Implementacija simetričnih blokovnih kriptosustava na Android platformi

Kovačević, Zoran

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij

IMPLEMENTACIJA SIMETRIČNIH BLOKOVNIH KRIPTOSUSTAVA NA ANDROID PLATFORMI

Diplomski rad

Zoran Kovačević

Osijek, 2016.

SADRŽAJ

1. UVOD

Simetrični blokovski kriptografski sustavi su najistaknutiji elementi u mnogim kriptografskim sustavima. Kao osnovni element u konstrukciji kriptografskih protokola pružaju povjerljivost, a njihova svestranost omogućuje kreiranje generatora pseudo nasumičnih brojeva, linijskih kriptografskih sustava, MAC (engl. *Message authentication codes*, kodova za autentifikaciju poruka) i *hash* funkcija [1]. Također mogu poslužiti kao centralna komponenta pri tehnikama autentifikacije poruka, mehanizmima provjere integriteta i izradi digitalnih potpisa.

Nisu svi blokovski kriptografski protokoli pogodni za sve primjene. To je rezultat nezaobilaznih prilagodbi zahtjevima konkretnih primjena, koje uključuju ograničenja brzine i memorije (veličina koda implementacije, veličina blokova, količina priručne memorije), programska ili hardverska ograničenja platforme na kojoj se implementiraju. Također, kompromis se mora raditi između efikasnosti i sigurnosti implementacije [2]. Zbog navedenih činjenica korisno je imati više kandidata prilikom odabira kriptografskog protokola [1].

U ovom radu primijenjeni su bivši i trenutni preporučeni protokoli DES i AES [2]. Trostruki DES (3DES) je također primijenjen zbog svoje raširenosti jer je zamišljen kao jednostavno proširenje sigurnosnih karakteristika DES protokola. DES je uzet kao primjer kriptografskog protokola baziranog na *Feistel* mreži. S obzirom na promatrane karakteristike u ovom radu, primjena ostalih protokola baziranih na istom principu nije potrebna.

Rad proučava efikasnost podjele procesa šifriranja na više niti s ciljem povećanja brzine.

U prvom poglavlju rad iznosi teorijsku podlogu potrebnu za razumijevanje problematike. Opisuje u detalje način rada pojedinog protokola, od broja rundi do strukture pojedine runde. Također, prikazani su najčešći načini primjene simetričnih kriptografskih protokola (ECB, CTR, OFB…)

Platforma na kojoj se vrši testiranje je Android operacijski sustav, u svojoj najraširenijoj verziji u trenutku pisanja rada [3]. Kreirana je Android aplikacija koja vrši testiranje ovisno o definiranim parametrima te pohranjuje rezultat u memoriju uređaja. Drugo poglavlje opisuje kreiranu aplikaciju. Izvorni kod aplikacije dan je u prilogu 1. Proučavane su razlike u veličini izvornog teksta, broju niti, primijenjenom protokolu te razlike između uređaja. Za veličine izvornog teksta uzet je raspon između 1 MB i 50MB. Prilikom testiranja svi uređaji su bili sposobni izvršiti testove s veličinom izvornog teksta od 40 MB, čime je prikazana gornja

granica veličine izvornog teksta koje se može uspješno šifrirati na svim uređajima. Sva testiranja izvršena su pri ECB načinu rada. Ostali načini, osim CTR, ne podržavaju paralelno izvođenje, dok se CTR ne razlikuje značajno da bi predstavlja razliku u rezultatima.

Treće poglavlje iznosi rezultate testiranja. Rezultati su obrađeni i prikazani tablično i grafički. Neobrađeni rezultati dani su u prilogu 2. Obrada rezultata obavljena je u Microsoft Excel programu. Datoteka s obrađenim rezultatima dana je u prilogu 3.

Četvrto poglavlje prikazuje raspravu o rezultatima mjerenja. Ponuđena su objašnjenja za rezultate mjerenja.

2. TEORIJSKA PODLOGA

2.1.Feistel mreža

Feistel mreža (*eng. Feistel network*) je princip za izradu simetričnih kriptografskih sustava. Nazvana je po Horstu Feistelu koji je prvi opisao proceduru prilikom razvoja protokola *Lucifer*, prethodniku DES standarda. Ostali poznatiji sustavi koji primjenjuju princip *Feistel* mreže su *IDEA, RC5, Skipjack* [4].

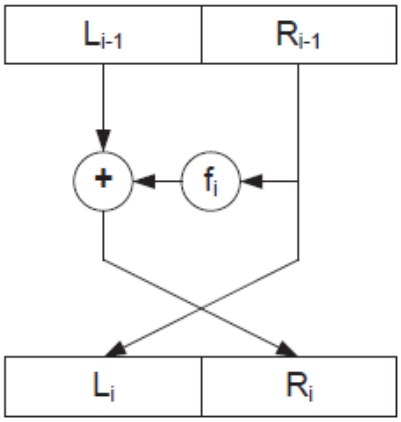
Feistel mreža opisana je funkcijom F: $\{0, 1\}^{2n}$ -> $\{0,1\}^{2n}$. Opisuju je broj rundi d ϵ N i skup funkcija rundi $f_1, \ldots, f_d$: $\{0, 1\}^n$ -> $\{0,1\}^n$. Tijekom *d* ciklusa blok podataka *B* poprima stanja $B_0 \ldots B_d$.

U *i*-tom ciklusu izvode se sljedeće operacije prikazane slikom:

- blok podataka B_{i-1} podjeli se na dva jednaka dijela $L_{i-1} \mid R_{i-1}$
- funkcija promjene f_i primjenjuje se na blok R_{i-1} pri čemu dobivamo $f_i(R_{i-1})$
- računa se isključivo ILI između dijela L_{i-1} i $f_i(R_{i-1})$ pri čemu dobivamo $L_{i-1} \oplus f_i(R_{i-1})$
- blok L_i poprima vrijednost bloka R_{i-1}, a blok R_i poprima vrijednost $L_{i-1} \oplus f_i(R_{i-1})$
- blok podataka B_i stvara se spajanjem blokova $L_i \mid R_i$

Funkcija *F* poprima oblik:

$$F_i(L_{i-1} \mid R_{i-1}) = R_{i-1} \mid L_{i-1} \oplus f_i(R_{i-1}) \qquad (2.1)$$



Sl. 2.1: Prikaz funkcije za šifriranje [4]

Ovako opisana Feistel mreža svodi se na skup permutacija za definirane funkcije f_i. Također, funkcije permutacije ne moraju biti reverzibilne.

2.1.1. Reverzibilnost f_i funkcija

Pretpostavimo kandidata za inverz Gi za ulaz $L_i \,||\, R_i$ kao $R_i \oplus f_i(L_i) \,||\, L_i$. Slijedeći izraz pokazuje da je kandidat uistinu inverz funkcije Fi.

$G_i(F_i(L_{i-1} \,||\, R_{i-1})) = G_i(R_{i-1} \,||\, L_{i-1} \oplus f_i(R_{i-1}))$

$= (L_{i-1} \oplus f_i(R_{i-1})) \oplus fi(R_{i-1}) \,||\, R_{i-1}$

$= L_{i-1} \,||\, R_{i-1}$

U slučaju da F funkcija ima *d* rundi, inverz definiramo kao $G = G_1 \circ ... \circ G_d$. Izraz pokazuje da je G uistinu inverz F:

$G_1(G_2(:::G_{d-1}(G_d(F_d(F_{d-1}(:::F_1(m):::))):::))$
$= G_1(G_2(:::G_{d-1}(F_{d-1}(:::F_1(m):::)):::))$
$= :::$
$= G_1(F_1(m))$
$= m:$

Iz priloženog vidimo da Feistel mreže imaju korisno svojstvo da se isti algoritam može koristiti za šifriranje i dešifriranje, što je bilo posebno važno prilikom razvoja prvih blokovski sustava u 60-tim i 70-tim kada je bila visoka cijena opreme. Razlika na koju treba obratiti pažnju je redoslijed f_i funkcija, što se svodi na promjenu redoslijeda parametara, tj. dijela ključa koji se primjenjuje u pojedinom ciklusu.

2.2.DES

Do početka 70-tih godina u Sjedinjenim Američkim Državama nije postojao standard za šifriranje podataka. Postojali su protokoli čije karakteristike u vidu sigurnosti nisu bile detaljno ispitane [4]. Američka NBS (*National Bureau of Standards,* danas NIST*, National Institute of Standards and Technology*) raspisuje natječaj za izradu standarda za šifriranje. Kriteriji su bili:

- visok nivo sigurnosti
- algoritam mora biti u potpunosti specificiran i razumljiv
- sigurnost mora biti bazirana u ključu; sigurnost ne smije ovisiti o tajnosti algoritma
- algoritam mora biti javno dostupan
- implementacija algoritma mora biti ekonomski isplativa
- algoritam mora biti efikasan
- algoritam mora biti verificiran
- algoritam mora biti neovisan o platformi i opremi

Kompanija IBM je prijavila algoritam pod imenom *Lucifer*, koji su razvili Roy Adler, Don Coppersmith, Edna Grossman, Alan Konheim, Carl Meyer, Bill Notz, Lynn Smith, Walt Tuchman, and Bryant Tuckerman. Također, u izradi je sudjelovao Horst Feistel, autor Fiestel principa na kojem se temelji prijavljeni algoritam. Nakon analize Agencije za nacionalnu sigurnost 1975. godine algoritam je objavljen te predan na analizu drugim agencijama i javnosti.

Jedan od argumenata protiv algoritma je da je NSA modificirala algoritam kako bi omogućila postojanje tajnih načina koji bi omogućili probijanje zaštite u znatno kraćem vremenu od *Brute force* napada [4]. Također, širina ključa od 56 bita je kritizirana kao prekratka. Unatoč kritikama, predloženi standard je prihvaćen pod imenom DES.

Ovo je prvi algoritam koji je procijenjen od strane Agencije za nacionalnu sigurnost koji je javno dostupan. Iako je standard nalagao isključivo hardversku implementaciju, objavljeno je dovoljno informacija koje su omogućile i programsku implementaciju. Po prvi puta je postojao algoritam odobren od strane NSA dostupan za analizu. Javna dostupnost algoritma može dodatno potvrditi njegovu kvalitetu, ali može otkriti slabosti čije posljedice mogu biti katastrofalne. Sljedeći kriptografski standard američke vlade koji se koristio, *Skipjack,* bio je povjerljiv.

Prvo uspješno probijanje algoritma za zadani skup šifriranih i izvornih blokova prijavljeno je 1997. godine od strane DESCHALL projekta [5]. DESCHALL je koristio prazne cikluse računala spojenih preko interneta. Sljedeći dokaz nepouzdanosti DES standarda pokazao je Electronic Frontier Foundation, koji su za cijenu od 250.000 dolara napravili računalo sposobno probiti par izvornog i šifriranog teksta za manje od 3 dana. Cilj njihovog projekta bio je pokazati da je probijanje DES algoritma moguće u praksi te su pokazali koja je cijena toga. Prijavljeno je probijanje za manje od jednog dana s opremom koja ima snagu manju od 1kW [6] i zauzima prostor od jedne 3HU kutije. Daljnji napori za probijanje nisu potrebni jer dostupna tehnologija čini DES standard u potpunosti zastarjelim.

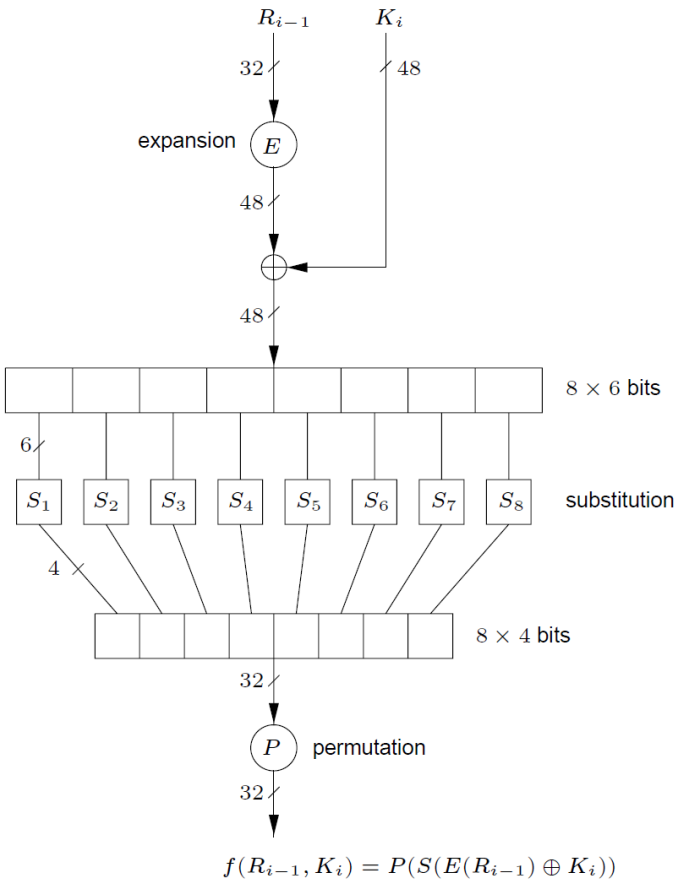
2.2.1. Opis rada

Data encryption standard (DES) je kriptografski sustav temeljen na Feistel principu. Sastoji se od 16 identičnih ciklusa koji obrađuju blok širine 64 bita. Veličina ključa je 56 bita. Uobičajeno se koristi širina ključa od 64 bita ili 8 bajta, gdje je svaki osmi bit zapravo provjera pariteta bajta. Protokol koristi standardne aritmetičke i logičke operatore, zbog čega je pogodan za upotrebu na jednostavnoj opremi.

Prije primjene Feistel mreže, na blok se primjenjuje IP (inicijalna permutacija) koja je dana tablicom 2.1. Prvi bit izlaza je 58. bit ulaza, drugi bit izlaza je 50. bit ulaza itd.

Tab. 2.1: Inicijalna permutacija

IP															
58	*50*	*42*	*34*	*26*	*18*	*10*	*2*	*60*	*52*	*44*	*36*	*28*	*20*	*12*	*4*
62	*54*	*64*	*38*	*30*	*22*	*14*	*6*	*64*	*56*	*48*	*40*	*32*	*24*	*16*	*8*
57	*49*	*41*	*33*	*25*	*17*	*9*	*1*	*59*	*51*	*43*	*35*	*27*	*19*	*11*	*3*
61	*53*	*45*	*37*	*29*	*21*	*13*	*5*	*63*	*55*	*47*	*39*	*331*	*23*	*15*	*7*



Sl. 2.2: Ciklus DES algoritma [1]

F_i funkcija ciklusa radi na sljedeći način, prikazano na slici 2.2.

Sastoji se od sljedećih koraka:

- Blok se proširuje na 48 bita dodavanjem prvih 16 bita te primjenom permutacije *E* danom tablicom 2.2
- Dobivenom bloku dodaje se Ki permutacija ključa logičkom operacijom isključivo ili
- Novonastali blok se dijeli u 8 blokova veličine 6 bita te se svaki blok koristi za ulaz u modul za zamjenu (S-Box engl. *Substitution box*) koji daje 4 bita. Izlazi modula za zamjenu se spajaju u jedan blok širine 8*4 bita = 32 bita
- Primjenjuje se permutacija P na dobiveni blok

Tab. 2.2: Permutacije E [1]

E:					
32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
14	25	26	27	28	29
28	29	30	31	32	1

Prvi bit izlaza je 32. bit ulaza. Drugi bit izlaza je prvi bit ulaza i tako dalje.

Permutacije ključa prikazane su tablicom 2.3. Prvih 48 bitova permutacije koristi se kao ključ K. Svaki bit se koristi u otprilike 14 od 16 ciklusa:

Tab. 2.3: Permutacije ključa DES algoritma [1]

K													
57	49	41	33	25	17	9	1	58	50	42	34	26	15
10	2	59	51	43	35	27	19	11	3	60	52	44	36
63	55	47	39	31	23	15	7	62	54	46	38	30	22
14	6	61	53	45	37	29	21	13	5	28	20	12	4

Permutacija P koja se primjenjuje kao zadnja operacija funkcije f_i dana je tablicom 2.3.

Tab. 2.4: Permutacije P DES algoritma [1]

P							
16	7	20	21	29	12	28	17
1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9
19	13	30	6	22	11	4	25

Moduli za zamjenu (S-box) prikazani su tablicama 2.5, 2.6, 2.7, 2.8, 2.9, 2.10, 2.11 i 2.12. Pridjeljivanje ulaza i izlaza obavlja se na sljedeći način: Za zadani $x_0x_1x_2x_3x_4x_5$ blok na ulaz, vrijednost x_0x_5 određuju redak, a vrijednost $x_1x_2x_3x_4$ stupac. Npr. 011001_2 daje redak $01_2 = 1$ i stupac $1100_2 = 12$ te S_1 na izlaz daje $9 = 0101_2$.

Tab. 2.5: S-box za prvi skup od 6 bita DES algoritma [1]

S1															
14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	O	6	13

Tab. 2.6: S-box za drugi skup od 6 bita DES algoritma [1]

S2:															
15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

Tab. 2.7: S-box za treći skup od 6 bita DES algoritma [1]

S3:															
10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

Tab. 2.8: S-box za četvrti skup od 6 bita DES algoritma [1]

S4:															
7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

Tab. 2.8: S-box za peti skup od 6 bita DES algoritma [1]

S5:															
2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	O	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

Tab. 2.9: S-box za šesti skup od 6 bita DES algoritma [1]

S6:															
12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

Tab. 2.11: S-box za sedmi skup od 6 bita DES algoritma [1]

S7:															
4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

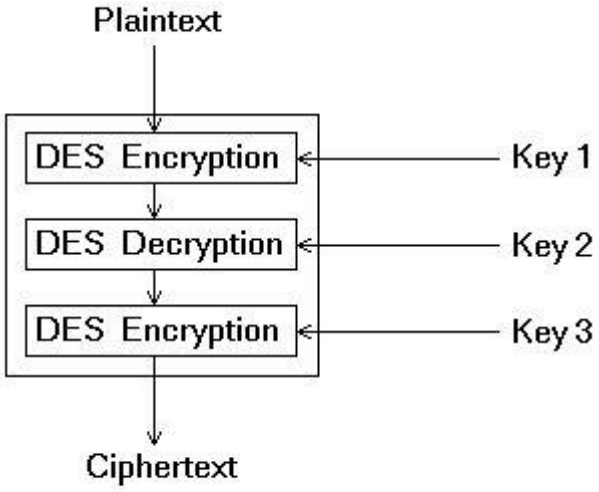
Tab. 2.12: S-box za osmi skup od 6 bita DES algoritma [1]

S8:															
13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

2.3. 3DES

Trostruki DES (eng. *Triple DES*, skraćeno *3DES*) je unaprijeđeni standard baziran na DES standardu [7]. Osnovna premisa nastanka protokola je jednostavnije poboljšanje postojećeg DES protokola [4].

Trostruki DES primjenjuje DES kriptografski standard 3 puta uzastopno na bloku izvornog teksta. Rezultat svake primjene služi kao izvorni tekst sljedeće primjene. Prva i treća primjena šifriraju izvorni tekst, dok druga primjena dešifrira izvorni tekst. Slika 2.2 prikazuje 3DES protokol [8].



Sl. 2.2: Princip rada 3DES sustava

Širina ključa može biti 56, 112 ili 168 bita. Primjenom širine ključa 56 isti se ključ upotrebljava u sve tri operacije. Kako prva operacija šifrira blok, a druga dešifrira, 3DES u ovom slučaju daje jednaki rezultat kao DES, jer je izvorni tekst nakon druge operacije jednak originalnom izvornom tekstu. Pri veličini ključa od 112 bita ključ se dijeli na dva pod ključa, gdje se prvi upotrebljava u prvoj i trećoj operaciji, a drugi u drugoj. Pri veličini ključa od 168 bita ključ se dijeli na 3 pod ključa, gdje se svaki koristi u jednoj operaciji.

2.4.AES

AES kriptografski sustav nasljednik je zastarjelog DES sustava. Standardiziran je 2001. godine [8]. Radi na blokovima od 128 bita i podržava veličinu ključeva od 128, 192 i 256 bita. Broj ciklusa je 10, 12 ili 14.

2.4.1. Konstrukcija

AES radi na bloku od 16 bajtova organiziranih u matricu 4x4. Inicijalizira se s porukom $m = m_0 \| m_1 \| ... \| m_{15}$ na sljedeći način, prikazan slikom 2.3.

M_0	M_4	M_8	M_{12}
M_1	M_5	M_9	M_{13}
M_2	M_6	M_{10}	M_{14}
M_3	M_7	M_{11}	M_{15}

$A_{0,0}$	$A_{0,1}$	$A_{0,2}$	$A_{0,3}$
$A_{1,0}$	$A_{1,1}$	$A_{1,2}$	$A_{1,3}$
$A_{2,0}$	$A_{2,1}$	$A_{2,2}$	$A_{2,3}$
$A_{3,0}$	$A_{3,1}$	$A_{3,2}$	$A_{3,3}$

Sl. 2.3: Preslikavanje poruke u matricu [8]

Svaka runda sastoji se od sljedećih koraka:

1. operacija supstitucije svakog bajta
2. permutacija redaka
3. manipulacija stupaca
4. isključivo ili primjenom ključa.

Iznimka je posljednja runda u kojoj se preskače manipulacija stupaca.

1. Operacija je slična primjenom *S-box* zamjene. Svaki bajt iz bloka mijenja se primjenom *S-box* zamjene. Sadržaj *S-box* ima algebarsku reprezentaciju te se pri implementaciji može koristiti potpuno kreirana tablica ili proračun, ovisno da li je brzina ili zauzeće prostora u memoriji važnije.
2. Permutacija redaka. Sl. 2.4 daje permutaciju redaka:

A0,0	A0,1	A0,2	A0,3
A1,0	A1,1	A1,2	A1,3
A2,0	A2,1	A2,2	A2,3
A3,0	A3,1	A3,2	A3,3

A0,0	A0,1	A0,2	A0,3
A1,1	A1,2	A1,3	A1,0
A2,2	A2,3	A2,0	A2,1
A3,3	A3,0	A3,1	A3,2

Sl. 2.4: Permutacija redaka [8]

3. Promjena stupaca dana je jednadžbom 2.1.

$$\begin{pmatrix} a'0i \\ a'1i \\ a'2i \\ a'3i \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} a0i \\ a1i \\ a2i \\ a3i \end{pmatrix}$$

Jednadžba 2.1: Manipulacija stupaca [8]

4. Dodavanje ključa primjenom isključivo ili funkcije. Varijante ključa koje se primjenjuju u pojedinoj rundi izvode se na sljedeći način:

- Računa se niz bajtova koji se grupiraju u blokove (riječi) veličine 4 bajta. Dobivene riječi se dalje grupiraju u blokove veličine 16 bajta (4 riječi) kako bi odgovarali veličini bloka koji se šifrira. Slika 2.5 prikazuje pseudo kod za generiranje bajtova.
- N_k odgovara jednoj četvrtini broja bajtova u ključu (256 b = 32B, N_k = 8). Prvih N_k riječi se kreira kopiranjem sadržaja ključa. Ostale riječi dobiju se primjenom isključivo ili operacije prijašnjeg ključa i w[i-N_k] ključa.

```
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
```

Sl. 2.5: Generiranje slijeda ključeva [9]

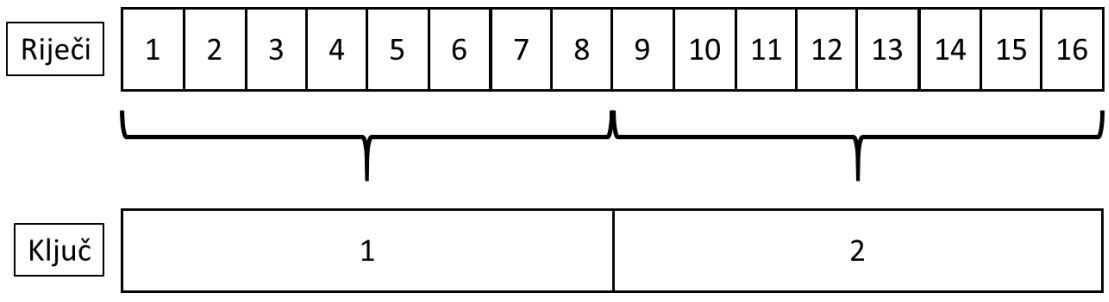
2.4.2. Kreiranje ključeva

Algoritam kreira niz 32-bitnih riječi *W* koristeći ključ za šifriranje. Kreira se ukupno N_b*(N_r + 1) riječi. N_b je broj riječi potrebnih u jednoj rundi, a N_r je broj rundi. Prvih N_k riječi dobiveno je popunjavanjem sadržaja ključa, gdje je N_k broj riječi čija ukupna duljina odgovara veličini ključa. Svaka sljedeća riječ dobivena je logičkom operacijom isključivo ILI prethodne riječi i riječi za N_k mjesta prije. Za riječi koje se nalaze na pozicijama koje su višekratnik N_k primjenjuje se drugačiji postupak: prethodna riječ se rotira, primjenjuje se supstitucija pomoću *S-boxa*. Primjenjuje se logičko ILI s dobivenom riječi i konstantom R_{con}, unaprijed definiranom za svaki korak.

Ključ za rundu *i* dobiva se spajanjem N_k riječi iz dobivenog niza. Na primjer, za veličinu ključa od 256 bita potrebno je ($Nk = \frac{256bita}{32bita/rije\check{c}i} rije\check{c}i$) 8 riječi. Ključ za pojedini ciklus *i* određuje se po formuli 2.2:

$$K_i = W_{(i-1)*8+1}\, W_{(i-1)*8+2} W_{(i-1)*8+3} W_{(i-1)*8+4},\ i \in [1,14] \qquad (2.2)$$

Ključ na poziciji *i* kreira se pomoću riječi na pozicijama (*i*-1)*8, (*i*-1)*8 + 1, (*i*-1)*8 + 2, (*i*-1)*8 + 3, kako prikazuje slika 2.6:

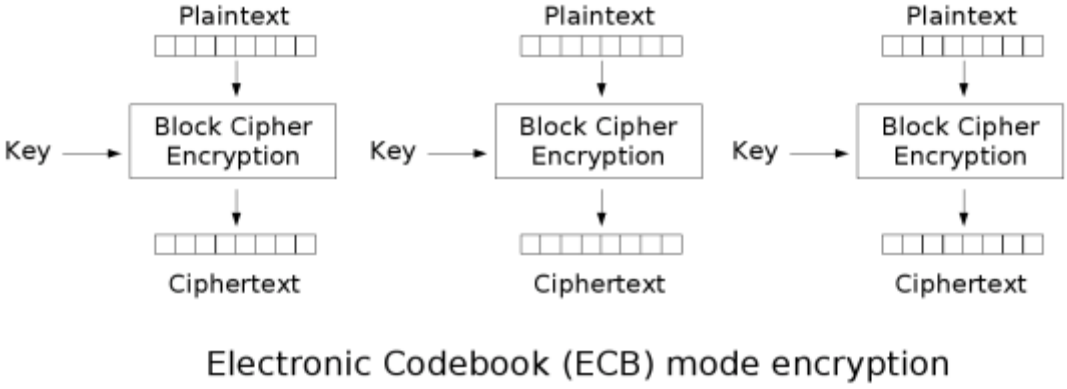


Sl. 2.6: Kreiranje ključeva iz sekvence riječi

2.5. Načini rada

2.5.1. *Electronic Codebook* (ECB)

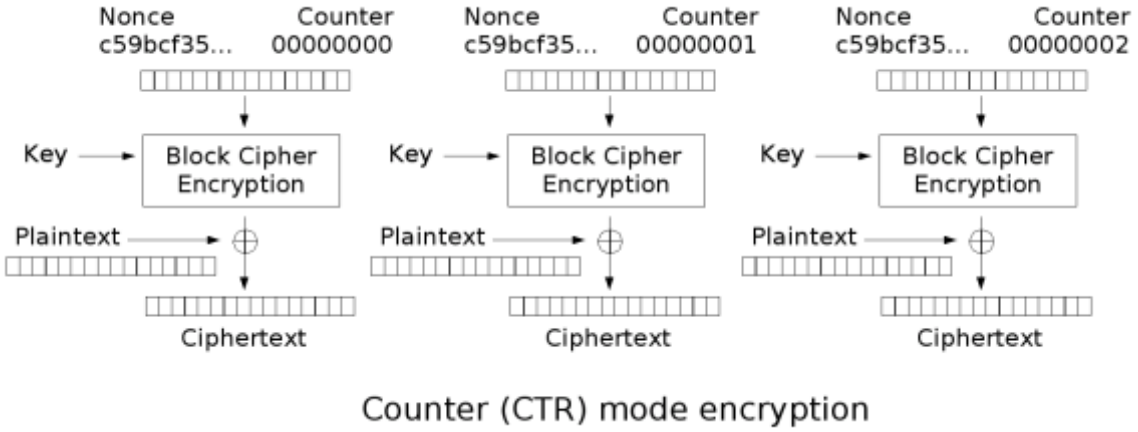
Jednostavna supstitucija ili *Electronic Codebook Mode* (skraćeno *ECB*) [10] je način šifriranja koji, za dani ključ, kreira blok šifrat (engl. *ciphertext*) određene duljine za zadani blok otvorenog teksta (engl. *plaintext*). Slika 2.7 prikazuje ECB način šifriranja.



Sl. 2.7: Šifriranja i dešifriranja ECB načinom [11]

2.5.2. *Counter Mode* (CTR)

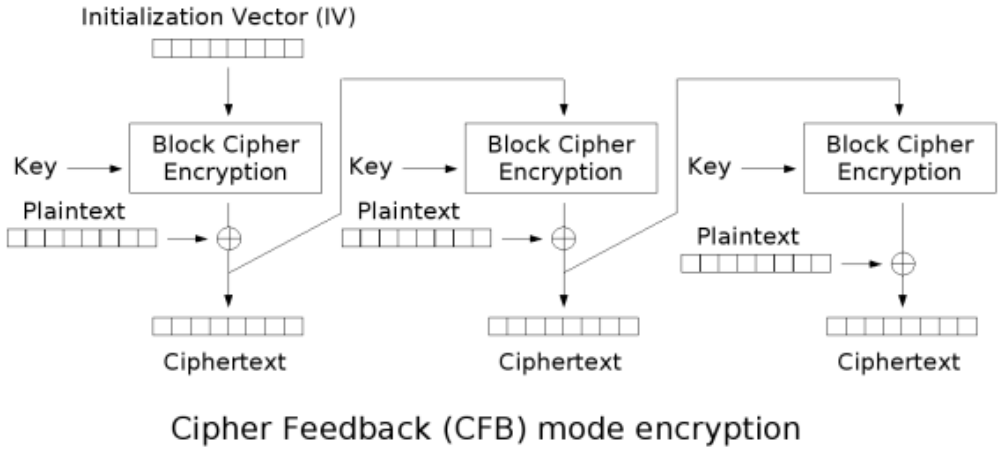
Otvoreni tekst (*plaintext*) dijeli se na blokove *P*. Za kreiranje prvog bloka potrebno je definirati početno stanje registra R_3 i R_4. Ono se naziva početni vektor (engl. *initialization vector IV*). *IV* ne mora biti tajan. Šifriranje svih blokova odvija se na isti način. Slika 2.8 prikazuje dijagram šifriranja CTR načinom.



Sl. 2.8: Dijagram CTR načina šifriranja [11]

2.5.3. *Cipher Feedback Mode* (CFB)

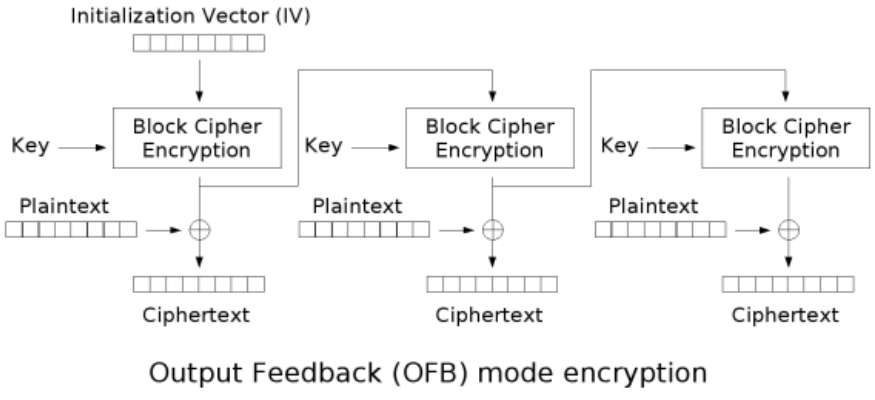
Šifriranje s povratnom vezom (eng. *Cipher Feedback Mode*, skraćeno CFB) je način šifriranja koji, za zadane početne uvjet *IV* i ključ K, stvara niz izlaznih blokova Q_i veličine 64 bita. Otvoreni tekst (*plaintext*) dijeli se na blokove P_i. Za kreiranje prvog bloka potrebno je definirati početno stanje registra R. Početni uvjet *IV* čini početno stanje registra R. *IV* ne mora biti tajan. Slika 2.9 prikazuje dijagram šifriranja CFB načinom.



3. Sl. 2.9: Dijagram šifriranja CFB načinom [11]

3.1.1. *Output Feedback Mode* (OFB)

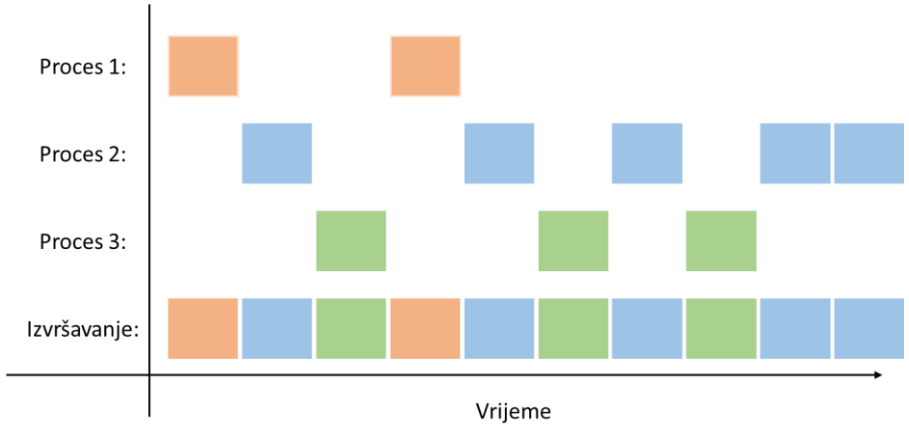
Slika 2.26 prikazuje šifriranje OFB načinom. Način je sličan prethodno opisanom CFB načinu, s razlikom u sadržaju izvornog teksta blokova osim prvog bloka. Ulazni tekst za svaku rundu jednak je rezultatu šifriranja runde prije primjene isključivo ILI operacije sa izvornim tekstom. Slika 2.10 prikazuje dijagram šifriranja OFB načinom.



Sl. 2.10: Dijagram OFB načina šifriranja [11]

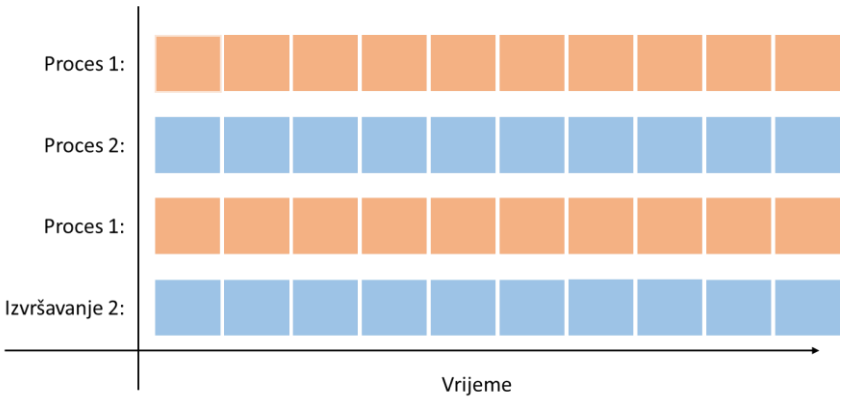
3.2. Paralelno izvođenje operacija

Kako bi omogućili bolje iskustvo korisnika prilikom upotrebe računala, javlja se potreba za paralelnim izvođenjem više procesa [12]. Međutim, svaki pojedini proces može imati dijelove koji se mogu podijeliti u manje jedinice te izvoditi paralelno. Hoće li se oni uistinu izvoditi paralelno ili prividno paralelno ovisi o arhitekturi opreme koja izvodi zadatke. Prividno paralelno izvođenje je postupak gdje jedan vršitelj, tj. procesor dijeli raspoloživo vrijeme na više procesa. Distribucija izvođenja procesa ovisno o parametrima procesa (prioritet, vrijeme izvođenja, vrijeme kreiranja…) neće biti detaljno objašnjena jer bi to izlazilo iz okvira rada. Prikazat će se slučajevi gdje svi procesi pristižu u istom trenutku, nisu podijeljeni po prioritetima te vrijeme izvođenja nije bitno. Slika 2.11 prikazuje ravnomjerno dijeljenje 3 procesa na jednom procesoru.



Sl. 2.11: Podjela vremena izvođenja

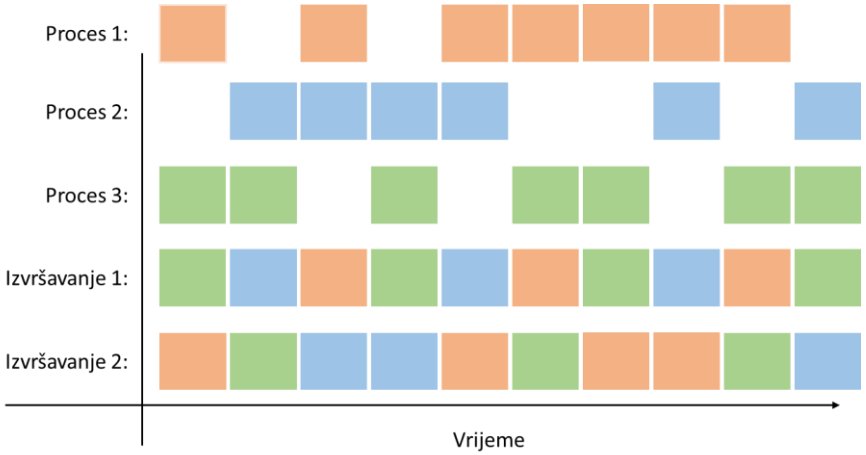
Stvarno paralelno izvođenje svaki zadatak izvodi neovisno o ostalim zadacima. Slika 2.12 prikazuje primjer sa dva zadatka.



Sl. 2.12: Podjela vremena izvođenja

Omogućiti paralelno izvođenje svakog zadatka neovisno o drugima zahtjevima traži poznavanje najvećeg broja zadataka u bilo kojem trenutku. Ovakav sustav je izrazito neprilagodljiv. S obzirom na to da pojedini procesi u raznim fazama svojeg odvijanja raznoliko troše pojedine dijelove računalnog sustava koji nisu u potpunosti paralelni (npr. čitanje iz memorije, komunikacija preko mreže) pojavljuju se slučajevi kada procesor čeka izvršavanje operacija zadanih od strane drugih procesa na komponentama.

Za optimalno izvršavanje svih zadataka koristi se kombinacija više procesa koji se raspodjeljuju na više procesora. Slika 2.13 prikazuje takav slučaj.



Sl. 2.13: Podjela vremena izvođenja više procesa na više procesora

Međutim, praktično je pojedine zadatke podijeliti na pod zadatke kako bi lakše svladala složenost izgradnje programskog sustava, a s druge strane omogućilo bolje iskorištavanje računalnih sredstava unutar jednog procesa. Uvode se mehanizmi koji omogućuju izvođenje procesa sa više niti. Niti dijele sva sredstva koja operacijski sustav stavlja na raspolaganje procesu. U razmatranju paralelnog izvođenja više niti treba pretpostaviti da je u sustavu osigurana pravilna promjena konteksta te da se niti mogu odvijati, prekidati i nastavljati.

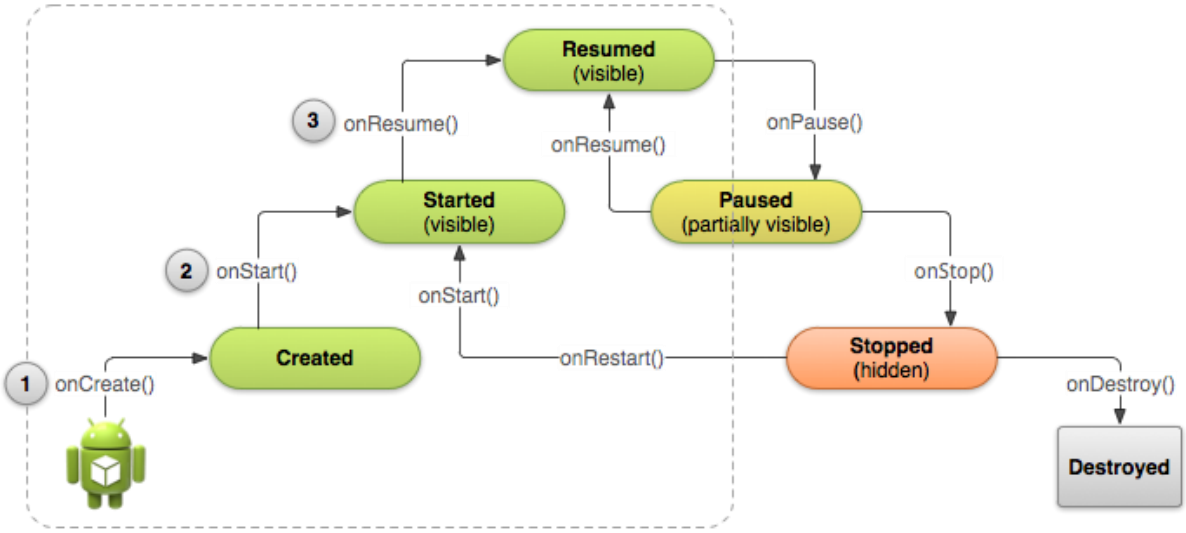
Većina programa može se podijeliti na 3 dijela: pod zadatak za učitavanje ulaznih podataka, pod zadatak za obradu i pod zadatak za ispis, tj. distribuciju rezultata. Navedeni zadaci se ne mogu izvoditi paralelno jer parametri za sljedeći zadatak ovise o rezultatima prethodnog zadatka. Međutim, pod zadaci se mogu dijeliti ovisno o strukturi. Primjer pod zadatka koji je pogodan za podjelu na više niti je simetrično šifriranje.

3.3. Android aplikacije

Android sustav definira nekoliko tipova komponenti koji čine osnovne dijelove svake Android aplikacije [3]. Svaka komponenta je drugačija ulazna točka u aplikaciju pomoću koje aplikacija komunicira s okolinom i ima posebnu funkciju unutar životnog ciklusa aplikacije. Postoje četiri tipa komponenti unutar aplikacije:

- Aktivnost (eng. *Activity*)
    - Aktivnost predstavlja jedan zaslon sa sučeljem. Na primjer, aplikacija može imati jedan zaslon za prijavu korisnika koji prikazuje polja za unos imena i lozinke. Po uspješnoj prijavi korisniku se prezentira sljedeći zaslon, npr. lista s porukama u slučaju aplikacije za dopisivanje. Aktivnost se implementira kao klasa koja nasljeđuje klasu *Activity*. Slika 2.14 prikazuje životni ciklus aktivnosti.
- Servis (eng. *Service*)
    - Servis je komponenta koja se odvija u pozadini i služi za izvođenje operacija koje nisu pogodne za izvršavanje na sučelju. Na primjer, servis može reproducirati glazbu dok korisnik upotrebljava druge aplikacije. Također, servis može dohvaćati podatke preko mreže koji će biti prezentirani korisniku na zahtjev, kao što je periodično provjeravanje sadržaja na društvenim mrežama. Servis se implementira kao klasa koja nasljeđuje klasu *Service*.
- Pružatelji sadržaja (eng. *Content Provider*)
    - Pružatelji usluga služe za dijeljenje informacija pod kontrolom aplikacije. Aplikacija može pohranjivati podatke u datoteke, u SQLite bazu podataka, web server i sl. Sustav ne dopušta pristup sadržaju unutar aplikacije iz drugih aplikacija. Na primjer, Android sustav ima pružatelja usluge koji upravlja podacima o korisniku. Ukoliko aplikacija ima odgovarajuća dopuštenja, ovaj pružatelj usluga će isporučiti podatke o korisniku. U tom slučaju aplikaciji će biti isporučen sadržaj u dokumentiranom formatu koji ne ovisi nužno o samoj implementaciji pohrane tih podataka. Pružatelji sadržaja se implementiraju kao klase koje nasljeđuju klasu *ContentProvider*.
- Primatelj emitiranih poruka (eng. *Broadcast receiver*)
    - Primatelj emitiranih poruka reagira na poruke emitirane od strane sustava ili drugih aplikacija. Na primjer, sustav može emitirati poruku o gašenju zaslona, isključivanje

pristupa mreži, paljenju *Bluetooth* protokola i sl. Primatelj emitiranih poruka implementira se kao klasa koja nasljeđuje *BroadcastReceiver* klasu.



Sl. 2.14: Životni ciklus Android aktivnosti [13]

4. OPIS APLIKACIJE

Rad proučava učinkovitost primjene postojećih implementacija simetričnih kriptografskih sustava na Android platformi. Zbog brzog zastarijevanja verzija platforme, u istraživanju je korištena najnovija verzija SDK na dan kada je započet razvoj. Aplikacija testira brzinu izvođenja procesa šifriranja. Implementacije protokola obavljene su u standardnim bibliotekama koje su dostupne na Android platformi (*javax.crypto*) [14].

Android platforma omogućuje razvoj aplikacija koje se izvode na više niti (engl. *threads*). Aplikacija razvijena u sklopu rada promatra vrijeme potrebno za šifriranje većeg broja blokova. Omogućena je distribucija blokova na više niti kako bi se utvrdilo do koje mjere Android podjela utječe na brzinu izvođenja. Promatrani su protokoli AES, DES i 3DES. Način rada koji se primjenjuje u testiranju je ECB. ECB i CTR načini omogućuju operacije na bloku neovisno o operacijama na ostalim blokovima te su pogodni za paralelno izvođenje. CFB i OFB nemaju navedeno svojstvo. Operacije na jednom bloku ovise od rezultatu operacija na prijašnjem bloku te kao takve nisu pogodne za paralelno izvođenje.

Operacijski sustav izvodi se na različitim uređajima čija svojstva mogu utjecati na brzinu izvođenja. Zbog toga su prikupljeni rezultati izvođenja na više uređaja navedenih u tablici 2.13. Navedeni uređaji predstavljeni su u razdoblju od najviše 4 godine prije objave rada. Karakteristike najranije predstavljenog uređaja možemo smatrati donjom granicom uređaja dostupnih na tržištu, uzevši u obzir najraniju verziju sustava čiji je ukupni udio veći od 90% tržišta. Iz tablice 2.14 vidimo da je najranija verzija s kojom udjel tržišta prelazi 90% *Jelly Bean*, najavljen 2012. godine [15].

Tab. 2.13: Popis testiranih uređaja

Uređaj	Broj jezgri	Takt jezgri	Količina radne memorije	Verzija sustava
LG Nexus 4	4	1.5 GHz	2 GB	5.1.1
Samsung Galaxy A5	8	1.6 GHz	2 GB	5.1.1
LG G Pro 7.0	4	1.2 GHz	1 GB	5.0.1

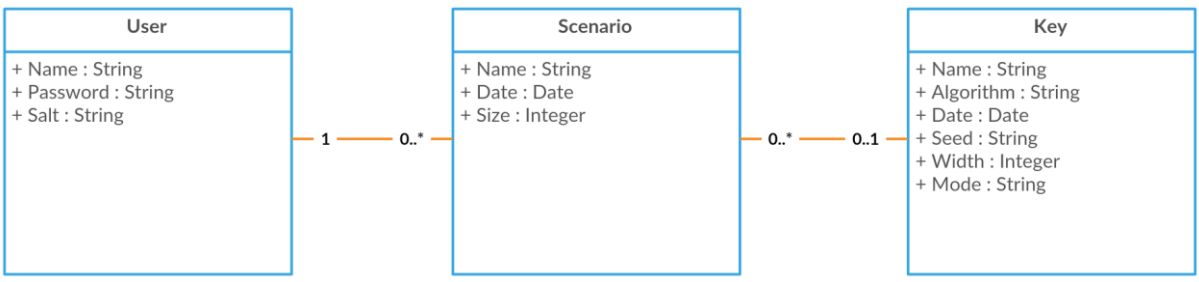
Tab. 2.14: Udio verzija Android sustava na tržištu [16]

Verzija	Kodno ime	Udio
2.2	Froyo	0.1%
2.3.3 2.3.7	Gingerbread	1.7%
4.0.3 4.0.4	Ice Cream Sandwich	1.6%
4.1.x	Jelly Bean	6.0%
4.2.x		8.3%
4.3		2.4%
4.4	KitKat	29.2%
5.0	Lollipop	14.1%
5.1		21.4%
6.0	Marshmallow	15.2%

Aplikacija omogućuje izvođenje testa ovisno o sljedećim parametrima:

- protokol
- veličina ključa
- veličina izvornog teksta
- broj niti

Postoje 3 entiteta unutar aplikacije: korisnik (*user*), ključ (*key*) i scenarij (*scenario*). Entiteti su pohranjeni unutar *SQLite* baze podataka. Ključ se sastoji od imena ključa, imena protokola i veličine ključa te pripada jednom ili više scenarija i jednom korisniku. Scenarij se sastoji od imena scenarija i veličine izvornog teksta, sadrži jedan ključ te pripada jednom korisniku. Korisnik se sastoji od imena, lozinke te više scenarija i ključeva. Slika 2.15 prikazuje UML dijagram relacija između entiteta:



Sl. 2.15: UML dijagram baze podataka

Uvođenjem entiteta korisnika omogućeno je grupiranje scenarija i ključeva u djelomično odvojene cjeline. Iako korisnici ne mogu vidjeli ključeve i scenarije drugih korisnika, imena ključeva i scenarija moraju biti unikatna na razini aplikacije. Prilikom kreiranja novog ključa ili scenarija s već postojećim imenom aplikacija operacija neće biti uspješna, ali korisnik

aplikacije neće biti obavješten o neuspjehu. Dodavanje validacija i obavijesti za ovakve rubne slučajeve produžuje razvoj, ali ne pospješuje značajno kvalitetu konačnog rezultata.

Aplikacija je razvijena u razvojnom okruženju Android Studio 2.1. Verzija SDK (*eng*. *Software Development Kit*) je 21 (Android 5.1 *Lollipop*). Tijekom razvoja operacijski sustav računala se mijenjao te je aplikacija razvijana na Windows 10, Ubuntu 14.04 i OSX 11.4 operacijskim sustavima.

4.1. Zahtjevi i mogućnosti aplikacije

Zahtjevi za uređaj na kojemu korisnik želi pokrenuti aplikaciju su:

- uređaj mora imati verziju 21 SDK-a, što odgovara kodnom imenu *Lollipop*; uređaji s nižom verzijom nisu podržani
- podržavanje SD kartice nije potrebno

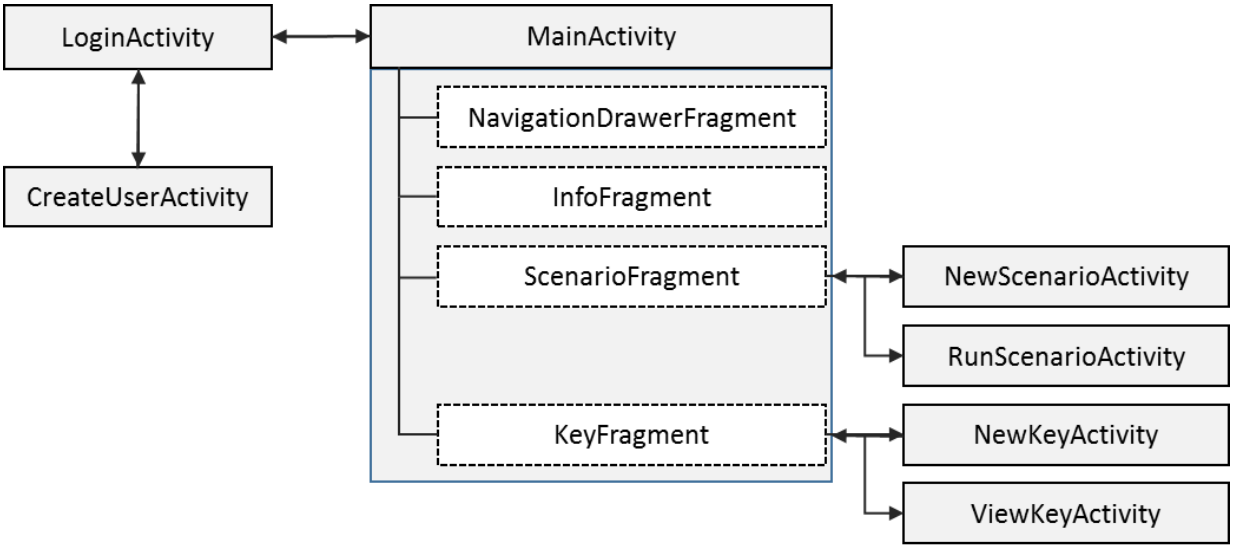
Mogućnosti aplikacije:

- kreiranje test slučajeva koji obuhvaćaju ime protokola, veličinu ključa, ukupnu veličinu izvornog teksta
- kreiranje korisnika aplikacije. Dodani slučajevi dostupni su samo korisniku koji ih je napravio
- izvođenje test slučajeva s mogućnosti odabira broja niti unutar kojih će se rasporediti proračun. Podržava odabir jednog od ponuđenih:
    - 1
    - 2
    - 4
    - 8
    - 16
- rezultat testa dodaje se u trajnu pohranu u memoriju uređaja

Oblik pohrane je tekst datoteka u CSV formatu. Spajanjem na računalo datoteka se može preuzeti i dalje obrađivati u programima koji podržavaju CSV format.

4.2.Struktura aplikacije

Aplikacija *CryptoBenchmark* sastoji se od 7 aktivnosti, prikazanih na slici 3.1. Nije bilo potrebe za implementacijom primatelja emitiranih poruka, pružatelja usluga i servisa. Uz aktivnosti razvijeno je 4 fragmenta za potrebe zaslona sa listama (*ScenarioListFragment*, *KeyListFragment*, *InfoFragment*, *NavigationDrawerFragment*), popratni adapteri za upravljanje sadržajem lista (*ScenarioCursorAdapter*, *KeyCursorAdapter*, *DrawerArrayAdapter*), servisi za upravljanje sadržajem baze podataka (*SQLiteHelper*, *DatabaseResponse*, *DataSource*), modela (*AsyncTaskModel*, *Colors*, *Key*, *NavigationItemModel*, *Scenario*, *User*) i *servisa za izvršavanje* (*CipherTask*, *BenchmarkAsyncTask*).



Sl. 3.1: Prikaz aktivnosti unutar aplikacije i veza između

Upravljanje fragmentima i listama pripada dijelu za upravljanje sučeljem i kao takav nije bitan za osnovnu funkciju aplikacije, stoga neće biti detaljno obrađen.

Servisi za upravljanje bazom podataka služe za olakšavanje pristupa bazi podataka. Sastoje se od *SQLiteHelper* klase koja kreira bazu podataka po specifikaciji prilikom pokretanja aplikacije ukoliko baza ne postoji i od klase *DataSource* koja sadrži sve metode potrebne za dohvaćanje podataka iz baze.

Za razvoj aplikacije potrebno je bilo dodati modele za svaku tablicu iz baze podataka i nekoliko pomoćnih modela.

Testiranje se vrši pomoću instance *TestAsyncTask* klase. Instanca kreira skup niti (eng. *thread pool*), gdje se svakoj niti prosljeđuje instanca klase *CipherTask*. Upravljanje instancom klase

CipherTask obavlja se pomoću sučelja *Callable* koje klasa implementira. Po završetku svih niti rezultat mjerenja prosljeđuje se na glavnu nit gdje se zapisuje u datoteku. Po potrebi kreira se novi *thread pool* i opisani postupak se ponavlja.

Kod koji pokreće testove prikazan je na slici 3.2.

```
threadCountArray = new int[]{1, 2, 4, 8, 16};
total_runs = 2;
for (int i = 0; i < total_runs; i++) {
  for(int     model_index=0;     model_index<     asyncTaskModels.length;
model_index++) {
    for (int thread_count_index = 0;
         thread_count_index < threadCountArray.length;
         thread_count_index++) {
      threadCount = threadCountArray[thread_count_index];
      byte[][] value = new byte[threadCount][];
      try {
        Scenario = asyncTaskModels[model_index].scenario;
        Key = asyncTaskModels[model_index].key;
        ExecutorService threadPool =
            Executors.newFixedThreadPool(threadCount);
        CompletionService<byte[]> pool =
            new ExecutorCompletionService<>(threadPool);
        List<Future<byte[]>> futures = new ArrayList<>(threadCount);
        byte[] seed = key.seed.getBytes(Charset.forName("UTF-8"));
        byte[] seedExpanded = new byte[key.width / 8];
        byte[] initVector;
        if (key.algorithm.equals("AES")) {
          initVector = new byte[(128 / 8)];
        } else {
          initVector = new byte[(64 / 8)];
        }
        new Random().nextBytes(initVector);
        SecretKeySpec skeySpec;
        IvParameterSpec iv;

        for (int j = 0; j < (key.width / 8); j++) {
          seedExpanded[j] = seed[(j % key.seed.length())];
        }

        int init_size = this.getBlockSize(scenario.size);
        for (int j = 0; j < threadCount; j++) {
          value[j] = new byte[init_size];
          new Random().nextBytes(value[j]);
        }
        long startTime = System.nanoTime();
        for (int j = 0; j < threadCount; j++) {
          skeySpec = new SecretKeySpec(seedExpanded, key.algorithm);
          iv = new IvParameterSpec(initVector);
          Log.d("THREAD", "Size: " + value[j].length);
          futures.add(
           pool.submit(new CipherTask(key, value[j], skeySpec, iv, j)));
```

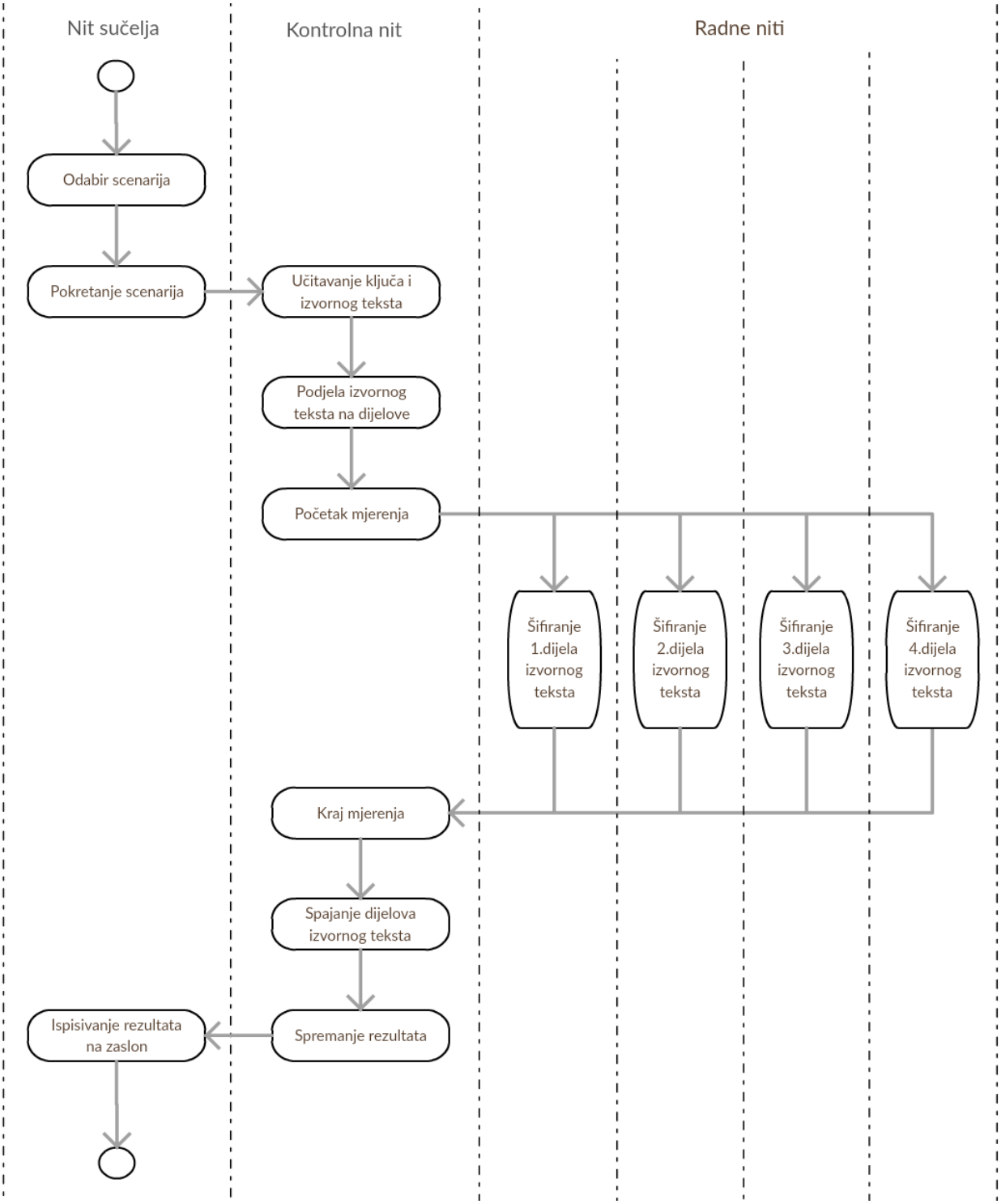
```
        }

        for (Future<byte[]> future : futures) {
           byte[] result = future.get();
        }

        threadPool.shutdown();
        long endTime = System.nanoTime();

        duration = (endTime - startTime) / 1000000; // milliseconds
        deliverResult.addResultWithKeyAndScenario(
          duration, threadCount, key, scenario);
        Log.d("Out", "" + duration + " ms");
      } catch (Exception e) {
          e.printStackTrace();
     }
    }
  }
}
```

Sl. 3.2: Izvorni kod za pokretanje testova



Sl. 3.3: Tijek izvođenja scenarija na 4 niti

Instanca klase *CipherTask* pokreće i izvršava šifriranje. *Thread pool* upravlja instancom pomoću sučelja *Callable*. Klasa sadrži konstruktor i implementaciju metode koje zahtjeva sučelje. Slika 3.4 prikazuje izvorni kod klase.

```
public class CipherTask implements Callable<byte[]> {
    private Cipher;
    private byte[] value;

    public CipherTask(Key key, byte[] value, SecretKeySpec secretKeySpec) {
        this.value = value;
        try {
            cipher = Cipher.getInstance(key.algorithm + "/"+ key.mode
+"/PKCS5PADDING");
            cipher.init(Cipher.ENCRYPT_MODE, secretKeySpec);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public byte[] call(){
        byte[] ret = new byte[1];
        try {
            ret =  cipher.doFinal(value);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return ret;
    }
}
```

Sl. 3.4: Izvorni kod klase CipherTask

4.3. Spremanje rezultata

Rezultati testiranja pohranjuju se u datoteku na uređaju. Prilikom razvoja kao testni uređaj koristio se LG Nexus 4, koji nema na raspolaganju mogućnost proširivanja memorije pomoću memorijskih kartica. Kako bi pojednostavili proces preuzimanja rezultata na osobno računalo, upotrjebljen je standardno definirani direktorij za pohranu datoteka specifičan za aplikaciju. Time je omogućeno preuzimanje rezultata na uniforman način za sve uređaje. Nakon spajanja uređaja na osobno računalo s instaliranim Android SDK-om, rezultati se prikazuju pomoću naredbe *"adb shell 'run-as com.zoran.cryptobenchmark cat files/rezultati.txt'"* unutar komandnog prozora. Aplikacija *adb* dio je Android SDK.

Za pohranu rezultata bilo je potrebno odabrati takav format koji se može obraditi pomoću programa za tablične proračune. Korišten je Microsoft Excel 2013. Također, zapis unutar aplikacije i prijenos na osobno računalo bi trebalo biti moguće izvesti što jednostavnije.

Iz navedenih razloga odlučeno je primijeniti *csv* format [17]. Podržan je od strane Excela [18] i moguće ga je kreirati zapisivanjem teksta u datoteku. Poredak vrijednosti za test slučaj prikazan je slikom 3.5 i dan je primjer nekoliko rezultata testova slikom 3.6.

{***ime korisnika***}{***algoritam***}{***širina ključa***}{***način rada***}{***veličina datoteke u MB***}{***broj niti***}{***rezultat u ms***}

Sl. 3.5: Poredak vrijednosti test slučajeva

```
pro7,AES,128,ECB,1,8,96
pro7,AES,128,ECB,40,1,3402
nexus4,AES,256,ECB,2,1,239
nexus4,DES,64,ECB,2,8,287
```

Sl. 3.6: Primjer rezultata testova

Kako bi pojednostavili diferenciranje rezultata pojedinih uređaja i omogućili agregiranje rezultata svih uređaja bez prethodne obrade, potrebno je uz svaki rezultat pohraniti identifikator uređaja. Identifikator može biti generiran od strane uređaja ili od strane korisnika. Upotreba generiranih identifikatora smanjuje broj koraka prilikom testiranja uređaja. Međutim, identifikatori uređaja mogu se razlikovati od marketinškog imena što otežava razlikovanje uređaja. Stoga je odlučeno omogućiti imenovanje rezultata od strane korisnika.

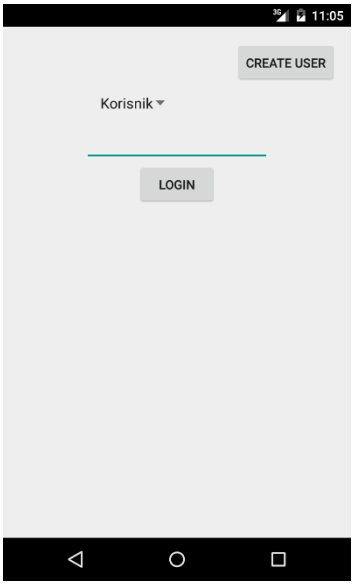
Implementiran je sustav registracije te ime korisnika služi za identifikaciju rezultata. Korisnik se registrira pomoću imena i zaporke. Pri implementaciji primijenjen je *salted hash* [19] za pohranu lozinke [20]. Za trenutne potrebe aplikacije ovo je previše složen pristup. Programiranje aplikacije započeto je s premisom izrade aplikacije za šifriranje datoteka te bi opisani pristup bio nužan za sigurnost podataka. Kasnije je izvorni kod aplikacije poslužio kao temelj aplikacije za testiranje, a implementirane mogućnosti nisu predstavljale zapreku za danji razvoj te nije bilo razloga za uklanjanje mogućnosti.

Izazov pri pohrani korisnika s autorizacijom je kako povećati zaštitu podataka korisnika u slučaju neovlaštenog pristupa bazi podataka. Pohrana zaporke unutar baze podataka nije opcija

jer su u slučaju neovlaštenog pristupa zaporke dostupne napadaču. Lozinke je potrebno pohraniti u sustav na takav način da provjera lozinke bude jednostavna, ali izvlačenje lozinke iz sustava dovoljno složeno. Primjena *hash* funkcija sa dodatkom nasumičnog teksta, tzv. *Salta* to omogućuje.

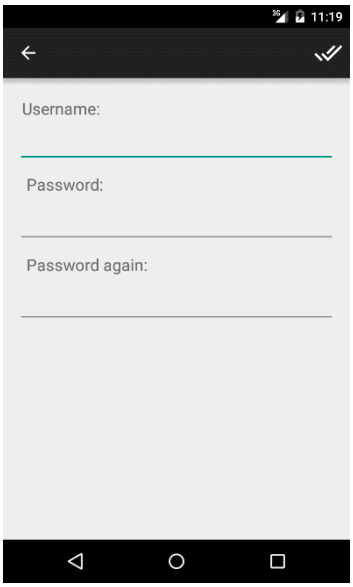
4.4.Sučelje aplikacije

Pokretanjem aplikacije korisniku je prikazan zaslon za prijavu korisnika, prikazan na slici 3.7



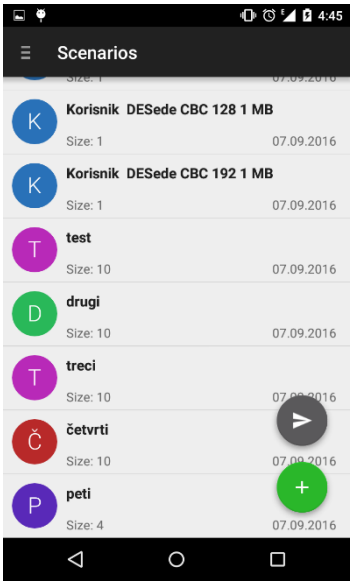
Sl. 3.7: Zaslon za prijavu korisnika

Zaslon se sastoji od padajućeg izbornika koji sadrži popis registriranih korisnika, polja za upis zaporke, tipke za prijavu (*Login*) i tipke za kreiranje novog korisnika (*Create User*). Polje za upis zaporke ima jednaku boju pozadine kao prazni prostor i njeno mjesto je naznačeno zelenim rubom s donje strane. Klikom na *Create User* otvara se zaslon za kreiranje novog korisnika prikazan na slici 3.8. Klikom na Login otvara se glavni zaslon programa prikazan slikom 3.9.



Sl. 3.8: Zaslon za registraciju korisnika

Zaslon za registraciju korisnika sastoji se od polja za upis imena, polja za upis zaporke te polja za potvrdu zaporke. Klikom na strjelicu u gornjem desnom uglu kreira se novi korisnik. Ukoliko unesene lozinke nisu identične, program će javiti grešku u obliku balona s tekstom u donjoj polovici zaslona. Također, ukoliko korisnik već postoji, program će javiti grešku na već opisan način.



Sl. 3.9: Glavni zaslon aplikacije

Dolaskom na glavni zaslon korisniku je prikazana lista kreiranih scenarija. Na slici 3.9 vidimo primjer liste sa tri kreirana scenarija. Elementi liste sastoje se od imena scenarija s podebljanim slovima, brojem ispod imena koji sadrži veličinu izvornog teksta predviđenog za taj scenarij, datumom stvaranja scenarija te krugom na lijevoj strani. Slovo unutar kruga je prvo slovo imena, dok je boja kruga izvedena iz prvog slova imena.

U donjem desnom uglu nalazi se tipka za kreiranje novog scenarija, u obliku zelenog kruga sa znakom plus.

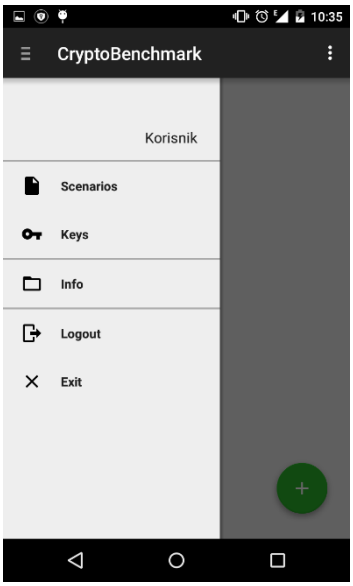
Za određivanje boje koristi se HSV model boja. H komponenta boje je broj iz intervala od 0 do 360. Ukoliko je prvi znak broj, H komponenta jednaka je umnošku broja i broja 10. Ukoliko je prvi znak slovo, on se pretvara u malo slovo i H komponenta jednaka je rednom broju slova u engleskoj abecedi uvećanog za 9 i umnoška broja 10. Npr. ako je prvo slovo je 'C', koje je treće slovo abecede, H komponenta poprima vrijednost:

$$\boldsymbol{H = (3 + 9) * 10 = 120}$$

Množenje s brojem deset je nužno kako bi se izvedene vrijednosti za sva slova i brojeve ravnomjerno rasprostirale unutar dopuštenog intervala H vrijednosti. Interval je od 0 do 360, a moguće vrijednosti boja su od 0 za '0' do 350 za 'z', razmaknute za 10.

Iznad tipke za kreiranje korisnika nalazi se tipka za izvršavanje svih kreiranih scenarija. Svi kreirani scenariji će se izvršiti 3 puta sa svim mogućnostima za broj niti te će rezultati biti zapisani u datoteku *rezultati.txt* na uređaju.

Kikom na ikonu s tri horizontalne linije u gornjem lijevom uglu otvara se izbornik animacijom otvaranja ladice s lijeve strane. Izbornik sadrži tipke za upravljanje aplikacijom. Slika 3.10 prikazuje izbornik.

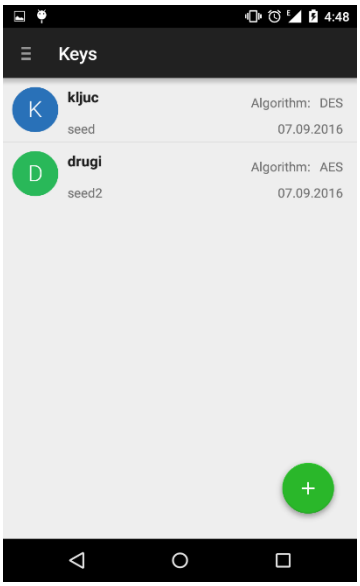


Sl. 3.10: Izbornik za upravljanje aplikacijom

Izbornik sadrži sljedeće elemente, od gore prema dolje redom:

- Ime korisnika
- *Scenarios* - Stranica s listom scenarija
- *Keys* - Stranica s listom ključeva
- *Info* - Stranica s infomacijama o trenutnok korisniku
- *Logout* - Tipka za povratak na zaslon za prijavu
- *Exit* – Tipka za izlaz iz aplikacije

Pritiskom na zatamnjeni prostor s desne strane izbornik se zatvara i korisnik se vraća na prijašnju stranicu. Stranica s listom ključeva prikazana je na slici 3.11.



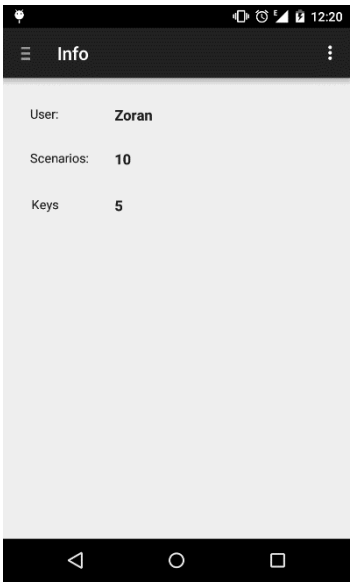
Sl. 3.11: Stranica s ključevima

Stranica s ključevima ima strukturu sličnu stranici s scenarijima. Sadrži listu ključeva te tipku za dodavanje novog ključa. Elementi liste sadrže:

- ime ključa u gornjem desnom uglu
- tekst za inicijalizaciju ključa u desnom donjem uglu
- ime algoritma u gornjem desnom uglu
- datum kreiranja ključa
- krug desno od imena

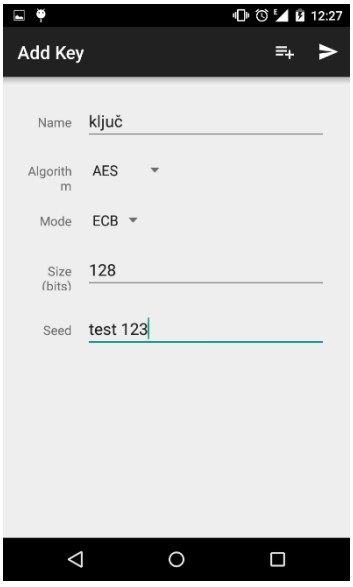
Boja i slovo unutar kruga određuju se na isti način kao u listi scenarija.

Odabirom opcije *Info* u izborniku otvara se stranica s informacijama, prikazana na slici 3.12. Sadrži ime korisnika, broj scenarija te broj ključeva koje je korisnik kreirao.



Sl. 3.12: Stranica s informacijama

Klikom na tipku za kreiranje ključa otvara se stranica za dodavanje ključa, prikazana na slici 3.13. Korisnik ima mogućnost definirati ime ključa, algoritam, način rada algoritma, veličinu ključa i tekst za inicijalizaciju ključa. Trenutno su podržani algoritmi AES, DES i 3DES.



Sl. 3.13: Stranica za dodavanje ključa

Načini rada koji se mogu odabrati su ECB i CTR. Pri upisu veličine ključa u polje *Size* potrebno je obratiti pažnju na podržanu veličinu ključa za odabrani algoritam. Primjerice, AES podržava

veličine ključa od 128, 192 i 256 bita. Upis bilo koje druge vrijednosti rezultirati će uspješnom kreacijom ključa, ali pokretanje scenarija kojem je dodijeljen taj ključ biti će neuspješno.

Pri određivanju veličine ključa za DES i 3DES protokole, potrebno je uključiti i paritetne bitove jer upotrjebljene implementacije DES i 3DES koriste niz bajtova za pohranu ključa, gdje je 8.bit svakog bajta paritetni. U slučaju DES protokola to bi iznosilo 56 + 8 = 64.

Klikom na tipku s dvije kvačice u gornjem desnom uglu kreira se ključ. Klikom na tipku s tri horizontalne linije i znakom plus kreira se predefinirani set ključeva. Set ključeva sadrži sve podržane kombinacije algoritma, moda i veličine ključa, prikazane tablicom 3.1.

Tab. 3.1: Primjer dodanih ključeva za korisnika pod imenom „Korisnik“

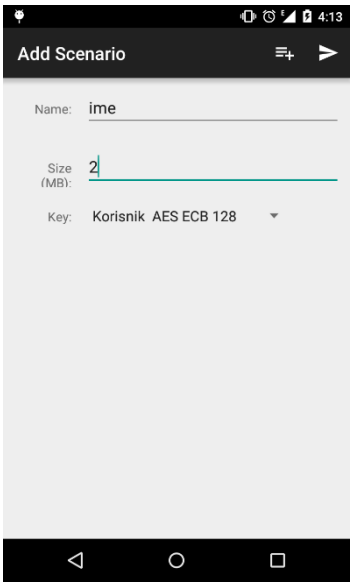
Algoritam	Način	Veličina ključa	Ime ključa
AES	ECB	128	Korisnik AES ECB 128
AES	ECB	256	Korisnik AES ECB 256
AES	CTR	128	Korisnik AES CTR 128
AES	CTR	256	Korisnik AES CTR 256
DES	ECB	64	Korisnik DES ECB 64
DES	CTR	64	Korisnik DES CTR 64
3DES (DESede)	ECB	128	Korisnik DESede ECB 128
3DES (DESede)	CTR	128	Korisnik DESede CTR 128

Ime ključa kreira se u formatu prikazanom slikom 3.14.

$$\{\boldsymbol{ime\ korisnika}\}\{\boldsymbol{algoritam}\}\{\boldsymbol{način}\}\{\boldsymbol{veličina\ ključa}\}$$

Sl. 3.14: Format automatskog imena ključa

Klikom na gumb za dodavanje scenarija otvara se stranica za dodavanje scenarija, prikazana na slici 3.15.



Sl. 3.15: Zaslon za kreiranje scenarija

Zaslon sadrži polja za upis imena scenarija, veličine izvornog teksta te ključa kojim se šifrira sadržaj. Klikom na simbol strjelice u gornjem desnom uglu scenarij se sprema, aktivnost se zatvara i korisnik se vraća na aktivnost s popisom scenarija.

4. REZULTATI

Promatrani protokoli su AES, DES i 3DES u ECB i CTR načinu rada. Izvorni tekst je 1, 5, 40 ili 50 MB veličine. Parametri ključa i sadržaj izvornog teksta nisu bitni za mjerenja te su nasumično generirani. Izvedeno je 3 ponavljanja svakog testa. U grafovima i proračunima koristi se srednja vrijednost ako nije drugačije naznačeno. Svi testovi izvršeni su na 3 stvarna uređaja.

Promatrani parametri su:

1. ovisnost vremena izvođenja ovisno o broju niti
2. ovisnost vremena izvođenja o veličini izvornog teksta
3. ovisnost vremena izvođenja ovisno o karakteristikama uređaja

Uređaj je prilikom testiranja spojen sa osobnim računalo. Napredak testiranja praćen je zapisnikom koji pruža Android Studio (*Logcat*). Aplikacija je kreirana u *debug* načinu rada. Rezultati testiranja dostupni su u prilogu *rezultati.csv*. Također je priložen Excel dokument sa uključenim rezultatima i proračunima za sve prikazane slike u poglavlju 5.

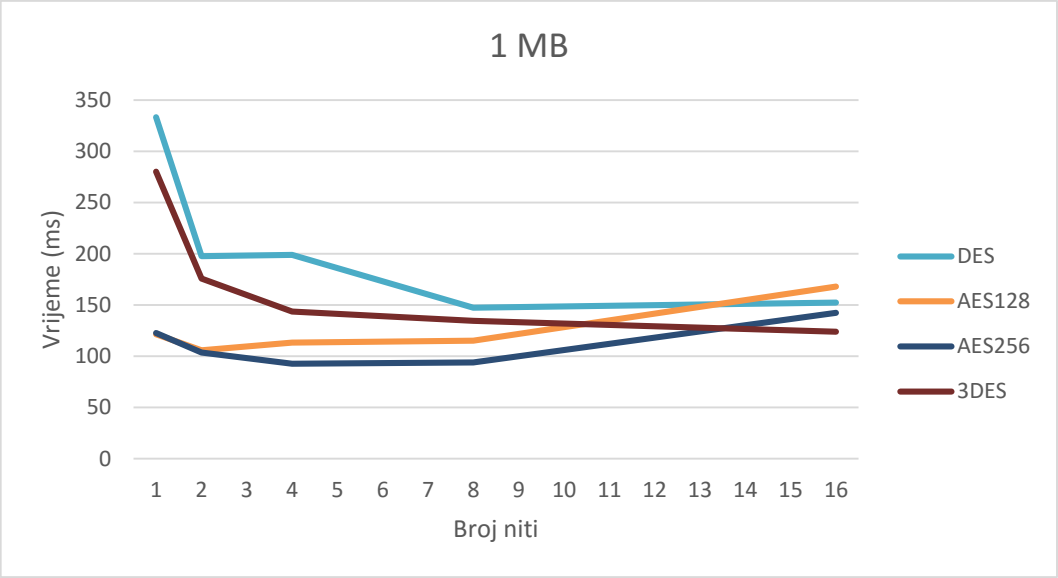
4.1. Ovisnost vremena izvođenja o broju niti

Slike 4.2, 4.3, 4.4 prikazuju vrijeme šifriranja ovisno o broju niti za veličinu izvornog teksta od 1, 5 ili 50 MB. Izvorni tekst ravnomjerno je raspoređen između niti, gdje je najveće odstupanje jedan blok podataka. Ukoliko se usporedi prosječno vrijeme izvršavanja svake niti pri veličini izvornog teksta od 50 MB, dobiju se rezultati prikazani tablicom 4.1. te slikama 4.1, 4.2 i 4.3.

Kod sustava gdje se niti izvršavaju paralelno očekivana brzina šifriranja po niti bila bi blizu konstante s povećanjem broja niti. Međutim, kod sustava gdje se niti izvršavaju prividno paralelno očekivana brzina šifriranja svake niti smanjivala bi se s povećanjem broja niti.

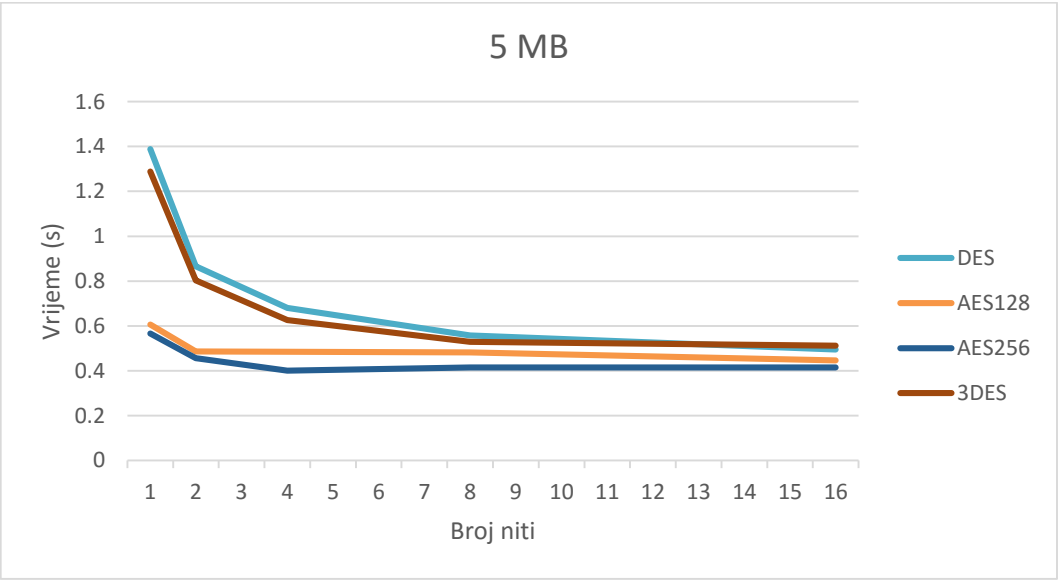
Tab. 4.1: Prosječno izvršavanje šifriranja

Protokol	Vrijeme izvršavanja (ms)	Broj niti	Prosječno vrijeme po niti (ms)	Prosječna brzina niti (Mbit/s)
AES128	606	1	606	8.65
	486.5	2	243.25	5.39
	484.5	4	121.125	2.71
	481.25	8	60.15625	1.36
	446	16	27.875	0.73
AES256	566	1	566	9.26
	456.5	2	228.25	5.74
	400.25	4	100.0625	3.27
	415.5	8	51.9375	1.58
	414.5	16	25.90625	0.79
DES	1388.25	1	1388.25	3.78
	866.5	2	433.25	3.03
	680.5	4	170.125	1.93
	557.5	8	69.6875	1.18
	495.25	16	30.95313	0.66
3DES	1287.75	1	1287.75	4.07
	803.75	2	401.875	3.26
	626.75	4	156.6875	2.09
	529.25	8	66.15625	1.24
	512.25	16	32.01563	0.64



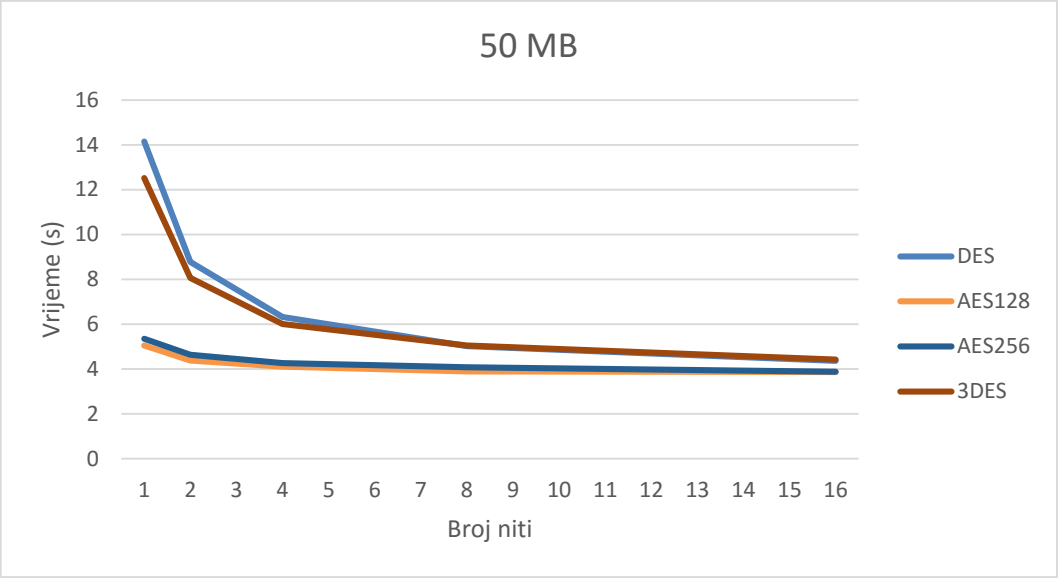
Sl. 4.1: Vrijeme šifriranje ovisno o broju niti pri veličini izvornog teksta od 1 MB

Na slici 4.1 možemo vidjeti da se promjena vremena izvođenja značajno razlikuje ovisno o protokolu.



Sl. 4.2: Vrijeme šifriranje ovisno o broju niti pri veličini izvornog teksta od 5 MB

Povećanjem veličine izvornog teksta vidi se da protokoli koji su slični imaju slične promjene vremena ovisno o broju niti, prema slici 4.2. Povećanjem broja niti iznad 8 razlike su manje uočljive.

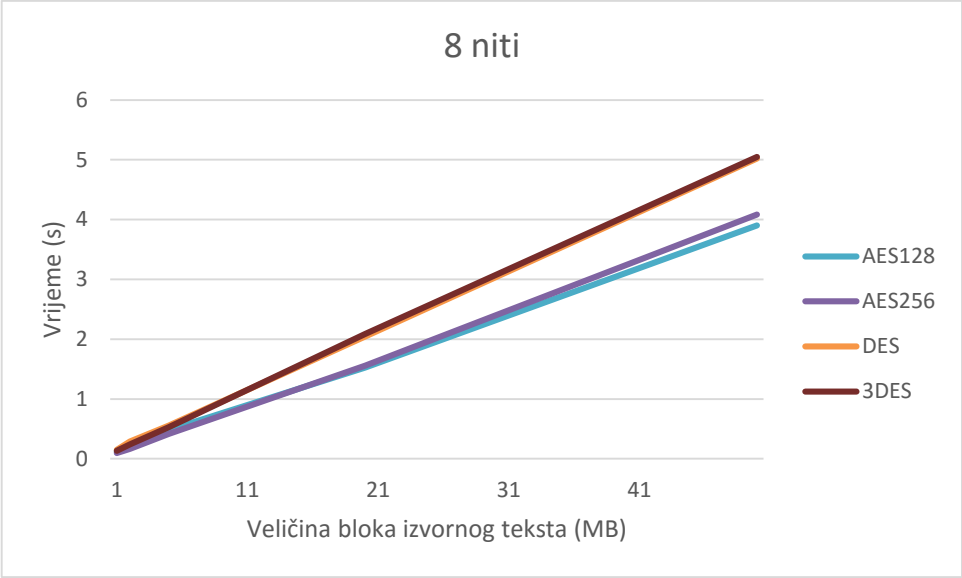


Sl. 4.3: Vrijeme šifriranje ovisno o broju niti pri veličini izvornog teksta od 50 MB

Povećanjem veličine izvornog teksta sa 5 na 50 MB sličnost između protokola DES i 3DES, odnosno AES u 128 i 256 bitnoj inačici je još izraženija.

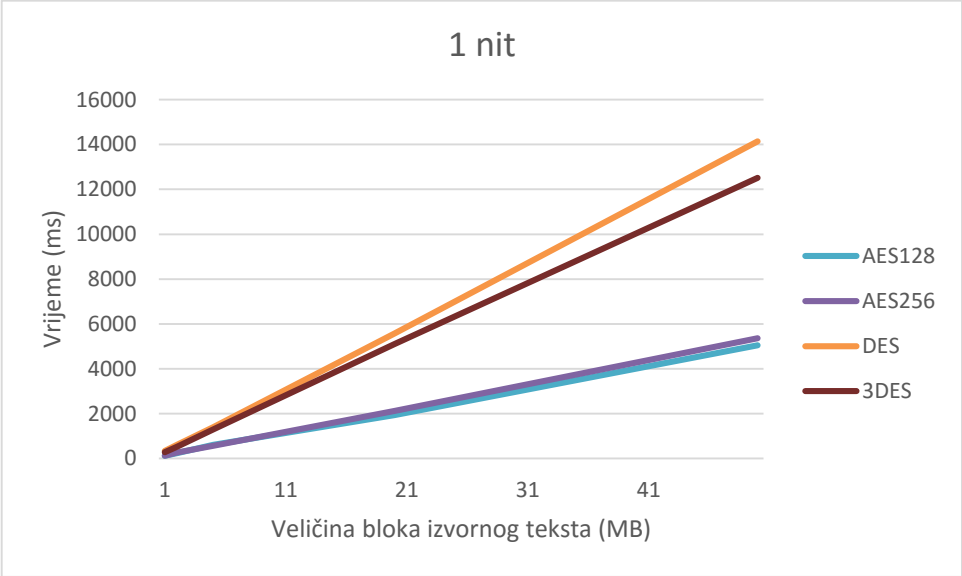
4.2. Ovisnosti vremena izvođenja o veličini izvornog teksta

Slike 4.4 i 4.5 i tablice prikazuju vrijeme šifriranja ovisno o veličini izvornog teksta bez podjele i s podjelom na 8 niti. Iz grafa se može pretpostaviti da je je vrijeme izvršavanja linearno povezano s veličinom izvornog teksta. To je u skladu s očekivanjem jer je kompleksnost šifriranja jednog bloka kod simetričnih kriptografskih protokola konstanta, dok je veličina izvornog teksta rasla.



Sl. 4.4: Vrijeme šifriranja ovisno o veličini izvornog teksta pri podjeli na 8 niti

Slika 4.4 također ističe sličnost između protokola kao na slici 4.3

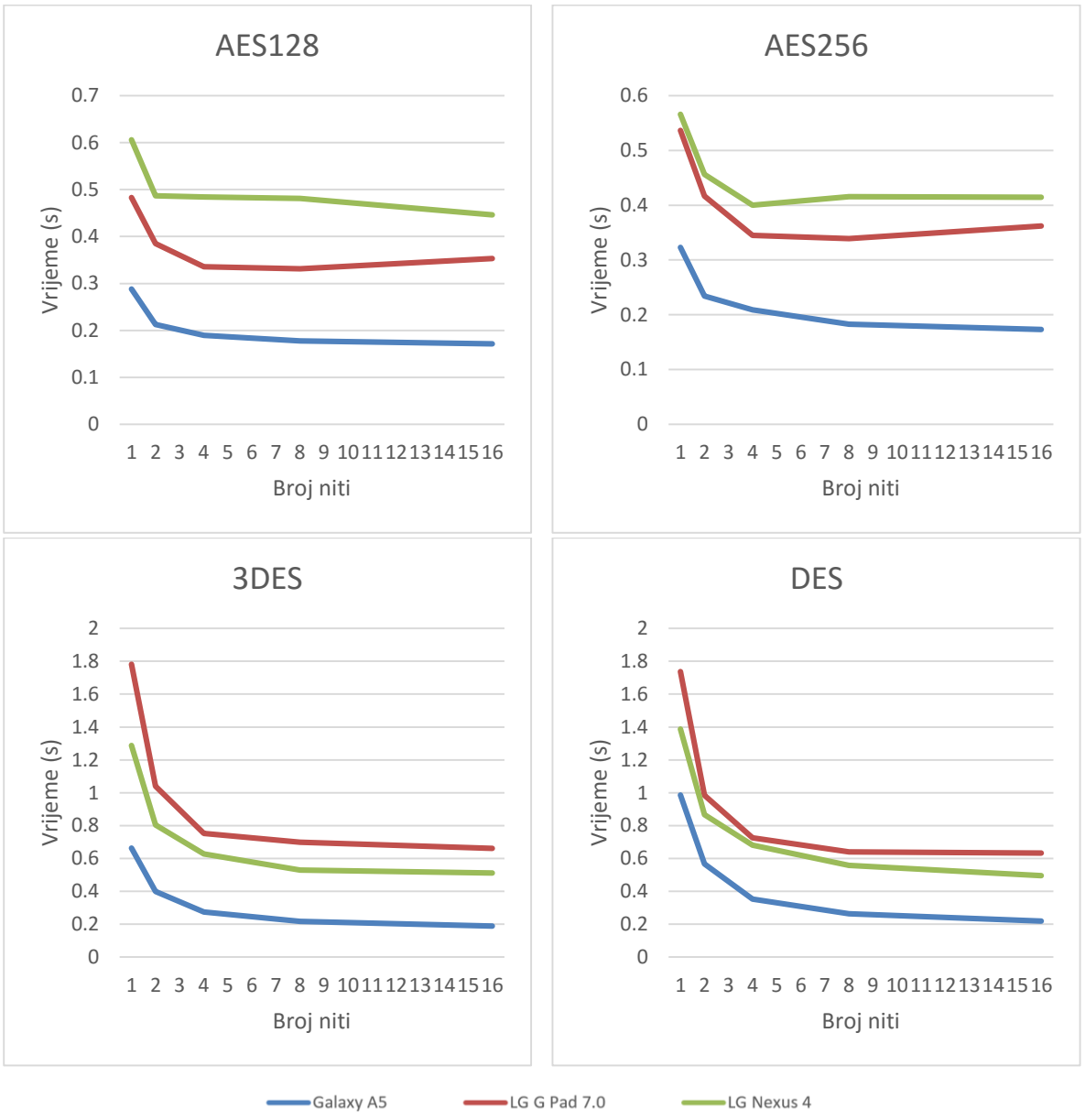


Sl. 4.5: Vrijeme šifriranja ovisno o veličini izvornog teksta bez podjele

Bez podjele na više niti lakše se uočava razlika između DES i 3DES, prikazano na slici 4.5. Međutim, vrijeme izvođenja 3DES protokola manja je od vremena izvođenja DES protokola. S obzirom na činjenicu da je 3DES u osnovi DES primijenjen tri puta, razlika u rezultatima može se objasniti razlikama u implementaciji.

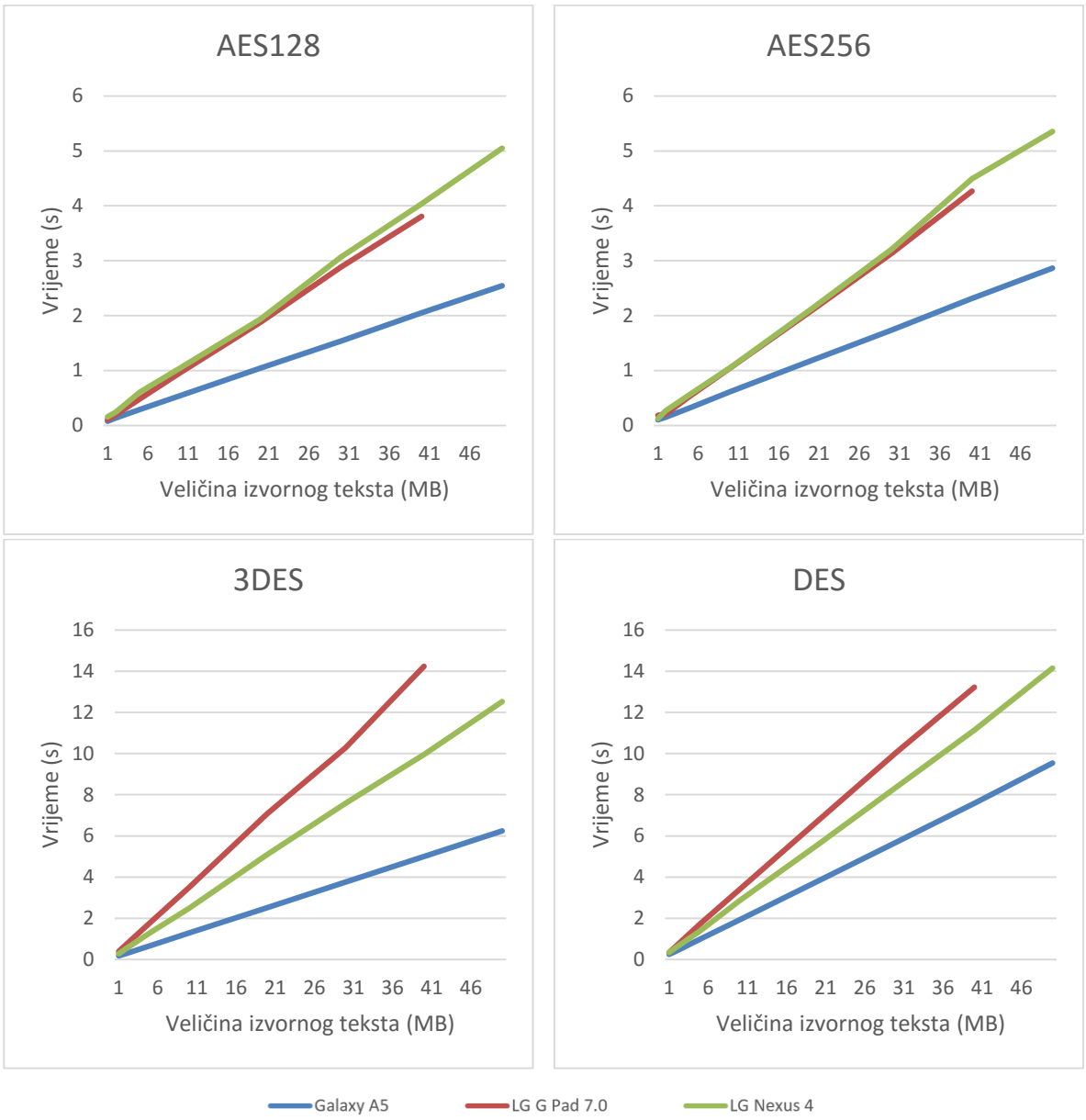
4.3. Ovisnost vremena izvođenja o uređaju

Usporedba više uređaja prikazana je slikom 4.6. Prikazana je usporedba vremena šifriranja između uređaja s obzirom na broj niti, uz veličinu izvornog teksta od pet MB.



Sl. 4.6: Usporedba vremena izvođenja na 3 uređaja ovisno o protokolu i broju niti

Usporedbom LG G Pad 7.0 i LG Nexus 4 uređaja ovisno o protokolu vidimo da LG Nexus 4 brži pri promjeni protokola AES, dok je LG G Pad 7.0 brži pri primjeni DES i 3DES protokola.



Sl. 4.7: Usporedba vremena izvođenja na 3 uređaja ovisno o protokolu i veličini izvornog teksta

Vrijeme izvođenja raste linearno s povećanjem veličine izvornog teksta. Odstupanje od linearnog rasta događa se kod AES 256 protokola pri povećanju veličine izvornog teksta na 50 MB. Uređaj LG G Pad 7.0 nije uspio uspješno izvršiti testiranje pri veličini izvornog teksta od 50 MB, dok LG Nexus 4 ima rezultat niži od očekivanog.

Tab. 4.2:Rezultati mjerenja bez podjele na niti

Protokol	Galaxy A5 (ms)	Nexus 4(ms)	LG G Pad(ms)
AES128	288	606	482
AES256	323	566	536
DES	986	1388	1737
3DES	662	1287	1781

Tablica 4.2 prikazuje vrijeme bez podjele na niti pri šifriranju izvornog teksta veličine 5 MB.

Tab. 4.3: Razlika u vremenu u odnosu na najbrži uređaj bez podjele na niti

	Galaxy A5	Nexus 4	LG G Pad
AES128	1	2.11	1.68
AES256	1	1.75	1.65
DES	1	1.41	1.76
3DES	1	1.94	2.69

Tablica 4.3 prikazuje omjer vremena potrebnom uređaju i vremena potrebnom najbržem uređaju.

Tab. 4.4:Rezultati mjerenja pri podjeli na 8 niti

Protokol	Galaxy A5 (ms)	Nexus 4 (ms)	LG G Pad (ms)
AES128	177	481	331
AES256	182	415	339
DES	264	557	639
3DES	216	529	699

Tablica 4.4 prikazuje vrijeme pri podjeli na 8 niti pri šifriranju izvornog teksta veličine 5 MB.

Tab. 4.5: Razlika u vremenu u odnosu na najbrži uređaj pri podjeli na 8 niti

Protokol	Galaxy A5	Nexus 4	LG G Pad
AES128	1	2.72	1.87
AES256	1	2.28	1.86
DES	1	2.11	2.42
3DES	1	2.45	3.2

Tablica 4.5 prikazuje omjer vremena potrebnom uređaju i vremena potrebnom najbržem uređaju.

5. RASPRAVA

Zanimljivo je proučiti obje testirane verzije AES protokola. Ukupno vrijeme izvršavanja ostaje gotovo konstanta pri povećanju broja niti na više od dva. Npr. povećanjem broja niti s dvije na šesnaest smanjeno je vrijeme izvršavanja za 26%. Smanjenje vremena izvršavanja podjelom na dvije niti u odnosu na jednu nit je 20%. Vidimo da povećanje broja niti ne dovodi do značajnog smanjenja vremena izvođenja.

Povećanje broja niti pri implementaciji različitih protokola donosi različita smanjenja u vremenu izvođenja. Postavlja se pitanje zašto se vrijeme ne smanjuje jednoliko za sve testirane implementacije protokola.

Možemo pretpostaviti da je pri implementaciji svih navedenih protokola uključena podjela na niti. Iz rezultata se ne može sa sigurnošću odrediti na koliko niti se izvršava pojedini protokol. Pri veličini teksta od 1 MB, ukupno vrijeme se ne smanjuje pri broju niti iznad osam za DES i 3DES te iznad četiri za AES. Rezultati sugeriraju da je broj niti na kojem se inicijalno raspoređuje posao veći kod implementacije AES protokola u usporedbi sa DES i 3DES. Ovaj slučaj možemo objasniti povećanjem broja niti iznad broja niti koje se mogu paralelno obrađivati, stoga dolazi do pauziranja izvođenja dijela niti za vrijeme izvršavanja ostalih.

Pauziranje jedne niti i pokretanje sljedeće nije trenutna operacija, nego ima svoje trajanje. Povećanjem broja niti može doći do učestalijeg prebacivanja između niti te vrijeme prebacivanja postaje značajni faktor. Broj prebacivanja ne mora nužno biti funkcija broja niti. Također, kreiranje pojedine niti nije trenutna operacija. U slučaju šifriranja izvornog teksta od 1 MB, pri podjeli na veći broj niti povećava se vrijeme izvođenja. Na primjer, prijelaz sa osam na šesnaest niti u slučaju AES protokola.

Uspoređujući vrijeme potrebno za šifriranje između različitih protokola dobijemo rezultate prikazane u tablicama 5.2 i 5.3. Najbrži protokol možemo uzeti kao referentni i uspoređivati koliko su puta ostali protokoli sporiji. Razlika između 128 i 256 bitne verzije je 6%. Uspoređujući broj ciklusa možemo zaključiti da će 256-bitna verzija biti 40% sporija jer ima 40% više ciklusa (14 za 256-bitnu verziju, u usporedbi s 10 za 128-bitnu verziju). Međutim, razlika je 6%. Stoga možemo zaključiti da na brzinu šifriranja utječe više faktora osim broja ciklusa.

Tab. 5.2: Usporedba brzine šifriranja primjenom 1 niti

Protokol	Vrijeme (ms)	Brzina (Mbit/s)	Koeficijent
AES128	5050	10.38	1
AES256	5355	9.79	1.0603
DES	14147	3.7	2.8014
3DES	15219	4.18	2.4790

Tab. 5.3: Usporedba brzine šifriranja primjenom 8 niti

Protokol	Vrijeme (ms)	Brzina (Mbit/s)	Koeficijent
AES128	3902	13.43	1
AES256	4086	12.85	1.0458
DES	5021	10.44	1.2868
3DES	5049	10.38	1.2939

Sukladno podacima na grafovima 4.2 i 4.3 vidimo da povećanje broja niti više utječe na smanjenje vremena izvođenja kod protokola DES i 3DES u usporedbi s protokolom AES.

Promatranjem ovisnosti vremena izvođenja o uređaju može se zaključiti da odnosi brzina nisu jednoliki između različitih uređaja. Na primjer, uspoređujući LG G Pad i LG Nexus 4, vidimo da je LG G Pad brži u šifriranju AES protokolom, dok je LG Nexus 4 brži sa DES i 3DES protokolima. Opisani uređaji razlikuju u većem broju parametara, stoga nije moguće niti pretpostaviti zašto se to događa.

Zanimljivo je usporediti Galaxy A5 [21] koji raspolaže sa osam jezgri s preostala dva uređaja četiri jezgrena uređaja [22, 23]. Ukoliko promatramo rezultate za 8 niti i rezultate bez podjele, uređaj sa četiri jezgre šifriranje obradi za od 86% do 269% više vremena.

Iz priloženog rezultata u tablicama 4.3 i 4.5 se može zaključiti da udvostručavanje broja jezgri ne utječe na performanse unutar aplikacije. Za potvrđivanje ove hipoteze bilo bi potrebno izvršiti testiranja na uređajima između kojih je razlika samo u broju jezgri. Idealno bi bilo testiranje izvršiti na uređaju kojemu se jezgre mogu isključiti/blokirati po potrebi. Unutar postavki svih testiranih uređaja ne postoji mogućnost manipulacije broja jezgri. Mijenjanje postavki upotrebom neprovjerenih aplikacija od treće strane nije bila moguća opcija zbog moguće povrede uvjeta garancije ili štete na uređajima.

6. ZAKLJUČAK

Cilj ovog rada je testiranje implementacija simetričnih kriptografskog sustava pri izvođenju na više niti na Android platformi. Rad nudi uvid u rad testiranih protokola na teorijskoj razini, bez zadiranja u implementacije. Opisani su dijelovi sustava i proces kriptiranja bloka podataka protokola koji su testirani. Također su objašnjeni načini kriptiranja koje simetrični kriptografski sustavi omogućuju. Opisane su osnovne komponente Android aplikacija.

Testiranje je izvedeno pomoću aplikacije napisane u Java programskom jeziku na platformi Android. Aplikacija za testiranje dio je rada. Opisana je struktura aplikacije. Prikazani su najvažniji dijelovi aplikacije. Prilikom dizajniranja sučelja težilo se ka jednostavnosti upotrebe i prikupljanja rezultata. Rezultati se pohranjuju u datoteku koji je moguće prenijeti na osobna računala na daljnju obradu. Također, implementirana je mogućnost automatskog generiranja svih testnih slučajeva te automatsko pokretanje svih testova bez nužne interakcije korisnika između testova.

Provedeno je testiranje pomoću tri uređaja. Prikazane su ovisnosti rezultata o broju niti i veličini datoteke. Uspoređeni su rezultati između različitih uređaja. Rezultati testiranja pokazuju da povećavanje broja niti koji izvršavaju testiranje drugačije utječe na rezultate ovisno o protokolu. DES i 3DES imaju veće povećanje performansi povećanjem broja niti u odnosu na AES. Razlog tome može biti podjela na više niti unutar AES protokola. Također je uočeno da povećanje broja jezgri ne utječe na performanse ovisno o povećanju broja niti. To može značiti da sustav ne daje sve jezgre na raspolaganje svakoj aplikaciji. Također, to može upućivati na pogrešan pristup prilikom testiranja.

Daljnje unaprjeđenje aplikacije bilo bi u obliku uključivanja podrške za više simetričnih kriptografskih protokola i više implementacija već postojećih protokola. Nadalje, obuhvaćanjem većeg broja uređaja prikazala bi se šira slika performansi uređaja.

LITERATURA

[1] Menezes, A. J.; van Oorschot, P. C.; Vanstone, S. A. Handbook of Applied Cryptography. ISBN 0-8493-8523-7

[2] B. Steiner, Applied Cryptography, Second Edition: Protocols, Algorithms, and Source Code in C, poglavlje 14.1, John Wiley & Sons, Inc, 1996.

[3] https://developer.android.com/guide/components/fundamentals.html, 1.9.2016

[4] M. Backes, Lecture Notes for CS-578 Cryptography, Lecture 3, Saarland University

[5] http://web.interhack.com/news/n2005/bruteforce, preuzeto 5.6.2016.

[6] http://www.sciengines.com/company/news-a-events/74-des-in-1-day.html, preuzeto 1.9.2016.

[7] William C. Barker, Elaine Barker, NIST Special Publication 800-67, Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher

[8] https://supportforums.cisco.com/document/6661/3des, preuzeto 1.9.2016

[9] Federal Information Processing Standards Publications 197, Advanced Encryption Standard (Aes), November 26, 2001

[10] M. Dworkin, Recommendation for Block Cipher Modes of Operation, Methods and Techniques, poglavlje 6.1, NIST Special Publication 800-38A, 2001.

[11] http://en.wikipedia.org/wiki/Block_cipher_mode_of_operation, preuzeto 3.9.2016

[12] Budin, Golub, Jakobović, Jelenković, Operacijski sustavi, 2.izdanje, Zagreb, 2011.

[13] https://developer.android.com/training/basics/activity-lifecycle/starting.html, preuzeto 3.9.2016

[14] https://docs.oracle.com/javase/7/docs/api/javax/crypto/package-summary.html, preuzeto 1.9.2016.

[15] http://arstechnica.com/gadgets/2012/06/android-4-1-jelly-bean-faster-smoother-more-delightful/, 1.9.2016

[16] https://developer.android.com/about/dashboards/index.html, preuzeto 1.9.2016.

[17] https://tools.ietf.org/html/rfc4180, preuzeto 2.9.2016.

[18] https://support.office.com/hr-hr/article/Uvoz-i-izvoz-tekstnih-datoteka-txt-ili-csv-5250ac4c-663c-47ce-937b-339e391393ba, preuzeto 4.9.2016.

[19] National Institute of Standards and Technology , FIPS PUB 180-4, Secure Hash Standard (SHS) , August 2015

[20] http://www.aspheute.com/english/20040105.asp, preuzeto 3.9.2016.

[21] http://www.samsung.com/uk/consumer/mobile-devices/smartphones/galaxy-a/SM-A510FZDABTU, preuzeto 1.9.2016.

[22] http://www.lg.com/us/tablets/lg-V410-g-pad-7.0-lte, preuzeto 1.9.2016.

[23] http://www.lg.com/hr/mobilni-telefoni/lg-Nexus-4-LG-E960-pametni-telefon, preuzeto 1.9.2016.

SAŽETAK

U radu je provedeno testiranje implementacija više simetričnih kriptografskog sustava pri izvođenju na više niti na Android platformi. Testirani protokoli su AES u 128 i 256 bitnoj verziji, DES u 64 i 3DES u 128 bitnoj verziji. Programski jezik Java kao glavni jezik za razvoj na Android platformi odabran je zbog raširenosti Android sustava na pametnim telefonima (eng. *smartphones*). Izrađena je aplikacija za izvršavanje testiranja. Aplikacija omogućuje izradu više scenarija koje definiraju parametri testiranja. Omogućeno je automatsko izvršavanje više scenarija kako bi pojednostavili prikupljanje rezultata. Opisana je struktura i funkcionalnost aplikacije. Provedeno je testiranje na tri uređaja s više ponavljanja svakog testa. Prikazani su rezultati testiranja tablično i grafički. Izvedeni su zaključci o implementaciji testiranih protokola.

Ključne riječi: Android, AES, DES, 3DES, kriptografija, testiranje, niti, više-nitnost, paralelno procesiranje

ABSTRACT

This research looks at symmetric cryptographic protocols performance on Android platform compared to the number of threads, plaint text size and device specifications. Included protocols are AES in 128 and 256 bit variants, DES in 64 and 3DES in 128 bit variant. Programming language Java is used as main development. Research was based on results collected with application developed for the purpose of this research. Application allows user to run multiple scenarios defined by the set of parameters. Automated testing is enabled for easier gathering of results. Application structure and functionality are described. Research was done on 3 devices with multiple runs per test. Results are presented in both graphical and in tabular form. Explanations are proposed over underlying implementation of protocols.

Key words: Android, AES, DES, 3DES, cryptography, testing, threads, multithreading, parallel processing

ŽIVOTOPIS

Zoran Kovačević je rođen 20.7.1991. godine u Osijeku. Osnovnu školu upisuje 1998. godine te potom pohađa Treću gimnaziju u Osijeku u razdoblju od 2006. do 2010. godine. Za vrijeme srednjoškolskog obrazovanja sudjeluje na državnom natjecanju iz programiranja. 2010. godine nastavlja školovanje upisom na Elektrotehnički fakultet u Osijeku, preddiplomski studij elektrotehnike. Razdoblje studiranja obilježava sudjelovanje na timskim IEEXtreme natjecanjima u programiranju od 2011. do 2015. godine. Posebno se ističe 2012. godina kada tim ostvaruje zapažen rezultat ulaskom u 10% najboljih. 2013. godine radeći na projektu razvoja aplikacije za video obradu na Android sustavu odrađuje ljetnu praksu u Madridu, Španjolska. Iste godine završava preddiplomski studij elektrotehnike te stječe titulu prvostupnika inženjera elektrotehnike. Od 2015. godine radi u *Code Consulting* d.o.o. na razvoju web aplikacija.

PRILOZI

[1] Izvorni kod aplikacije *CryptoBenchmark*

[2] Rezultati testiranja o obliku csv datoteke

[3] Obrađeni rezultati u obliku Excel dokumenta