey[a] ]
  tmpkey[0] = tmpkey[0] XOR Rcon(i)
  a = a + 1
end while
end

keyExpansion320_512(byte init_key_len)
begin
byte c = 40
byte user_def = 40
buff_len = 272;
byte i = 1
byte a
switch case init_key_len of
    case 320
         c := 40;
         user_defined := 40;
         buff_len := 272;
    end case 320
    case 384
         c := 48;
         user_defined := 48;
```

```
          buff_len := 304;
     end case 384
     case 448
          c := 56;
          user_defined := 56;
          buff_len := 336;
     end case 448
     case 512
          c := 64;
          user_defined := 64;
          buff_len := 368;
     end case 512
end switch case

while(c < buff_len)
   a = 0
   while(a <= 3)
      tmpkey[a] = key[a + c - 4]
      a = a + 1
   end while
   if ((c mod user_defined) = 0)
      Call keyExpBase(i)
      i = i + 1
   end if
   if ((c mod user_defined) = 16)
      tmpkey[a] = SBox[ tmpkey[a] ]
   end if
   a = 0
   while(a <= 3)
      key[c] = key[c - user_defined] XOR tmpkey[a]
      c = c + 1
      a = a + 1
   end while
end while
end
```

Listing 48: Pseudo kod rutina za proširenje ključa za slučajeve 320-bitnih i 512-bitnih ključeva

U slučaju modifikovanih verzija rutina za proširenje 320-bitnog inicijalnog ključa poziva funkciju *keyExpBase* na svakih 40 bajta (odnosno svakih deset blokova od po četiri bajta), dok se npr. rutina za proširenje 512-bitnog inicijalnog ključa poziva na svakih 64 bajta, kako je to prikazano u isječku međurezultata (Listing 49):

```
A)
Ključ 320 bita
000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f2122232425262728
 |          | After    | After    |          | After    |           | key[i]=temp
 | temp     | RotWord  | SubWord  | Rcon     | xor Rcon | key[i-40] | xor key[i-40]
 |----------|----------|----------|----------|----------|-----------|--------------
 | 25262728 | 26272825 | F7CC343F | 01000000 | F6CC343F | 08090A0B  | FEC53E34
 | FEC53E34 |          |          |          |          | 0C0D0E0F  | F2C8303B
 | F2C8303B |          |          |          |          | 10111213  | E2D92228
 | E2D92228 |          |          |          |          | 14151617  | F6CC343F
 | F6CC343F |          | 424B1875 |          |          | 18191A1B  | 5A52026E
 | 5A52026E |          |          |          |          | 1C1D1E1F  | 464F1C71
 | 464F1C71 |          |          |          |          | 21222324  | 676D3F55
```

```
 | 676D3F55 |          |          |          |          | 25262728  | 424B187D
 | 424B187D |          |          |          |          | FEC53E34  | BC8E2649
 | BC8E2649 |          |          |          |          | F2C8303B  | 4E461672
 | 4E461672 | 4616724E | 5A47402F | 02000000 | 5847402F | E2D92228  | BA9E6207
 | BA9E6207 |          |          |          |          | F6CC343F  | 4C525638
 | 4C525638 |          |          |          |          | 5A52026E  | 16005456
 | 16005456 |          |          |          |          | 464F1C71  | 504F4827
 | 504F4827 |          | 538452CC |          |          | 676D3F55  | 34E96D99
 | 34E96D99 |          |          |          |          | 424B187D  | 76A275E4
 | 76A275E4 |          |          |          |          | BC8E2649  | CA2C53AD
 | CA2C53AD |          |          |          |          | 4E461672  | 846A45DF
 | 846A45DF |          |          |          |          | BA9E6207  | 3EF427D8
 | 3EF427D8 |          |          |          |          | 4C525638  | 72A671E0
 | 72A671E0 | A671E072 | 24A3E140 | 04000000 | 20A3E140 | 16005456  | 36A3B516
...
...
...
...


B) Ključ 512 bita
000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f212223242526272829 2a2b
2c2d2e2f313233343536373839 3a3b3c3d3e3f3132

 |          | After    | After    |          | After    |           | key[i]=temp
 | temp     | RotWord  | SubWord  | Rcon     | xor Rcon | key[i-64] | xor key[i-64]
 |----------|----------|----------|----------|----------|-----------|-------------
 | 3E3F3132 | 3F31323E | 75C723B2 | 01000000 | 74C723B2 | 21222324  | 55E50096
 | 55E50096 |          |          |          |          | 25262728  | 70C327BE
 | 70C327BE |          |          |          |          | 292A2B2C  | 59E90C92
 | 59E90C92 |          |          |          |          | 2D2E2F31  | 74C723A3
 | 74C723A3 |          | 92C6260A |          |          | 32333435  | A0F5123F
 | A0F5123F |          |          |          |          | 36373839  | 96C22A06
 | 96C22A06 |          |          |          |          | 3A3B3C3D  | ACF9163B
 | ACF9163B |          |          |          |          | 3E3F3132  | 92C62709
 | 92C62709 |          |          |          |          | 55E50096  | C723279F
 | C723279F |          |          |          |          | 70C327BE  | B7E00021
 | B7E00021 |          |          |          |          | 59E90C92  | EE090CB3
 | EE090CB3 |          |          |          |          | 74C723A3  | 9ACE2F10
 | 9ACE2F10 |          |          |          |          | A0F5123F  | 3A3B3D2F
 | 3A3B3D2F |          |          |          |          | 96C22A06  | ACF91729
 | ACF91729 |          |          |          |          | ACF9163B  | 00000112
 | 00000112 |          |          |          |          | 92C62709  | 92C6261B
 | 92C6261B | C6261B92 | B4F7AF4F | 02000000 | B6F7AF4F | C723279F  | 71D488D0
 | 71D488D0 |          |          |          |          | B7E00021  | C63488F1
 | C63488F1 |          |          |          |          | EE090CB3  | 283D8442
 | 283D8442 |          |          |          |          | 9ACE2F10  | B2F3AB52
 | B2F3AB52 |          | 370D6200 |          |          | 3A3B3D2F  | 0D365F2F
 | 0D365F2F |          |          |          |          | ACF91729  | A1CF4806
 | A1CF4806 |          |          |          |          | 00000112  | A1CF4914
 | A1CF4914 |          |          |          |          | 92C6261B  | 33096F0F
 | 33096F0F |          |          |          |          | 71D488D0  | 42DDE7DF
 | 42DDE7DF |          |          |          |          | C63488F1  | 84E96F2E
 | 84E96F2E |          |          |          |          | 283D8442  | ACD4EB6C
 | ACD4EB6C |          |          |          |          | B2F3AB52  | 1E27403E
 | 1E27403E |          |          |          |          | 0D365F2F  | 13111F11
 | 13111F11 |          |          |          |          | A1CF4806  | B2DE5717
 | B2DE5717 |          |          |          |          | A1CF4914  | 13111E03
 | 13111E03 |          |          |          |          | 33096F0F  | 2018710C
 | 2018710C | 18710C20 | ADA3FEB7 | 04000000 | A9A3FEB7 | 42DDE7DF  | EB7E1968...
...
...
...
```

Listing 49: Dio međurezultata proizveden pri ekspanziji 320 i 512-bitnog inicijalnog ključa

U rutini za ekspanziju ključeva u 256-bitnoj verziji AES algoritma, na svakih 16 bajta nakon poziva funkcije *keyExpBase*, četiri bajta ključa se izmiješaju pomoću AES - ove supstitucione kutije (transformacija *SubWord*). Iako je u ojačanoj verziji algoritma bilo mogućnosti da se S-Box poziva češće, pri čemu bi algoritam ostao potpuno funkcionalan, nastavljena je praksa da se S-Box poziva na svakih 16 bajta nakon poziva funkcije *keyExpBase*, da bi algoritam što manje odstupao od standardom definisanog.

Modifikovana verzija algoritma AES je implementirana na jasan i razumljiv način korišćenjem dva programska jezika (Java i Pascal) i to u dvije faze. U toku prve faze nisu korišćene ideje za ubrzanje izvršenja izvornog koda da bi on ostao što čitljiviji i razumljiviji. Na ovaj način su stvoreni preduslovi da ovakva verzija algoritma bude iskorišćena kao uvod i za upoznavanje sa standardom definisanim AES algoritmom. U slijedećoj fazi implementirana su proširenja ovoga algoritma u kojima su primjenjena ubrzanja njegovog izvršenja pomoću T-tabela prema idejama autora algoritma.

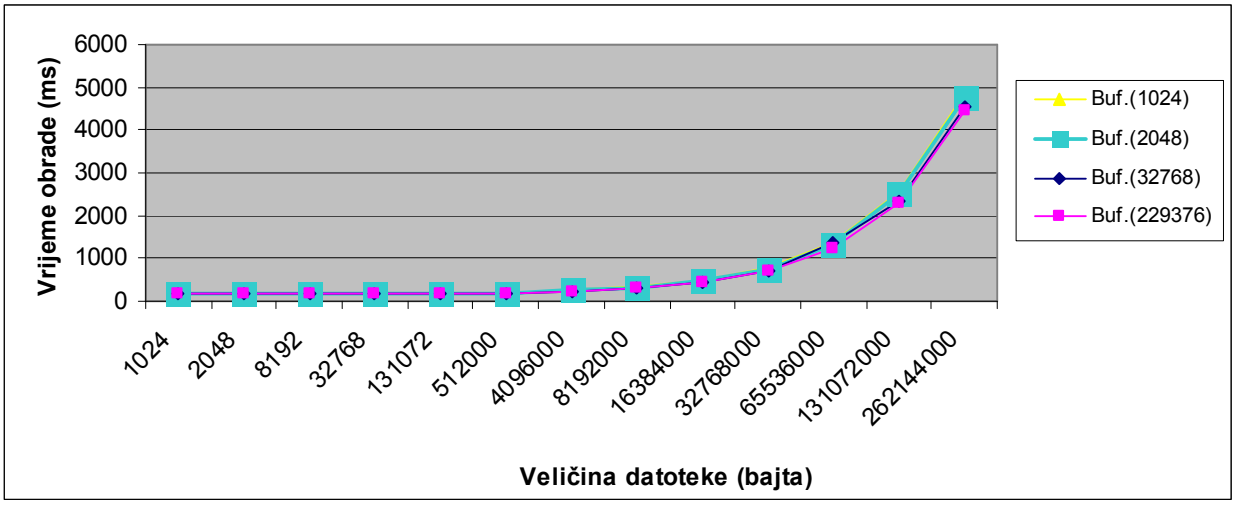
9.1.3 Rezultati mjerenja brzine izvršenja modifikovanih verzija algoritma kao trening podaci za kreiranje modela podataka

Kako je navedeno, implementacija modifikovane verzije AES algoritma razvijana je tokom dvije faze. U drugoj fazi implementirana je verzija bazirana na idejama autora algoritma (Daemen and Rijmen, 2002) i dr Briana Gladmana (Gladman, 2007). I ova aplikacija je kreirana tako da u radu koristi osam tabela sa unaprijed pripremljenim međurezultatima (T tabele). Izvršenje ove aplikacije karakteriše kompletno vrijeme koje joj je potrebno za šifrovanje određene datoteke, gdje je uključena inicijalizacija, UI operacije i vrijeme potrebno za obradu podataka.

Na Dell Inspiron N5110 testnoj platformi ponovo je obavljena serija mjerenja vremena koje je potrebno za šifrovanje različitih datoteka sa različitim veličinama bafera.

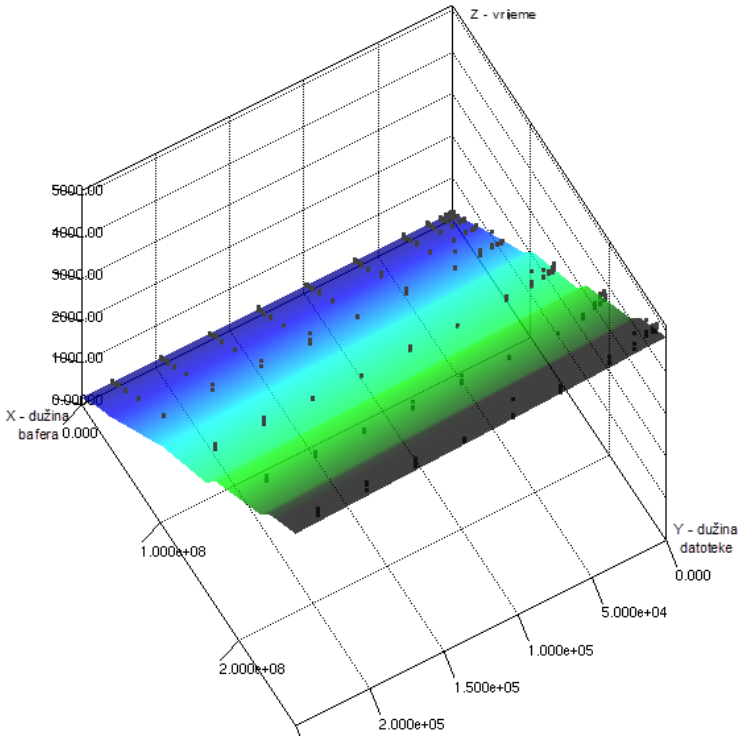
9.1.3.1.1 Rezultati mjerenja 32-bitnog proširenja

Kada se koristi modul EAesG za 32 bitno ECB šifrovanje uz komunikaciju putem neimenovanih cijevi uz minimalan skup poruka, trening podaci se grupišu kako je to prikazano na slici 61.



Slika 61: Uticaj dužina datoteka na vrijeme potrebno za šifrovanje (EAesGCECB321p, N5110), kom.pomoću neimenovanih cijevi uz minimalan skup poruka

Ako poredimo rezultate mjerenja za 128 bitnu ECB enkripciju pomoću jedne niti, sa rezultatima za 32 bitnu ECB enkripciju pomoću jedne niti, kada se komunikacija obavlja pomoću neimenovanih cijevi (obje u istom EaesG modulu) za datoteku dužine 256MB se u postiže očekivano bolje vrijeme i to za prosječno 1251 milisekundu. Na slici 62 je dat prikaz slaganja prediktivnog modela EAesGCECB321p sa trening podacima.



Slika 62: Vizuelni prikaz slaganja trening podataka i kreiranog EAesGCECB321p prediktivnog modela

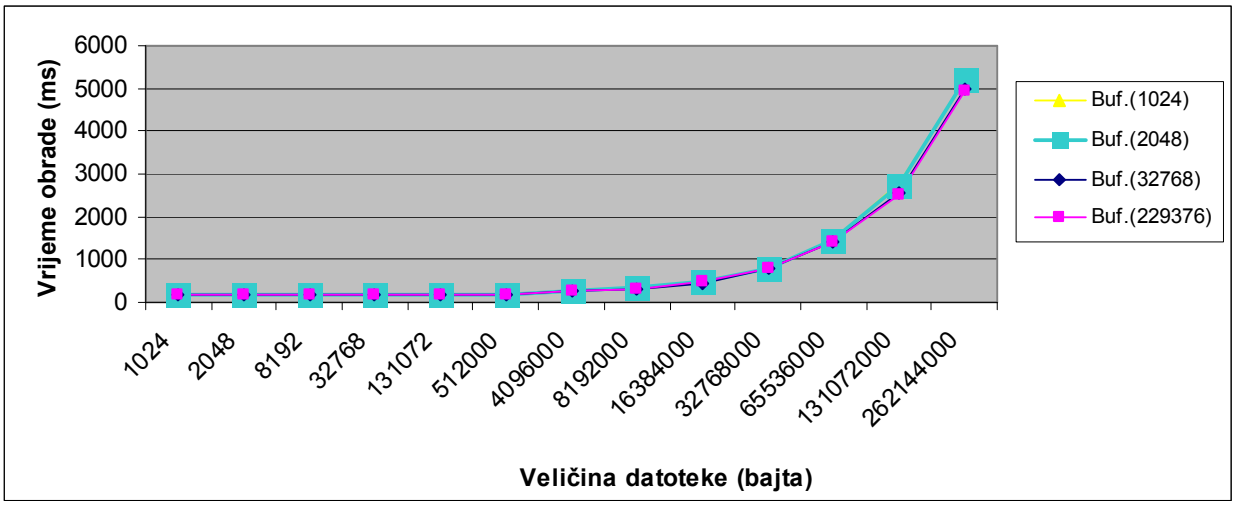
```
Koef.korelacije                  0.9995
Srednja apsolutna greška        23.8455
Srednja kvadratna greška        42.1405
Relativna apsolutna greška       2.1797%
Relativna kvadratna greška       3.0366%
Broj instanci                  168
```

Listing 50: Koef.korelacije, srednja apsolutna i srednja kvadratna greška, te relativna apsolutna i relativna kvadratna greška modela EAesGCECB321p, trening podaci

Na slici 62 dat je grafički prikaz iz kojega je moguće vidjeti kako se kreirani model EAesGCECB321p koji koristi neimenovane cijevi uklapa u trening podatke. U listingu 50 prikazani su podaci koji govore da navedeni model ima visok stepen slaganja sa pojavom koju opisuje (koef.korelacije = 0.9995) te da ima vrlo malu relativnu standardnu (2.1797%) i relativnu kvadratnu (3.0366%) grešku.

9.1.3.1.2 Rezultati mjerenja 64-bitnog proširenja

Dio trening podataka koji je izmjeren kada se koristio EAesG modul za 64 bitno ECB šifrovanje uz komunikaciju putem niemenovanih cijevi, prikazan je na slici 63.



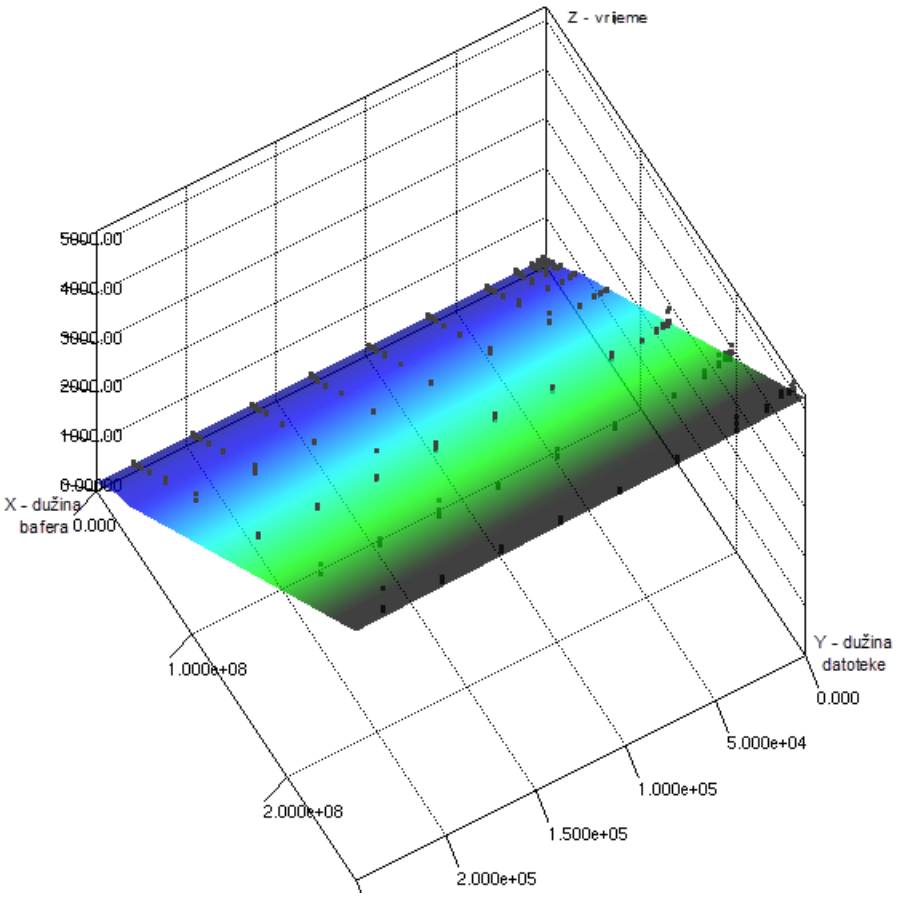
Slika 63: Uticaj dužina datoteka na vrijeme potrebno za šifrovanje (EAesGCECB641p, N5110), kom.pomoću neimenovanih cijevi uz minimalan skup poruka

I u slučaju 64 bitne enkripcije, poređenje dobijenih rezultata sa mjerenjima za 128 bitnu ECB enkripciju pomoću jedne niti, kada se komunikacija obavlja pomoću neimenovanih cijevi (obje u istom EaesG modulu) pokazuje da je 64 bitni modul očekivano brži i to za prosječno 882. milisekunde.

Na slici 64 je dat prikaz slaganja prediktivnog modela EAesGCECB641p sa trening podacima, a u listingu 51 prikazani su podaci koji govore da navedeni model ima vrlo visok stepen slaganja sa pojavom koju opisuje (koef.korelacije = 0.9995) te da ima vrlo malu relativnu standardnu (2.2088%) i relativnu kvadratnu (3.0623%) grešku.

Prethodno prikazana proširenja standardom definisanog AES algoritma ne bi trebalo koristiti u prikazanom obliku, jer se, prema preporukama agencije NIST (Barker and Roginsky, 2011), već od 2011. godine preporučuje da bi najslabija zaštita treba biti bar

112 bita, dok bi proces tranzicije na (najmanje) 112. bitnu zaštitu morao biti završen do kraja 2014. godine.



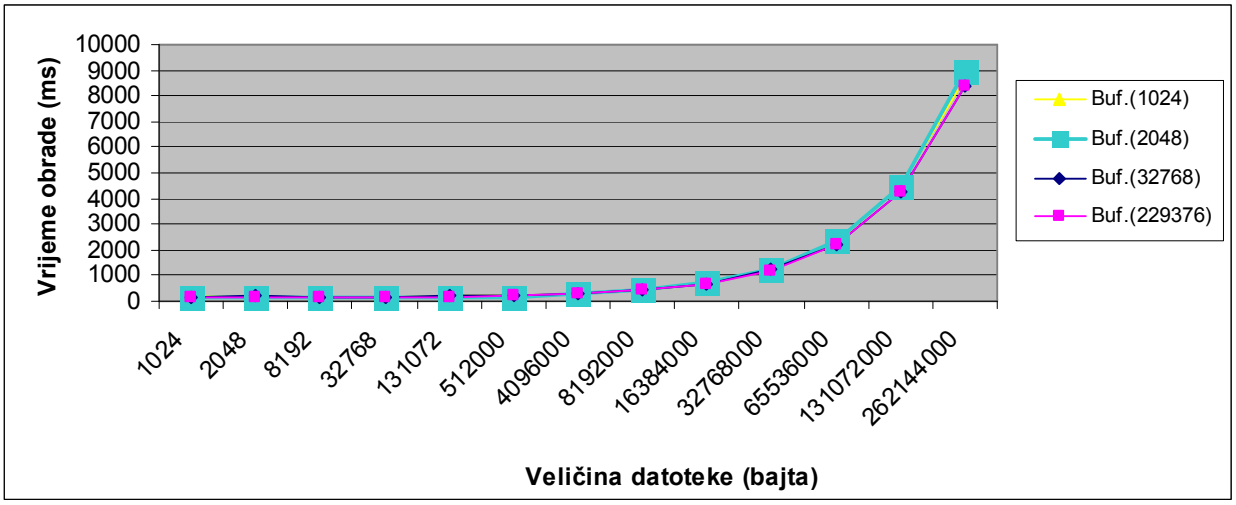
Slika 64: Vizuelni prikaz slaganja trening podataka i kreiranog EAesGCECB641p prediktivnog modela

Koef.korelacije	0.9995
Srednja apsolutna greška	26.1776
Srednja kvadratna greška	46.0506
Relativna apsolutna greška	2.2088%
Relativna kvadratna greška	3.0623%
Broj instanci	168

Listing 51: Koef.korelacije, srednja apsolutna i srednja kvadratna greška, te relativna apsolutna i relativna kvadratna greška modela EAesGCECB641p, trening podaci

9.1.3.1.3 Rezultati mjerenja 320-bitnog proširenja

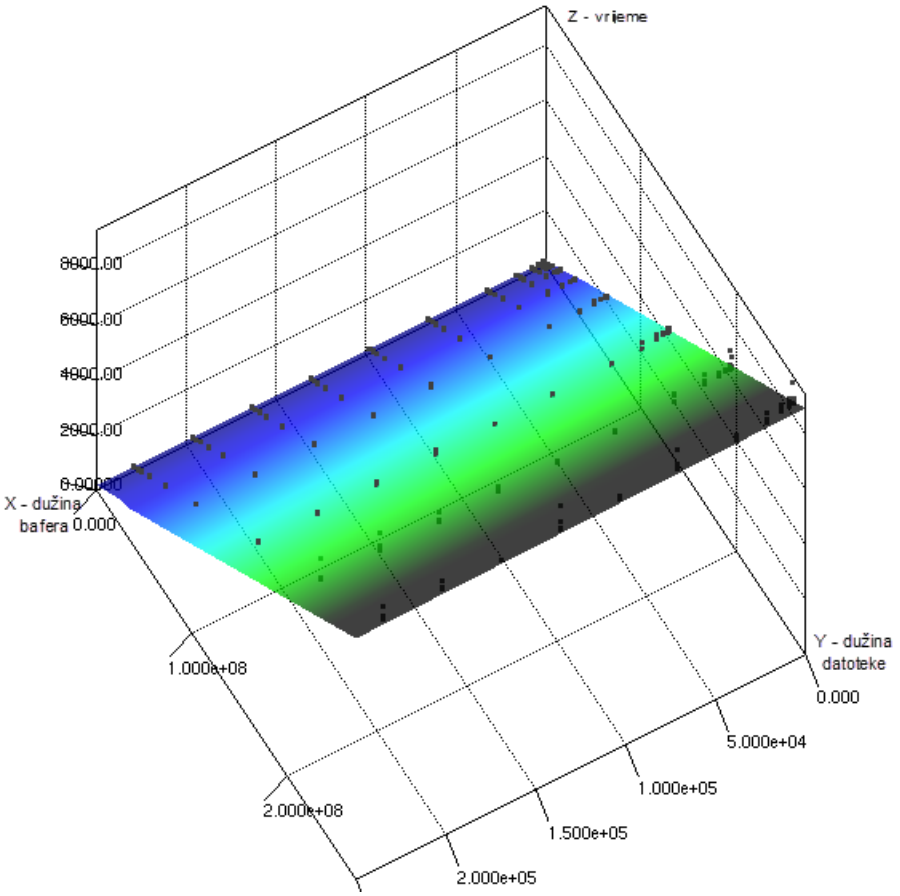
Dio trening podataka koji se dobiju kada se pomoću EAesG modula 320 bitnim ECB šifrovanjem uz komunikaciju putem neimenovanih cijevi uz minimalan skup poruka prikazan je na slici 65.



Slika 65: Uticaj dužina datoteka na vrijeme potrebno za šifrovanje (EAesGCECB3201p, N5110), kom.pomoću neimenovanih cijevi uz minimalan skup poruka

Rezultat koji je dobijen mjerenjem brzine šifrovanja 256MB datoteke pomoću 320-bitnog EAesGCECB3201p je u prosjeku sporiji za 2685 milisekundi nego šifrovanje EAesGCECB1281p modulom.

Kada se komunikacija između adaptibilnog virtuelnog kriptografskog fajl sistema i modula obavlja pomoću neimenovanih cijevi, na osnovu njih nastaje prediktivni model EAesGCECB3201p. Stepen slaganja ovog modela sa trening podacima dat je na slici 66 i u listingu 52. U listingu je naveden visok koeficient korelacije (0.9998) i mala relativna standardna (1.4274%) i relativna kvadratna (2.1725%) greška, što ukazuje na dobro kreiran model.



Slika 66: Vizuelni prikaz slaganja trening podataka i kreiranog EAesGCECB3201p prediktivnog modela

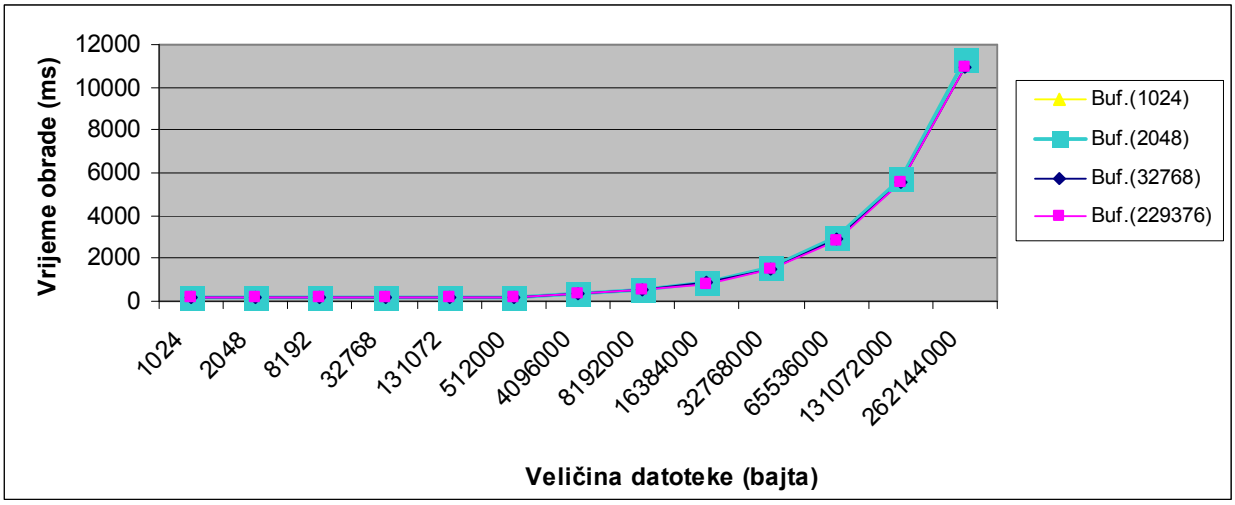
```
Koef.korelacije                  0.9998
Srednja apsolutna greška        29.3409
Srednja kvadratna greška        56.8053
Relativna apsolutna greška       1.4274%
Relativna kvadratna greška       2.1725%
Broj instanci                  168
```

Listing 52: Koef.korelacije, srednja apsolutna i srednja kvadratna greška, te relativna apsolutna i relativna kvadratna greška modela EAesGCECB3201p, trening podaci

9.1.3.1.4 Rezultati mjerenja 512-bitnog proširenja

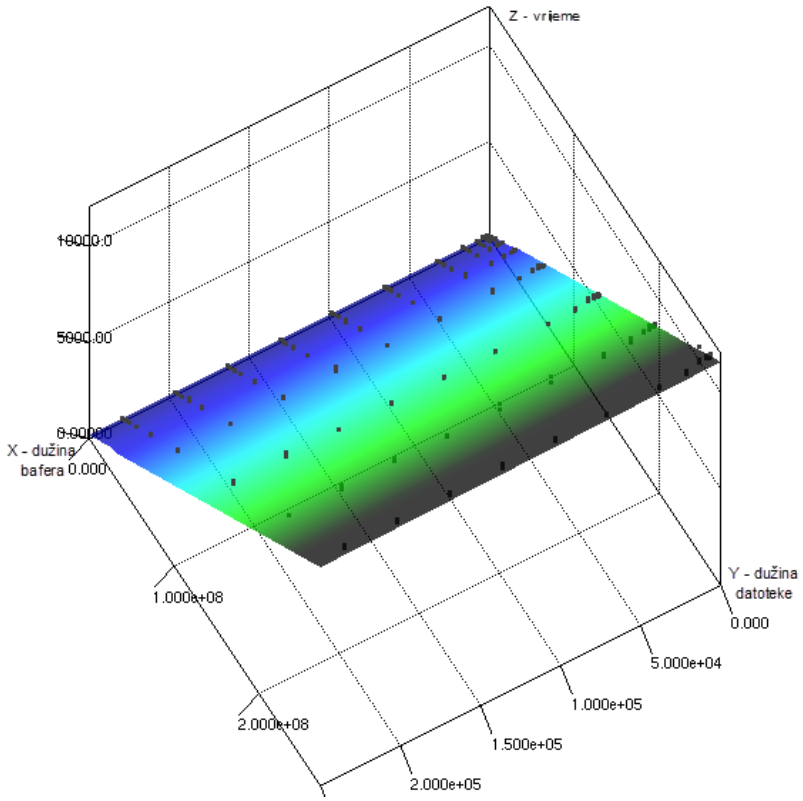
Kada se koristi modul EaesG za 512 bitno ECB šifrovanje uz komunikaciju putem neimenovanih cijevi, trening podaci se grupišu kako je to prikazano na slici 67.

Poređenje dobijenih rezultata 512 bitne enkripcije sa 128 bitnom enkripcijom pri šifrovanju datoteke dužine 256MB kada je u pitanju komunikacija putem neimenovanih cijevi uz minimalan skup poruka daje prosječnu razliku od 5233 milisekunde.



Slika 67: Uticaj dužina datoteka na vrijeme potrebno za šifrovanje (EAesGCECB5121p, N5110), kom.pomoću neimenovanih cijevi uz minimalan skup poruka

Na slici 68 i listingu 53 predstavljen je prediktivni model koji predviđa brzinu šifrovanja modula EAesG pomoću ECB moda, koristeći 512 bitni ključ, jedno jezgro i komunikaciju putem neimenovanih cijevi. Ovaj model ima veoma veliki koeficient korelacije (0.9999) sa malom relativnom standardnom (1.1508%) i relativnom kvadratnom 1.6159%) greškom.



Slika 68: Vizuelni prikaz slaganja trening podataka i kreiranog EAesGCECB5121p prediktivnog modela

```
Koef.korelacije                          0.9999
Srednja apsolutna greška                30.8981
Srednja kvadratna greška                55.1004
Relativna apsolutna greška               1.1508%
Relativna kvadratna greška               1.6159%
Broj instanci                          168
```

Listing 53: Koef.korelacije, srednja apsolutna i srednja kvadratna greška, te relativna apsolutna i relativna kvadratna greška modela EAesGCECB5121p, trening podaci

10 Razmatranja mogućnosti paralelizacije kriptografskog algoritma AES

Pojava različitih kriptografskih načina rada (*modes of operation*) bila je uslovljena uočenim nedostacima simetričnih blokovskih algoritama, jer će neki simetrični blok algoritam dva identična bloka koja se nalaze u okviru otvorenog teksta kojeg šifrujemo istim ključem šifrovati na identičan način - kao rezultat će dati identične blokove šifrata. Iako se u literaturi najčešće spominje 5 osnovnih ili tradicionalnih načina rada (Chakraborty and Rodriguez-Henriquez, 2009), u (Rogaway, 2011) se opisuje čak 17 ovakvih načina rada.

Kako je navedeno u poglavlju (4.10), zajednička karakteristika prikazanih kriptografskih načina rada je da oni bilo koji simetrični blok algoritam pretvaraju u protočni algoritam. Iako funkcionišu na bitno različite načine, CFB, OFB i CTR načini rada, kako je prikazano na listinzima i slikama u poglavljima (4.10.1, 4.10.2 i 4.10.3), koriste samo funkciju šifrovanja (*Ciph)*, dok ni u jednom prikazanom slučaju ne koriste funkciju dešifrovanja (*InvCiph*). Bilo koji blokovski algoritam koji se koristi sa CFB, OFB i CTR načinima rada funkcioniše kao tok ključeva (*keystream*). Navedni tok se u transformacijama šifrovanja pomoću XOR operacije kombinuje sa otvorenim tekstom da bi se proizveo šifrat i obrnuto, šifrat se pomoću XOR operacije kombinuje sa tokom ključeva da bi se ponovo proizveo otvoreni tekst.

Činjenica da prikazanim načinima rada nije potrebna funkcija dešifrovanja znači da bi se kao algoritam mogao upotrijebiti neki generator toka ključeva ili neki drugi kriptografski algoritam koji uopšte nema funkciju dešifrovanja (*InvCiph*). Ova ideja može se iskoristiti za školski prikaz modifikacije i paralelizacije pojedinih kriptografskih algoritama. Istraživanje prikazano u ovom poglavlju svoje izvore ima u objavljenim publikacijama (Damjanović and Simić, 2011), (Damjanović i Simić, InfoM46, 2013) te konačno u (Damjanović and Simić, 2015).

10.1.1.1 Paralelizacija - osnovni koncept

Algoritam AES može da koristi ključeve dužine 128, 192 i 256 bita. U slučaju najsnažnije zaštite koristi se inicijalni ključ dužine 256 bita koji se u toku postupka proširenja ključeva povećava na potrebnih 240 bajta. Zatim niz State u kojem se nalazi ulazni vektor (otvoreni tekst ili inicijalizacioni vektor kojeg šifrujemo, koji je veličine 16 bajta) prolazi kroz inicijalnu rundu, kada se miješa sa bajtima proširenog ključa. Potom se niz State transformiše i pomoću 14 rundi tokom kojih prolazi kroz transformacije *SubBytes*, *ShiftRows*, *MixColumns* i *AddRoundKey*, pri čemu se u finalnoj rundi ne koristi transformacija *MixColumns*. S druge strane, najslabija zaštita se postiže korišćenjem 128-bitnog ključa koji se proširuje na 178 bajta, kada se za transformisanje ulaznog niza *State* koriste navedene transformacije *SubBytes*, *ShiftRows*, *MixColumns* i *AddRoundKey* tokom 10 rundi.

Prema (Menezes et al. 1996), karakteristika pomenutih kriptografskih načina rada je da bilo koji sa njima primijenjeni blokovski algoritam funkcioniše kao tok ključeva (*keystream*). Kako iz (Dworkin, 2001) i (Chakraborty and Rodriguez-Henriquez, 2009) slijedi, za funkcionisanje CFB, OFB i CTR načina rada dovoljan je algoritam koji ima

samo funkciju šifrovanja i koji se uz pomoć inicijalizacionog vektora ponaša kao generator toka ključeva. Zahvaljujući njegovim karakteristikama na osnovu standardom definisanog algoritma AES je moguće, radi demonstracije, veoma jednostavno kreirati novi algoritam koji ima samo funkciju šifrovanja. Takav algoritam na ulazu uzima blok otvorenog teksta veličine 16 bajta i smiješta ga u matricu Stanje, te ključ dužine 256 bita. Ovakav inicijalni ključ se zatim dijeli na dva jednaka dijela veličine 128 bita. Ta dva dijela ključa se koriste da se dva puta šifruje blok otvorenog teksta 128-bitnom enkripcijom. Dobijena dva rezultirajuća bloka šifrata se zatim stapaju pomoću XOR operacije. Ako imamo P1, P2,..., PN blokova otvorenog teksta, šifrovanje *j*-tog bloka je dato pomoću slijedećeg pseudo koda:

$K1_{128}, K2_{128} = Split(K_{256})$
$O1_J = CIPH(P_J, K1_{128})$
$O2_J = CIPH(P_J, K2_{128})$
$C_J = O1_J \text{ XOR } O2_J$
gdje je:
P_J: j-ti blok otvorenog teksta,
Split – funkcija koja dijeli inicijalni ključ
$K1_{128}$, $K2_{128}$: dijelovi ključa koji su nastali na osnovu inicijalnog 256-bitnog ključa;
$O1_J$, $O2_J$: međurezultati šifrovanja paralelnim nitima;
C_J : konačan rezultat šifrovanja J-tog bloka.

Listing 54: Pseudo kod demonstracionog algoritma prilagođenog za paralelno šifrovanje

Ovakav demonstracioni algoritam za svoje funkcionisanje zahtjeva 256-bitni ključ koji se zatim dijeli na dva jednaka dijela koji se zatim daju dvjema rutinama koje vrše enkripciju. Svaki blok otvorenog teksta je dva puta šifrovan 128-bitnom enkripcijom, a zatim se dobijeni međurezultati stapaju pomoću XOR operacije u konačan rezultat šifrovanja bloka podataka. Ovakav demonstracioni algoritam je dobar kandidat za paralelizaciju, jer se postupak može implementirati tako da se umjesto dva uzastopna šifrovanja u okviru jedne programske niti (*thread*) koriste dvije niti koje bi paralelno vršile šifrovanje.

Međutim, pomoću prikazanog algoritma nije moguće dešifrovati jednom šifrovani otvoreni tekst, pa se zbog toga može koristiti samo kao jednosmjerna funkcija šifrovanja odnosno kao generator toka ključeva. A kako je već rečeno, kada se koriste sa CFB, OFB i CTR načinima rada, blokovski algoritmi funkcionišu kao generatori toka ključeva (*keystream*). Blokovskim algoritmima sa CFB, OFB i CTR načinima rada nije ni potrebna funkcija dešifrovanja, jer se ona realizuje samom konstrukcijom ovih kriptografskih načina rada (*mode of operation*).

OFB mod kreira protočni algoritam od blokovskog, kod kojeg jedan bit greške u šifratu proizvodi jedan bit greške u otvorenom tekstu. Sa jedne strane, ova osobina ga čini osjetljivim na napad obrtanjem bitova (*bit flipping attack*). Istovremeno, ova osobina čini OFB mod izuzetno korisnim za satelitske komunikacije u uslovima kada ima mnogo šuma u transmisionim kanalima (Tirro, 1993) (Cheltha and Velayutham, 2011) (Ramadevi, 2013).

10.1.1.2 Paralelno OFB šifrovanje sa odloženom sinhronizacijom niti u kom se koriste XEX i XE konstrukcije- formalna reprezentacija

U uvodu u poglavlje 10 pominjano je da za OFB režim rada nije potrebno implementirati transformacije dešifrovanja (*InvCipher*) podređenog algoritma. Zahvaljujući ovoj činjenici, moguće je kreirati novi algoritam koji implementira samo funkciju šifrovanja. Broj rundi u 256 i 128 bitnoj verziji AES algoritma je 14 i 10, respektivno. Moguće je kreirati algoritam koji će pomoću dvije niti šifrovati podatke tokom 10 rundi u svakoj niti. Pri ovakvom dizajnu, 256 bitni ključ se dijeli na dva 128 bitna dijela. Da bi se poboljšala njegova otpornost na kriptoanalizu, korišćen je izmjenljivi parametar (tweak) sličan onom koji je prikazan u (Rogaway, 2004) i (IEEE, 2008). U nastavku slijedi listing pseudo koda koji ilustruje navedenu ideju:

OFB paralelno šifrovanje
$K1,K2 = Split(K_{256})$

Prva nit
$EIV' = Ciph(IV, K2)$
$F'_1 = EIV'$
$T'_1 = \alpha^1 \otimes EIV'$
$In'_1 = T'_1 \oplus F'_1$
$O'_1 = Ciph(In'_1, K1)$
$C'_1 = T'_1 \oplus O'_1$

$T'_i = \alpha^i \otimes T'_{i-1}$ For i = 2,…,n;
$F'_i = C'_{i-1}$ For i = 2,…,n;
$In'_i = T'_i \oplus I'_i$ For i = 2,…,n;
$O'_i = Ciph(In'_i, K1)$ For i = 2,…,n;
$C'_i = T'_i \oplus O'_i$ For i = 2,…,n;

Druga nit
$EIV'' = Ciph(IV, K1)$
$F''_1 = EIV''$
$T''_1 = \alpha^1 \otimes EIV''$
$In''_1 = T''_1 \oplus F''_1$
$O''_1 = Ciph(In''_1, K2)$
$C''_1 = T''_1 \oplus O''_1$

$T''_i = \alpha^i \otimes T''_{i-1}$ For i = 2,…,n;
$F''_i = C''_{i-1}$ For i = 2,…,n;
$In''_i = T''_i \oplus I''_i$ For i = 2,…,n;
$O''_i = Ciph(In''_i, K2)$ For i = 2,…,n;
$C''_i = T''_i \oplus O''_i$ For i = 2,…,n;

$C_i = P_i \oplus C'_i \oplus C''_i$ For i = 1,…,n-1;
Zadnji blok
$C^*_N = P^*_N \oplus MSB_U(C^*_N \oplus C^*_N)$

OFB paralelno dešifrovanje

$K1,K2 = Split(K_{256})$

Prva nit
$EIV' = Ciph(IV, K2)$
$F'_1 = EIV'$
$T'_1 = \alpha^1 \otimes EIV'$
$In'_1 = T'_1 \oplus I'_1$
$O'_1 = Ciph(In'_1, K1)$
$C'_1 = T'_1 \oplus O'_1$

$T'_i = \alpha^i \otimes T'_{i-1}$ For i = 2,…,n;
$F'_i = C'_{i-1}$ For i = 2,…,n;
$In'_i = T'_i \oplus I'_i$ For i = 2,…,n;
$O'_i = Ciph(In'_i, K1)$ For i = 2,…,n;
$C'_i = T'_i \oplus O'_i$ For i = 2,…,n;

Druga nit
$EIV'' = Ciph(IV, K1)$
$F''_1 = EIV''$
$T''_1 = \alpha^1 \otimes EIV''$
$In''_1 = T''_1 \oplus I''_1$
$O''_1 = Ciph(In''_1, K2)$
$C''_1 = T''_1 \oplus O''_1$

$T''_i = \alpha^i \otimes T''_{i-1}$ For i = 2,…,n;
$F''_i = C''_{i-1}$ For i = 2,…,n;
$In''_i = T''_i \oplus I''_i$ For i = 2,…,n;
$O''_i = Ciph(In''_i, K2)$ For i = 2,…,n;
$C''_i = T''_i \oplus O''_i$ For i = 2,…,n;

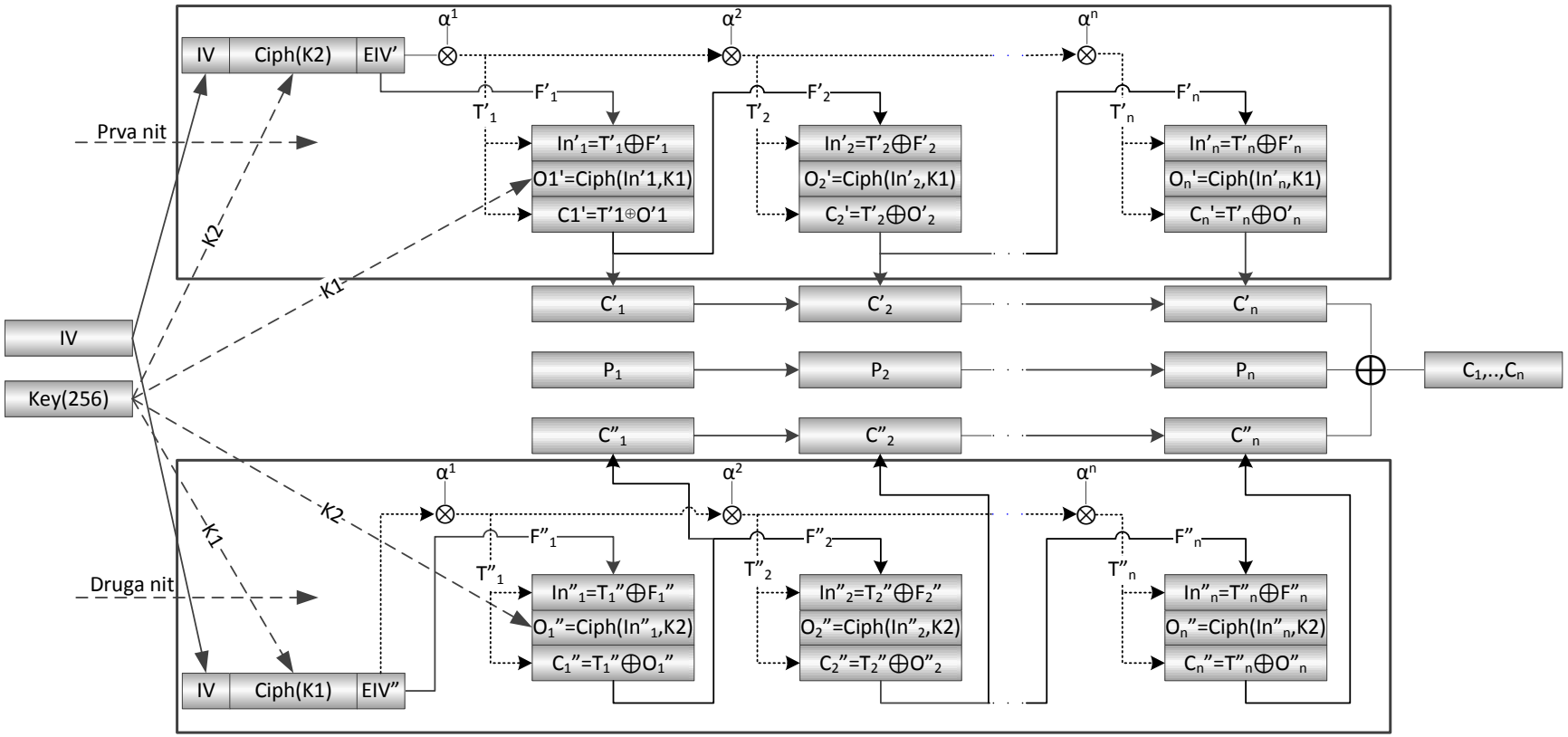
$P_i = C_i \oplus C'_i \oplus C''_i$ For i = 1,…,n-1;
Zadnji blok
$P^*_N = C^*_N \oplus MSB_U(C^*_N \oplus C^*_N)$

Gdje je:
IV- inicijalizacioni vektor,
Split - funkcija koja dijeli inicijalni ključ
K1, K2 - dijelovi ključa dužine 128 bita koji su nastali na osnovu inicijalnog ključa njegovim dijeljenjem
T'_i, T''_i: - izmjenjivi parametri (tweaks)
O'_i, O''_i: - međurezultati šifrovanja paralelnim nitima
F'_i, F''_i: - ulazni blokovi nastali povratnom spregom (Feedback)
P_i - i-ti blok otvorenog teksta,
C_i - i-ti blok šifrata,
P^*_N - zadnji blok otvorenog teksta; može btiti samo dio bloka,
C^*_N - zadnji blok šifrata teksta; može btiti samo dio bloka,
$MSB_M(X)$ - Najznačajnijih M bita od niza bita X,

Listing 55: Algoritam za paralelno OFB šifrovanje i dešifrovanje sa odloženom sinhronizacijom niti realizovan pomoću XEX konstrukcije

OFB način rada je moguće paralelizovati i modifikacijom algoritma koji je prikazan u listingu 54, ali bi on tada bio osjetljiv na *meet-in-the-middle* napad. Da bi pojačali njegovu otpornost, opisani algoritam je ojačan korišćenjem ideje iz XEX (XOR Encrypt XOR) konstrukcije. Predloženi način rada koristi par 128 bitnih ključeva za enkripciju pomoću dvije niti. Svaka nit koristi primarni 128 bitni ključ za enkripciju podataka i sekundarni 128 bitni ključ za koji se koristi da bi se kreirao izmjenjivi parametar (*tweak*) šifrovanjem inicijalizacionog vektora (IV). U prvoj niti, sekundarni ključ K2 se koristi za kreiranje *tweak-a*, a primarni ključ K1 za proces šifrovanja. U drugoj niti, kao primarni ključ za enkripciju sadržaja se koristi K2, dok se za kreiranje *tweak-a* koristi K1.

U prikazanom algoritmu koristi se IV i za kreiranje *tweak-a* i za funkcionisanje OFB načina rada. Osim po korišćenju dvije niti, ovo rješenje se razlikuje od onog koje je primjenjeno u XTS-AES algoritmu i po tome što se u XTS-AES-u za kreiranje *tweak-a* koristi broj sektora na hard disku.



Slika 69: Algoritam za paralelno OFB šifrovanje i dešifrovanje sa odloženom sinhronizacijom niti realizovan pomoću XEX konstrukcije

U prikazanom rješenju, vrijednost inicijalizacionog vektora IV je šifrovana pomoću sekundarnog ključa navedene niti. Na taj način se formiraju međurezultati EIV' i EIV". Dobijeni međurezultati su potom uzastopno transformisani pomoću redefinisane operacije množenja kako je to prikazano u poglavlju 4.11. Predloženo rješenje koristi isti metod za multiplikaciju kao XTS-AES, ali umjesto da koristi adresu sektora, ono koristi šifrovani IV (EIV' i EIV") pri čemu sekvencijalno uvećava eksponent *i* (α^i).

Kako je to prikazano u listingu 55 i na slici 69, obje niti koriste proceduru koja je slična XEX konstrukciji. U obje niti, prvi ulaz u glavnu funkciju šifrovanja je formiran XOR operacijom stapanjem šifrovanih vrijednosti inicijalizacionih vektora (EIV'= F'_1 i EIV"= $F"_1$) sa odgovarajućim vrijednostima izmjenjivih parametara (*tweaks* T'_1 i $T"_1$) koje su kreirane pomoću navedene procedure množenja. Svi ostali ulazi u glavnu proceduru šifrovanja su kreirani miješanjem izlaza koji je nastao enkripcijom prethodnog bloka sa odgovarajućim vrijednostima izmjenjivih parametara (*tweak*).

Međurezultati (O'_i i $O"_i$) koji čine izlaze iz glavne procedure za šifrovanje su formirani pomoću odgovarajućih ključeva. U obje niti, konačni rezultati šifrovanja jednog bloka kreirani su miješanjem međurezultata (O'_i i $O"_i$) sa odgovarajućim vrijednostima izmjenjivih parametara (*tweaks* T'_1 i $T"_1$).

Predloženi algoritam podržava enkripciju bafera čija dužina nije cjelobrojni umnožak veličine AES-ovog bloka (128 bita). Ako završni blok ima manje od 128 bita, preostali dio završnog bloka se obrađuje pomoću tehnike *ciphertext stealing* kako je opisano u (IEEE, 2008).

Konstrukcija predloženog algoritma je takva da izlazni šifrati iz obje niti moraju da budu generisani a potom i izmješani pomoću XOR operacije međusobno i sa otvorenim tekstom da bi proizveli blok šifrata. Ako bi algoritam bio implementiran na takav način da niti međusobno čekaju jedna drugu da bi izmiješale međurezultate nakon svakog bloka, to bi prema (Chen et al. 2011) i (Tarvo, 2014) imalo izrazito negativan uticaj na performanse. Da bi se ovo izbjeglo, algoritam treba implementirati na takav način da se međusobno miješanje međurezultata i otvorenog teksta obavi odjednom nakon šifrovanja većeg broja blokova. Na ovaj način se postiže da se negativan ticaj sinhronizacije niti umjesto nakon svakog bloka javlja nakon svakog n-tog bloka. Na navedeni način dobijeni su rezultati mjerenja brzine izvršenja paralelnog OFB moda sa XEX konstrukcijom koji su na testiranim računarima bili do 10.14% brži u odnosu na serijsko 256 bitno izvršenje AES algoritma u OFB modu.

U tekstovima (Liskov et al. 2002) i (Rogaway, 2004) se navodi da jednonitna XEX kostrukcija pruža dobru zaštitu protiv napada odabranim šifratom ili CCA napada (*chosen-ciphertext attack*). Ako bi se izbacio vanjski XOR, takva jednonitna konstrukcija bi postala osjetljiva na napad odabranim otvorenim tekstom ili CPA napad (*chosen-plaintext attack*). Prethodno opisani javnim parametrom modifikovani (*tweakable*) dvonitni OFB mod sa XEX konstrukcijom može veoma jednostavno da se pretvori u dvonitni algoritam sa XE kontrukcijom ako se izostavi vanjska XOR operacija. Korišćenjem XE konstrukcije sa dvije niti, postignuti su rezultati koji su do 19.35% brži u odnosu na serijsko 256 bitno izvršenje AES algoritma u OFB načinu rada. Listing 56 pseudo koda i slika 70 ilustruju javnim parametrom modifikovani (*tweakable*) paralelni OFB mod sa XE konstrukcijom.

OFB paralelno šifrovanje
$K1,K2 = Split(K_{256})$

Prva nit
$EIV' = Ciph(IV, K2)$
$F'_1 = EIV'$
$T'_1 = \alpha^1 \otimes EIV'$
$In'_1 = T'_1 \oplus F'_1$
$O'_1 = Ciph(In'_1, K1)$
$C'_1 = O'_1$

$T'_i = \alpha^i \otimes T'_{i-1}$ For i = 2,…,n;
$F'_i = O'_{i-1}$ For i = 2,…,n;
$In'_i = T'_i \oplus F'_i$ For i = 2,…,n;
$O'_i = Ciph(In'_i, K1)$ For i = 2,…,n;
$C'_i = O'_i$ For i = 2,…,n;

Druga nit
$EIV'' = Ciph(IV, K1)$
$F''_1 = EIV''$
$T''_1 = \alpha^1 \otimes EIV''$
$In''_1 = T''_1 \oplus F''_1$
$O''_1 = Ciph(In''_1, K2)$
$C''_1 = O''_1$

$T''_i = \alpha^i \otimes T''_{i-1}$ For i = 2,…,n;
$F''_i = O''_{i-1}$ For i = 2,…,n;
$In''_i = T''_i \oplus F''_i$ For i = 2,…,n;
$O''_i = Ciph(In''_i, K2)$ For i = 2,…,n;
$C''_i = O''_i$ For i = 2,…,n;

$C_i = P_i \oplus C'_i \oplus C''_i$ For i = 1,…,n-1;
Zadnji blok
$C^*_N = P^*_N \oplus MSB_U(C'^*_N \oplus C''^*_N)$

OFB paralelno dešifrovanje
$K1,K2 = Split(K_{256})$

Prva nit
$EIV' = Ciph(IV, K2)$
$F'_1 = EIV'$
$T'_1 = \alpha^1 \otimes EIV'$
$In'_1 = T'_1 \oplus F'_1$
$O'_1 = Ciph(In'_1, K1)$
$C'_1 = O'_1$

$T'_i = \alpha^i \otimes T'_{i-1}$ For i = 2,…,n;
$F'_i = O'_{i-1}$ For i = 2,…,n;
$In'_i = T'_i \oplus F'_i$ For i = 2,…,n;

$O'_i = Ciph(In'_i, K1)$ For i = 2,…,n;
$C'_i = O'_i$ For i = 2,…,n;

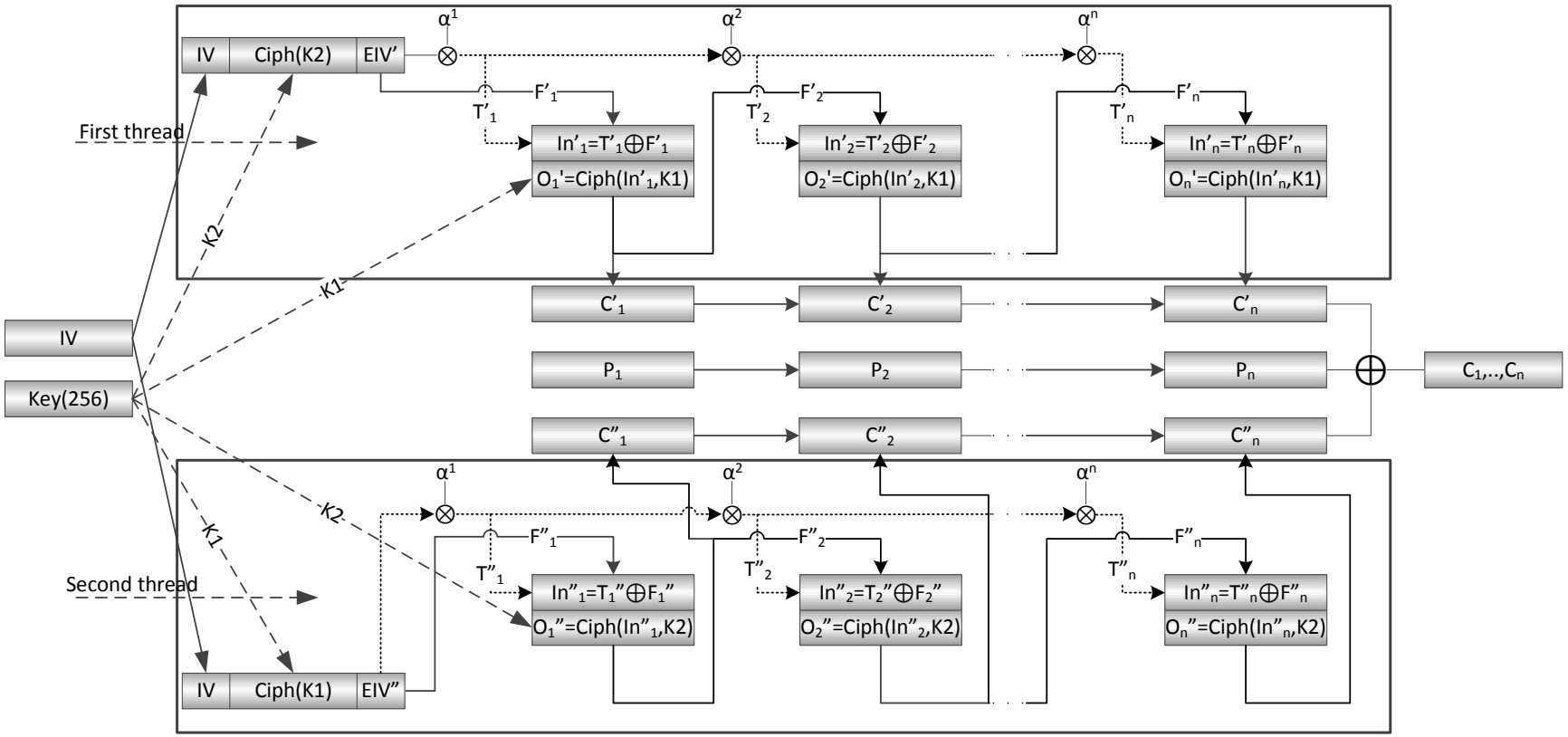
Druga nit
$EIV'' = Ciph(IV, K1)$
$F''_1 = EIV''$
$T''_1 = \alpha^1 \otimes EIV''$
$In''_1 = T''_1 \oplus F''_1$
$O''_1 = Ciph(In''_1, K2)$
$C''_1 = O''_1$

$T''_i = \alpha^i \otimes T''_{i-1}$ For i = 2,…,n;
$F''_i = O''_{i-1}$ For i = 2,…,n;
$In''_i = T''_i \oplus F''_i$ For i = 2,…,n;
$O''_i = Ciph(In''_i, K2)$ For i = 2,…,n;
$C''_i = O''_i$ For i = 2,…,n;

$P_i = C_i \oplus C'_i \oplus C''_i$ For i = 1,…,n-1;
Zadnji blok
$P^*_N = C^*_N \oplus MSB_U(C'^*_N \oplus C''^*_N)$

Gdje je:
IV- inicijalizacioni vektor,
Split - funkcija koja dijeli inicijalni ključ
K1, K2 - dijelovi ključa dužine 128 bita koji su nastali na osnovu inicijalnog ključa njegovim dijeljenjem
T'_i, T''_i: - izmjene (tweaks)
O'_i, O''_i: - međurezultati šifrovanja paralelnim nitima
C'_i, C''_i: - međurezultati šifrovanja paralelnim nitima
F'_i, F''_i: - ulazni blokovi nastali povratnom spregom
In'_i, In''_i: - ulazni blokovi nastali miješanjem sa vrijednošću *tweak-a*
P_i - i-ti blok otvorenog teksta,
C_i - i-ti blok šifrata,
P^*_N - zadnji blok otvorenog teksta; može btiti samo dio bloka,
C^*_N - zadnji blok šifrata teksta; može btiti samo dio bloka,
$MSB_M(X)$ - Najznačajnijih M bita od niza bita X,

Listing 56: Algoritam za paralelno OFB šifrovanje i dešifrovanje sa odloženom sinhronizacijom niti realizovan pomoću XE konstrukcije



Slika 70: Algoritam za paralelno OFB šifrovanje i dešifrovanje sa odloženom sinhronizacijom niti realizovan pomoću XE konstrukcije

U prethodnom poglavlju opisani su koncepti pomoću kojih je moguće paralelizovati izvršenje algoritma AES u OFB načinu rada. Međutim, prikazano rješenje može se primijeniti na bilo koji blokovski algoritam koji dozvoljava različite dužine ključa kao i brojeve rundi, poput algoritama Camellia, CLEFIA, RC6 itd.

10.1.2 Neki aspekti paralelizacije u Java okruženju

U okviru ovoga istraživanja su radi prikupljanja empirijskih podataka pomoću programskog jezika Java kreirani moduli koji koriste javnim parametrom modifikovane (*tweakable*) paralelne OFB načine rada sa XEX i sa XE konstrukcijama. Kako se navodi u (Georges et al. 2007), (Georges et al. 2008), (Boyer1, 2008), (Boyer2, 2008) i (Krieger and Strout, 2010), višenitne Java aplikacije zavise od velikog broja faktora, poput JIT (*Just In Time*) kompajlera, mehanizma za raspodjelu niti (*Thread Scheduling*), mehanizma za sakupljanje smeća (*garbage collection*), broja pokrenutih procesa i sinhronizacije niti. U (Chen et al. 2011) autori ističu tri najvažnija pitanja koja određuju performanse višenitne Java aplikacije. Na prvom mjestu, takmičenje niti (*lock contention*) može u velikoj mjeri da naruši performanse takvog sistema. Kao slijedeću tačku, autori ističu zastoje u radu memorije, odnosno izgubljene memorijske cikluse (*memory stall cycles*) koji su uzrokovani promašajima u okviru drugostepenog L2 keša i vremenu potrebnom za realizaciju međusobnih transfera između različitih keš memorija. Loše performanse memorijskog sistema mogu bitno umanjiti profit koji se ostvari pomoću paralelnog izvršavanja koda na više jezgara i niti. U publikaciji (Tarvo and Reiss, 2014) se navodi da operacije vezane za sinhronizaciju niti i limitirani računarski resursi u smislu broja jezgara mogu da proizvedu kompleksne zavisnosti između višenitne aplikacije i njenih performansi. U ovoj publikaciji autori dodatno ističu da mehanizam skupljanja smeća može da postane uticajan faktor za degradaciju performansi Java aplikacija. U publikacijama (Zhang et al. 2007) i (Gu et al. 1999) se navodi da kreiranje niti negativno utiče na performanse Java aplikacija.

U nekom realnom sistemu u radu vrlo često je pokrenut veći broj procesa. Prema (Oracle JPT, 2015), svaka Java aplikacija posjeduje najmanje jednu nit koja se naziva glavna nit. Ali ako se broje sistemske niti koje npr. upravljaju memorijom ili razmjenom signala, aplikacija može da ima nekoliko niti koje su posvećene njenom izvršenju. Prema (Chen et al. 2011) postoji 6 tipova JVM niti: *vm_thread*, *cgc_thread*, *pgc_thread*, *java_thread*, *compiler_thread* i *watcher_thread*. Prema (Goetz, 2010) aktivne niti troše sistemske resurse. Kada se u sistemu nalazi više aktivnih niti nego procesora, neke niti će biti besposlene. Mnogo besposlenih niti vezuje mnogo memorije, a mnogo zauzete memorije zadaje mnogo posla skupljaču smeća (*garbage collecotor*). Međutim, mnogo aktivnih niti koje se nadmeću za resurse, posebno za CPU, može ozbiljno da utiče na performanse. Drugim riječima, ako u sistemu ima dovoljno niti koje su u potpunosti zaposlile jezgra procesora, svako dalje dodavanje niti (*threads*) neće pomoći, već može samo da uspori izvršenje postojećih niti.

10.1.3 Višenitni Java modul sa izmjenljivim parametrom nastao na osnovu OFB rješenja sa odloženom sinhronizacijom niti za paralelizaciju AES algoritma

U listinzima pseudo koda 55 i 56 i na slikama 69 i 70 predstavljeni su prijedlozi za nove kriptografske načine rada. U njima su prikazani javnim parametrom modifikovani (*tweakable*) paralelni OFB mod sa XE konstrukcijom i javnim parametrom modifikovani (*tweakable*) paralelni OFB mod sa XEX konstrukcijom. Oba algoritma u svom izvršenju pokušavaju da nadmaše serijsko izvršenje 256 bitnog OFB načina rada.

U okviru ovoga istraživanja je radi prikupljanja emprijskih podataka pomoću programskog jezika Java kreiran modul za jednonitno 256 bitno OFB šifrovanje i dešifrovanje, te dva modula koja koriste javnim parametrom modifikovane (*tweakable*) paralelne OFB modove sa XEX i sa XE konstrukcijama. Pri tome je za paralelizaciju korišćena dva puta standardna 128-bitna enkripcija tokom 10 rundi po svakoj niti. Implementacija AES rutine koja vrši 128 i 256 bitno šifrovanje u OFB modu kreirana je na standardan način, bez korišćenja XEX i XE konstrukcija, kako je prikazano u poglavlju 4.10.2.

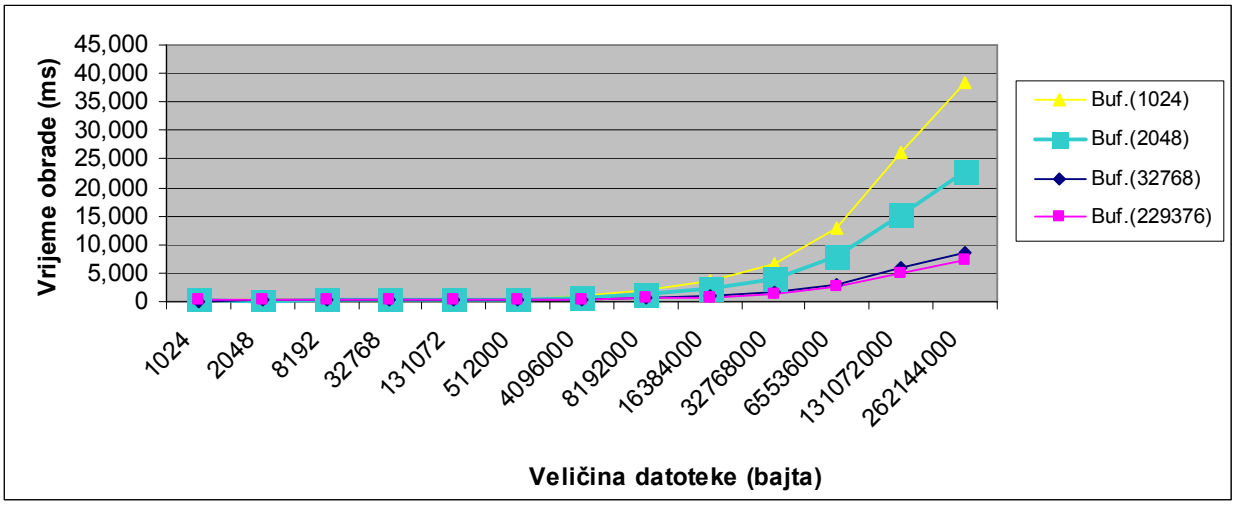
Svi testirani procesi su kreirani, pokretani i zaustavljani na identičan način. Izvorni kod svih rješenja je bio gotovo identičan, sa minimalnim razlikama koje se odnose na potrebu stapanja bafera u kojima se nalaze međurezultati sa baferima u kojima je otvoreni tekst odnosno šifrat. Svaka pojedina nit je, po učitavanju podataka iz datoteke u bafer koji se nalazi u memoriji, uzastopno kreirana i pokretana za svaki novi bafer. Nastavak rada niti (*threads*) je zavisio od toga da li su ostale niti koje učestvuju u šifrovanju završile sa radom.

Ideja upotrebe OFB moda kao načina za paralelizaciju izvršenja algoritma AES opisana je u listinzima pseudo koda 55 i 56. U implementaciji je korišćen *Java cached thread pool* mehanizam (klasa *Executor*, *newCachedThreadPool* metod) koji pokreće parlelno izvršenje nekoliko niti odjednom pa zatim čeka završetak svih niti da bi nastavio dalje. Slike 69 i 70 prikazuju kako se može izvesti paralelno 256 bitno šifrovanje pomoću dvije programske niti. Sa datih dijagrama se može vidjeti da, ako bi se stapanje međurezultata i otvorenog teksta (odnosno sinhronizacija niti) vršilo nakon svakog bloka od 16 bita, takvo rješenje ne bi donijelo bitna poboljšanja performansi u odnosu na čisto serijsko izvršenje. Za nastavak izvršenja algoritma neophodno je obezbjediti da su sve niti završile svoje izvršenje. Ako se to dešava svakih 16 bajta, tada nakon svakog bloka dolazi do pominjanog gubitka vremena koje se potroši na međusobno čekanje niti.

Da bi bilo moguće postići poboljšanje performansi, moralo se pristupiti drugačijem rješenju kod kojega je odložen momenat stapanja međurezultata koje obrađuju različite niti. Kod ovog rješenja svaka od navedenih niti se *N* puta izvršava nad ulaznim blokovima koji predstavljaju izlaze šifrovanja prethodnih blokova te na takav način svaka nit stvara bafere sa međurezultatima dužine 16**N*. Tek nakon *N* šifrovanja koje se obavi u svim programskim nitima, ovakvi baferi se pomoću XOR operacije kombinuju međusobno i sa otvorenim tekstom, zbog čega je tek u tom momentu potrebno obezbjediti istovremeni završetak rada svih niti. Pored toga, potrebno je obezbjediti da i bafer u kojem se nalazi otvoreni tekst bude dužine

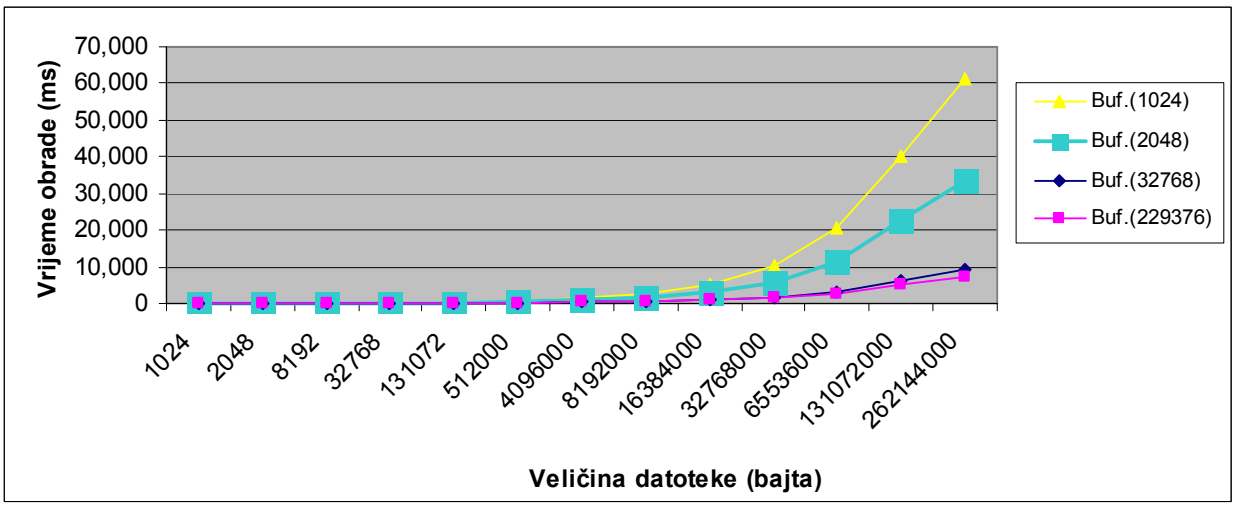
$$16*(n-1) + B;\ \ n = 1, 2, 3,...,N;\ \ 0 <= B < 16 \qquad (78)$$

da bi se moglo uspješno izvršiti njegovo stapanje sa međurezultatima šifrovanja pojedinih niti. Dakle, sa povećanjem dužine bafera praktično se postiže efekat odlaganja momenta sinhronizacije niti.



Slika 71: Uticaj dužina datoteka na vrijeme potrebno za šifrovanje (AesCOFB2561p / 1x256, N5110), kom.pomoću neimenovanih cjevi sa minimalnim skupom poruka.

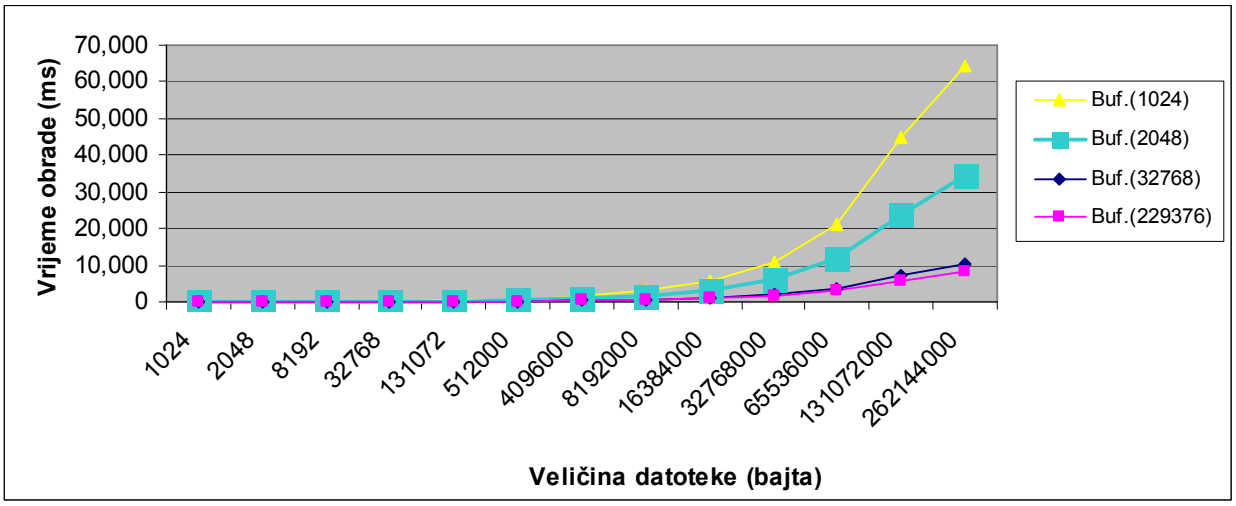
Za prikupljanje empirijskih podataka, pored ranije opisane N5110 mašine sa Core i7-2630QM procesorom koji ima 4 fizička i 8 logičkih jezgara i sa 6 GB operativne memorije sastavljene od DDR3 1333 čipova, zbog specifičnih potreba istraživanja koja su bila vezana za brzinu operativne memorije, korišćena je Toshiba Satellite L50-A-1D5 mašina sa Core i7-4700MQ procesorom i 8GB operativne memorije koja se sastojala od DDR3-1600 memorijskih modula. Na drugoj mašini, koja će u nastavku biti referencirana kao L50A bio je instaliran Windows 8.1 operativni sistem, te Java(TM) SE Runtime Environment (build 1.8.0_25-b18) i Java HotSpot(TM) 64-bit server (build 25.25-b02). Ovaj modul je testiran na isti način na koji je vršeno testiranje svih ostalih modula.



Slika 72: Uticaj dužina datoteka na vrijeme potrebno za šifrovanje (AesXE_COFB2562p / 2x128, N5110), kom.pomoću neimenovanih cjevi sa minimalnim skupom poruka.

Na N5110 mašini sa sporijim procesorom i sporijom operativnom memorijom, pri najvećoj dužini bafera i datoteke, paralelno 2x128 bitno OFB šifrovanje sa odloženom sinhronizacijom niti u kom se koristi XE konstrukcija pomoću dvije niti postiže 13% bolje vrijeme u odnosu na 256 bitno OFB šifrovanje jednom niti.

U svim izvršnim mjerenjima primjetno je da sa redukcijom veličine bafera postepeno dolazi do sve veće degradacije performansi paralelnog OFB rješenja sa XE konstrukcijom koje koristi dvije niti u odnosu na jednonitno 256 bitno OFB šifrovanje.



Slika 73: Uticaj dužina datoteka na vrijeme potrebno za šifrovanje (AesXEX_COFB2562p / 2x128, N5110), kom.pomoću neimenovanih cjevi sa minimalnim skupom poruka.

Paralelno OFB šifrovanje sa odloženom sinhronizacijom niti u kom se koristi XEX konstrukcija u svim mjerenjima pokazuje lošije rezultate u odnosu na 256 bitno OFB šifrovanje jednom niti. U tabeli 21 prikazani su rezultati šifrovanja datoteke dužine 128MB pomoću sva tri načina u kojoj je vidljivo da paralelno rješenje sa XEX konstrukcijom uvijek daje lošije rezultate u odnosu na jednonitno šifrovanje, dok rješenje koji koristi XE konstrukciju za paralelno šifrovanje pri dužinama bafera većim od 32768 postiže bolje rezultate u odnosu na jednonitno 256 bitno OFB šifrovanje.

	512000	64000	20000	16000	14000	12000	8000
1x256 OFB (ms)	4128.4	4715.8	5510.33	5726.8	5797	5835.5	6439.5
2x128 OFB XEX (ms)	4219	4895	6445.4	6794	7077.8	7275	8367
ubrzannje	0.979	0.963	0.855	0.843	0.819	0.802	0.770
%	-2.15	-3.66	-14.51	-15.71	-18.10	-19.79	-23.04
1x 128 XEX (ms)	3159.2	3741.4	4611.7	4811.9	4826.8	4980	5699.8
1x 256 XEX (ms)	4041.2	4749.9	5685	5908.2	5997.8	6193.8	6699

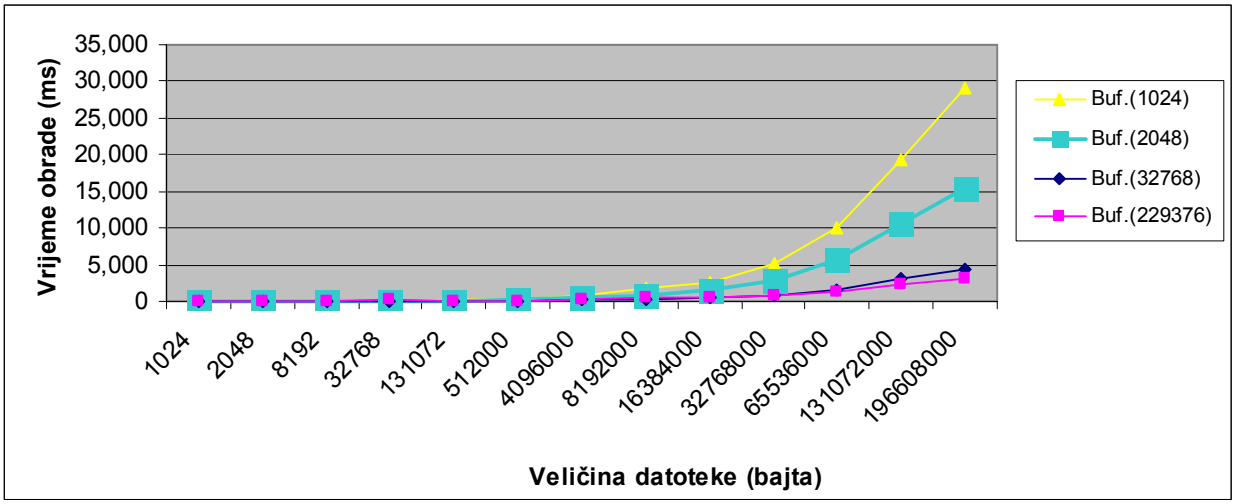
Tabela 21: Uporedni rezultati šifrovanja datotke dužine 128MB bajta pomoću jednonitnih i dvonitnih XEX OFB rješenja na N5110 mašini

	512000	64000	20000	16000	14000	12000	8000
1x256 OFB (ms)	4128.4	4715.8	5510.33	5726.8	5797	5835.5	6439.5
2x128 OFB XE (ms)	3654.2	4290	5632	5960.4	6198.8	6537.6	7520.4
ubrzannje	1.130	1.099	0.978	0.961	0.935	0.893	0.856
%	***12.98***	9.93	-2.16	-3.92	-6.48	-10.74	-14.37

1x 128 XE (ms)	3107.6	3787.6	4745.2	4881.2	4958.3	5120.6	5672.4
1x 256 XE (ms)	4058.5	4771.8	5644.1	6033	6047.1	6115.5	6691

Tabela 22: Uporedni rezultati šifrovanja datotke dužine 128MB bajta pomoću jednonitnih i dvonitnih XE OFB rješenja na N5110 mašini

Rezultati koji su postignuti na L50A mašini koja ima brži procesor i operativnu memoriju su bitno različiti.



Slika 74: Uticaj dužina datoteka na vrijeme potrebno za šifrovanje (AesCOFB2561p / 1x256, L50A), kom.pomoću neimenovanih cjevi sa minimalnim skupom poruka.

U tabeli 24 su prikazani rezultati šifrovanja datoteke dužine 128MB u slučajevima kada se na L50A mašini koristi paralelno 2x128 bitno OFB šifrovanje sa odloženom sinhronizacijom niti sa XEX konstrukcijom pomoću dvije niti i 256 bitno OFB šifrovanje jednom niti.

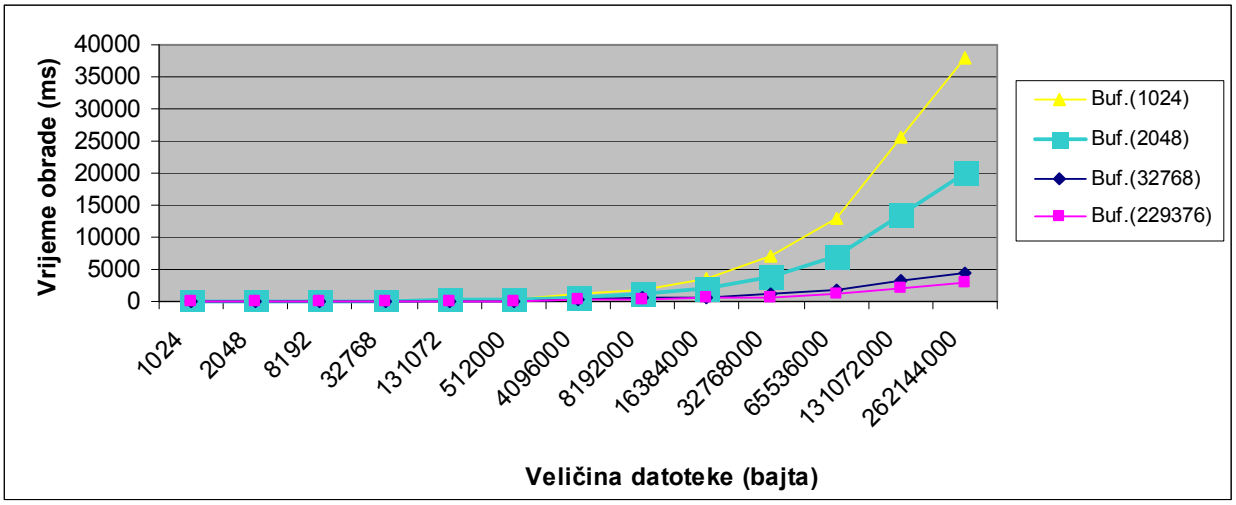
	512000	64000	20000	16000	14000	12000	8000
1x256 OFB (ms)	1653.2	2380	3093.6	3274.2	3421.83	3530	4441
2x128 OFB XEX (ms)	1552.6	2160.8	2972.8	3250	3408.2	3665	4715.2
ubrzanje	1.065	1.101	1.041	1.007	1.004	0.963	0.942
%	6.48	***10.14***	4.06	0.74	0.40	-3.68	-5.82
1x 128 XEX (ms)	1290.6	1948.4	2603.9	2667.7	2846	3141.2	3804.3
1x 256 XEX (ms)	1611.1	2341.4	3047.5	3325.5	3472.5	3673.3	4747

Tabela 23: Uporedni rezultati šifrovanja datotke dužine 128MB pomoću jednonitnog i dvonitnih OFB rješenja na L50A mašini

	512000	64000	20000	16000	14000	12000	8000
1x256 OFB (ms)	1653.2	2380	3093.6	3274.2	3421.83	3530	4441
2x128 OFB XE (ms)	1458	1994.2	2850.8	3141.6	3347.2	3673.8	4667.8
ubrzanje	1.134	1.193	1.085	1.042	1.022	0.961	0.951
%	13.39	***19.35***	8.52	4.22	2.23	-3.91	-4.86
1x 128 XE (ms)	1268.6	1951.8	2480.3	2778.7	2840.6	3068.7	3856.8
1x 256 XE (ms)	1872.1	2345.9	3083.6	3213.2	3440.4	3683.5	4678.6

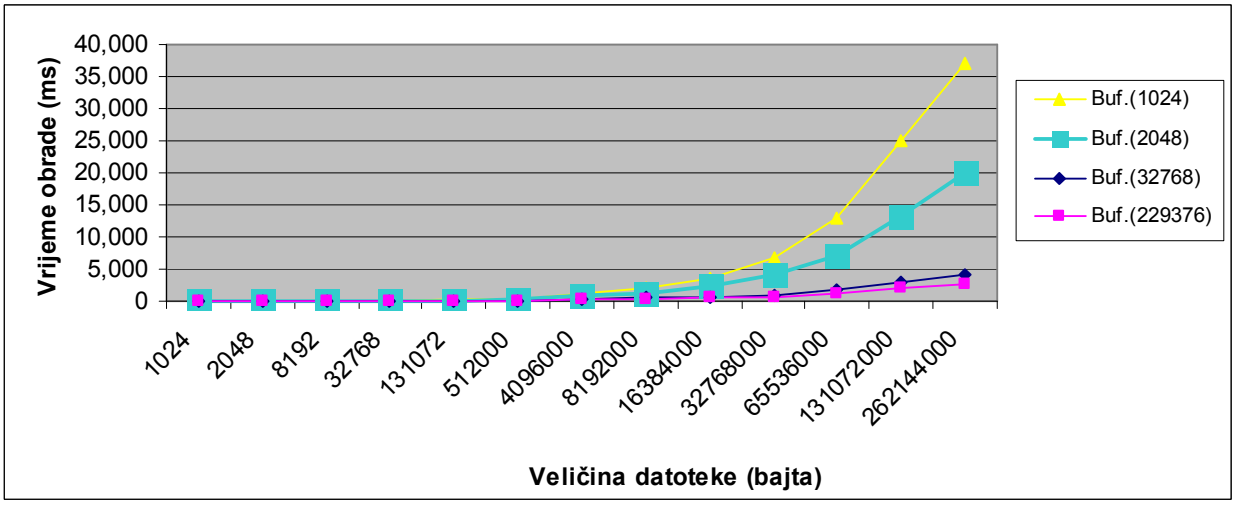
Tabela 24: Uporedni rezultati šifrovanja datotke dužine 128MB pomoću jednonitnog i dvonitnih OFB rješenja na L50A mašini

U zavisnosti od veličine ulaznog niza, rezultati koji su dobijeni na L50A testnoj platformi pokazuju poboljšanje performansi do 10.14% u slučaju kada se koristi paralelno OFB šifrovanje sa odloženom sinhronizacijom niti uz XEX konstrukciju u odnosu na jednonitno šifrovanje.



Slika 75: Uticaj dužina datoteka na vrijeme potrebno za šifrovanje (AesXEX_COFB2562p / 2x128, L50A), kom.pomoću neimenovanih cjevi sa minimalnim skupom poruka.

Na istoj testnoj mašini paralelno OFB šifrovanje sa korišćenjem XE konstrukcije pokazuje poboljšanje performansi do 19.35% u odnosu na jednonitno OFB šifrovanje.



Slika 76: Uticaj dužina datoteka na vrijeme potrebno za šifrovanje (AesXE_COFB2562p / 2x128, L50A), kom.pomoću neimenovanih cjevi

Kako je pomenuto u uvodu u poglavlje 10, takmičenje niti (*lock contention*) i loše performanse operativne memorije mogu u velikoj mjeri da naruše performanse neke višenitne aplikacije.

Operatvna memorija u prvoj testnoj platformi (N5110) je sporija od one koja je ugrađena u L50A mašini. Kako je prikazano u listingu 55 i na slici 69, paralelni XEX OFB mod se pri šifrovanju prvog bloka razlikuje od jednonitnog rješenja po tome što svaka nit provodi

dodatnu enkripciju jednog bloka, nakon koje slijedi jedno dodatno XTS množenje i dvije dodatne XOR operacije kojim se kombinuje vrijednost izmjenjivog parametra (*tweak*) sa ulazom u funkciju šifrovanja i sa izlazom iz nje. Pri šifrovanju svakog slijedećeg bloka, svaka nit obavlja jedno dodatno XTS množenje i dvije dodatne XOR operacije kojim se kombinuje vrijednost izmjenjivog parametra (*tweak*) sa ulazom u funkciju šifrovanja i sa izlazom iz nje.

U slučaju šifrovanja pomoću paralelnog XE moda, izostavljeno je stapanje izlaznog bloka i izmjenjivog parametra (*tweak*) pomoću XOR operacije. Kako se operacija stapanja obavljala pri šifrovanju svakog bloka, izostavljanjem ove operacije kod paralelnog XE OFB moda postižu se značajno bolji rezultati nego u slučaju paralelnog XEX OFB moda.

Kako je navedeno u uvodu u poglavlje 10, sinhronizacija niti prema (Chen et al. 2011) i (Tarvo, 2014) ima izrazito negativan uticaj na performanse. Za rješenja koja koriste dvije niti i XE i XEX konstrukcije, pri manjim dužinama bafera niti se kreiraju i sinhronizuju mnogo češće nego pri većim dužinama bafera. Konkretno, kada se koristi bafer dužine 512 bajta za šifrovanje datoteke dužine 256MB, potrebno je 512000 puta obezbjediti istovremeni završetak rada niti, dok se za bafer dužine 65536 bajta sinhronizacija niti obavlja svega 4000 puta za datoteku iste dužine. Zbog toga se povećanje dužine bafera javlja kao faktor koji vrši uticaj na broj sinhronizacija niti, čime se uvećava vrijeme potrebno za šifrovanje. Prikazano rješenje u velikoj mjeri zavisi od broja fizičkih i logičkih jezgara procesora i od broja procesa i njihovih niti koje su pokrenute u sistemu.

Na slikama 71, 72 i 73 te u tabeli 21 prikazani su rezultati mjerenja modula za OFB šifrovanje za slučaj serijskog (jednonitnog) 256-bitnog OFB šifrovanja, te za slučajeve paralelnog 2x128 bitnog OFB šifrovanja sa odloženom sinhronizacijom niti u kom se koriste XE i XEX konstrukcije pomoću dvije niti.

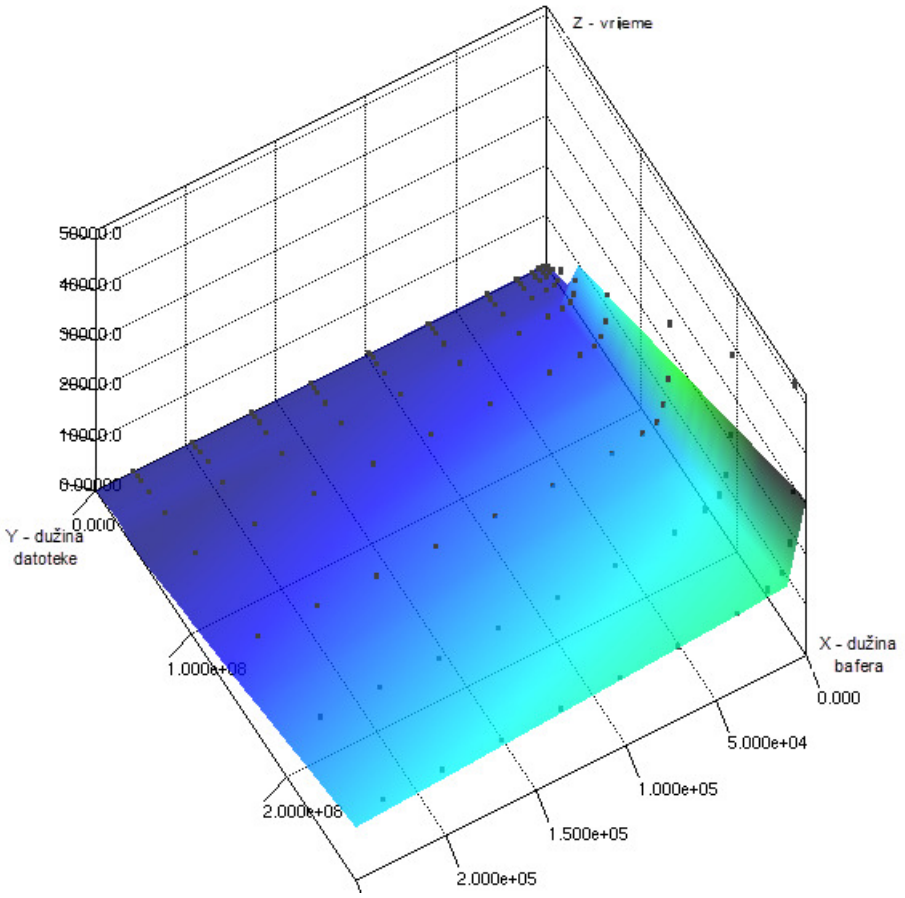
Na osnovu prikazanih trening podataka pomoću metode M5' ponovo je izgrađen određen broj prediktivnih modela koji predviđaju brzinu šifrovanja modula OFBnAes. U konkretnom slučaju, kreirano je

$$N = cip\_inv * modes * keys * threads * comm = 2 * 3 * 2 * 1 * 2 = 48 \quad (79)$$

- 3 načina rada – OFB, OFBXEX i OFBXE.
- Broj niti je postavljen na 1, jer se podrazumjeva da će npr. OFB uvijek koristiti jednu, a OFBXE uvijek dvije niti

Kao i u ranije prikazanim slučajevima, uticaj načina rada, dužine ključa i broja niti na predviđanje brzine ispoljava se kroz klasifikaciju, da bi se na kraju na osnovu dužine datoteke i kriptografskog bafera kreirao trening skup podataka na osnovu kojeg nastaje model. U jednačini 79 prikazano je da na kreiranje klasa utiču samo dvije dužine ključa (128 i 256) i 2 kombinacije paralelnih niti (1x i 2x).

Na slici 77 je prikazan stepen slaganja kreiranog prediktivnog modela OFBnAesCOFB2561p koji koristi jednonitno šifrovanje sa trening podacima. Model pokazuje visok koeficient korelacije (0.9383), ali i nešto veće iznose za relativnu standardnu (20.2061%) i relativnu kvadratnu (36.9758%) grešku.

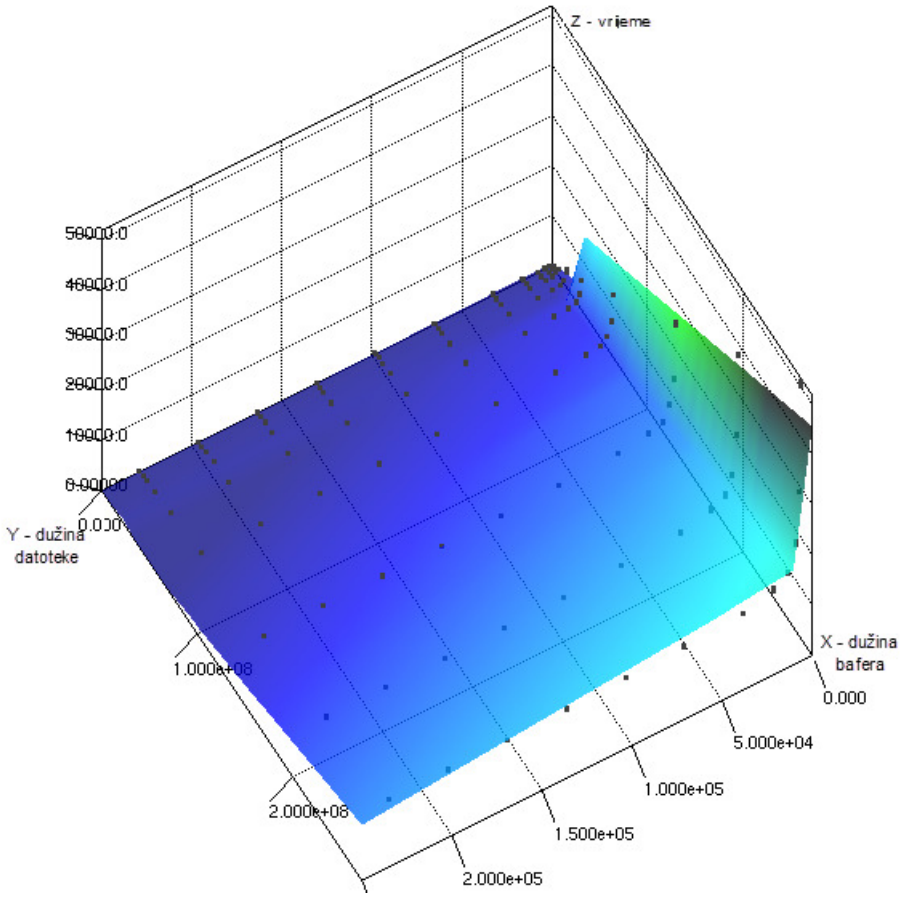


Slika 77: Vizuelni prikaz slaganja trening podataka i kreiranog OFBnAesCOFB2561p modela na N5110 mašini

Koef.korelacije	0.9383
Srednja apsolutna greška	845.0611
Srednja kvadratna greška	2508.5923
Relativna apsolutna greška	20.2061%
Relativna kvadratna greška	36.9758%
Broj instanci	168

Listing 57: Koef.korelacije, srednja apsolutna i srednja kvadratna greška, te relativna apsolutna i relativna kvadratna greška modela OFBnAesCOFB2561p, trening podaci na N5110 mašini

Kao i u slučaju modula koji se oslanja na CUDA arhitekturu koji je prikazan u poglavlju 8, navedene greške su nastale na osnovu podataka koji su skoncentrisani na veoma malom prostoru uz Y osu, tako da imaju relativno mali uticaj na predviđanje, kako je prikazano u listinzima 44 i 45 i na slici 60.

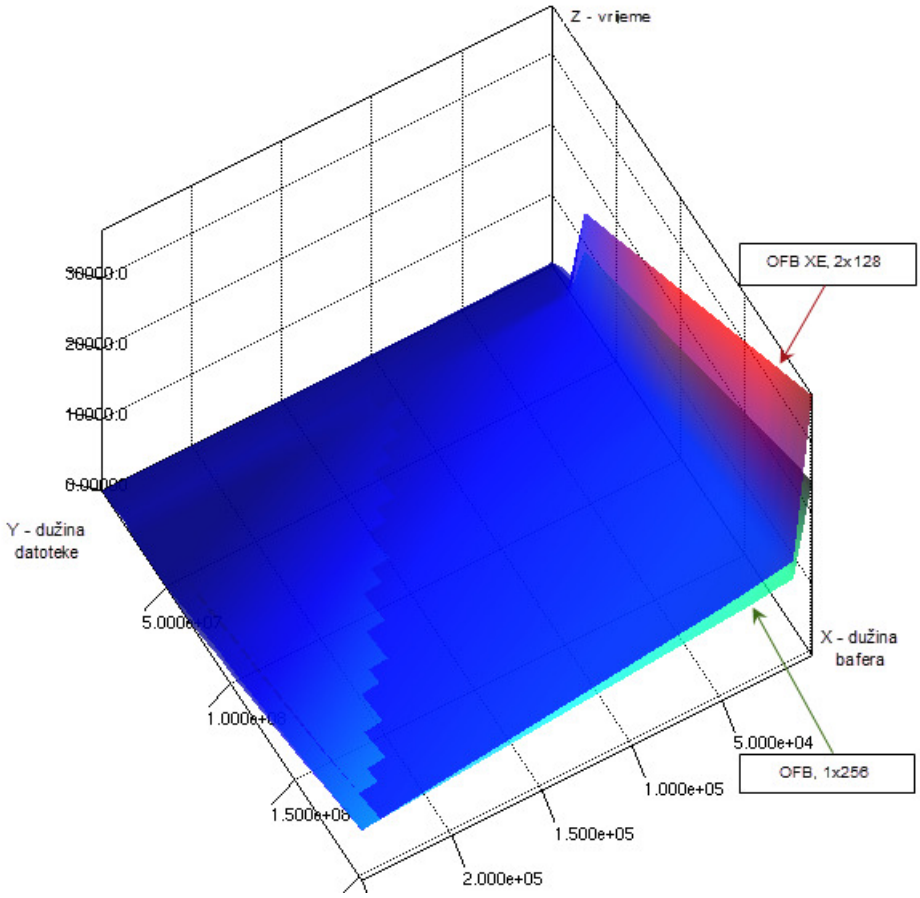


Slika 78: Vizuelni prikaz slaganja trening podataka i kreiranog OFBnAesCOFB2x128XEp modela na N5110 mašini

Koef.korelacije	0.92
Srednja apsolutna greška	1463.5526
Srednja kvadratna greška	4372.2542
Relativna apsolutna greška	26.772%
Relativna kvadratna greška	43.3232%
Broj instanci	168

Listing 58: Koef.korelacije, srednja apsolutna i srednja kvadratna greška, te relativna apsolutna i relativna kvadratna greška modela OFBnAesCOFB256XEp, trening podaci na N5110 mašini

Kada se 256-bitno OFB šifrovanje obavlja pomoću dvije niti kao 2x128 bitno OFB šifrovanje sa odloženom sinhronizacijom niti sa XE konstrukcijom i kada se komunikacija između adaptibilnog virtuelnog kriptografskog fajl sistema i modula obavlja putem neimenovanih cijevi sa minimalnim skupom poruka, nastaje novi model pod nazivom OFBnAesCOFB2x128XEp. Stepen slaganja ovog modela sa trening podacima dat je na slici 78 i u listingu 58. U listingu je naveden visok koeficient korelacije (0.92) i ponovo nešto veća relativna standardna (26.772%) i relativna kvadratna (43.3232%) greška.



Slika 79: Poređenje paralelnog 2x128 bitnog OFB XE šifrovanja sa seriijskim 1x256 bitnim OFB šifrovanjem na N5110 mašini

Na slici 79 prikazan je odnos vizuelizacije modela koji predviđaju brzinu 2x128 bitne paralelne OFB XE enkripcije prema serijskoj 256-bitnoj OFB enkripcji. Model ispravno predviđa da ovo paralelno rješenje u navedenom testnom okruženju postiže to bolje rezultate što se sinhronizacija niti više odlaže. Zahvaljujući sporosti operativne memorije, rješenje koje koristi paralelno 2x128 bitno OFB šifrovanje sa odloženom sinhronizacijom niti sa XEX konstrukcijom na N5110 mašini ne postiže poboljšanje performansi. Ipak, oba paralelna rješenja imaju veliki praktični potencijal jer na bržim mašinama postižu unapređenje performansi u odnosu na jednonitno rješenje, kako je to prikazano za slučaj L50A testne platforme.

11 Evaluacija sistema i prikazanih rješenja

Evaluacija jednog ovakvog kompleksnog sistema morala je biti izvršena na nekoliko načina.

Na prvom mjestu, mora da se prikaže koliko je sistem u cjelini uspješan u onome za šta je namjenjen, odnosno u kojim slučajevima i koliko poboljšanje donosi upotreba prikazanog adaptibilnog modela kada se on primjeni u praksi. Da bi ovo bilo moguće, najprije je prikazano koliko dobro pojedini prediktivni modeli predviđaju brzine odgovarajućih kriptografskih resursa. Potom je izvršena komparacija pojedinih modela koji pripadaju istim kategorijama, a različitim kriptografskim resursima da bi se mogla utvrditi tačnost rada sistema u cjelini. Bitan doprinos evaluaciji dao je opis uticaja pojedinih softverskih i hardverskih tehnologija na performanse i upotrebljivost ovakvog sistema.

U okviru deskripcije svakog pojedinog kriptografskog resursa prikazano je kako i koliko se modeli koji su nastali na osnovu obrade podataka pomoću datog resursa uklapaju u skup trening podataka. U tabeli 25 dat je komparativni prikaz koeficienata korelacije, srednjih apsolutnih i srednjih kvardratnih grešaka koji su dobijeni na osnovu trening podataka.

Koef. korelacije	Relativna apsolutna greška	Relativna kvadratna greška	Model
0.9999	1.1508%	1.6159%	EAesGCECB5121p
0.9998	1.4274%	2.1725%	EAesGCECB3201p
0.9997	1.6535%	2.3584%	EAesGCECB1281p
0.9995	2.1797%	3.0366%	EAesGCECB321p
0.9995	2.2088%	3.0623%	EAesGCECB641p
0.9985	3.1297%	5.3858%	MSVCAesCECB1281p
0.9962	5.8407%	8.6775%	AesNICECB1281p
0.9938	5.9798%	11.0878%	OraSunCECB1281p
0.9200	26.7720%	43.3232%	OFBnAesCOFB2561XEp
0.9082	37.2068%	47.9832%	CudaCECB128np

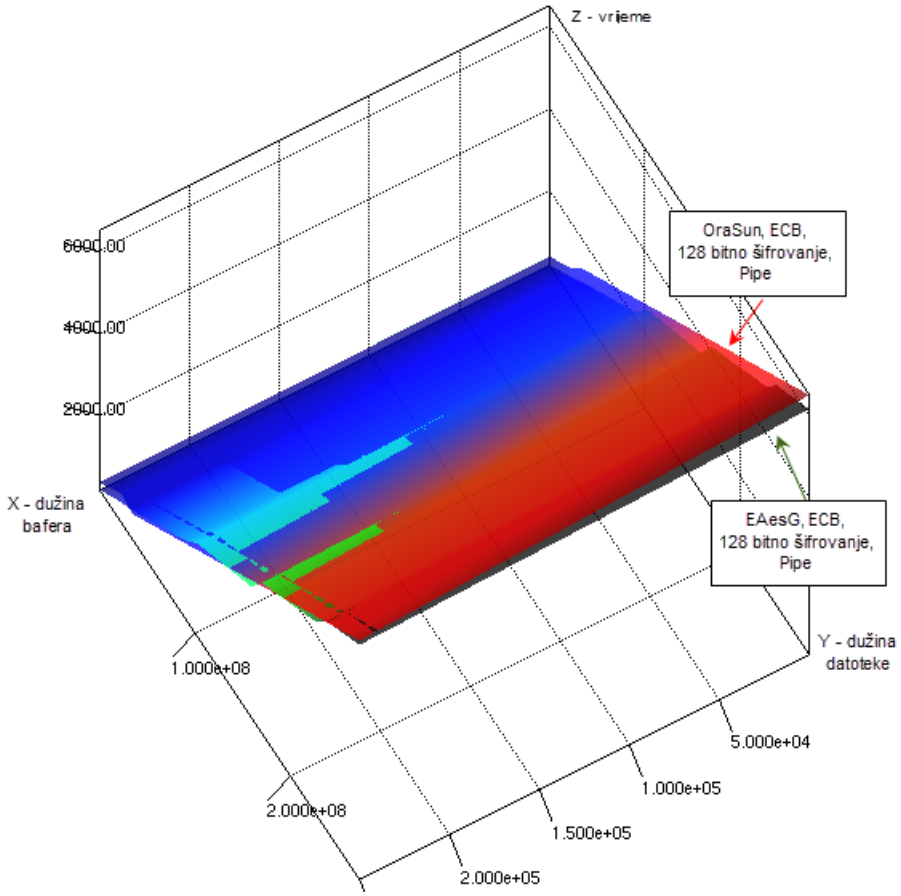
Tabela 25: Komparativni prikaz koeficienata korelacije, srednjih apsolutnih i srednjih kvardratnih grešaka, trening podaci

Iz tabele je vidljivo da svi modeli imaju veoma visok koeficient korelacije što ukazuje na visok stepen slaganja kreiranih modela i podataka. U osam od deset prikazanih modela relativna apsolutna i relativna kvadratna greška su niske. U dva prikazana slučaja rezultati u jednom dijelu domena (male vrijednosti X, u blizini Y ose) prilično raspršeni, zbog čega se javljaju nešto veće vrijednosti grešaka, kako je već komentarisano u poglavlju 8. Posmatranjem koeficienata korelacije, može se reći da modeli dosta dobro predviđaju brzinu izvršavanja posmatranih kriptografskih resursa. Uz uslov da ne predviđaju brzinu izvršenja za male vrijednosti bafera, u blizini Y ose, svi modeli će proizvesti male vrijednosti za relativnu apsolutnu i relativnu kvadratnu grešku.

Komparacija pojedinih kriptografskih resursa će biti predstavljena komparacijom pojedinih njihovih prediktivnih modela, koji pripadaju istim kategorijama (npr. 128 bitna ECB enkripcija, komunikacija putem neimenovanih cijevi) a različitim resursima. Ovakva

komparacija istovremeno navodi koji resurs bi u kom slučaju bio odabran od sistema da se u njemu nalaze samo navedeni kriptografski resursi.

Kada posmatramo jednonitne Java module, prema izvršenim mjerenjima platforme originalni 128-bitni modul EAESG pokazao je veoma dobre rezultate. Posebno dobar rezultat se pokazuje u slučaju malih i srednjih datoteka koje su učestvovale u testovima, pri čijem šifrovanju je navedeni modul brži od OraSUN modula. Ova razlika se sa povećanjem datoteka postepeno smanjuje kada je u pitanju modul koji se oslanja na Oracle SunJCE provajder, da bi se za najveće testirane datoteke gotovo potpuno izgubila.

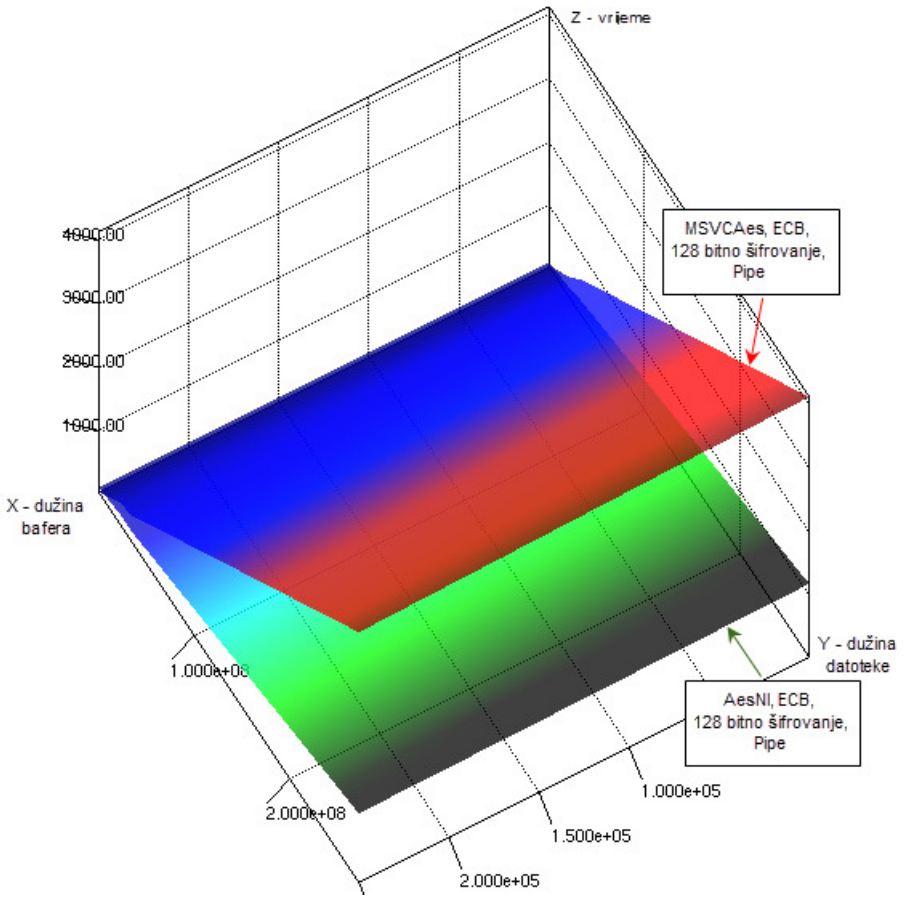


Slika 80. Poređenje modela OraSunECB1281cp modula i EAesGECB1281cp modula, prva testna platforma

Lošiji rezultati koje pokazuje navedena implementacija pri opadanju količine podataka koja se obrađuje i bolji rezultati sa povećanjem istih su uslovljeni pomenutom činjenicom da Java program prilikom startovanja prolazi kroz faze pod nazivima "zagrijavanje koda" (*Code warmup*), učitavanje klasa (*Class loading*) i miješani režim (*Mixed mode*), zbog kojih je virtuelnoj mašini potrebno nešto više vremena da postigne punu brzinu izvršavanja. Velikim kriptografskim paketima potrebno je nešto više vremena za učitavanje klasa i za fazu "zagrijavanja koda" (*Code warmup*). Zbog toga je originalna implementacija bila uvjerljivo bolja u slučaju malih i srednjih datoteka

(prikazanih u rezultatima) i malih i srednjih veličina bafera pomoću kojih se vršila obrada podataka. Kako su sva mjerenja provedena tako da je mjereno vrijeme izvršenja šifrovanja jednog programa u jednom pozivu virtuelne mašine, programski paketi velikih proizvođača zahvaljujući velikom broju linija koda ne uspjevaju odmah da postignu punu brzinu izvršavanja.

Poređenje modela koji predviđaju brzinu jednonitnih Java EAesGECB1281cp i C/C++ MSVCAesECB1281cp modula pokazuju da C/C++ modul postiže očekivano bolje rezultate.

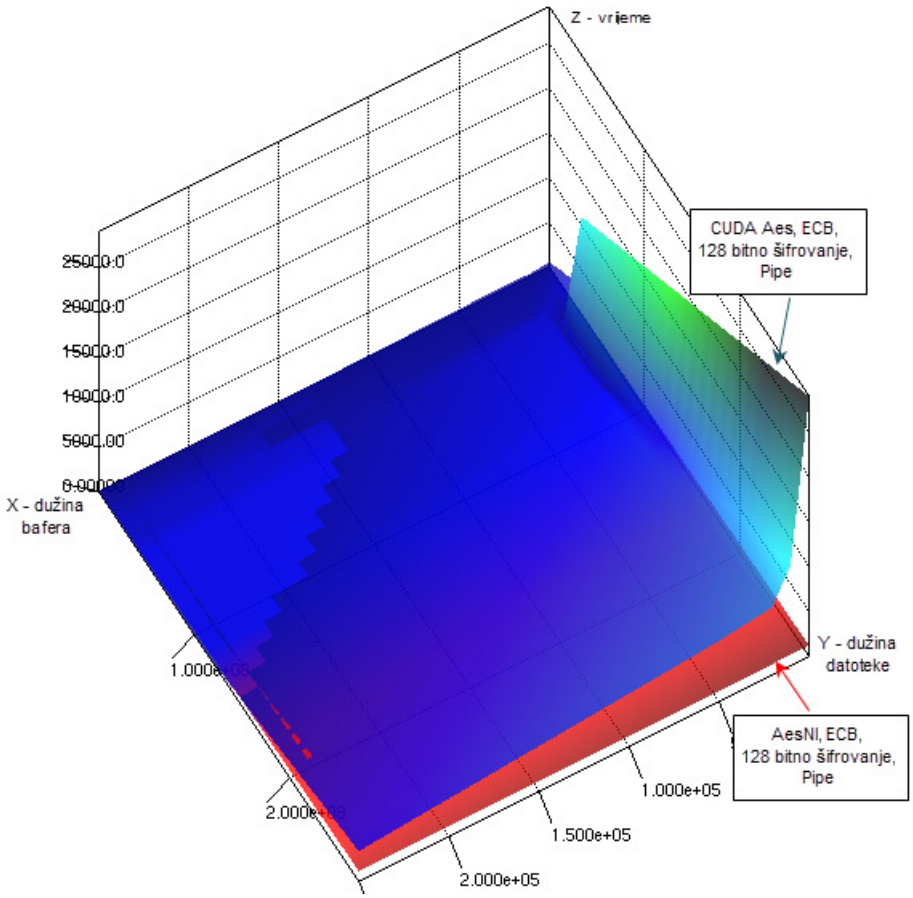


Slika 81. Poređenje modela AesNIECB1281cp modula i MSVCAesECB1281cp modula, prva testna platforma

Najbolje rezultate očekivano pokazuje modul koji se oslanja na hardversku akceleraciju zasnovanu na AES-NI skupu instrukcija. Poređenje njegovog modela sa MSVCAes modulom pokazuje da će, bez obzira na uslove, kriptografski modul koji se oslanja na hardversku akceleraciju višestruko nadmašiti sve ostale.

Među prikazanim rješenjima se svojim velikim potencijalom izdvaja CUDA C modul, koji koristi posaban sklop za hardversku akceleraciju i koji paralelno obrađuje podatke u ECB načinu rada. Model koji opisuje ovu implementaciju pokazuje veliki uticaj koji dužina datoteke i dužina bafera ispoljavaju na brzinu obrade podataka. Ovakvo ponašanje je u skladu sa *coarse grain* dizajnom koji je prikazan u poglavlju 8 a kojim se

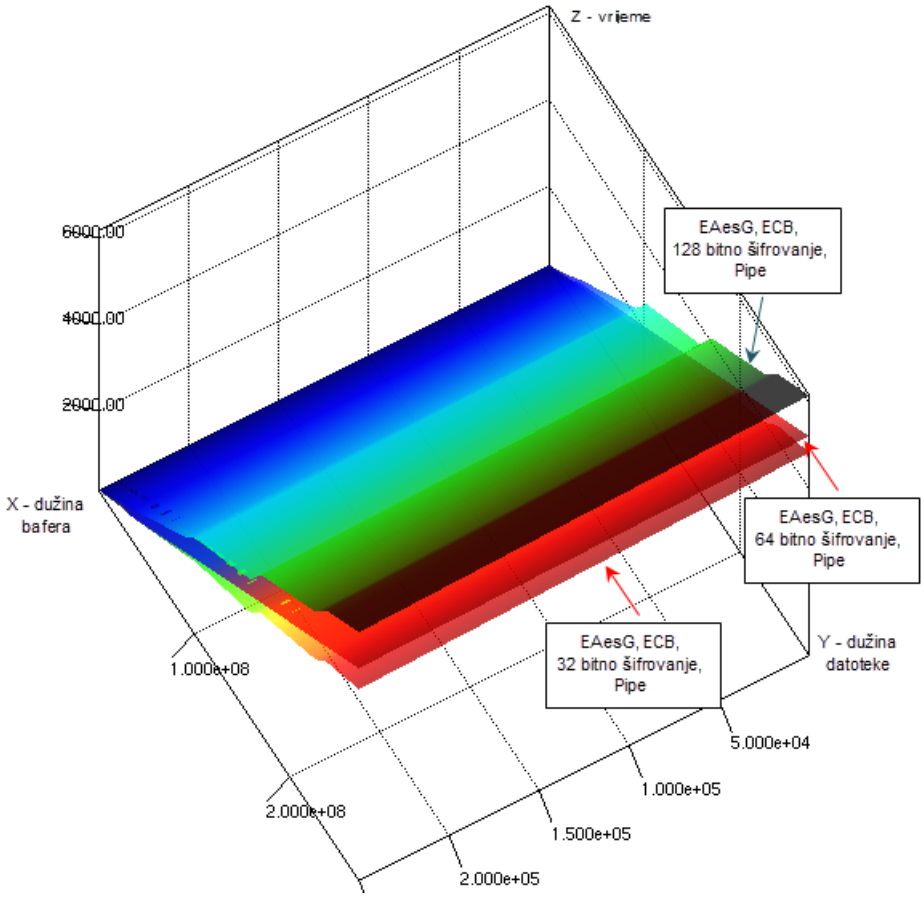
pomoću poziva kernel funkcije vrši paralelno šifrovanje pojedinih blokova otvorenog teksta. Zbog toga šifrovanje neke male količine podataka (npr. 64 bajta) upošljava samo 4 niti, dok šifrovanje veće količine (npr 64KB) koristi 4096 niti koje obrađuju 4096 AES-ovih blokova Stanje paralelno. Veći fragmenti podataka se prenose u globalnu memoriju uređaja pomoću manjeg broja memorijskih transfera, čime se smanjuje njihov negativan uticaj na performanse, kako je navedeno u 4.1.3.1.



Slika 82. Poređenje modela CudaECB128ncp i AesNIECB1281cp modula, prva testna platforma

Iako u datim testnim uslovima na datoj platformi rezultati mjerenja nisu prestigli na hardver oslonjen AesNI modul, rezultati CUDA C modula tek neznatno zaostaju za njim. Navedeni testovi su rađeni na mobilnom NVidia GeForce GT 525M CUDA GPU uređaju. Dobijeni rezultati i prediktivni model ukazuju na to da bi navedene dvije implementacije vrijedilo testirati i u drugim hardverskim uslovima.

Poređenje modela u kojima su primjenjene modifikacije algoritma Aes, očekivano daju različite rezultate u odnosu na modul koji koristi standardom definisanu enkripciju. Prediktivni model koji predstavlja EAesGECB1281cp Java implementaciju (Slika 83) je zbog svojih 10 rundi tokom kojih se vrši šifrovanje svakog bloka pokazao lošije rezultate od 32-bitne (7 rundi, kraća ekspanzija ključeva) i 64-bitne Java implementacije (8 rundi, ponovo kraća ekspanzija ključeva).



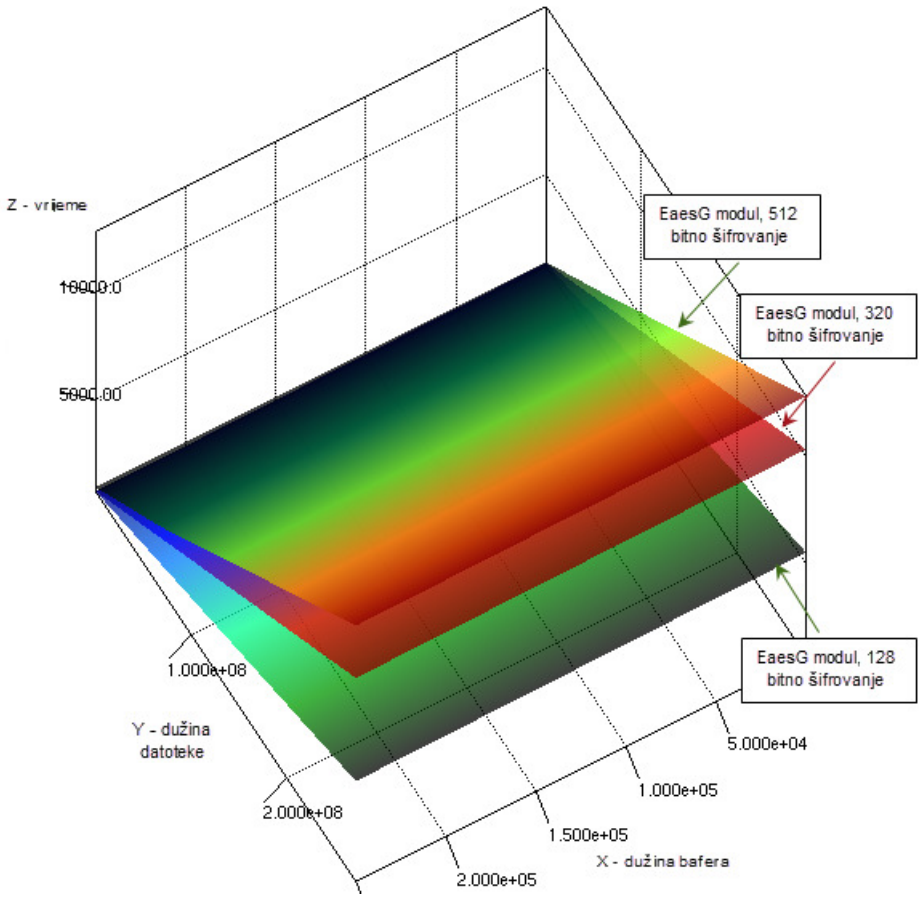
Slika 83. Poređenje modela EAesGECB321cp, EAesGECB641cp i EAesGECB1281cp modula, prva testna platforma

Na slici 84 prikazano je poređenje modela koji predstavljaju 320-bitni i 512-bitni EAESG modul. Jasno je vidljiv zaostatak koji pokazuje 512-bitni modul a koji je prouzrokovan mnogo većim brojem rundi (22 runde, duža ekspanzija ključeva) u odnosu na 320-bitni modul (16 rundi) i posebno u odnosu na 128-bitni (10 rundi).

Obavljenim mjerenjima i kreiranim modelima koji predviđaju brzinu izvršavanja obuhvaćeni su najvažniji dijelovi arhitekture prikazane u poglavlju 6, uz prezentaciju i poređenje različitih tehnologija na kojima su pojedini od njih nastali. Prikazan je omotač oko *third party* biblioteke Oracle SUN JCE (Java), modul nastao pomoću *native* kompajlera MS Visual C/C++, moduli koji se oslanjaju na hardverske sklopove (Aes-NI i CUDA GPU) koji su kreirani *native* kompajlerima MS Visual C/C++ odnosno CUDA C, te originalna implementacija koja koristi jednu ili više niti (Java). Poređenje navedenih implementacija pokazalo je jasnu i očekivanu prednost u brzini kada se koriste hardverski akceleratori. Osim uticaja hardvera, očekivana prednost se javlja pri korišćenju *native* kompajlera u odnosu Java tehnologiju.

Velika razlika se javlja i u slučaju prikazane dvije vrste komunikacije između adaptibilnog virtuelnog kriptografskog fajl sistema i pojedinih modula. Svi izmjereni trening podaci i na osnovu njih kreirani modeli pokazuju jasnu prednost komunikacije

pomoću neimenovanih cijevi uz minimalan skup poruka u odnosu na komunikaciju pomoću soketa na obje testne platforme. U slučajevima kada se komunikacija obavljala pomoću neimenovanih cijevi sa minimalnim skupom poruka, sistem je prosječno dodavao po 100 milisekundi na obje testne platforme, dok je pri korišćenju soketa sa maksimalnim skupom poruka sistem dodavao između 300 i 600 milisekundi ukupnom vremenu potrebnom za obradu. Zbog toga ova mogućnost komunikacije na platformama na kojima je testirana nikada nije odabrana od strane sistema. Ovim je u praksi provjerena mogućnost sistema koja je navedena u poglavlju 5.3.1, da se za povezivanje modula i sistema može koristiti više od jednog načina interprocesne komunikacije, pri čemu je ostavljena mogućnost da adaptibilni kriptografski fajl sistem odredi koji je način najbrži za upotrebu.



Slika 84. Poređenje modela EAesGECB1281cp, EAesGECB3201cp i EAesGECB5121cp modula, prva testna platforma

Pri evaluaciji rada cjelokupnog sistema, mora se pomenuti i rad modula za koordinaciju. Ovaj sistem unosi određeno kašnjenje koje se javlja zbog obrade podataka u toku izračunavanja najefikasijeg resursa koje obavlja modul selektor odnosno njegovi podsistemi za klasifikaciju i za nominaciju kriptografskog resursa. Uticaj vremena potrebnog za rad biblioteke programa za mašinsko učenje i vremena potrebnog za komunikacuju neutrališe se pomoću podsistema za određivanje praga iskoristivosti, kako je to opisano u poglavlju 6.5.

Iz svega navedenog proizilazi da je razvojem modela pomoću kojega se na najbolji način mogu iskoristiti hardverski i softverski resursi računara za izbor najefikasnije mogućnosti za enkripciju i dekripciju algoritmom AES zadovoljen osnovni zahtjev istraživanja. Razvijeni model je konfigurabilan, proširiv i modularan. On se u radu oslanja na modul za koordinaciju sa svojim podsistemima za trening i kategorizaciju, na modul za selekciju sa podsistemima za klasifikaciju i nominaciju i sa podsistemom za određivanje praga iskoristivosti. Model kompletnog sistema se u radu oslanja na predstavljanje pojedinih kriptografskih resursa pomoću više kategorija prediktivnih modela koji se kreiraju metodama mašinskog učenja. Karakteristika cjelokupnog sistema je da on može efikasno da popravlja eventualne pogrešne procjene zahvaljujući mogućnosti evidentiranja novih podataka i ponovnog kreiranja prediktivnih modela. Istraživanje je obuhvatilo kategorizaciju, klasifikaciju i prikaz realizacije pojedinih kriptografskih resursa (modula) i prikaz mogućih načina za komunikaciju između adaptibilnog virtuelnog kriptografskog fajl sistema i pojedinih kriptografskih resursa. Prikazani model je nezavisan od operativnog sistema, zbog čega se može primjenjivati na različitim platformama.

11.1 Provjera hipoteza

Primarni cilj ovog istraživanja predstavlja razvoj modela za adaptibilnu primjenu algoritma AES na nivou operativnog sistema radi poboljšanja performansi računarskog sistema pri izvršavanju ovog algoritma. Predmet istraživanja je analiza iskorišćenosti raspoloživih kriptografskih hardverskih i softverskih resursa od strane modernih operativnih sistema i ispitivanje mogućnosti adaptibilne primjene algoritma AES u okviru operativnog sistema u zavisnosti od dostupnih resursa.

Hipoteza **H0.1.3** navodi da, ako se modeluju postupci za kategorizaciju, evidentiranje i pretraživanje empirijskih podataka koji opisuju različite kriptografske resurse (module), tada se stvaraju pretpostavke za kreiranje trening podataka koji će biti korišćeni od strane metoda za mašinsko učenje. U poglavlju 6.1.1 je prikazan postupak kategoritacije kriptografskih resursa radi grupisanja izvršenih mjerenja i njihovog objedinjavanja u klasifikovane skupove podataka, te je opisan način za njihovo evidentiranje u bazi podataka. Kako je opisano, na ovaj način se od svakog kriptografskog resursa kreira odgovarajući broj modela podataka.

Prema hipotezi **H0.1.2**, ako se uspješno kreiraju kategorizovani trening podaci, tada se stvaraju pretpostavke za primjenu metoda mašinskog učenja radi kreiranja perzistentnih prediktivnih modela koji predviđaju brzinu izvršenja pojedinih kriptografskih resursa u određenim uslovima. U poglavlju 6.1.1 se navodi da se za predviđanje numeričkih podataka između ostalih mogu koristiti metode M5', M5Rules, LWL i *Multilayer perceptron*. U istom poglavlju je predstavljen prediktivni model sa koeficientom regresije i statističkim greškama koji je kreiran pomoću metode M5', a koji se može snimiti u bazi podataka ili datoteci, te njegova grafička reprezentacija.

Kako navodi hipoteza **H0.1.1**, pravilna realizacija postupka kreiranja prediktivnih modela koji predviđaju ponašanje pojedinih kriptografskih resursa predstavlja preduslov za kreiranje algoritma za donošenje odluke o izboru najefikasnijeg kriptografskog resursa.

U poglavlju 6.1.2 je prikazan modul selektor i u okviru njega podsistemi za klasifikaciju i nominaciju. U okviru podsistema za klasifikaciju (poglavlje 6.1.2.1), uticaj većeg broja parametara se klasifikacijom se svodi na dva parametra. Zadatak podsistema za nominaciju je da pomoću u poglavlju 6.1.2.2 prikazanog algoritma odredi najefikasniji kriptografski resurs kojim treba obraditi podatke u datim uslovima.

Hipoteza **H0.2.2** navodi da se kreiranjem standardnog modela za komunikaciju sistema sa okruženjem stvaraju pretpostavke za korišćenje raspoloživih kriptografskih resursa. U poglavlju 6.2 prikazani su modeli komunikacije za povezivanje kriptografskih resursa (modula) i adaptibilnog virtuelnog kriptografskog fajl sistema, gdje su prikazani načini komunikacije uz razmjenu minimalnog i maksimalnog skupa poruka korišćenjem različitih tehnika interprocesne komunikacije. Svi navedeni kriptografski resursi koji su učestvovali u kreiranju empirijskih podataka su na identičan način koristili predstavljene načine za komunikaciju sa sistemom. Sistem je od jednog kriptografskog resursa kreirao više prediktivnih modela (po jedan za svaki način komunikacije) koji su dalje ravnopravno učestvovali u izboru najefikasnijeg resursa.

Prema hipotezi **H0.2.1**, kreiranjem modela za mapiranje kriptografskih resursa od strane sistema ili korisnika stvaraju se pretpostavke za uključenje raspoloživih kriptografskih resursa (modula) u sistem. U poglavlju 6.2.3 predstavljen je model za mapiranje pojedinih kriptografskih resursa sa osnovnim karakteristikama pojedinih resursa i načinima njihovog pokretanja.

U hipotezi **H0.1** stoji da ako se uspješno kreiraju metoda za klasifikaciju i algoritam za nominaciju koji uz pomoć ranije kreiranih prediktivnih modela (i pomoću metoda mašinskog učenja) donose odluku o izboru najefikasnijeg raspoloživog kriptografskog resursa u datim uslovima, tada je moguće kreirati elastičan model koji se na sistemskom nivou prilagođava raspoloživim hardverskim i softverskim kriptografskim resursima. Prema hipotezi H0.1.1, u poglavlju 6 predstavljeni su podsistemi za klasifikaciju i nominaciju koji donose odluku o izboru najefikasnijeg raspoloživog resursa na osnovu prediktivnih modela, a korišćenjem metode M5'. U poglavljima 5 i 6 predstavljeni su model i detaljna arhitektura rješenja za adaptibilnu primjenu AES algoritma na nivou operativnog sistema, gdje je prikazan elastičan model pomoću kojeg sistem bira najefikasniji kriptogtafski resurs za obradu podataka. Kreirani model je elastičan u smislu da može samog sebe da popravlja sa svakim novim izvršenim šifrovanjem ili dešifrovanjem.

Prema hipotezi **H0.2**, mogućnost izbora najefikasnijeg raspoloživog resursa zavisi od mogućnosti mapiranja i komunikacije različitih kriptografskih resursa sa sistemom. Prema hipotezi H0.2.1 i H0.2.2 u poglavlju 6.2 predstavljeni su modeli za mapiranje pojedinih kriptografskih resursa od strane sistema i za komunikaciju resursa sa sistemom.

U okviru ovog istraživanja je kreiran model adaptibilnog virtuelnog kriptografskog fajl sistema (poglavlje 5) zajedno sa njegovom detaljnom arhitekturom (poglavlje 6). Kreirani adaptibilni virtuelni kriptografski fajl sistem služio je kao reprezent adaptibilnog kriptografskog operativnog sistema. U okviru istraživanja je izvršena i deskripcija većeg broja kriptografskih resursa koji su direktno ili indirektno uključeni u adaptibilni virtuelni kriptografski fajl sistem, te dio prediktivnih modela koji opisuju njihovo ponašanje sa

evaluacijom istih. Ono što je posebno karakteristično za izgrađeni model je da su sve njegove komponente na kraju implementirane u programskim jezicima Java, C++ ili Cuda C. Zbog toga sve prethodne hipoteze zajedno pomoću hipotetičkog silogizma potvrđuju generalnu hipotezu istraživanja **H1** koja glasi: Postoji uticaj koji se na poboljšanje performansi i kvaliteta zaštite algoritmom AES ostvaruje putem adaptacije operativnih sistema raspoloživim hardverskim i softverskim kriptografskim resursima.

12 Zaključak

U ovom radu analiziran je problem iskorišćenja raspoloživih kriptografskih resursa za enkripciju podataka pomoću algoritma AES od strane modernih operativnih sistema i mogućnost njegove adaptibilne primjene u zavisnosti od raspoloživih resursa.

Primarni cilj ovog istraživanja je bio razvoj modela za adaptibilnu primjenu algoritma AES na nivou operativnog sistema. U istraživanju je predstavljen i detaljno opisan elastičan model rješenja koji je u stanju da bira najefikasniji resurs u nekom heterogenom sistemu. Navedeni model je u stanju da popravlja svoje procjene, te da na osnovu preferencija korisnika u obzir uzima i efikasnost drugih parametra, poput načina komunikacije i/ili veličine bafera kojim se vrši šifrovanje.

U okviru ovog istraživanja je kreiran model adaptibilnog virtuelnog kriptografskog fajl sistema (poglavlje 5) i prikazana njegova detaljna arhitektura (poglavlje 6). U okviru istraživanja je data i deskripcija većeg broja kriptografskih resursa (modula). Ovi moduli se oslanjaju na vlastita rješenja, zatim na third-party rješenja (kao omotači) i rješenja koja koriste posebne hardverske sklopove za akceleraciju.

Takođe je predstavljena detaljna arhitektura sistema i podsistema koji čine adaptibilni virtuelni kriptografski fajl sistem. Jedan od najvažnijih je modul za koordinaciju koji je zadužen da donese odluku o izboru kriptografskog resursa koji pokazuje najbolje performanse. On se sastoji od modula za trening i kategorizaciju, zatim od modula selektora sa podsistemima za klasifikaciju i nominaciju, ulazno-izlaznog podsistema, podsistema za određivanje praga iskoristivosti, te veza ka biblioteci funkcija za mašinsko učenje i sistemu za upravljanje bazama podataka. Prikazani model je pri tome postao nezavistan od izbora tehnologije i od izbora hardverske i softverske platforme.

Treba reći da je tokom istraživanja proveden određen broj eksperimenata sa proširenjem algoritma AES, koji su implementirani u programskom jeziku Java i predstavljeni u radu uporedo sa empirijskim prikazom njihovih performansi. Posebnu oblast u okviru podistraživanja predstavlja istraživanje novog rješenja za paralelizaciju izvršenja OFB n i algoritma AES.

Dio rezultata istraživanja koja su obavljena pri izradi disertacije su već objavljeni u časopisima domaćeg i međunarodnog značaja i saopšteni na odgovarajućim naučnim skupovima.

Najznačajniji doprinos ovog istraživanja je razvoj modela za adaptibilnu primjenu algoritma AES na nivou operativnog sistema. Kao praktičan rezultat istraživanja realizovan je adaptibilni virtuelni kriptografski fajl sistem u okviru kojega je implementiran modul za koordinaciju koji na osnovu ranije kreiranih i sačuvanih prediktivnih modela donosi odluku o tome koji kriptografski resurs treba upotrebiti za zaštitu podataka pomoću AES algoritma.

Naučni doprinos istraživanja ogleda se u slijedećem:

1. istraživanju adaptibilnosti operativnih sistema za efikasnu upotrebu algoritma AES;

2. formalnom opisu modela sistema i metoda korišćenih u razvoju adaptibilnog virtuelnog kriptografskog fajl sistema;
3. formalnom opisu modela za mapiranje kriptografskih resursa i za njihovu komunikaciju sa sistemom;
4. sistematizaciji i detaljnoj analizi dosadašnjih pristupa enkripciji unutar operativnih sistema;
5. istraživanju mogućnosti proširenja algoritma AES;
6. istraživanju novih ideja za paralelizaciju algoritma AES korišćenjem odgovarajućih kriptografskih načina rada (modes of operation);
7. istraživanju ideja za uključenje GPU jedinica kao kriptografskih resursa operativnog sistema;
8. istraživanju metoda koje mogu da se koriste za predviđanje ponašanja odgovarajućih kriptografsih resursa.

Stručni doprinos istraživanja se ogleda u:

1. implementaciji proširenja ovoga algoritma
2. implementaciji ideja za paralelizaciju algoritma AES korišćenjem odgovarajućih kriptografskih načina rada (modes of operation);
3. implementaciji ideja za uključenje GPU jedinica kao kriptografskih resursa operativnog sistema;
4. sistematizaciji i detaljnoj analizi implementacije različitih modula koji su u stanju da koriste brojne hardverske i softverske kriptografske resurse koji stoje na raspolaganju modernim operativnim sistemima;
5. implementaciji metoda koje mogu da se koriste za predviđanje ponašanja odgovarajućih kriptografsih resursa;

Prikazano istraživanje će dovesti do niza društvenih doprinosa, među kojima se izdvajaju:

1. rezultati istraživanja će pomoći pri identifikaciji resursa koji od strane operativnog sistema mogu da budu iskorišćeni za ubrzanje šifrovanja i dešifrovanja pomoću algoritma AES;
2. rezultati istraživanja će doprinijeti pri analizi i deskripciji procesa implementacije kriptografske adaptibilnosti operativnih sistema algoritmu AES;
3. rezultati istraživanja mogu da budu iskorišćeni za dalja unapređenja algoritma AES;
4. rezultati istraživanja će doprinijeti daljem razvoju metoda kriptografske adaptibilnosti i efikasnijem korišćenju kriptografije u okvirima operativnih sistema.

Karakteristika današnjih klasičnih računarskih sistema je usporavanje rasta brzine jednoprocesorskih sistema i okretanje proizvođača hardvera višeprocesorskim sistemima i hardverskim akceleratorima. Sve veća potreba za zaštitom podataka na pametnim telefonima i sličnim uređajima obavezuje proizvođače softvera da vode više

računa o memorijskim zahtjevima uređaja. Obzirom da pored hardverskih postoji velika raznolikost softverskih resursa koji koriste algoritam AES, mogućnosti primjene rezultata istraživanja su velike. Standardizacijom komunikacije na relaciji operativni sistem - kriptografski modul stvorena je mogućnost da se nezavisni proizvođači hardvera i softvera na jednostavan način povežu sa operativnim sistemom. Pri ovome bi operativni sistem, odnosno adaptibilni virtuelni kriptografski fajl sistem, koji koristi navedene koncepte dobio bogatiji i raznovrsniji izbor kriptografskih resursa iz kojega bi opisanim metodama bio u prilici da bira najbolji resurs.

Ova disertacija otvara mogućnosti za dalja istraživanja koja se odnose poboljšanje performansi i povećanje efikasnosti i prilagodljivosti savremenih operativnih sistema pri primjeni AES algoritma u nekoliko pravaca:

- istraživanje mogućnosti oporavka sistema od greške u slučaju otkaza ili uklanjanja nekog kriptografskog resursa iz sistema, te od drugih nedefinisanih grešaka;

- istraživanje efikasnosti iz perspektive utroška operativne memorije;

- istraživanje mogućih načina upravljanja ključevima u adaptibilnim operativnim sistemima;

- istraživanje mogućnosti primjene novih metoda mašinskog učenja radi poboljšanja performansi sistema;

- fino podešavanje postojećih i razvoj novih algoritama za paralelizaciju korišćenjem različitih hardverskih i softverskih resursa predstavlja još jedan od mogućih pravaca istraživanja koji proističu iz ovdje prikazanih rješenja.

13 Referentna literatura

(Akdemir et al. 2010) Akdemir, K., Dixon, M., Feghali, W., Fay, P., Gopal, V., Guilford, J., Ozturk, E., Wolrich, G. and Zohar, R. (2010), Breakthrough AES Performance with Intel AES New Instructions, Intel White Paper, Intel Corporation

(Alpaydın, 2010) Alpaydın, E. (2010), E.Introduction to machine learning , Second edition, The MIT Press, Massachusetts Institute of Technology, Cambridge, MA, USA

(Amdahl, 1967) Amdahl, G. M., (1967), Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities, Proc. AFIPS '67, ACM, New York, 483–485.

(Android Open Source Project, 2016) Android Open Source Project, (2016) Full disk encryption in Android 5.0, Android open source project Teach Info, https://source.android.com/security/encryption/

(Apache Fondation, 2014) Apache Fondation, (2013). Apache Commons Virtual File System (VFS), Available at: http://commons.apache.org/proper/commons-vfs/

(Apple Corp., 2012) Apple Corp. (2012). Best Practices for Deploying FileVault 2, Available at: http://training.apple.com/pdf/WP_FileVault2.pdf

(Arch Linux , 2016) Arch Linux documentation, (2016). Disk Encryption, Available at: https://wiki.archlinux.org/index.php/Disk_Encryption

(Barker and Roginsky, 2011) Barker, E. and Roginsky, A. (2011), Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths, NIST Special Publication 800-131A, National Institute of Standards and Technology, U.S. Department of Commerce

(Basu and Bhargav-Spantzel, 2012) Basu, A. and Bhargav-Spantzel, A. (2012), AES-NI Performance Testing on Linux/Java Stack, Intel vPro Developer Community, Intel Corporation

(Berk Guder, 2009) Berk Guder, C., AES on CUDA, (2009), dostupno na:http://github.com/cbguder/aes-on-cuda (januar 2016).

(Bertoni et al. 2002) Bertoni, G., Breveglieri, L., Fragneto, P., Macchetti, M. and Marchesin, S. (2002), Efficient Software Implementation of AES on 32-Bit Platforms. CHES 2002: 159-171

(Bidulock, 2008) Bidulock, B.F.G. (2008), STREAMS-based vs. Legacy Pipe Performance Comparison - Experiment Test Results for Linux, OpenSS7 documentation, OpenSS7 Corporation

(Biryukov et al. 2009) Biryukov, A., Khovratovich, D. (2009), Related-key Cryptanalysis of the Full AES-192 and AES-256, ASIACRYPT '09 Proceedings of the 15th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology, Pages 1 - 18 , Springer-Verlag Berlin, Heidelberg.

(Biryukov et al. 2010) Biryukov, A., Dunkelman, O., Keller, N., Khovratovich, D. and Shamir, A. (2010), Key recovery attacks of practical complexity on AES-256 variants with up to 10 rounds, EUROCRYPT'10 Proceedings of the 29th Annual international conference on Theory and Applications of Cryptographic Techniques, Pages 299-319

(Blaze, 1993) Blaze, M. (1993), A Cryptographic File System for Unix, ACM Conference on Computer and Communications Security

(Blevins and Ramamoorthy, 1976) Blevins, P.R. and Ramamoorthy, C.V. (1976), Aspects of a Dynamically Adaptive Operating System, IEEE Transactions on Computers, Volume:C-25, Issue:7

(Bogdanov et al. 2011) Bogdanov A., Khovratovich D. and Rechberger C., (2011), Biclique Cryptanalysis of the Full AES. ASIACRYPT 2011:344-371

(Boyer1, 2008) Boyer, B., (2008), Robust Java benchmarking, Part 1: Issues, Understand the pitfalls of benchmarking Java code, IBM developerWorks

(Boyer1, 2008) Boyer B. (2008), Robust Java benchmarking, Part 1: Issues, Understand the pitfalls of benchmarking Java code, IBM developerWorks

(Boyer2, 2008) Boyer, B., (2008), Robust Java benchmarking, Part 2: Statistics and solutions, Introducing a ready-to-run software benchmarking framework, IBM developerWorks

(Boyer2, 2008) Boyer B. (2008), Robust Java benchmarking, Part 1: Statistics and solutions, Introducing a ready-to-run software benchmarking framework, IBM developerWorks

(Breimanet al. 1984) Breiman, L., Friedman, J. H. , Olshen, R. A., Stone, C. J. (1984), Classification and Regression Trees, Wadsworth, Belmont, California, USA

(Carrell, 2005) Carrell J.B., (2005), Fundamentals of Linear Algebra, University of British Columbia, Canada

(Chakraborty and Rodriguez-Henriquez, 2009) Chakraborty D. and Rodriguez-Henriquez, F. (2009). Block Cipher Modes of Operation from a Hardware Implementation Perspective, Cryptographic Engineering, Springer

(Cheltha and Velayutham, 2011) Cheltha JNC, Velayutham R. (2011) A novel error-tolerant method in AES for satellite images. IEEE International Conference on Emerging Trends in Electrical and Computer Technology (ICETECT), Tamil Nadu, 2011, str. 937–940.

(Chen et al. 2011) Chen, K, Morris Chang, J., Hou, T., (2011), Multithreading in Java: Performance and Scalability on Multicore Systems", IEEE Transactions on Computers , 60, 11, 1521-1534.

(Christopher, 2010) Christopher, C.J. (2010), Cryptographic Accelerators on the UltraSPARC T2 with the Solaris Cryptographic Framework, School of Computer Science, ANU, Available at: http://courses.cecs.anu.edu.au/courses/CS_PROJECTS/10S2/Reports/Cody%20Christopher.pdf

(Chung-Wei Phan, 2002) Chung-wei Phan R. (2002), Mini Advanced Encryption Standard (Mini-AES): A Testbed for Cryptanalysis, Cryptologia, 26(4):283–306

(CM W3C, 2005) Character Model for the World Wide Web 1.0, W3C Recommendation, (2005), Available at: http://www.w3.org/TR/charmod/

(Daemen and Rijmen, 1999) Daemen J., Rijmen V., (1999), The Rijndael Block Cipher - AES Proposal (from original submission), NIST, Available at: http://csrc.nist.gov/archive/aes/rijndael/Rijndael-ammended.pdf

(Daemen and Rijmen, 2002) Daemen, J. and Rijmen, V. (2002), The Design Of Rijndael, Springer-Verlag, Inc.

(Daemen and Rijmen, 2012) Daemen, J. and Rijmen, V. (2012), ON THE RELATED-KEY ATTACKS AGAINST AES, Proceedings Of The Romanian Academy, Series A, Mathematics, Physics, Technical Sciences, Information Science, Volume 13, Number 4/2012, pp. 395–400

(Damjanović and Simić, 2011) Damjanović, B. and Simić, D. (2011), Comparative Implementation Analysis of AES Algorithm, Journal of Information Technology and Applications, Banja Luka

(Damjanović and Simić, 2013) Damjanović, B. and Simić, D. (2013), Performance evaluation of AES algorithm under Linux operating system, Proceedings Of The Romanian Academy, Series A, Mathematics, Physics, Technical Sciences, Information Science, Volume 14, Number 2, pp. 177–183

(Damjanović and Simić, 2015) Damjanović, B. and Simić, D. (2015), Tweakable parallel OFB mode of operation with delayed thread synchronization, Wiley, Security and Communication Networks, Security Comm. Networks (2015), Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/sec.1404.

(Damjanović i Simić, InfoM46, 2013) Damjanović, B. i Simić, D. (2013), Eksperimenti sa mogućim modifikacijama AES algoritma, InfoM 46/2013

(Damjanović i Simić, InfoM47, 2013) Damjanović, B. i Simić, D. (2013), Pregled primjenjenih pristupa za softversku enkripciju podataka u različitim operativnim sistemima, InfoM 47/2013

(Damjanović, 2010) Damjanović, B. (2010), Implementacija i proširenje AES algoritma, Master rad, Fakultet organizacionih nauka, Beograd

(Dandalisand and Prasanna, 2000) Dandalisand, A. and Prasanna, V.K. (2000), An Adaptive Cryptographic Engine for IPSec Architectures, IEEE Symposium on Field-Programmable Custom Computing Machines

(Delaune et al. 2008) Delaune, S., Kremer S. and Steel, G. (2008), Formal Analysis of PKCS#11, Research Report LSV-4

(Demirci et al. 2008) Demirci, H. and Selçuk. A.A. (2008), A Meet-in-the-Middle Attack on 8-Round AES, Fast Software Encryption, Springer-Verlag Berlin, Heidelberg

(Dent and Mitchell, 2005) Dent A.W. and Mitchell C.J., (2005), User's Guide to Cryptography and Standards, Artech House, Boston, London
(Di Biagio et al. 2009) Di Biagio, A., Barenghi, A., Agosta and G. and Pelosi, G. (2009), Design of a Parallel AES for Graphics Hardware using the CUDA framework, IEEE International Symposium onParallel & Distributed Processing
(Diesburg, 2010) Diesburg S.M. and Wang, A.A. (2010), A Survey of Confidential Data Storage and Deletion Methods, ACM Computing Surveys (CSUR), Volume 43 Issue 1, Article No. 2
(Diffie, 1979) Diffie W., Hellman M.E., (1979), Privacy And Authentication: An Introduction to Cryptography, Proceedings of the IEEE 67.
(Dworkin, 2001) Dworkin M., (2001), Recommendation for Block Cipher Modes of Operation, NIST Special Publication 800-38A
(Dworkin, 2010) Dworkin M. (2010) Recommendation for Block Cipher Modes of Operation: The XTS-AES Mode for Confidentiality on Storage Devices. Tech. report,NIST Special Publication 800-38E.
(Edge, 2011) Edge, J. (2011), Desktop Summit: Crypto consolidation, Available at: https://lwn.net/Articles/454491/
(Ermelindo, 1997) Ermelindo, M. (1997), Transparent Cryptographic File System, Linux Journal, Volume 1997 Issue 40es, Article No. 3
(Feistel, 1971) Feistel, H., (1971), Block Cipher Cryptographic System, US Patent 3,798,359, IBM.
(Ferguson et al, 2010) Ferguson N., Schneider B., Kohno T., (2010), Cryptography Engineering: Design Principles and Practical Applications
(Ferguson, 2006) Ferguson, N. (2006), AES-CBC + Elephant difuser A Disk Encryption Algorithm for Windows Vista, Microsoft White Paper
(FIPS PUB 81, 1980) FIPS PUB 81, DES MODES OF OPERATION, (1980), Federal Information Processing Standards Publication 81, December 1980, available at http://www.itl.nist.gov/fipspubs/fip81.htm (april 2013)
(FIPS197, 2001) FIPS197, Specification for the Advanced Encryption Standard (AES). (2001), Federal Information Processing Standards Publication 197, Available at: http://csrc.nist.gov/publications/
(Fried and Jarden, 2008) Fried, M.D., Jarden, M., (2008), Field Arithmetic, Third Edition, Springer-Verlag
(Fruhwirth, 2008) Fruhwirth, C., (2008). LUKS On-Disk Format Specification, Version 1.1.1
(Gebali, 2011) Gebali, F. (2011), Algorithms and Parallel Computing, John Wiley & Sons, Inc., Hoboken, New Jersey, USA
(Georges et al. 2007) Georges, A., Buytaert, D. and Eeckhout, L. (2007), Statistically Rigorous Java Performance Evaluation, OOPSLA '07 Proceedings of the 22nd annual ACM SIGPLAN conference on Object-oriented programming systems and applications, Vol. 42, pg 57-76
(Georges et al. 2008) Georges, A., Eeckhout, L. and Buytaert, D. (2008), Java Performance Evaluation through Rigorous Replay Compilation, OOPSLA '07 Proceedings of the 22nd annual ACM SIGPLAN conference on Object-oriented programming systems and applications, Vol. 43, pp. 367-384
(Gervasi et al. 2010) Gervasi O., Russo D., Vella F., (2010), The AES implantation based on OpenCL for multi/many core architecture, International Conference of Computational Science and Its Applications, ICCSA 2010, Fukuoka, Japan, pp. 129-134
(Gladman, 2007) Gladman, B., (2007), A Specification for Rijndael, the AES Algorithm, Available at: http://gladman.plushost.co.uk/oldsite/cryptography_technology/rijndael/
(Goetz, 2010) Goetz B., Peierls T., Bloch J., Bowbeer J., Holmes D., Lea, D., (2010), Java Concurrency in Practice, Addison-Wesley, New York, USA.
(Gordon, 2013) Gordon, S. (2013). Simplified AES Example, Security and Cryptography, Sirindhorn International Institute of Technology (SIIT) in Thammasat University, Bangkok, Thailand, dostupno na: http://hw.siit.net/files/001283.pdf
(Gosling et al. 2013) Gosling, J., Joy, B., Steele, G., Bracha G., Buckley, A., (2013), The Java® Language Specification, Java SE 7 Edition, Oracle Corporation
(Gough, 2008) Gough, V. (2008), EncFS: Encrypted file system, http://arg0.net/wiki/encfs

(Gu et al. 1999) Gu Y., Lee, B.S., Cai, W., (1999), Evaluation of Java Thread Performance on Two Different Multithreaded Kernels"; ACM SIGOPS Operating Systems Review, 33, 1, 34 - 46.

(Gueron, 2009) Gueron S. (2009), Intel's New AES Instructions for Enhanced Performance and Security, Fast Software Encryption, Lecture Notes in Computer Science Volume 5665, pp 51-66, Springer Berlin Heidelberg

(Gueron, 2012) Gueron, S. (2012), Intel Corporation, White Paper, Intel Advanced Encryption Standard (AES) New Instructions Set

(Gupta et al. 2013) Gupta, A., Ababneh, E., Han, R. and Keller. E. (2013), Towards Elastic Operating Systems, HotOS XIV, 14th Workshop on Hot Topics in Operating Systems

(Halcrow, 2007) Halcrow, M. (2007), eCryptfs: a Stacked Cryptographic Filesystem, Linux Journal, Volume 2007 Issue 156

(Hall et al. 2009) Hall M., Frank E., Holmes G., Pfahringer B., Reutemann P., and Witten I.H. (2009). The WEKA data mining software: an update. SIGKDD Explor. Newsl. 11, 1 (November 2009), 10-18. DOI=http://dx.doi.org/10.1145/1656274.1656278

(Han and Kamber, 2006) Han J. and Kamber M., (2006) Data Mining: Concepts and Techniques, Second Edition, Morgan Kaufmann Publishers

(Harrington, 2012) Harrington, P. (2012), Machine Learning in Action, Manning Publications Co.

(Harrison et al. 2010) Harrison, O. and Waldron, J. (2010), GPU accelerated cryptography as an OS service, Transactions on computational science XI, Springer-Verlag Berlin, Heidelberg

(Hoban et al. 2013) Hoban, A., Laurent, P., Betts, I. and Tahhan, M. (2013), Unleashing Linux*-Based Secure Storage Performance with Intel AES New Instructions, Intel Corporation, White Paper, Februar

(Hohmann, 2008) Hohmann, C. (2008) CryptoFS. http://reboot.animeirc.de/cryptofs/

(Hook, 2005) Hook D. (2005), Beginning Cryptography with Java, Wrox Press

(IBM Blueprints, 2009) IBM Corporation, (2009), IBM Blueprints, Securing Sensitive Files With TPM Keys

(IBM Corporation, 2015) IBM Corporation, (2015), z/OS V2R2 XL C/C++ Programming Guide

(IEEE, 2008) IEEE Std. 1619-2007. (2008) Cryptographic Protection of Data on Block-Oriented Storage Devices. IEEE.

(Im et al. 2013) Im, C., Jeong, M., Lee, J. and Lee, S. (2013), A Dynamically Reconfigurable Operating System for Manycore Systems, SAC '13 Proceedings of the 28th Annual ACM Symposium on Applied Computing

(Intel Corporation, 2013) Intel Corporation, (2013). Intel Advanced Encryption Standard (AES-NI) Ecosystem March 2013 update, Available at: http://www.intel.com/content/dam/www/public/us/en/documents/specification-updates/aes-ni-ecosystem-update.pdf

(ISO/IEC 10116, 2006) ISO/IEC, (2006), ISO/IEC 10116 (3rd edition): Information technology - Security techniques - Modes of operation for an n-bit block cipher.

(Iwai, 2010) Iwai, K., Kurokawa, T. and Nisikawa, N. (2010), AES Encryption Implementation on CUDA GPU and Its Analysis, First International Conference on Networking and Computing (ICNC)

(Iyer et al. 2010) Iyer, V., Perry, E.H.,, Wright, B., Pfaeffle, T., (2010), Oracle Database JDBC Developer's Guide and Reference, 10g Release 2 (10.2), Oracle corporation

(Jacobson, 2009) Jacobson, N., (2009), Basic algebra I (Second ed.), W.H.Freeman And Company, New York

(Jacquin, 2010) Jacquin L., Roca V., Roch J.L., Al Ali M., (2010), Parallel arithmetic encryption for high-bandwidth communications on multicore/GPGPU platforms, PASCO '10 Proceedings of the 4th International Workshop on Parallel and Symbolic Computation

(JCA PROV, 2016) Java Cryptography Architecture, Oracle Providers Documentation, (2016), Oracle Corporation, Available at: http://docs.oracle.com/javase/7/docs/technotes/guides/security/SunProviders.html

(JCA RFG, 2016) Java Cryptography Architecture (JCA) Reference Guide, (2016), Oracle Corporation, Available at: http://docs.oracle.com/javase/7/docs/technotes/guides/security/crypto/CryptoSpec.html

(Keromytis et al. 2006) Keromytis, A.D., Wright, J.L., DE Raadt, T. and Burnside, M. (2006), Cryptography As An Operating System Service: A Case Study, ACM Transactions on Computer Systems (TOCS), Volume 24 Issue 1

(Krieger and Strout, 2010) Krieger CD. Strout MM. (2010), Performance Evaluation of an Irregular Application Parallelized in Java. Proceedings of the 39th International Conference on Parallel Processing Workshops (ICPPW), San Diego, CA, USA

(Lagger et al. 2006) Lagger, A., Upegui, A., Sanchez, E. and Gonzalez, I. (2006), Self-Reconfigurable Pervasive Platform for Cryptographic Application, FPL 2006, International Conference on Field Programmable Logic and Applications

(Lantz, 2013) Lantz, B. (2013), Machine learning with R, Packt Publishing Ltd., UK

(Lee et al. 2007) Lee, H.K., Malkin, T. and Nahum, E. (2007), Cryptographic Strength of SSL/TLS Servers: Current and Recent Practices, IMC '07 Proceedings of the 7th ACM SIGCOMM conference on Internet measurement

(Li et al. 2010) Li, G., Zheng, H. and Li, G. (2010), Building A Secure Web Server Based on OpenSSL and Apache, International Conference on E-Business and E-Government ICEE2010

(Lim et al. 2002) Lim, J., Gan, B. and Ting-Ting, C. (2002), A survey on NSS adoption intention, Proceedings of the 35th Annual Hawaii International Conference on System Sciences HICSS-35

(Lipmaa et al. 2000) Lipmaa H., Rogaway P and Wagner D., (2000), Comments to NIST Concerning AES-modes of Operations: CTR-mode Encryption. In Symmetric Key Block Cipher Modes of Operation Workshop, Baltimore, Maryland, USA

(Liskov et al. 2002) Liskov, M. Rivest, R. and Wagner D. (2002), Tweakable block ciphers. Proc. Advances in Cryptology CRYPTO 02, vol.2442 of Lecture Notes in Computer Science, Springer, Germany

(Lloyd, 2013) Lloyd J. (2013), Botan C++ crypto library, Available at: http://botan.randombit.net/index.html

(Lovrić et al. 2008) Lovrić, M., Komić, J., Stević, S., Zečević, T., Žižić, M. and Kočović J. (2008). Statistička analiza - metodi i primjena, Ekonomski fakultet, Univerzitet u Banja Luci, Banja Luka, BiH

(Malita, 2013) Malita, F. (2013). LUFS: Linux Userland FileSystem. http://lufs. sourceforge.net

(Manangi et al. 2010) Manangi, S.J., Chaurasia, P. and Singh, M.P. (2010), Simplified AES for Low Memory Embedded Processors, Global Journal of Computer Science and Technology, Vol. 10 Issue 14

(Manavski, 2007) Manavski, S.A. (2007), CUDA Compatible GPU as an Efficient Hardware Accelerator for AES Cryptography, IEEE International Conference on Signal Processing and Communications,

(Mavrogiannopoulos and Josefsson, 2013) Mavrogiannopoulos, N. and Josefsson, (2013). The GnuTLS manual, Free Software Foundation, Inc., Nikos Mavrogiannopoulos

(MDN NSS RN, 2016) Mozilla Developer Network, (2016). NSS 3.14.2 release notes, Available at: https://developer.mozilla.org/en-US/docs/NSS/NSS_3.14.2_release_notes

(MDN NSS, 2016) Mozilla Developer Network, (2016). Network Security Services (NSS), Available at: https://developer.mozilla.org/en-US/docs/NSS

(Mei et al. 2010) Mei, C., Jiang H., Jenness J., (2010), CUDA-based AES Parallelization with Fine-Tuned GPU Memory Utilization, IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2010, Atlanta, USA

(Menezes et al. 1996) Menezes A.J., van Oorschot P.C., Vanstone S.A., (1996), Handbook Of Applied Cryptography, CRC Press, Inc. Boca Raton, FL, USA.

(Metakides, 1996) Metakides, G. and Nerode, A. (1996), Principles of logic and logic programming, Elsevier B.V.

(Microsoft Corp. 2005) Microsoft Corp. (2005), Using Encrypting File System, Tech. rep. Available at: http://technet.microsoft.com/en-us/library/bb457116.aspx.

(Microsoft Corp. 2008) Microsoft Corp. (2008), How EFS works. Tech. Rep. Available at: http://technet.microsoft.com/en-us/library/cc962103.aspx

(Microsoft Corp. 2014) Microsoft Corp. (2014). FIPS 140 Evaluation, Available at: http://technet.microsoft.com/en-us/library/cc750357.aspx

(Microsoft Corp. 2016) Microsoft Corp. (2016) Microsoft CryptoAPI and Cryptographic Service Providers, TechNet. http://technet.microsoft.com/en-us/library/ cc962093.aspx
(Microsoft Corp. IPC, 2016) Microsoft Corp. (2016), Interprocess Communications, MSDN, Available at: https://msdn.microsoft.com/en-us/library/windows/desktop/aa365574(v=vs.85).aspx
(Microsoft Corp. KB1, 2013) Microsoft Corp. (2013), Slow performance occurs when you copy data to a TCP server by using a Windows Sockets API program, Microsoft Knowledge Base, Article ID: 823764 Available at:http://support.microsoft.com/kb/823764
(Mihailović 2012) Mihailović, D. (2012), Metodologija naučnih istraživanja, Fakultet organizacionih nauka, Beograd. ISBN: 978-86-7680-265-4
(Miljević 2007) Miljević, M. (2012), Metodologija naučnog rada, Filozofski fakultet, Univerzitet u Istočnom Sarajevu, Pale
(Moffat, 2012) Moffat, D. (2012), How to Manage ZFS Data Encryption, Oracle technetwork
(Mollin, 2010) Mollin R. A., (2010), An Introduction to Cryptography, Second Edition, Taylor & Francis
(Mukherjee and Schwan, 1994) Mukherjee, B., and Schwan, K. (1994), Adaptive Operating System Abstractions: A Case Study of Multiprocessor Locks, Technical Report GIT-CC-94/39, College of Computing, Georgia Tech, June 1994.Submitted to ACM TOCS
(Navalgund et al. 2013) Navalgund, S. S., Desai, A., Ankalgi, K. and Yamanur H., (2013) Parallelization of AES Algorithm Using OpenMP, Lecture Notes on Information Theory (LINT) Volume 1, No. 4, December 2013, ROWLAND HEIGHT, USA
(Nishikawa et al. 2011) Nishikawa, N,. Iwai K. and Kurokawa T., High-Performance Symmetric Block Ciphers on CUDA, 2011 Second International Conference on Networking and Computing (ICNC), Osaka, Japan pp. 221-227
(NVIDIA, 2015) NVIDIA Corporation. (2015), NVIDIA CUDA C Programming Guide, Version 7.5
(Ogbuji, 2004) Ogbuji, U., (2004), A survey of XML standards: Part 1, IBM developerWorks, Available at: http://www.ibm.com/developerworks/xml/library/x-stand1/index.html
(OpenSSL Software Foundation, 2015) OpenSSL Software Foundation, (2015). User Guide for the OpenSSL FIPS Object Module v2.0
(Oracle Corporation 2016) Oracle Corporation, (2016), Java™ Platform, Standard Edition 7 API Specification
(Oracle JPT, 2015) Processes and Threads, The Java tutorials, (2015), Oracle Corporation, Available at: http://docs.oracle.com/javase/tutorial/essential/concurrency/procthread.html
(Ortega et al. 2011) Ortega, J., Trefftz, H. and Trefftz, C., (2011), Parallelizing AES on multicores and GPUs, Mankato, USA, pp. 1-5
(Paranjape, 2005) Paranjape, K.H. (2005) Right To Your Own Devices, Linux Gazette, May 2005 (#114)
(Petullo, 2007) Petullo, W.M. (2007), Disk encryption in Fedora: Past, present and future, RedHat Magazine, Available at: http://magazine.redhat.com/2007/01/18/
(Pleskonjić et al. 2007) Pleskonjić D., Maček N., Đorđević B. and Carić M., (2007), Sigurnost računarskih sistema i mreža, Mikro knjiga, Beograd, str.53
(Quinlan, 1992) Quinlan, J.R. (1992), Learning with continuous classes, Proceedings 5th Australian Joint Conference on Artificial Intel ligence, Singapore: World Scientific, 343-348.
(Ray, 2001) Ray, E.T., (2001), Learning XML, O'Reilly Media
(Ramadevi, 2013) Ramadevi G, Sujatha R. (2013) Robust code based fault tolerant architecture using OFB mode for onboard EO satellites. International Journal of Soft Computing and Engineering (IJSCE) 2013; 3(2), str.146–149.
(Ray, 2003) Ray, E.T., (2003), Learning XML, 2nd Edition, O'Reilly Media
(Red Hat Inc, 2016) Red Hat Inc, (2016), Fedora CryptoConsolidation, https://fedoraproject.org/wiki/FedoraCryptoConsolidation
(RFC 5652, 2009) Housley, R. (2009) RFC 5652, Cryptographic Message Syntax (CMS)
(Rijmen, 2010) Rijmen V., (2010), Practical-Titled Attack on AES-128 Using Chosen-Text Relations. IACR Cryptology ePrint Archive (IACR) 2010:337
(Roback and Dworkin, 1998) "Roback E. and Dworkin M., (1998), Conference Report From First Advanced Encryption Standard (AES) Candidate Conference, Journal of Research of the National Institute of Standards and Technology, Ventura, USA

(Rogaway, 2004) Rogaway P. (2004) Efficient instantiations of tweakable blockciphers and refinements to mode OCB and PMAC. Proc. Asiacrypt 2004, Springer, pp. 16-31; doi: 10.1007/978-3-540-30539-2 2.
(Rogaway, 2011) Rogaway, P. (2011), Evaluation of Some Blockcipher Modes of Operation, Cryptography Research and Evaluation Committees (CRYPTRECl)
(Roman, 2006) Roman S., (2006), Field Theory, second edition, Springer-Verlag Berlin-. Heidelberg-New York-London-Paris-Tokyo-Hong Kong
(Roshen 2009) Roshen W. (2009), SOA-Based Enterprise Integration: A Step-by-Step Guide to Services-Based Application Integration, The McGraw-Hill
(Rott 2011) Rott J. (2011), Intel AESNI Sample Library, Available at: http://software.intel.com/en-us/articles/download-the-intel-aesni-sample-library
(Russell and Norvig, 2003) Russell, S.J., Norvig, P. (2003), Artificial Intelligence: A Modern Approach (2nd ed.), Upper Saddle River, New Jersey: Prentice Hall
(Saarinen, 2005) Saarinen, M.J.O. (2005), Encrypted watermarks and linux laptop security, Proceeding WISA'04 Proceedings of the 5th international conference on Information Security Applications, Springer-Verlag Berlin, Heidelberg
(Samara et al. 2009) Samara, S., Orfanus, D. and Janacik, P. (2009), Towards biologically inspired decentralized self-adaptive OS services for distributed Reconfigurable System on Chip (RSoC), ACM SIGBED Review - Special Issue on the 2nd International Workshop on Adaptive and Reconfigurable Embedded Systems (APRES'09)
(Sandberg, 1985) Sandberg, R., Goldberg, D., Kleiman, S., Walsh, D. and Lyon, B. (1985), Design and Implementation of the Sun Network Filesystem, USENIX Conference Proceedings, USENIX Association, Berkeley, CA, Summer
(Sanders and Kandrot, 2011) Sanders, J. and Kandrot, E. (2011), CUDA by Example, An Introduction to General- Purpose GPU Programming, Addison-Wesley
(Shalizi, 2009) Shalizi, C. R., (2009), The Statistical Analysis of Complex Systems Models, Cambridge University Press
(Silberschatz et al. 2011) Silberschatz A., Galvin P.B. and Gagne G., (2011), Operating System Concepts E ssentials, JOHN WILEY & SONS. INC, Hoboken, USA
(Smola and Vishwanathan, 2008) Smola A., and Vishwanathan, S.V.N., (2008), Introduction to Machine Learning, Cambridge University Press
(Sotirović i Adamović, 2005) Sotirović V. i Adamović Ž., Metodologija naučno-istraživačkog rada, Univerzitet u Novom Sadu, Tehnički fakultet "Mihajlo Pupin", Zrenjanin, 2005. god.
(Szeredi, 2008) Szeredi, M. (2008), Filesystem in USEr space, Available at: http://sourceforge.net/projects/avf
(Tanenbaum and Bos, 2014) Tanenbaum A.S. and Bos H. (2014), Modern Operating Systems (4th ed.). Prentice Hall Press, Upper Saddle River, NJ, USA.
(Tarvo and Reiss, 2014) Tarvo A. Reiss SP. (2014) Automated analysis of multithreaded programs for performance modeling. Proceedings of the 29th IEEE/ACM International Conference on Automated Software Engineering, Vasteras, Sweden
(Tarvo and Reiss, 2014) Tarvo A. Reiss SP. (2014) Automated analysis of multithreaded programs for performance modeling. Proceedings of the 29th IEEE/ACM International Conference on Automated Software Engineering, Vasteras, Sweden
(Tirro, 1993) Tirro S., (1993), Satellite Communication Systems Design, Springer Science+Business Media LLC: New York, 1993, str. 101.
(TrueCrypt, 2009) TrueCrypt Foundation, (2009). TrueCrypt User's Guide. Available at: www.truecrypt.org
(von Neumann,1945) von Neumann, J. (1945), First Draft of a Report on the EDVAC, Available at: https://sites.google.com/site/michaeldgodfrey/vonneumann/vnedvac.pdf?attredirects=0&d=1
(Wang and Witten, 1996) Wang, Y. & Witten, I. H. (1996). Induction of model trees for predicting continuous classes. (Working paper 96/23). Hamilton, New Zealand: University of Waikato, Department of Computer Science.
(Wells, 2009) Wells, G.C., (2009), Interprocess Communication in Java, Parallel and Distributed Processing Techniques and Applications, PDPTA 2009: Las Vegas, Nevada, USA
(Williams and Bergmann, 2004) Williams, J.A. and Bergmann, N.W. (2004), Embedded Linux as a platform for dynamically self-reconfiguring systems-on-chip The International

Conference on Engineering of Reconfigurable Systems and Algorithms, Las Vegas, Nevada, USA

(Wilt, 2013) Wilt, N. (2013), The CUDA Handbook (A Comprehensive Guide to GPU Programming), Addison-Wesley

(Witten et al. 2011) Witten, I.H., Frank, E., Hall, M.A., (2011), Data Mining - Practical Machine Learning Tools and Techniques, Third Edition, Elsevier

(Wright, 2004) Wright, C.P. (2004), Operating System Support for Extensible Secure File Systems, Technical Report FSL-04-02

(XML W3C 1.0, 2008) Extensible Markup Language (XML) 1.0 (Fifth Edition), W3C Recommendation, (2008), Available at: http://www.w3.org/TR/xml/

(XML W3C 1.1, 2006) Extensible Markup Language (XML) 1.1 (Second Edition), W3C Recommendation, (2006), Available at: http://www.w3.org/TR/xml11/

(Xu, 2010) Xu, L. (2010), Intel Corporation, White Paper, Securing the Enterprise with Intel AES-NI

(Youseff et al. 2012) Youseff, L., Beckmann, N., Kasture, H., Gruenwald, C., Wentzlaff, D. And Agarwal A. (2012),The Case for Elastic Operating System Services in fos, DAC '12 Proceedings of the 49th Annual Design Automation Conference

(Zadok and Badulescu, 1999) Zadok, E., Badulescu, I. and Shender, A. (1999), Extending File Systems Using Stackable Templates, Proceedings of the USENIX Annual Technical Conference, Monterey, California, USA

(Zadok and Nieh, 2000) Zadok, E., and Nieh, J. (2000), FiST: A language for stackable file systems. In Proceedings of the Annual USENIX Technical Conference. USENIX Association, Berkeley, CA, 55–70

(Zadok et al. 1998) Zadok, E., Badulescu, I. and Shender, A. (1998), Cryptfs: A Stackable Vnode Level Encryption File System, Technical Report CUCS-021-98. Computer Science Department, Columbia University, Available at http://www.cs.columbia.edu/~library/.

(Zadok, 2001) Zadok, E. (2001), FiST: A System for Stackable File - System Code Generation, PhD Thesis, COLUMBIA UNIVERSITY

(Zhang et al. 2007) Zhang, L., Krintz, C., Nagpurkar, P., (2007), Language and Virtual Machine Support for Efficient Fine-Grained Futures in Java, Proc. PACT '07, IEEE Computer Society Washington, DC, USA, 130-139.

(Zola, 2012) Zola W.M.N., De Bona L.C.E., (2012), Parallel Speculative Encryption of Multiple AES Contexts on GPUs, Innovative Parallel Computing (InPar).

BIOGRAFIJA AUTORA

Boris Damjanović je rođen 02.06.1965. godine u Banja Luci. Završio je srednju Elektro-tehničku školu u Prijedoru 1983. godine. Visoku školu za ekonomiju i informatiku u Prijedoru završava 2008. godine sa prosječnom ocenom 9.61. Nakon toga upisuje diplomske akademske studije - Master na Fakultetu organizacionih nauka u Beogradu, studijski program Informacioni sistemi i tehnologije, studijsko područje Informacioni sistemi. Master rad pod naslovom „Implementacija i proširenje AES algoritma“ pod mentorstvom prof. dr Simić Dejana odbranio je sa ocjenom 10 i završio studije sa prosječnom ocjenom 10.00. Boris Damjanović upisao je dana 05.10.2010. godine doktorske studije, izborno područje Informacioni sistemi, na Fakultetu organizacionih nauka Univerziteta u Beogradu. Dana 25.11.2013. godine, Boris Damjanović je na Fakultetu organizacionih nauka Univerziteta u Beogradu odbranio Pristupni rad pod nazivom „ADAPTIBILNA PRIMJENA AES ALGORITMA KOD SAVREMENIH OPERATIVNIH SISTEMA“.

Programiranjem i razvojem informacionih sistema se bavi od 1988. godine. Od 1991. zaposlen je u ITC Kozarski Vijesnik - Radio Prijedor, najprije kao realizator programa, a kasnije i kao šef tehnike. Od 1998. godine radi u preduzeću Esprit Radio u Prijedoru, najprije kao šef tehnike a zatim kao menadžer. Od 2008. godine zaposlen je u preduzeću Commercial D.O.O. kao programer i suvlasnik. Autor je velikog broja raznorodnih aplikacija i ERP rješenja (Media Light, Media Pro, RadioAMP, Sinapsa, KFKK, FCMidi, FCMidiVP, FCMini, KasaPDV...) i internet prezentacija (Geno Balkan, Esprit Radio, AGK, ...). Paralelno sa radom u privredi, počevši od školske 2008/2009 godine Boris Damjanović je kao saradnik u nastavi na Visokoj školi za ekonomiju i informatiku Prijedor učestvovao u pripremi, izvođenju vežbi i predavanja, kao i ispita na predmetima: Zaštita računarskih sistema i Arhitektura i funkcija računara. Počevši od školske 2010/2011 godine, kandidat je kao predavač na Visokoj školi za ekonomiju i informatiku učestvovao u pripremi, izvođenju vežbi, predavanja, kao i ispita na predmetima: Baze podataka 1, Baze podataka 2, Zaštita računarskih sistema i Arhitektura i funkcija računara. Od školske 2014/2015 godine Boris Damjanović je kao saradnik u nastavi Univerziteta za poslovni inženjering i menadžment u Banja Luci učestvovao u pripremi, izvođenju vežbi i predavanja, kao i ispita na predmetima: Kriptografija, Uvod u WWW, Softverski studio 2, Softverski studio 3, Programiranje internet aplikacija i Operativni sistemi.

Tokom svog dosadašnjeg naučno-istraživačkog rada, Boris Damjanović je objavio, u svojstvu prvog autora 2 rada u časopisima međunarodnog značaja (SCIe lista sa Impakt faktorom), u svojstvu prvog autora ili koautora 5 radova u časopisima domaćeg značaja i 6 radova na domaćim konferencijama.

Prilog 1.

Izjava o autorstvu

Potpisani-a Damjanović Boris

broj indeksa 5005/2010

Izjavljujem

da je doktorska disertacija pod naslovom

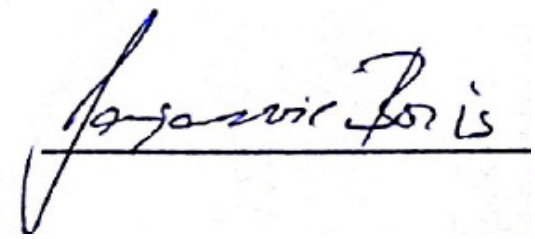
Adaptibilna primjena AES algoritma kod savremenih operativnih sistema

---

- rezultat sopstvenog istraživačkog rada,
- da predložena disertacija u celini ni u delovima nije bila predložena za dobijanje bilo koje diplome prema studijskim programima drugih visokoškolskih ustanova,
- da su rezultati korektno navedeni i
- da nisam kršio/la autorska prava i koristio intelektualnu svojinu drugih lica.
-

Potpis doktoranda

U Beogradu, 15.04.2016



Prilog 2.

Izjava o istovetnosti štampane i elektronske verzije doktorskog rada

Ime i prezime autora _Boris Damjanović_____

Broj indeksa _5005/2010_____

Studijski program _Informacioni sistemi_____

Naslov rada _**Adaptibilna primjena AES algoritma kod savremenih operativnih sistema**_

Mentor __prof.dr. Dejan Simić_____

Potpisani/a ___ Boris Damjanović _____

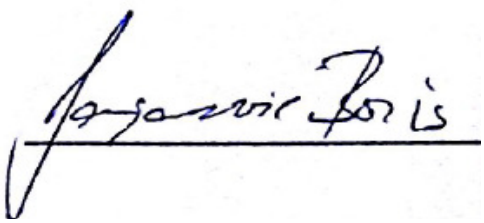
Izjavljujem da je štampana verzija mog doktorskog rada istovetna elektronskoj verziji koju sam predao/la za objavljivanje na portalu **Digitalnog repozitorijuma Univerziteta u Beogradu**.

Dozvoljavam da se objave moji lični podaci vezani za dobijanje akademskog zvanja doktora nauka, kao što su ime i prezime, godina i mesto rođenja i datum odbrane rada.

Ovi lični podaci mogu se objaviti na mrežnim stranicama digitalne biblioteke, u elektronskom katalogu i u publikacijama Univerziteta u Beogradu.

Potpis doktoranda

U Beogradu, ___15.04.2016_____



Prilog 3.

Izjava o korišćenju

Ovlašćujem Univerzitetsku biblioteku „Svetozar Marković“ da u Digitalni repozitorijum Univerziteta u Beogradu unese moju doktorsku disertaciju pod naslovom:

Adaptibilna primjena AES algoritma kod savremenih operativnih sistema

koja je moje autorsko delo.

Disertaciju sa svim prilozima predao/la sam u elektronskom formatu pogodnom za trajno arhiviranje.

Moju doktorsku disertaciju pohranjenu u Digitalni repozitorijum Univerziteta u Beogradu mogu da koriste svi koji poštuju odredbe sadržane u odabranom tipu licence Kreativne zajednice (Creative Commons) za koju sam se odlučio/la.

1. Autorstvo

2. Autorstvo - nekomercijalno

3. Autorstvo – nekomercijalno – bez prerade

4. Autorstvo – nekomercijalno – deliti pod istim uslovima

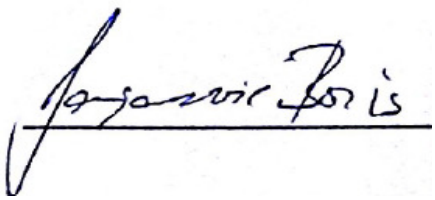
5. Autorstvo – bez prerade

6. Autorstvo – deliti pod istim uslovima

(Molimo da zaokružite samo jednu od šest ponuđenih licenci, kratak opis licenci dat je na poleđini lista).

Potpis doktoranda

U Beogradu, 15.04.2016



1. Autorstvo - Dozvoljavate umnožavanje, distribuciju i javno saopštavanje dela, i prerade, ako se navede ime autora na način određen od strane autora ili davaoca licence, čak i u komercijalne svrhe. Ovo je najslobodnija od svih licenci.

2. Autorstvo – nekomercijalno. Dozvoljavate umnožavanje, distribuciju i javno saopštavanje dela, i prerade, ako se navede ime autora na način određen od strane autora ili davaoca licence. Ova licenca ne dozvoljava komercijalnu upotrebu dela.

3. Autorstvo - nekomercijalno – bez prerade. Dozvoljavate umnožavanje, distribuciju i javno saopštavanje dela, bez promena, preoblikovanja ili upotrebe dela u svom delu, ako se navede ime autora na način određen od strane autora ili davaoca licence. Ova licenca ne dozvoljava komercijalnu upotrebu dela. U odnosu na sve ostale licence, ovom licencom se ograničava najveći obim prava korišćenja dela.

4. Autorstvo - nekomercijalno – deliti pod istim uslovima. Dozvoljavate umnožavanje, distribuciju i javno saopštavanje dela, i prerade, ako se navede ime autora na način određen od strane autora ili davaoca licence i ako se prerada distribuira pod istom ili sličnom licencom. Ova licenca ne dozvoljava komercijalnu upotrebu dela i prerada.

5. Autorstvo – bez prerade. Dozvoljavate umnožavanje, distribuciju i javno saopštavanje dela, bez promena, preoblikovanja ili upotrebe dela u svom delu, ako se navede ime autora na način određen od strane autora ili davaoca licence. Ova licenca dozvoljava komercijalnu upotrebu dela.

6. Autorstvo - deliti pod istim uslovima. Dozvoljavate umnožavanje, distribuciju i javno saopštavanje dela, i prerade, ako se navede ime autora na način određen od strane autora ili davaoca licence i ako se prerada distribuira pod istom ili sličnom licencom. Ova licenca dozvoljava komercijalnu upotrebu dela i prerada. Slična je softverskim licencama, odnosno licencama otvorenog koda.