



Univerzitet Singidunum
Fakultet za informatiku i računarstvo

Sinteza sopstvenog izvora slučajnosti za generisanje kriptoloških ključeva

- Diplomski rad -

Beograd, 2015. godine

UNIVERZITET SINGIDUNUM

FAKULTET ZA INFORMATIKU I RAČUNARSTVO

Beograd, Danijelova 32

Broj: _____/2015.

Kandidat: Milutin Đukanović

Broj indeksa: 2011/200072.

Smer: Informatika i računarstvo

Tema: Sinteza sopstvenog izvora slučajnosti za generisanje kriptoloških ključeva

Datum odobrenja teme:

Beograd, _____.____.2015.

MENTOR

Doc. dr Saša Adamović

DEKAN

Prof. dr Dragan Cvetković

SINTEZA SOPSTVENOG IZVORA SLUČAJNOSTI ZA GENERISANJE KRIPTOLOŠKIH KLJUČEVA

Sažetak – Osnova svih šifarskih sistema je izvor slučajnosti na osnovu koga se generišu potrebni kriptološki ključevi koji uz pomoć algoritama obezbeđuju zaštitu informacija. Razvojem tehnologija, posebno u domenu hardvera, kriptanalizom je sve lakše odrediti periodičnost pseudoslučajnih generatora PRNG (eng. Pseudo Random Number Generator) što predstavlja ranjivost kompletnog sistema za kriptološku zaštitu informacija. Rešenje prikazano u ovom radu predstavlja izvor slučajnosti kao generator istinski slučajnih brojeva TRNG (eng. True Random Number Generator) zasnovan na neodređenosti položaja većeg broja vozila u geografskom koordinatnom sistemu. Sistem predstavlja zatvorenu celinu sa ulaznim informacijama koje mogu biti poznate eventualnom napadaču ali na osnovu kojih napadač ne može predvideti generisane slučajne vrednosti. Sistem nalazi primenu, osim u kriptologiji, i u raznim sistemima za simulaciju, kompjuterskim igrama, naučnim aplikacijama, pa čak i u muzici i umetnosti. Savremeni kriptološki sistemi su danas nezamislivi bez istinski slučajnih izvora brojeva TRNG i u tom pogledu ovo rešenje ima značajnu upotrebnu vrednost.

Ključne reči: istinski izvor slučajnosti, generatori kriptoloških ključeva, šifarski sistemi, neodređenost brojeva

SYNTHESIS OF OWN SOURCE OF RANDOMNESS TO GENERATE CRYPTOLOGIC KEYS

Abstract – The basis of all encryption systems is a source of randomness which is used to generate necessary encryption keys which provide protection of information with help of algorithms. With the development of the technology, especially in the domain of hardware, with crypto-analysis it is becoming easier to determine the periodicity of pseudorandom generators PRNG what represents vulnerability of a complete system for cryptography protection of information. The solution presented in this project represents a source of randomness as true random number generator TRNG based on the unpredictability of the position of a large number of vehicles in a geographic coordinate system. The system represents a closed unit with the input information that may be known to a possible attacker but on that basis an attacker can not predict generated random values. The system is usable, not only in cryptography, but also in various systems for simulation, computer games, scientific applications, and even music and art. Modern cryptography systems are unthinkable today without a source of true random numbers TRNG, and in that respect this solution has significant practical value.

Keywords: source of true randomness, encryption keys generators, encryption systems, unpredictability of numbers

SADRŽAJ

1. Uvod.....	4
1.1 Predmet istraživanja	4
1.2 Doprinosi rada	4
1.3 Struktura rada.....	5
2. Pregled u oblasti istraživanja	6
2.1 Primena generatora slučajnih brojeva.....	6
2.1.1 Primena generatora slučajnih brojeva u kriptografiji	6
2.1.2 Primena generatora slučajnih brojeva u simulacijama	7
2.1.3 Primena generatora slučajnih brojeva u kompjuterskim igrama	7
2.1.4 Primena generatora slučajnih brojeva u estetskom izražavanju	8
2.1.5 Aplikacije servisa za generisanje slučajnih brojeva Random.org	8
2.2 Tipovi generatora	9
2.2.1 Generatori istinski slučajnih brojeva	9
2.2.1.1 TRN generator na osnovu atmosferskog šuma – Random.org	10
2.2.1.2 TRN generator na osnovu nuklearnog raspada atoma – HotBits	11
2.2.1.3 TRN generator na osnovu koordinata avionskog saobraćaja – FlyBit.....	11
2.2.2 Generatori pseudoslučajnih brojeva	12
2.2.2.1 PRN generatori u PHP programskom jeziku	13
2.2.2.2 PRN generatori u Java programskom jeziku	14
2.2.2.3 Drugi PRN generatori	14
2.2.3 Poređenje TRN i PRN generatora	15
2.2.4 Sopstveni izvor slučajnosti	16
2.3 Provajderi kriptografskih funkcionalnosti.....	16
2.3.1 JCA kriptološko proširenje	16
2.3.2 JCE kriptološko proširenje.....	17
2.3.3 Podržani izvori slučajnosti	17
2.3.3.1 Primer upotrebe generatora SHA1PRNG.....	18
3. Teorijske osnove	19
3.1 Šta je izvor slučajnosti?	19

3.2	Kodovanje izvora slučajnosti	20
3.3	Generisanje ključeva za simetrične i asimetrične algoritme.....	22
3.4	Specifikacije za generisanje simetričnih i asimetričnih ključeva	23
4.	Eksperimentalni rad	24
4.1	Opis eksperimentalnog okruženja	24
4.2	Generička šema razvijenog rešenja	26
4.2.1	GPS -> Izvor -> Kodovanje -> Slučajni brojevi	28
4.2.2	GPS -> Izvor -> Kodovanje -> Postdigitalna obrada -> Slučajni brojevi	29
4.3	Statistička analiza – rezultati	30
4.3.1	Statistička analiza šema 1 (NIST)	30
4.3.1.1	Šema 1 - Ispitivanje učestanosti u nizu.....	30
4.3.1.2	Šema 1 - Serijski test.....	31
4.3.1.3	Šema 1 – Ispitivanje uzastopnih ponavljanja	31
4.3.1.4	Šema 1 – Entropija.....	32
4.3.2	Statistička analiza šema 2 (NIST)	32
4.3.2.1	Šema 2 - Ispitivanje učestanosti u nizu.....	32
4.3.2.2	Šema 2 - Serijski test.....	33
4.3.1.3	Šema 1 – Ispitivanje uzastopnih ponavljanja	34
4.3.1.4	Šema 1 – Entropija.....	34
4.3.3	Razlike performansi šeme 1 i šeme 2	34
4.4	Prikaz funkcionalnosti razvijene Java aplikacije	35
4.4.1	Primeri koda	35
5.	Zaključak	40
6.	Literatura	41

Spisak slika

Slika 1 - Izgled grafikona entropije izvora Random.org	8
Slika 2 - Grafička prezentacija rezultata PHP funkcije rand()	11
Slika 3 - LFSR pseudoslučajni generator sa 5-bitnim registrom	13
Slika 4 - Dijagram CSPRN generatora u Javi.....	15
Slika 5 - Optički izvor slučajnosti na osnovu ponašanja fotona.....	17

Slika 6 - Princip rada aplikacije za praćenje vozila GPSTracker.com.....	23
Slika 7 –Generička šema izvora slučajnih brojeva za generisanje kriptoloških ključeva	24
Slika 8 –Konverzija prikupljenih podataka	25
Slika 9 -Dijagram izvora slučajnih brojeva šema 1.....	27
Slika 10 - Dijagram proširenja šeme 1 sa postdigitalnom obradom sekvenci – šema 2	28

Spisak tabela

Tabela 1 –Dužine ključeva za simetrične kriptološke algoritme	21
Tabela 2 –Šema 1-Ispitivanje učestanosti u nizu.....	30
Tabela 3 – Šema 1-Rezultati serijskog testa	31
Tabela 4 – Šema 1-Ispitivanje uzastopnih ponavljanja	31
Tabela 5 –Šema 1-Ispitivanje entropije.....	32
Tabela 6 –Šema 2-Ispitivanje učestanosti u nizu.....	32
Tabela 7 – Šema 2-Rezultati serijskog testa.....	33
Tabela 8 –Šema 2-Ispitivanje uzastopnih ponavljanja	33
Tabela 9 - Šema 1-Ispitivanje entropije.....	34

1. Uvod

1.1 Predmet istraživanja

Izvori slučajnosti predstavljaju osnov za generisanje kriptoloških ključeva i jedan od najosetljivijih elemenata zaštite informacija. Od neodređenosti izvora slučajnih binarnih sekvenci zavisi kvalitet generisanih ključeva a sa time i pouzdanost kriptoloških alata. Potreba za generisanjem slučajnih binarnih sekvenci se proteže od generisanja šifri za pristup raznim ličnim nalogima korisnika do zaštite tokova informacija. Najčešće korišćeni vid generisanja slučajnih binarnih sekvenci je pomoću determinističkih algoritama u računarskim sistemima. Kao početne vrednosti algoritama koriste se pseudo slučajne vrednosti tako da ovaj način nije adekvatan za dobijanje jakih kriptoloških ključeva.

Najbolji izvori neodređenosti se mogu pronaći u fizičim pojavama i one se uzimaju kao osnov za generisanje istinski slučajnih sekvenci brojeva. Danas postoji veći broj javno dostupnih generatora istinski slučajnih brojeva a u ovom radu biće predstavljeno rešenje izvora slučajnosti koji se može upotrebiti za generisanje kriptoloških ključeva jer za osnov koristi izvor podataka visoke neodređenosti.

Rešenje u ovom radu je zasnovano na slučajnosti koja se dobija iz generisanih geografskih podataka putem GPS praćenja većeg broja vozila u pokretu. S obzirom da je vremenski interval generisanja trenutnog položaja kratak i da broj vozila može biti jako velik, dobijamo dovoljnu količinu stalno generisanih podataka za potrebe jednog ovakvog izvora slučajnosti.

U radu će biti upotrebljeni i softverski moduli koji obezbeđuju obradu dobijenih slučajnih sekvenci. Implementirani kod će obezbediti stabilnost sistema i prevenciju od anomalija izvora kao i zaštitu od krypto-analitičkih napada na generator radi realizacije algoritma za predviđanje narednih sekvenci.

Istraživanje sprovedeno u okviru ovog rada jasno je potvrdilo pretpostavku da jedan ovakav izvor može dati zaista klavitetne istinski slučajne binarne sekvence koje se mogu koristiti za generisanje kriptoloških ključeva.

1.2 Doprinosi rada

Doprinos ovog rada se ogleda u sledećem:

1. Realizacija izvora slučajnih binarnih sekvenci iz čisto fizičkog izvora koji je generator neodređenosti na osnovu podataka drugog servisa koji te podatke proizvodi u druge svrhe. To znači da se podaci generisani za potrebe praćenja vozila koriste u još jednu svrhu čime se povećava isplativost kompletne infrastrukture.

2. Mogućnost korišćenja generisanih binarnih sekvenci kako za kriptološke ključeve tako i za zaštitu tokova podataka. Sa dovoljno velikim brojem vozila može se dobiti velika količina slučajnih sekvenci.
3. Visok kvalitet i neodređenost binarnih sekvenci koja će biti potvrđena NIST testovima. Upotrebljeni algoritmi za kodovanje i obradu binarnih sekvenci obezbediće dobru uniformnu raspodelu

1.3 Struktura rada

U drugom poglavlju ovog rada biće predstavljen pregled istraživanja u ovoj oblasti. Biće navedeni primeri upotrebe generatora slučajnih brojeva kao i načini njihovog rada. Predstavljeni će biti i najpoznatiji javni servisi za generisanje istinski slučajnih brojeva i principi na osnovu kojih rade kao i fizički izvori slučajnosti koje koriste. U ovom poglavlju biće prikazani i tipovi generatora sa njihovim karakteristikama i realizacijama. Programski jezici takođe poseduju svoje generatore pseudoslučajnih brojeva i u ovom poglavlju ćemo opisati generatore iz dva najpopularnija programska jezika.

Treće poglavlje ovog rada je posvećeno teorijskim osnovama koje su neophodne za realizaciju rešenja izvora slučajnosti. Tehnike kodovanja, načini generisanja ključeva i specifikacije ključeva su teme opisane u ovom poglavlju.

U okviru četvrtog poglavlja prikazaćemo predloženo rešenje sa detaljnim opisom strukture u dve varijante, sa i bez postdigitalne obrade. Prikazaćemo i rezultate testova obe varijante rešenja kao i primere koda za realizaciju pisanog u Java programskom jeziku.

Na kraju, u petom poglavlju, opisaćemo postignute rezultate rešenja, koje su prednosti u odnosu na postojeća rešenja, u koje svrhe se može izvor slučajnosti koristiti i koje su smernice za budući rad.

2. Pregled u oblasti istraživanja

2.1 Primena generatora slučajnih brojeva

Generatori slučajnih brojeva danas imaju široku primenu kako u nauci (kriptografija, simulacije) tako i u svakodnevnom životu (kompjuterske igre, estetskom izražavanju, muzici i umetnosti). Savremeni kriptološki sistemi se ne mogu zamisliti bez upotrebe generatora slučajnih brojeva za potrebe zaštite informacija, stohastički simulacioni modeli, čija je karakteristika slučajno ponašanje, koriste slučajne promenljive za ispitivanje sistema, zanimljivost kompjuterskih igara ne bi postojala da se kreatori ne koriste slučajnim vrednostima za dinamiku događaja, a slučajne vrednosti se koriste i za dobijanje muzickih sekvenci i grafičkih fraktala.

2.1.1 Primena generatora slučajnih brojeva u kriptografiji

Generisanje slučajnih brojeva je osnov skoro svih kriptografskih algoritama koji se zasnivaju na upotrebi ključeva za šifrovanje informacija. Od kvaliteta generisanog kriptološkog ključa i upotrebljenog algoritma zavisi tajnost kompletnog sistema za zaštitu informacija a s obzirom da se za generisanje ključeva koriste generatori slučajnih brojeva, njihov značaj u kriptološkom sistemu je izuzetno velik [1]. Najbolji primer za to je upotreba *One-Time-Pad* kriptološkog algoritma čiji ključ predstavlja sekvencu slučajnih brojeva iste dužine kao i informacija koja se šifrjuje. U kriptografiji generatori slučajnih brojeva se koriste pretežno u sledećim slučajevima:

- Za distribucione potrebe ključeva i autorizacione algoritme. U ovom slučaju generišu se rečenice slučajnih brojeva koje predstavljaju sesijske identifikatore koji su jedinstveni za svaki zahtev komunikacione strane upućen centru za distribuciju ključeva,
- Za generisanje sesijskih ključeva u simetričnim kriptografskim algoritmima koji se koriste za jednu sesiju između komunikacionih strana i imaju određeno kratko vreme validnosti.
- Za generisanje para ključeva *RSA* algoritma sa javnim ključem
- Za generisanje povorke bitova koja se koristi u simetričnom šifrovanju toka podataka (primer: *One-Time-Pad* algoritam).

Ovakva primena slučajnih brojeva u kriptografiji postavlja dva osnovna zahteva pred generatore slučajnih brojeva: slučajnost i nepredvidivost. Slučajnost mora ispunjavati dva kriterijuma: uniformnu raspodelu bitova i međusobnu nezavisnost sekvenci generisanih bitova. Nepredvidivost zahteva od generatora slučajnih brojeva ne

samo da generiše slučajne brojeve već i da se ti brojevi ne mogu predvideti na osnovu drugih brojeva u sekvenci.

Realizacija i primena generatora slučajnih brojeva u kriptografiji a i u simulacionim modelima, zahteva posebnu pažnju i nosi sa sobom odgovornost zbog važnosti aplikativne primene njihovih generisanih vrednosti jer od kvaliteta proizvedenih sekvenci zavise mnogo veći sistemi zaštite i simulacije koji se na njih oslanjaju.

2.1.2 Primena generatora slučajnih brojeva u simulacijama

Generatori slučajnih brojeva se koriste u raznim simulacionim modelima koji koriste Monte Carlo method za simulacione sisteme. Monte Carlo metod je klasa računskih operacija i algoritama koja se zasniva na uzastopnoj upotrebi slučajnih uzoraka vrednosti na ulazu sistema za dobijanje numeričkih rezultata na izlazu. Pretežno se koristi za ispitivanje sistema u fizici i matematici, ali nalazi primenu i u ekonomiji[6].

U simulacionim modelima nije moguće ispitati svaku mogućnost ponašanja sistema u zavisnosti od parametara pa se zato korišćenjem slučajnih vrednosti dobijaju rezultati ispitivanja sistema u širem dijapazonu. Na osnovu rezultata dobijenih korišćenjem slučajnih vrednosti, može se doneti zaključak o stabilnosti sistema bez potrebe korišćenja determinističkih metoda. Korišćenje slučajnih brojeva i promenljivih u simulacionim modelima omogućava prikaz ponašanja elemenata sistema bez potrebe postojanja modela koji detaljno opisuje to ponašanje.

2.1.3 Primena generatora slučajnih brojeva u kompjuterskim igrama

Kompjuterske igre sve više koriste generatore slučajnih brojeva za postizanje neodređenosti svojih funkcija i algoritama. Slučajne vrednosti omogućavaju veću dinamičnost i nepredvidivost igara i time za igrača postavljaju teže i izazovnije zadatke. U kompjuterskim igrama se najčešće koriste paterni za razne nivoe i ponašanja karaktera. Da bi se postigla dinamičnost igre bez upotrebe velikog broja paterna koriste se generatori slučajnih vrednosti koji omogućavaju da se jedan patern koristi više puta sa različitim ponašanjem. Najčešće se generatori slučajnih vrednosti u kompjuterskim igrama koriste u sledećim paternima:

- Jedinična statistika (eng. *Unit Stats*) – određivanje jedinstvenih karakteristike svakog „neprijateljskog“ karaktera u igrici, kao što su: snaga, moć, oružje, brzina, prostim kloniranjem jednog paterna ili kreiranjem ponašanja za svakog nezavisno je nezanimljiv i nepraktičan način i dodavanjem slučajnih vrednosti dobijamo realnije i zanimljivije igračko iskustvo korisnika kompjuterske igrice.
- Generisanje mapa – Generatori slučajnih brojeva se koriste i za kreiranje slučajnih terena. Dug i mukotrpan posao bi bio crtati posebno svaki deo terena i mape a pored toga, ovaj važan deo igre ostaje određen brojem delova terena koji su urađeni, dok primenom generatora slučajnih vrednosti tereni i mape mogu imati neodređen broj elemenata.

- Generisanje sprajtova (eng. *Sprite*) – Korišćenjem generatora slučajnih brojeva možemo sprajtove nasumično generisati i rasporediti u igri i postići željene efekte kompleksnosti dizajna.
- Veštačka inteligencija – Gotovo je nezamislivo odrediti kako će se karakteri u igri ponašati i najčešći slučaj u igrama danas je da se slučajne vrednosti koriste za donošenje odluka o ponašanju karaktera i elemenata kompjuterske igre, sto ostavlja na igrača utisak o postojanju veštačke inteligencije u igri.

Pored arkadnih i strateških kompjuterskih igara, generatori slučajnih vrednosti posebno mesto zauzimaju u kompjuterskim simulacijama igara na sreću (rulet, poker, klađenje) i tu je njihova upotreba od vitalnog značaja kako za utisak o realnosti tako i za bezbednost.

2.1.4 Primena generatora slučajnih brojeva u estetskom izražavanju

Muzičari na pojam generatora slučajnih brojeva gledaju drugačije od matematičara. Dok je za matematičare očekivan rezultat generatora slučajnih brojeva ujednačena raspodela svih očekivanih vrednosti, za muzičare takav rezultat može proizvesti veoma dosadne i jednolične muzičke sekvence. Ipak, u muzici postoje algoritmi koji generišu muzičke sekvence zasnovane na slučajnim vrednostima koje koriste kao ulazne vrednosti (eng. *seed*). Jedan od primera je upotreba zvučnih efekata u muzici. Kao primer se može navesti upotreba zvuka zvončića koje pokreće vetar. Generisanje tona zvonceta je inicirano nekim od generatora slučajnih brojeva. Primera ima jos dosta i naravno, ne mogu se tretirati kao kreativni ali u muzičkom izražavanju imaju svoje mesto.

U pogledu grafičkog izražavanja generatori slučajnih brojeva pored upotrebe za generisanje grafičkih prikaza terena u kompjuterskim igrama, važno mesto imaju i u generisanju grafika i slika. Opseg upotrebe se kreće od generisanja objekata zasnovanih na teksturama, kao sto su oblaci ili plamen vatre, do složenijih kao što su ostrva, planinski venci i šume. Veliku primenu generatori slučajnih brojeva imaju i u pogledu kreiranja tekstura za vektorske objekte generišući različite oblike i kombinacije boja.

2.1.5 Aplikacije servisa za generisanje slučajnih brojeva Random.org

Jedan od vodećih servisa za generisanje pravih slučajnih brojeva, Random.org, u svojoj ponudi ima usluge koje su zasnovane na sekvencama istinskih slučajnih brojeva. Pored standardne ponude binarnih sekvenci, sekvenci celih brojeva, decimalnih brojeva u određenom opsegu, tu su i aplikacije za generisanje slučajnih stringova, lozinki, datuma i vremena, geografskih koordinata, bitmapa koji reprezentuju sekvence slučajnih brojeva, kao i generator belog šuma.

2.2 Tipovi generatora

Za potrebe generisanja slučajnih brojeva danas su najčešće u upotrebi dva tipa generatora: generatori istinski slučajnih brojeva (eng. *True Random Number Generators - TRNG*) i generatori pseudoslučajnih brojeva (eng. *PseudoRandom Number Generators - PRNG*). Pri generisanju ključeva za upotrebu u kriptografiji preporuka je upotreba generatora istinski slučajnih brojeva, a ukoliko se koristi pseudoslučajni generator, na njegov ulaz kao početnu vrednost (*seed*) potrebno je dovesti vrednost generisanu generatorom istinski slučajnih brojeva. Generatori na svojim izlazima daju binarne sekvence, s tim što je za dobijanje binarne sekvence kod *TRN* generatora potreban hardver koji će neku fizičku pojavu konvertovati u binarni niz a kod *PRN* generatora binarne sekvence se dobijaju korišćenjem posebnih programskih algoritama koji na osnovu istinski slučajne inicijalne vrednosti ulaza generišu slučajne binarne nizove na izlazu.

2.2.1 Generatori istinski slučajnih brojeva

Za *TRN* generatore kao izvori slučajnosti najčešće se koriste fičke pojave sa visokim nivoom neodređenosti, kao što su termalni šum u elektronskim kolima, radioaktivni raspad, atmosferski šum, nepravilnosti čitanja signala sa harddiska usled vibracija rotirajuće magnetne ploče, koordinate dobijene pokretanjem miša i druge. Ovi generatori na svom izlazu generišu binarni niz čija entropija ne može biti veća od entropije izvora slučajnosti koji koriste. Entropija izvora je i najkritičnija komponenta pri realizaciji generatora s obzirom da od njene vrednosti zavisi kvalitet generisanih sekvenci. Upotrebom posebnih sistema kodovanja i postdigitalnih obrada dobijenih binarnih sekvenci moguće je poboljšati kvalitet i obezbediti otpornost na promene i ometanja primarnog izvora slučajnosti. Postdigitalna obrada generisanih slučajnih vrednosti nije neophodna tehnika ali se često primenjuje kod velikih sistema za generisanje istinski slučajnih brojeva.

Jedan od nedostataka *TRN* generatora je da ne mogu generisati velike količine slučajnih binarnih vrednosti i u direktnoj su zavisnosti od količine proizvedenih impulsa primarnog izvora i kreću se u rasponu do nekoliko Mbit-a u sekundi. Upotrebom savremenih optičkih komponenata posebno lasera i superluminentnih dioda kao generatora šuma, raspon generisanih binarnih sekvenci je povećan do nekoliko Gbit-a u sekundi. Postdigitalnom obradom vrednosti sa izvora dobija se kvalitetniji binarni niz ali u većini slučajeva se smanjuje količina generisanih bitova na izlazu, što se može delom kompenzovati upotrebom različitih algoritama za obradu dobijenih rezultata iz izvora.

S obzirom da se *TRN* generatori oslanjaju na fizičke pojave, neophodan je hardver koji ponekad može višestruko povećati troškove implementacije. Uticaj okruženja generatora kao i drugi spoljni faktori mogu doprineti nestabilnosti izvora slučajnosti što dovodi do pojave devijacije impulsa na izvoru i pojave generisanja dužih nizova 1 ili 0. To se koriguje u okviru postdigitalne obrade binarne sekvence, pretežno kompresionim algoritmima i na taj način se povećava vrednost entropije po jednom bitu sekvence.

Danas postoji više servisa za generisanje istinski slučajnih brojeva koji svoje usluge nude i preko internet aplikacija a dostupni su javno. Neki od njih su: Random.org, HotBits i FlyBit.

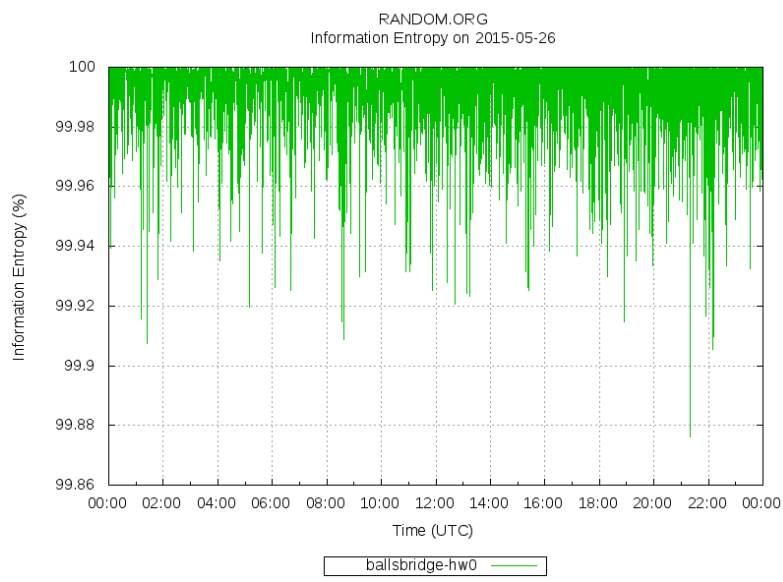
2.2.1.1 TRN generator na osnovu atmosferskog šuma – Random.org

Jedan od izvora slučajnosti za potrebe generisanja istinski slučajnih brojeva svakako može biti atmosferski šum. Upravo ovu prirodnu pojavu za svoj generator je iskoristio provajder Random.org i sa velikim uspehom pruža svoje usluge od 1997.g. do danas.

Random.org za prikupljanje atmosferskog šuma koristi radio prijemnike čije su frekvencije prijema podešene na frekventnom prostoru između komercijalnih radio stanica na FM i AM području. Svoje prijemnike imaju postavljene širom sveta i povezane u jedinstvenu mrežu koja obezbeđuje dovoljnu količinu informacija i bezbednost od mogućih napada i ometanja [11].

Atmosferski šum je prirodni proces nastao elektrostatičkim pražnjenjem u atmosferi usled grmljavine. Procene su da se širom sveta u jednoj sekundi dogodi oko 40 takvih pražnjenja. Udaljena pražnjenja u atmosferskom šumu proizvode više belog šuma dok bliža učestvuju impulsno i sa većom amplitudom. Prosečan nivo indukovnog signala u anteni je jako mali tako da se taj nivo pojačava elektronskim pojačavačem koji u ukupnom šumu učestvuje i sa svojim unutrašnjim termičkim šumom. Tako dobijeni pojačani šum se dalje prosleđuje u demodulacioni deo hardvera kao i u komparatore da bi na izlazu hardverskog dela konvertora dobili povorku jedinica i nula. Takav kontinualni binarni niz se uvodi u računarski sistem koji vrši filtriranje i postdigitalnu obradu dobijenih rezultata. Prema objavljenim podacima iz 2009.g. Random.org je generisao oko 3000 bita u sekundi po jedinici prijemnika. S obzirom na zahteve klijenata i potrebe za istinski slučajnim brojevima u raznim sferama života, Random.org danas poseduje mnogo kompleksniju hardversku arhitekturu koja obezbeđuje daleko veće količine generisanih brojeva.

U okviru svog servisa, Random.org nudi korisnicima i niz statističkih izveštaja o stanju i kvalitetu generisanih slučajnih binarnih nizova putem grafova za određeni datum. Izgled jednog od grafova na kome se može očitati entropija izvora za određeni datum prikazan je na slici 1.



Slika 1 – Izgled grafikona entropije izvora Random.org

2.2.1.2 TRN generator na osnovu nuklearnog raspada atoma – HotBits

HotBits je još jedan provajder koji se koristi prirodnim pojavom za generisanje istinski slučajnih brojeva. Generator je zasnovan na tehnici merenja vremena između radioaktivnih raspada Cezijuma 137 koji se detektuju u Geiger-Müller-ovoj cevi[8].

Pod određenom pobudom od atoma Cezijuma nastaje atom Barijuma a pri tom procesu se oslobađa beta čestica. Ukoliko nastavimo da pobuđujemo Barijum nastaje gama zračenje. Vreme između pojedinačnih beta raspada je neodređeno i idealno za generisanje slučajnih vrednosti. Cezijum ima ukupno vreme poluraspada od 30g kada se gubi oko 50% energije, a potpuni prestanak radioaktivnosti je nakon 240g [10].

Generisanje binarnih nizova se sastoji u merenju vremenskog intervala između 4 raspada. Između prvog i drugog, i trećeg i četvrtog. Ukoliko je prvi vremenski interval kraći od drugog generisaće se bit 0 a ukoliko je prvi interval duži od drugog generisaće se bit 1. Ako su vremenski intervali jednaki rezultat merenja se odbacuje.

U okviru hardverskog dela HotBits infrastrukture određeni su posebni računarski sistemi koji vrše isključivo merenje vremenskih intervala i potpuno su odvojeni od spoljašnjeg okruženja i prekida u procesima koje obavljaju da bi merenje bilo preciznije a samim tim sprečilo narušavanje entropije izvora. Takvi sistemi binarne nizove šalju direktno u skladište gde se oni memorišu i čekaju na upotrebu. Primena skladišta omogućava brži odziv na zahteve za slučajnim binarnim sekvencama od strane korisnika a i omogućavaju nesmetan rad servisa prilikom gašenja sistema za merenje radi održavanja. Svako od skladišta ima kapacitet od oko 7 miliona bita i dopunjavaju se u pozadinskim procesima između zahteva korisnika.

2.2.1.3 TRN generator na osnovu koordinata avionskog saobraćaja – FlyBit

Ovaj TRN generator za izvor istinski slučajnih brojeva nema prirodnu pojavu ali ima izvor koji se zasniva na koordinatama civilnih aviona prikupljenih pomoću sistema za praćenje letilica u vazduhu[9]. Letilice svoju lokaciju određuju preko posebnih ugrađenih GPS uređaja koji signal sa satelita koriste za određivanje položaja. Ti podaci se zatim smeštaju u radio pakete i emituju kao poruka o položaju letilice koja je namenjena kontrolama leta kao i drugim letilicama u vazduhu. Sistem se naziva ADS-B i danas većina letilica je opremljena ADS-B uređajima.

Sistem je realizovan sa dve antenske pozicije na kojima se sakupljaju podaci o letilicama. Podaci sa obe antene se zaštićenom SSL vezom šalju na glavni server i na njemu se vrši poređenje rezultata prikupljanja. Za generisanje istinski slučajnih binarnih nizova iskorišćene su geografska širina i dužina jer su to vrednosti koje se brzo menjaju u jedinici vremena. Svaka od ovih vrednosti ima udela od 8 bita u ukupnom generisanom nizu.

Količina generisanih binarnih sekvenci ovog sistema zavisi direktno od gustine saobraćaja koji varira u toku dana. Web servis omogućava preuzimanje binarnih sekvenci veličine do 1Mbit a preuzeta sekvenca se uklanja iz skladišta trajno. Generisane sekvence su testirane i namenjene pretežno za kriptografske ključeve.

2.2.2 Generatori pseudoslučajnih brojeva

Za razliku od generatora istinski slučajnih brojeva koji su nedeterministički izvori slučajnosti, generatori pseudoslučajnih brojeva se tretiraju kao deterministički izvori na osnovu toga što se njihovi izvori slučajnosti nalaze u računarskim sistemima koji su po prirodi ponašanja određeni [5]. *PRN* generatori imaju tri važne karakteristike koje ne poseduju *TRN* generatori: *efikasnost*, *određenost* i *periodičnost*.

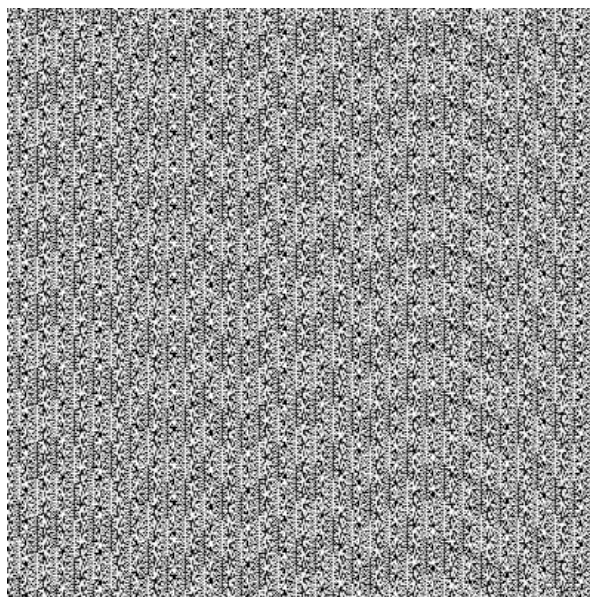
- *Efikasnost* ovih generatora se ogleda u brzini generisanja slučajnih binarnih sekvenci odnosno u generisanju većeg broja slučajnih bitova u kraćem vremenskom intervalu. Ovo je poželjna karakteristika ukoliko imate potrebe za većom količinom slučajnih binarnih sekvenci. Od brzine generisanja i upotrebljenog algoritma zavisi i kvalitet izlazne binarne sekvence tako da je potrebno naći najadekvatnije rešenje za realizaciju ovog generatora. Efikasnost ovih generatora se najviše ogleda u kriptološkoj zaštiti tokova velikih količina podataka jer *TRN* generatori ne mogu generisati dovoljnu količinu binarnih sekvenci u realnom vremenu.
- *Određenost* kao osobina *PRN* generatora ne predstavlja njihovu dobru karakteristiku i ogleda se kroz zavisnost izlaza od upotrebljenih algoritama i početne vrednosti. Naime, *PRN* generator proizvodi slučajnu binarnu sekvencu na osnovu početne vrednosti (eng.*seed*) koju koristi u algoritmima za transformaciju i enkodovanje tako da posmatrač bez znanja početne vrednosti ne može predvideti izlaznu vrednost. Ali ukoliko imate dva ista generatora i na njihove ulaze dovedete istu početnu vrednost, oba generatora će na svojim izlazima dati istu binarnu sekvencu čime dolazimo do zaključka da je mehanizam generisanja slučajnih sekvenci određen strukturom i načinom rada generatora. Ova karakteristika ima i svoju dobru stranu. Ukoliko između dva udaljena mesta želite zaštititi tok informacija dovoljno je da posedujete na oba mesta istu početnu vrednost i time generišete potpuno jednake slučajne sekvence za šifrovanje i dešifrovanje informacija na oba mesta.
- Periodičnost binarnih sekvenci koje proizvode ovi generatori dolazi od upotrebe početne vrednosti koja se koristi za generisanje izlaznih vrednosti. *PRN* generator proizvodi binarnu sekvencu koja zavisi od početne vrednosti i posebnim funkcijama transformiše tu vrednost u novu početnu vrednost koja se koristi za generisanje nove binarne sekvence. Pošto veličina početne vrednosti određuje broj mogućih kombinacija, generator će posle završetka ciklusa da proizvede ponovo iste binarne sekvence⁽⁵⁾. Od veličine ulazne početne vrednosti zavisi i periodičnost dobijene izlazne binarne sekvence odnosno veličina ciklusa ponavljanja. Za upotrebu *PRN* generatora u kriptološke svrhe ova karakteristika predstavlja veliki problem. Napadač bi mogao, sa dovoljno

velikom kompjuterskom moći, da pokuša otkriti koja je početna vrednost sa detaljnom kriptanalitičkom pretragom.

Kako kvalitet generisanih binarnih sekvenci *PRN* generatora zavisi od upotrebljenog algoritma, danas postoji veliki broj različitih implementacija ovih generatora. Neki programski jezici imaju svoju implementaciju koja se koristi u funkcijama za autentifikaciju i zaštitu podataka prilikom razvoja aplikacija.

2.2.2.1 *PRN* generatori u PHP programskom jeziku

PHP programski jezik poseduje set funkcija za generisanje slučajnih brojeva. To su funkcije *rand()* i *mt_rand()* kao i funkcija *openssl_pseudo_random_bytes()* iz paketa OpenSSL. Prve dve funkcije su prosti pseudoslučajni generatori koji generišu slučajne brojeve u zadatom rasponu od minimuma do maksimuma. Razlika između *rand()* i *mt_rand()* je u algoritmu kao i u tome da funkciji *mt_rand()* možemo odrediti početnu vrednost. Takođe, *rand()* funkcija se zasniva na generatoru koji je deo *libc* biblioteke, dok je *mt_rand()* funkcija zasnovana na *Mersenne Twister* algoritmu za generisanje pseudo slučajnih brojeva i 4 puta je brža od *rand()* funkcije a periodičnost je izrazito smanjena. Na slici 2 je prikazan rezultat *rand()* funkcije i jasno se može uočiti periodičnost sekvence generisanih binarnih vrednosti.



Slika 2 - Grafička prezentacija rezultata PHP funkcije *rand()*

Ove dve funkcije zbog loših osobina se ne preporučuju za korišćenje u kriptološke svrhe. *openssl_pseudo_random_bytes()* funkcija je kreirana da zadovolji potrebe za kriptološki sigurnim generisanim brojevima u PHP okruženju. Ova funkcija generiše pseudoslučajni niz bajtova u vidu niza karaktera. Ima zadovoljavajuće karakteristike i

može se primeniti za generisanje kriptoloških ključeva i inicijalnih vektora tako i za šifrovanje toka podataka.

Od verzije 7 programski jezik PHP je bogatiji za dve nove funkcije zadužene za generisanje sigurnih kriptoloških slučajnih sekvenci. To su funkcije *random_bytes()* i *random_int()*. Ove dve funkcije koriste drugačije algoritme od prethodnih, nezavisne su od operativnog sistema i daju dobre rezultate na testovima za ispitivanje slučajno generisanih brojeva.

2.2.2.2 PRN generatori u Java programskom jeziku

Java programski jezik takođe poseduje osnovni generator slučajnih brojeva koji se realizuje kroz upotrebu klase *Random* iz paketa *java.util* kao i funkciju *random()* koja se nalazi u klasi *java.lang.Math*, ali ove funkcije nisu pogodne za upotrebu u kriptografske svrhe jer se koriste deterministički algoritmi.

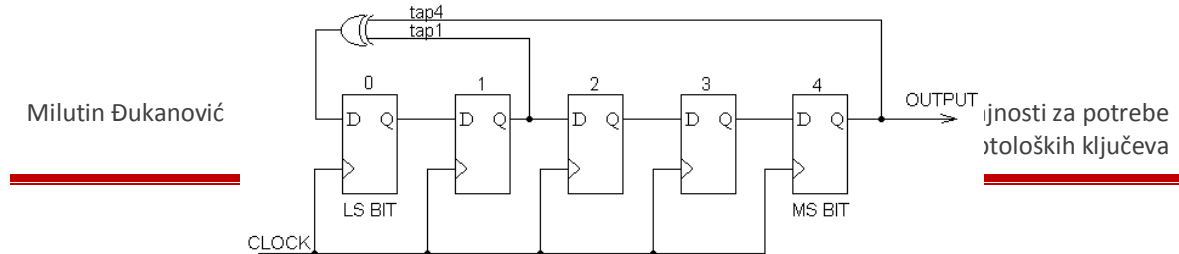
JCA ekstenzija u programskom jeziku Java uključuje klasu *SecureRandom* koja je predviđena da obezbedi kvalitetne pseudoslučajne binarne sekvence za kriptografske aplikacije. Ova klasa zapravo implementira algoritam koji na osnovu inicijalnog stanja, koji se prosleđuje instanci klase pri kreiranju objekta klase, generiše pseudoslučajne sekvence određenih tipova podataka. Za inicijalno stanje *SecureRandom* klase koristi se uobičajeno *SHA1PRNG* pseudo generator slučajnih brojeva *SUN* provajdera. U Linux operativnim sistemima moguće je koristiti *NativePRNG* pseudo generator koji se koristi */dev/random* i */dev/urandom* vrednostima za početno stanje i nema funkciju ukoliko ovi fajlovi ne postoje. Većina Java aplikacija koje koriste *SecureRandom* klasu, za generisanje slučajnih nizova koristi *SHA1PRNG* algoritam [2].

SecureRandom klasa nije predviđena za generisanje velike količine pseudoslučajnih binarnih sekvenci niti ima izuzetne performase u pogledu brzine, ali je sasvim dovoljna za generisanje sigurnih kriptoloških ključeva. Mana ove klase je što je zavisna od operativnog sistema i na Linux platformama može doći do blokiranja njenog rada usled sistemskog nedostatka entropije, što ponekad u blokadama dovodi do zastoja u generisanju sekvenci izraženog čak i u minutama.

2.2.2.3 Drugi PRN generatori

Pored navedenih proširenja programskih jezika za generisanje pseudoslučajnih binarnih sekvenci postoje i rešenja koja se mogu upotrebiti u skoro svakom programskom jeziku i nalaze svoju upotrebnu vrednost u različitim aplikacijama.

Jedan od primera takvog generatora pseudoslučajnih binarnih sekvenci je i generatora sa pomeračkim registrima sa linearnom povratnom spregom (eng. *Linear Feedback Shift Registers - LFSR*). Ovaj generator je veoma jednostavan i efikasan ali ne pruža dovoljnu sigurnost u pogledu upotrebe u kriptološke svrhe. Na slici 3 je prikazan diagram realizacije generatora sa 5-bitnim pomeračkim registrom a povratna veza se ostvaruje preko uzimanja vrednosti sa dva bita registra i XOR-ovanjem njihovih vrednosti koja se zatim vodi na prvi bit kao ulazna vrednost. Dužina niza koji ovaj generator može da generiše je maksimalne vrednosti $2^n - 1$ ali realna dužina niza zavisi od veličine registra kao i od inicijalnog stanja registra.



Nedostatak ovakvih generatora pseudoslučajnih binarnih sekvenci se ogleda u tome da ih je moguće dovesti u kružno stanje promene vrednosti kao i u stanje blokade kada svi bitovi registra dobiju stanje 0.

Za generisanje pseudoslučajnih brojeva mogu se koristiti neki šifarski algoritmi. PRN generator ANSI X9.31 je baziran na upotrebi AES i DES šifarskih algoritama. U varijanti sa AES šifarskim algoritmom ovaj generator koristi 128-bitni AES tajni ključ, 128-bitni vremenski vektor i početnu vrednost koja treba da bude takođe dužine 128 bita. Pri svakoj iteraciji početna vrednost se regeneriše. Preporuku za način njihove upotrebe

Slika 3 - LFSR pseudoslučajni generator sa 5-bitnim registrom

objavio je i NIST u svom dokumentu [12].

2.2.3 Poređenje TRN i PRN generatora

Po svojim karakteristikama TRN i PRN generatori se veoma razlikuju. Sa druge strane, u realnim kriptografskim rešenjima generatora slučajnih brojeva ove dve vrste generatora se često koriste zajedno u svrhu poboljšanja kvaliteta generisanih binarnih sekvenci. TRN generatori su nedeterministički izvori slučajnosti, što je njihova glavna prednost u odnosu na PRN generatore. Prednost PRN generatora u odnosu na TRN generatore se ogleda u njihovoj efikasnosti. Oni mnogo brže generišu binarne sekvence i u mogućnosti su da bez problema generišu velike količine slučajnih vrednosti što nije slučaj kod TRN generatora koji zbog prirode izvora slučajnosti i potrebnih konverzija fizičkih veličina u digitalne nisu u stanju da konkurišu PRN generatorima. Takođe, periodičnost je karakteristika PRN generatora i ona može narušiti bezbednosne aspekte generisanih binarnih sekvenci i omogućiti kriptanalitičarima predviđanje izlaznih vrednosti. TRN generatori nemaju periodičnost ili je ona neprimetno izražena i zbog toga su otporni na analizu.

U savremenim kriptografskim aplikacijama potreba da se generišu što kvalitetniji ključevi, inicijalni vektori i sertifikati, dovela je do kombinovanja ove dve vrste generatora sa ciljem da se od svake iskoriste dobre osobine. Tako se za upotrebu PRN generatora u kriptološke svrhe koriste inicijalne vrednosti generisane TRN generatorom i time povećava entropija izlaznih binarnih sekvenci a takođe i sprečava eventualni napadač da uz pomoć periodičnosti regeneriše početnu vrednost.

Loša efikasnost TRN generatora jedna od mana ovih generatora koja se ne može jednostavno rešiti zbog prirode izvora. Kombinacijom izlaznih sekvenci TRN generatora i algoritama koji su sastavni deo PRN generatora u mogućnosti smo da na izlazu dobijemo duže binarne sekvence od onih koje dobijamo na izvoru slučajnosti. Na isti način se popravljaju još jedna loša karakteristika TRN generatora a to je neotpornost na promene i ometanja.

2.2.4 Sopstveni izvor slučajnosti

Pod sopstvenim generatorom slučajnih vrednosti smatramo takav sistem u kome svi delovi generatora (izvor, algoritmi, postdigitalna obrada) čine jednu celinu, nezavisnu od trećeg lica koje bi na bilo koji način moglo da kompromituje neki od delova sistema. Primer za sistem koji ne zadovoljava uslove sopstvenog izvora slučajnosti bi bila upotreba pseudogeneratora slučajnih brojeva koji za početnu vrednost uzima binarne sekvence generisane od strane Random.org generatora istinski slučajnih brojeva koji se nalazi na udaljenom serveru. Svaka neispravnost servera, posedovanje znanja o izdatim slučajnim sekvencama kao i ometanja u informacionom kanalu mogu ozbiljno ugroziti bezbednost generisanih binarnih sekvenci od strane PRN generatora a na taj način i bezbednost generisanih kriptoloških ključeva.

Da bi sopstveni generator slučajnih vrednosti bio nezavisan, izvor slučajnosti mora biti u sklopu istog sistema. U ovom radu opisano je rešenje u kome je izvor slučajnosti sastavni deo prikazanog generatora slučajnih vrednosti.

2.3 Provajderi kriptografskih funkcionalnosti

U okviru Java programskog jezika nalaze se dva proširenja koja u sebi sadrže sve kriptografske funkcije. To su *JCA* (eng. *Java Cryptography Architecture*) i *JCE* (eng. *Java Cryptography Extension*) biblioteke. Za oba ova proširenja zajedničke su karakteristike algoritamska nezavisnost, algoritamska proširivost, nezavisna implementacija i interoperabilnost [13]. *JCA* predstavlja radno okruženje za korišćenje i razvoj kriptografskih funkcionalnosti u Java programskom jeziku. Karakteristika ovog radnog okruženja je upotreba arhitekture koja se zasniva na provajderima kriptografskih funkcionalnosti. Ovim je postignuta nezavisna implementacija koja obezbeđuje korišćenje različitih kriptografskih servisa. *JCE* proširuje *JCA API* sa dodatnim klasama i provajderima, funkcijama za šifrovanje i razmenu i generisanje kriptoloških ključeva. Provajderi su setovi paketa koji implementiraju konkretne kriptološke algoritme.

2.3.1 JCA kriptološko proširenje

JCA je Java proširenje koje je standardno integrisano u jezgru JAVA API-ja. *JCA* se nalazi u okviru paketa *java.security*. Funkcionalnost ovog proširenja se ogleda u mogućnosti upotrebe sa različitim nivoima interakcije između aplikacije i mehanizma *JCA*. Na primer, programer u svojoj aplikaciji može jednostavno zahtevati od *JCA* bilo koju *hash* funkciju ili u zahtevu eksplicitno navesti određenu hash funkciju određenog provajdera kriptoloških funkcija. Prva karakteristika vodilja pri dizajniranju *JCA* ekstenzije je bila algoritamska nezavisnost koja predstavlja mehanizam za klasifikaciju kriptografskih operacija u kategorije. *JCA* ove kategorije predstavlja kao Java klase i funkcionalnosti. *JCA* sadrži sledeće klase i funkcionalnosti:

- *MessageDigest* – generiše hash vrednost za datu poruku
- *Signature* – generiše digitalni potpis za dati dokument

- *KeyPairGenerator* – generiše par ključeva (PKI) sa kojima se, na primer, može potpisati neki dokument
- *KeyFactory* – konvertuje ključeve u specifikacije ključeva i obatno
- *KeyStore* – obezbeđuje upravljanje i čuvanje ključeva
- *SecureRandom* – generiše slučajne brojeve prihvatljive za kriptografsku upotrebu
- *AlgorithmParameters* – upravlja enkodovanjem i dekodovanjem parametara za dati kriptografski algoritam
- *AlgorithmParameterGenerator* – generiše kompletan set parametara za dati kriptografski algoritam
- *CertificateFactory* – generiše digitalne sertifikate
- *CertPathBuilder* – realizuje lančane veze između sertifikata
- *CertStore* – upravlja i čuva digitalne sertifikate

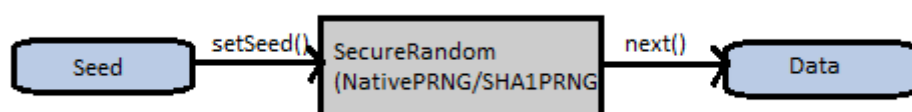
2.3.2 JCE kriptološko proširenje

JCE je proširenje *JCA* kriptološkog paketa i obezbeđuje više klasa kao i dodatnog provajdera *SunJCE*. *JCE* svoje klase drži u drugom paketu, *javax.crypto*[13]. *JCE* sadrži sledeće klase i funkcionalnosti:

- *Cipher* – realizuje operacije sifrovanja i dešifrovanja
- *KeyGenerator* – generiše tajne ključeve
- *SecretKeyFactory* – kao u *JCA* paketu ali samo nad instancom *SecretKey*
- *KeyAgreement* – predstavlja protokol za usaglašavanje i razmenu ključeva između više strana
- *Mac* – predstavlja funkcionalnost autentičnosti poruke

2.3.3 Podržani izvori slučajnosti

U okviru *JCA* proširenja za generisanje pseudoslučajnih brojeva se koristi klasa *SecureRandom* koja generiše objekat *CSPRNG* (eng. *Cryptographically Secure Pseudo-Random Number Generator*). Za inicijalno stanje unutrašnjeg algoritma koristi se istinski slučajni generator koji generiše potrebni binarni niz. Pretežno se koristi *SHA1PRNG* generator Sun provajdera.



Slika 4 – Dijagram CSPRN generatora u Javi

SHA1PRNG algoritam predstavlja pseudorandom generator koji generisani binarni niz daje izračunavajući *SHA-1* hash vrednost nad istinski slučajnom početnom vrednošću koja se povezuje sa 64-bitnim brojačem čija se vrednost povećava za 1 pri svakoj iteraciji. Od 160 bita koliko se dobija na izlazu *SHA-1* hash funkcije koristi se samo 64 bita. Za konstruisanje novog generatora pseudoslučajnih brojeva u *JCA* paketu potrebno je instancirati klasu *SecureRandom* kao parametar proslediti *SHA1PRNG*. U većini slučajeva *SHA1PRNG* početne vrednosti uzima iz operativnog sistema i veoma je mali potrošač resursa jer mu je potrebno svega nekoliko sistemskih poziva.

Sa druge strane, pseudo generator *NativePRNG* više koristi resurse sistema i pretežno je u upotrebi na Linux operativnim sistemima a za izvor slučajnosti koristi */dev/random* i */dev/urandom* sistemski generisane entropije. Razlika između ova dva generatora se ogleda u tome što *NativePRNG* konstantno dobija entropiju od izvora odnosno operativnog sistema dok *SHA1PRNG* dobija samo početnu vrednost.

2.3.3.1 Primer upotrebe generatora SHA1PRNG

```
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;

public class GeneratorRandomBrojeva {

    public static void main(String[] args) {

        try {

            // Kreiranje objekta pseudoslučajnog generatora
            // saSHA1PRNG algoritmom
            SecureRandom RGenerator = SecureRandom.getInstance("SHA1PRNG");

            // Generisanje 128 random byta
            byte[] randomBytes = new byte[128];
            RGenerator.nextBytes(randomBytes);

            // Generisanje i upotreba početne vrednosti
            int seedByteNumber = 5;
            byte[] seed = RGenerator.generateSeed(seedByteNumber);

            SecureRandom Rand = SecureRandom.getInstance("SHA1PRNG");
            Rand.setSeed(seed);

        } catch (NoSuchAlgorithmException e) {

        }

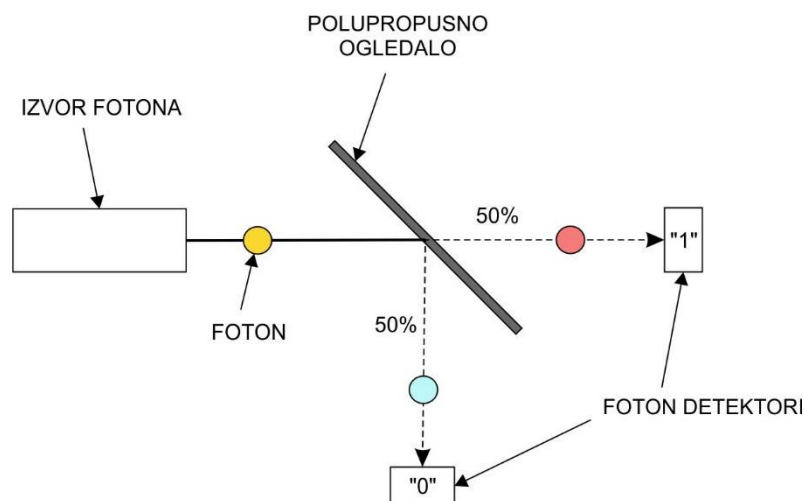
    }

}
```

3. Teorijske osnove

3.1 Šta je izvor slučajnosti?

Fizičke pojave a posebno pojave u kvantnoj fizici su idealni izvori slučajnosti. Kvantna fizika je jedina oblast fizike koja u velikoj meri integriše slučajnost. Pri projektovanju generatora slučajnih brojeva kao izvor slučajnosti se može koristiti neki jednostavan proces u kvantnoj fizici. Neki fizički proces se može jednostavno modelivati i pratiti njegovo funkcionisanje radi utvrđivanja pravilnog generisanja slučajnih sekvenci. Primer korišćenja procesa u kvantnoj fizici je ponašanje fotona, čestica od kojih se sastoji svetlost, tačnije od njihovog slučajnog ponašanja u određenim situacijama [17]. Na slici 5 je prikazan optički izvor slučajnosti koji koristi tu pojavu u kvantnoj fizici.



Slika 5 – Optički izvor slučajnosti na osnovu ponašanja fotona

Za neku pojavu ili događaj se kaže da je slučajan ukoliko nismo u mogućnosti da sa sigurnošću odredimo naredni proizvod te pojave ili događaja. Za jedan objekat u nizu možemo reći da je slučajan ukoliko ima najveću složenost koja je moguća u datom nizu. Niz je slučajan ukoliko se algoritamski ne može kompresovati niti svoditi na prostije oblike.

Jedan od načina da odredimo količinu slučajnosti, ili neodređenosti, izvora je da izračunamo entropiju generisanog binarnog niza. Entropija je veličina koja opisuje nedređenost izvora i može se definisati kao količina informacije koju generiše izvor. Idealna vrednost entropije je 1, odnosno, kada jedan generisani bit sadrži jedan bit informacije. Ukoliko jedan izvor generiše binarne sekvence i ako uzmemo u obzir da postoje dve vrednosti izlaza izvora predstavljene kao 0 i 1, tada imamo verovatnoću pojavljivanja ovih stanja kao $P(0)$ i $P(1)$. Da bi smo pojednostavili izračunavanje

predstavljamo verovatnoću $P(0)$ kao p , a verovatnoću $P(1)$ kao $1-p$. Vrednost entropije u tom slučaju možemo dobiti sledećim izrazom:

$$H = -p \log p - (1 - p) \log (1 - p)$$

U slučaju TRN generatora vrednost entropije predstavlja veoma važnu komponentu i s obzirom da je kod ovih generatora izvor slučajnosti pretežno neka fizička pojava, entropija uobičajeno ima visoku vrednost. Kod PRN generatora entropija izlazne binarne sekvence isključivo zavisi od entropije početne vrednosti koja bi trebalo da bude generisana pomoću TRN generatora.

Kada koristimo izvor slučajnosti za realizaciju generatora slučajnih brojeva, podrazumevamo da je taj izvor savršeno slučajan sa generisanim bitovima koji su potpuno nezavisni i bez korelacije među njima. To uvek nije slučaj i dešava se da usled ometanja izvora ili drugih anomalija izlazna vrednost nije uvek zadovoljavajuća. Da bi se takvi problemi rešili koriste se *ekstraktori slučajnosti*, algoritmi koji optimizuju izlazne vrednosti izvora i ispravljaju njihove nedostatke[14].

Ekstraktori slučajnosti su efikasni algoritmi koji popravljaju karakteristike slabih izvora slučajnosti. Ekstraktori slučajnosti se koriste pre svega da poprave uniformnu raspodelu generisanih slučajnih vrednosti izvora. Svojom funkcionalnošću ekstraktori pomažu da se iz slabih izvora neodređenosti, ili u momentima kada dobri izvori dolaze u stanje anomalija, dobiju najbolji mogući rezultati. Jedan od primera ekstraktora slučajnosti je i *Von Neumann-ovalgoritam* za destilaciju koji se zasniva na uklanjanju grešaka na osnovu mapiranja prelaza. Princip rada algoritma: podeliti binarnu sekvencu izvora na parove, za svaki par 01 na izlazu ekstraktora generisati 0, za par 10 generisati 1 a parove 00 i 11 ukloniti iz binarne sekvence [15]. Ovakav princip obrade binarne sekvence izvora daje bolju uniformnu raspodelu slučajnih vrednosti ali smanjuje količinu generisanih slučajnih binarnih sekvenci.

Postoje tri osnovna parametra ekstraktora slučajnosti. Prvi je dužina binarne sekvence izvora koja se uvodi u algoritam. Drugi je dužina izlazne sekvence koja teži da bude jednaka ulaznoj, a treći je smanjenje statističkog odstupanja izlaza ekstraktora od uniformne raspodele.

3.2 Kodovanje izvora slučajnosti

Svaki izvor slučajnosti generiše vrednosti ili neke događaje vezane za jednu slučajnu vrednost i ukoliko posmatramo promene neodređenosti možemo izvesti zaključke o stanju slučajne vrednosti. Na taj način dobijamo količinu informacije koju proizvodi izvor slučajnosti. Količina informacije i entropija su usko povezani pojmovi. *Claude E. Shannon* je u svom radu (1948) naveo da postoji takav odnos između količine informacije i entropije izvora koji se ogleda u limitu stepena kompresije informacije bez gubitaka koji ne može biti veći od entropije samog izvora.

Kao jedan od primera možemo navesti bacanje novčića. Ako uzmemo u obzir da rezultat može biti samo jedno od dva stanja, ili 0 ili 1, onda je količina informacija posle svakog bacanja tačno jedan bit. Posle 100 bacanja novčića imaćemo 100 bita informacije koja opisuje sekvencu bacanja novčića 100 puta. Sa druge strane, ukoliko imamo novčić koji na obe strane ima istu vrednost i rezultat bacanja će biti uvek isti, posle 100 bacanja novčića količina informacije se može opisati samo jednim bitom. Uzmimo sada kao primer slučaj da imamo novčić koji je teži na jednoj strani i da posle niza bacanja jedna strana se pojavljuje u pretežno 75% slučajeva. Tada na osnovu formule za izračunavanje entropije dobijamo sledeći iskaz: $-(0.75\log 0.75 + 0.25\log 0.25) = 0.8113$, što znači da nam je za opis informacije dobijene posle 100 bacanja ovakvog novčića potrebno 82 bita.

Bez obzira na to da li koristimo izvor slučajnosti kao osnovni generator slučajnih vrednosti kod TRN generatora ili njegove sekvence koristimo samo kao početne vrednosti PRN generatora potrebno je vrednosti izvora obraditi i povećati kvalitet dobijenih sekvenci, odstraniti duge nizove istovetnih vrednosti nastale usled anomalija izvora kao i povećati bezbednost izlaznih sekvenci. Za neki binarni generator slučajnih vrednosti se kaže da poseduje devijacije u izlaznoj sekvenci kada verovatnoća jednog ishoda nije jednaka verovatnoći drugog. Ova devijacija se pojavljuje usled nemogućnosti da se realizuje precizno izbalansiran fizički proces. U tu svrhu je potrebno izvršiti kodovanje sirovih informacija sa izvora i post-digitalna obrada dobijenih binarnih sekvenci [7]. Kodovanje u većini slučajeva obezbeđuje kvalitetnije sekvence ali zbog konstrukcije algoritama obrađene sekvence su manje dužine zbog uklanjanja redundantnih bita koji u informaciji nemaju vrednost.

Von Neumann-ova procedura, prethodno opisana, je jedan od načina sprovođenja post-digitalne obrade binarnih sekvenci izvora ali njena primena na sirovu sekvencu izvora dovodi do toga da će proizvedena dužina nedevijantnog niza biti oko 25% od ukupne dužine sirovog binarnog niza.

Kompresija binarnih sekvenci može povećati jačinu i kvalitet izvora slučajnosti a sa tim i jačinu generatora slučajnih brojeva koji koriste taj izvor. Kada se govori o kompresiji uobičajeno je shvatanje da kompresija služi da bi se jedna ulazna sekvenca transformisala u kraću izlaznu sekvencu sa potpuno istom količinom informacije tako da se može iz nje dekompresijom povratiti originalna ulazna informacija [3]. Pri korišćenju kompresije za kodovanje binarnih sekvenci izvora slučajnosti svrha upotrebe je drugačija. Izlazna sekvenca izvora slučajnosti može posedovati slabosti koje nastaju usled devijacije ili determinističkih svojstava procesa koji se koriste. Cilj kompresije je suprotan kompresiji podataka radi prenosa komunikacionim kanalima i ogleda se u tome da se na izlazu isporuči sekvenca koja će dati što manje informacija o ulaznoj sekvenci a da pri tome ostane što duža.

Kompleksnost binarnih sekvenci slučajnog izvora usled devijacija na izvoru može varirati a sa tim i rezultati testova nad sekvencama. Prilikom generisanja kriptoloških ključeva veoma je bitno odabrati binarne sekvence koje imaju veliku kompleksnost. Kolmogorova kompleksnost nam daje mogućnost da dobijemo informaciju o kompleksnosti nekog binarnog niza i da analiziramo prisustvo šablonskih podnizova. Kolmogorova kompleksnost binarnog niza predstavlja broj bita koji su potrebni računarskim resursima da bi se taj niz opisao [16]. Pretpostavimo da imamo tri binarna niza svaki dužine 100 bita:

```

11111111.....1111
10101010.....1010
1011011010.....0111

```

ukoliko ove nizove analiziramo uočićemo da u prva dva niza postoje šablonski podnizovi dok u trećem nizu ih nema. Kompleksnost prvog niza je najmanja moguća i ona se može opisati sa 1 bitom (naravno, ukoliko se prethodno zna dužina niza od 100 bita). Ako drugi niz podelimo u podnizove uočićemo šablonski podniz koji se ponavlja i njega možemo opisati sa „50 puta 01“, dok treći niz je potrebno podeliti u podnizove opisati ga što sa većim brojem podnizova povećava kompleksnost binarnog niza. Podnizovi binarnog niza se izdvajaju u n-grame. Što je veći broj n-grama to je veća i kompleksnost generisanih binarnih sekvenci a od nje zavisi i upotreba za generisanje kriptoloških ključeva.

Korišćenjem procesa postdigitalne obrade binarnih sekvenci izvora slučajnosti dobija se bolja entropija izvora kao i kvalitetniji binarni nizovi koji su otporniji na korelaciju i zavisnost izazvanu devijacijama hardverskog izvora.

3.3 Generisanje ključeva za simetrične i asimetrične algoritme

Dugi niz godina dužina ključa je bila limitirana na 40 bita. Danas ključevi od 40 bita predstavljaju slabu zaštitu od napada. Međutim, i dalje su na snazi u USA zakoni koji ne dozvoljavaju izvoz kriptoloških proizvoda čiji ključevi prelaze 64 bita.

Simetrični kriptološki algoritmi zahtevaju tajni ključ, koji se koristi kako za proces šifrovanja podataka tako i za dešifrovanje. JCA arhitektura poseduje klasu *javax.crypto.KeyGenerator* koja je odgovorna za generisanje tajnih ključeva za simetrične kriptološke algoritme. Nakon kreiranja instance generatora ključeva potrebno je inicijalizovati generator sa potrebnom veličinom ključa a zatim pokrenuti metodu za generisanje. Pregled dužine ključeva za simetrične algoritme:

Simetrični algoritam	Dužina ključa
AES	128, 192 i 256 bita
RC4	40 – 1024 bita
Blowfish	128 bita
DES	56 bita
DESede	168 bita
HmacMD5	512 bita
HmacSHA1 HmacSHA256 HmacSHA384 HmacSHA512	256, 384 i 512 bita

Tabela 1 – Dužine ključeva za simetrične kriptološke algoritme

Za potrebe asimetričnih šifarskih algoritama, konkretno RSA algoritma generišu se po dva ključa za svaku stranu koja učestvuje u komunikaciji, po jedan javni i jedan privatni ključ. Princip generisanja ključeva je sledeći:

- Izabrati dva različita velika prosta broja p i q od 512 bita
- Izračunati n koje ima vrednost proizvoda p i q
- Izabrati mali neparan celi broj e uzajamno prost sa vrednošću Eulerove funkcije $\phi(n)$, koja za n iznosi $(p - 1)(q - 1)$, i vrednosti $1 < e < \phi(n)$.
- Izračunati d kao inverzni element za operaciju množenja broju e , po modulu $\phi(n)$.
- Objaviti uređeni par $P = (e, n)$ kao javni RSA ključ.
- Sačuvati u tajnosti uređeni par $S = (d, n)$ kao privatni RSA ključ.

Za generisanje para ključeva za potrebe RSA algoritma u Java programskom jeziku u okviru JCA arhitekture koriste se klase `javax.crypto.KeyPair` i `javax.crypto.KeyGenerator`.

3.4 Specifikacije za generisanje simetričnih i asimetričnih ključeva

Specifikacija ključa predstavlja izlaganje metapodataka o ključu[2]. U Java programskom jeziku zadatak svakog provajder je da dokumentuje podržane specifikacije ključa. U JCA arhitekturi klasni interfejs *KeySpec* omogućava transparentni pristup ključu čime se mogu utvrditi karakteristike i kvalitet ključa za primenu u aplikaciji.

Ako se želi specifikacija ključa nezavisno od provajdera u JCA arhitekturi se može koristiti klasa *SecretKeySpec* ali to je moguće samo za simetrične šifre koje koriste jedan niz bajtova.

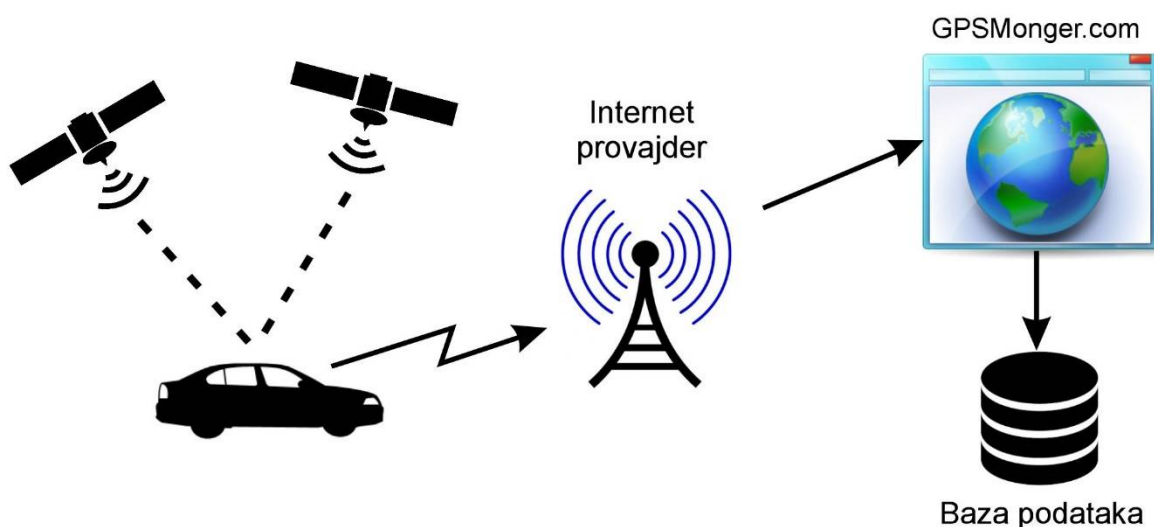
Nekad zbog jednostavnosti rukovanja odričemo se mogućnosti da imamo pregled metapodataka ključa što nije dobra praksa.

4. Eksperimentalni rad

4.1 Opis eksperimentalnog okruženja

Izvor slučajnosti na kome je baziran ovaj rad predstavlja podatke dobijene u sistemu GPS praćenja vozila na osnovu njihove lokacije. Aplikacija za praćenje vozila, GPSTracker.com koristi posebne uređaje koji se nalaze u vozilima i na osnovu podataka dobijenih putem internet mreže određuje njihov trenutni položaj. Podaci se radi analize i pregleda čuvaju u bazi podataka na istom serveru gde je postavljena i aplikacija. Princip rada aplikacije GPSTracker.com prikazan je na slici 6 i sastoji se u sledećem:

- Uređaj za GPS praćenje putem signala sa geostacionarnih satelita određuje svoj položaj na osnovu podataka o geografskoj širini i dužini
- Prikupljeni podaci se konvertuju u standardni GPS format
- Putem internet mreže uređaj ostvaruje konekciju na *web-socket* server aplikacije GPSTracker.com i tim kanalom šalje paket informacija o svom trenutnom položaju svakih 5 sekundi
- Aplikacija GPSTracker.com dobijeni paket informacija konvertuje u adekvatan format za čuvanje u bazi podataka zajedno sa identifikacionim parametrima koji se odnose na vlasnika uređaja, vozilo i sam uređaj
- Na web stranici aplikacije autorizovani korisnici imaju mogućnost pregleda trenutnog položaja vozila kao i uvid u podatke u bazi o prethodnim stanjima i putanjama vozila



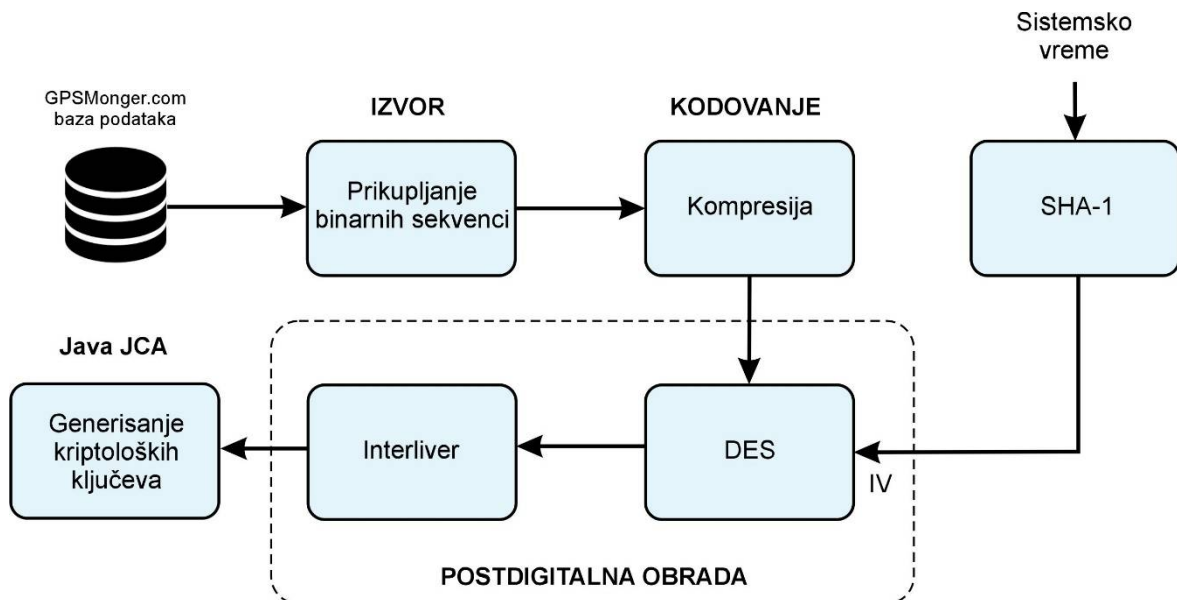
Slika 6 – Princip rada aplikacije za praćenje vozila GPSTracker.com

Podaci o položaju vozila dobijeni putem interneta se čuvaju u posebnoj tabeli u bazi podataka. Bazi podataka je moguće pristupiti direktno putem komunikacionog kanala uz potrebna autorizaciona ovlašćenja. Podaci se u bazu podataka upisuju posredstvom *MySQL* servera koji radi paralelno sa aplikacijom. U okviru *MySQL* servera postoji posebna funkcija zadužena za obradu zahteva upućenih sa aplikacije koja je predmet ovog projekta.

Aplikacija koja predstavlja realizaciju predloženog rešenja za generator slučajnih brojeva prikazanog u ovom radu, izrađena je u sledećem okruženju:

- Programski jezik: Java
- Radno okruženje: Netbeans 8.0.2
- OS u toku razvoja: Windows 10
- OS u produkciji: Ubuntu 14.04
- Server hardware: DigitalOcean.com, 512Mb RAM, 20Gb Storage, 1 Core
- Server software: Apache Tomcat 8, Oracle Java JDK 1.8.0_45

4.2 Generička šema razvijenog rešenja

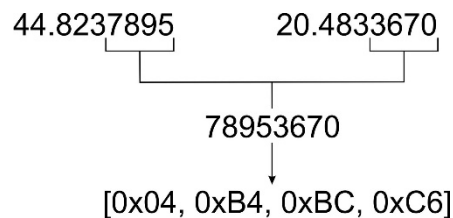


Slika 7 – Generička šema izvora slučajnih brojeva za generisanje kriptoloških ključeva

Na slici 7 prikazana je generička šema predloženog rešenja izvora slučajnih brojeva. Rešenje se sastoji od više zasebnih funkcionalnih jedinica. Svaka jedinica u blok šemi ima svoju ulogu u obradi i generisanju slučajnih binarnih sekvenci. U ovom radu ćemo uporediti dobijene rezultate binarnih sekvenci sa i bez upotrebe postdigitalne obrade. Predloženo rešenje se sastoji od sledećih blokova:

- **IZVOR** – Ovaj blok predstavlja ulaznu jedinicu aplikacije i zadužen je za maipulaciju sa podacima koji se prikupljaju iz baze podataka GPSMonger.com servisa. U sebi sadrži više klasa i to: klasu za kreiranje objekta konekcije sa bazom podataka, klasu za modelovanje i obradu ulaznih podataka kao i klasu za vremensko okidanje prikupljanja podataka. Podaci koji se prikupljaju iz baze podataka predstavljaju decimalne vrednosti geografske širine i dužine. S obzirom da se celobrojni deo decimalne vrednosti malo menja, posebno ako se primena praćenja odnosi na usko geografsko područje, za potrebe ovog izvora slučajnosti koriste se vrednosti iza decimalnog zareza, i to poslednje 4 cifre od svakog podatka. Na taj način dobijamo, pri učitavanju svake lokacije upisane u bazu, brojčanu vrednost sa 8 cifara. Prilikom eksperimentalne analize uočena je razlika u broju korišćenih cifara iz svakog podatka o lokaciji. Ukoliko izaberemo programski da koristimo veći broj cifara, i time dobijemo veću količinu slučajnih vrednosti iz iste količine upisanih podataka u bazi,

dobićemo manju entropijsku vrednost generisanih nizova. Ova pojava je rezultat jednoličnih podataka sa GPS uređaja kada vozilo stoji ili se nalazi u jako uskom radijusu kretanja. Suprotno tome kada koristimo manji broj cifara, entropija se povećava. Pored neodređenosti koju proizvode podaci o položaju vozila, na entropiju izvora utiču i drugi faktori a pre svega greške u preciznosti određivanja položaja vozila GPS uređajai varijacije u signalu sa satelita. Optimalan broj cifara za ovaj rad je postavljen na 4 po koordinati, što nam kao rezultat daje osmocifrenu vrednost. Tako dobijena vrednost se konvertuje u niz bajtova koji se smešta u bafer prikupljenih slučajnih vrednosti. Način konverzije prikazan je na slici 8. Nakon prikupljanja i konverzije svakog reda iz tabele baze podataka dobijeni nizovi bajtova se smeštaju u kolekciju tipa reda i pripremljeni su za dalju obradu. Vremenski okidač prikupljanja podataka kontrolise veličinu bafera i ukoliko je ona manja od donje granice učitava nove podatke iz baze.



Slika 8 – Konverzija prikupljenih podataka

- **KODOVANJE**– Dobijeni niz bajtova u kolekciji sadrži sirove podatke iz izvora slučajnosti. Takav niz može usled anomalija izvora imati veliku količinu redundantnih podataka. Iako servis GPSMonger.com obezbeđuje da se u bazu ne upisuju vrednosti koje ne zadovoljavaju formu lokacije vozila, veliku količinu redundantnih podataka možemo očekivati usled većeg broja približno istih lokacija. Da bi smo izbegli velike nizove nula i jedinica kao i pojavu sličnih okvira podataka, u okviru kodovanja upotrebićemo kompresioni metod. Java programski jezik poseduje klasu *Deflater* koja se oslanja na *ZLIB* biblioteku. Klasa *Deflater* kompresuje podatke na nivou bitova i zasnovana je na *Huffman*-ovom algoritmu za kodovanje podataka. Ovaj blok aplikacije radi kompresiju kompletne kolekcije bafera tako da posle kompresije kolekcija bafera poseduje kompresovani niz bajtova sa daleko boljim karakteristikama od ulaznog. Izbacivanjem redundantnih podataka popravljena je količina entropije, tako da su podaci na izlazu kompresora mnogo bolji od ulaznih.
- **POSTDIGITALNA OBRADA** – Da bi se dodatno popravio kvalitet generisane sekvence iz izvora slučajnosti često je potrebna postdigitalna obrada podataka. Ova funkcionalnost nam obezbeđuje i dodatnu zaštitu od

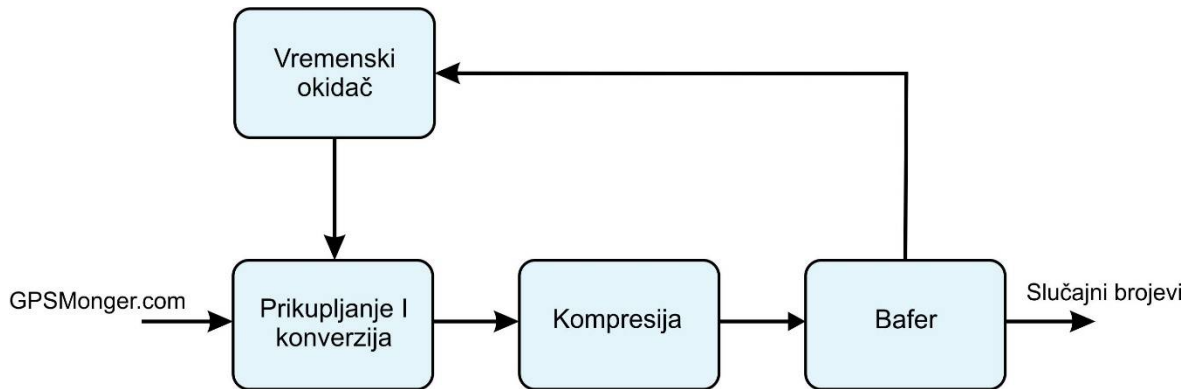
kriptoanalize i eventualnih napada sa namerom da se otkriju informacije o izvoru i primenjenih algoritmima. U tu svrhu smo upotrebili DES algoritam za šifrovanje podataka koji ima dobre karakteristike u pogledu normalne raspodele i kompleksnosti izlaznih podataka. Kao inicijalnu vrednost DES algoritma koristićemo sistemsko vreme koje smo prethodno propustili kroz algoritam za *hash*-ovanje SHA-1 i time dobili 20 bajtova od kojih za inicijalni vektor koristimo samo 8. Šifrovani podaci se zatim propuštaju kroz interliver da bi se još više poboljšala zaštita i popravile karakteristike.

- **Java JCA** – Ovaj blok predstavlja deo aplikacije u kojoj se uz pomoć Javinog proširenja JCA i kolekcije slučajnih brojeva generišu kriptografski ključevi za razne šifarske algoritme. Takođe ovaj blok može obezbediti i potrebne korisničke nizove slučajnih vrednosti za razne potrebe. S obzirom da u aplikaciji svaka procedura se obavlja na kolekciji tipa reda i rezultat obrade se vraća u istu kolekciju koja je dostupna u čitavoj aplikaciji, jasno je da aplikacija ima mogućnost proširivanja sa dodatnim modulima.

4.2.1 GPS -> Izvor -> Kodovanje -> Slučajni brojevi

Izvor slučajnih brojeva prikazan u ovom radu može biti realizovan na dva načina. Prvi način je korišćenjem izvora slučajnosti bez upotrebe postdigitalne obrade dobijenih sekvenci slučajnih vrednosti. U ovom slučaju vrednosti koje se dobijaju konverzijom GPS koordinata propuštaju se kroz blok za kompresiju, uklanjaju se redundantni podaci i obezbeđuje imunitet na anomalije izvora.

Na ovaj način neodređenost vrednosti koje se dobijaju na izlazu izvora zavisi isključivo od neodređenosti prikupljenih podataka. Vremenski okidač koji pokreće deo aplikacije za prikupljanje podataka iz baze GPSMonger.com podešen je na period od 15 sekundi i ukoliko je bafer pun ne aktivira proces preuzimanja podataka. Ukoliko se stanje bafera promeni i njegova veličina padne ispod zadate vrednosti, vremenski okidač će aktivirati klasu za prikupljanje podataka i bafer će se dopuniti. Na slici 9 je prikazan dijagram funkcionalnih delova izvora slučajnih brojeva i njihova međusobna veza, i ovo rešenje u ovom radu nazivamo šema 1.

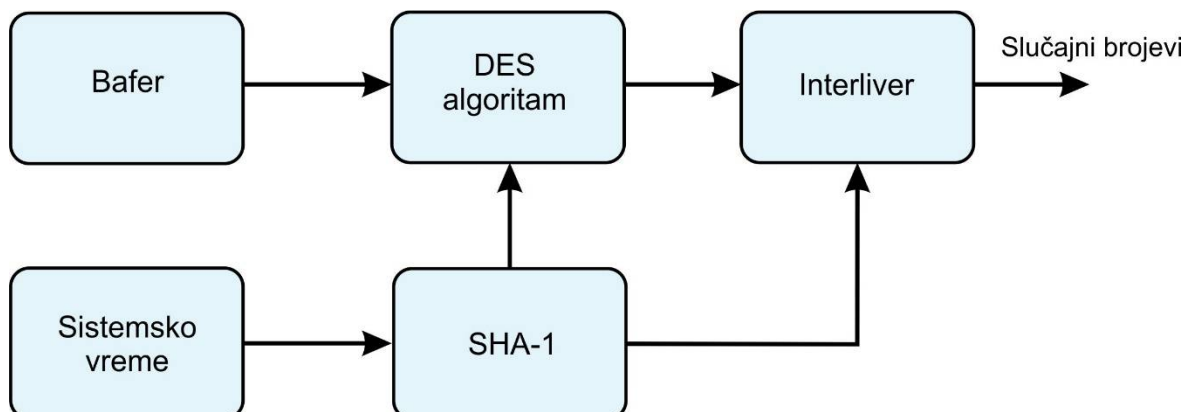


Slika 9 – Dijagram izvora slučajnih brojeva šema 1

Nakon prikupljanja i konverzije učitanih sekvenci niz bajtova se prosleđuje modulu za kompresiju koji je zasnovan na Deflater klasi. Ova klasa kompresiju podataka zasniva na upotrebi Kolmogorove kompleksnosti i Huffmanovog algoritma za kodovanje. Klasa prvo analizira niz bitova i generiše rečnik podnizova a zatim nad nizom primenjuje Huffmanovo kodovanje. Tako dobijeni niz bitova se konvertuje u niz bajtova i dodaje u bafer. Slučajni brojevi iz ovog izvora prema šemi 1 se mogu koristiti direktnim preuzimanjem iz bafera tako da se pri svakom preuzetom bajtu on briše iz bafera.

4.2.2 GPS -> Izvor -> Kodovanje -> Postdigitalna obrada -> Slučajni brojevi

Izvor slučajnosti prethodno prikazan prema šemi 1 može se dopuniti postdigitalnom obradom dobijenih sekvenci. Time povećavamo kvalitet i bolju raspodelu bigrama i trigrama unutar binarne sekvence. Takođe, postdigitalnom obradom povećavamo otpornost na kriptanalizu izvora slučajnosti kao i njegovih unutrašnjih mehanizama obrade sekvenci. Na slici 10 je prikazan dijagram proširenja rešenja šeme 1. Ovo rešenje u ovom radu nazivamo šema 2.



Slika 10 – Dijagram proširenja šeme 1 sa postdigitalnom obradom sekvenci – šema 2

Primarni deo postdigitalne obrade sekvenci dobijenih iz izvora predstavlja DES algoritam za šifrovanje. Poseban modul u aplikaciji je zadužen za obradu sekvenci pomoću ovog algoritma. DES algoritam je odabran jer ima najbolje karakteristike po pitanju manipulacije sa podacima otvorenog teksta i pre svega zbog velikih promena u izlaznoj sekvenci na osnovu malih promena na ulazu. Blok ulazne sekvence se koduje u 16 rundi transformacija i zamena sa podključem od 48 bita u svakoj rundi [1].

U ovoj šemi pored slučajnog niza koji je generisan iz izvora slučajnosti uvodimo ješ jednu vrednost koja je neodređena i sa kojom povećavamo slučajnost izvora. To je korišćenje sistemskog vremena, vrednosti koja se generiše u momentu kada se od strane korisnika uputi zahtev za izdavanjem slučajnih brojeva. Ova vrednost se hešuje sa SHA-1 algoritmom i proizvodi niz od 20 bajtova. Prvih 8 bajtova se ekstrahuje iz ovog niza i služi kao inicijalni vektor za DES algoritam sa kojim šifrujemo slučajni niz sa izvora. Za interliver je korišćen blokovski interliver sa blokom 8x8 bajtova.

4.3 Statistička analiza – rezultati

Da bi smo ustanovili da su proizvedene sekvence izvora slučajnih brojeva koji je predloženo rešenje u ovom radu, potrebno je rezultate generisanja slučajnih binarnih sekvenci podvrgnuti statističkoj analizi. Za statističku analizu slučajnih sekvenci postoje *NIST* testovi koji proveravaju da li binarne sekvence zadovoljavaju potrebnu slučajnost za upotrebu u kriptološke svrhe, posebno za generisanje kriptoloških ključeva. Za potrebe analize generisan je slučajni niz od 10000 bitova na osnovu preporuke *NIST*-a o adekvatnoj veličini uzorka.

4.3.1 Statistička analiza šema 1 (*NIST*)

4.3.1.1 Šema 1 - Ispitivanje učestanosti u nizu

Ovaj test ima za zadatak da ispita odnos jedinica i nula u binarnoj sekvenci. Potrebno je da ovaj odnos bude što više jednak, odnosno da broj jedinica i nula u nizu bude približno isti.

Parametar	Rezultat testa
P	0.4122161071716264
Broj nula	5041
Broj jedinica	4959

Tabela 2 – Šema 1- Ispitivanje učestanosti u nizu

Nakon testiranja dobijena je vrednost P i s obzirom da je $P \geq 0.01$, možemo smatrati da je binarna sekvenca slučajna.

4.3.1.2 Šema 1 - Serijski test

Ovaj test se koristi za određivanje uzastopnog ponavljanja nula i jedinica i mogućeg preklapanja n -bitnog niza u binarnoj sekvenci. U tabeli 3 su prikazani rezultati test nad ispitivanom binarnom sekvencom.

Parametar	Rezultat testa
000	1312
001	1234
010	1294
011	1201
100	1235
101	1260
110	1201
111	1263
P_1	0.10399602122148983
P_2	0.030678291247744006

Tabela 3 – Šema 1-Rezultati serijskog testa

Nakon testiranja dobijena je vrednost P i s obzirom da je $P \geq 0.01$, možemo smatrati da je binarna sekvenca slučajna.

4.3.1.3 Šema 1 – Ispitivanje uzastopnih ponavljanja

Ovaj test se sprovodi radi utvrđivanja ukupnog broja ponavljanja jedinica i nula u posmatranoj binarnoj sekvenci.

Parametar	Rezultat testa
pi	0.4959
tau	0.02
P	0.8624253292487001

Tabela 4 – Šema 1-Ispitivanje uzastopnih ponavljanja

Nakon testiranja dobijena je vrednost P i s obzirom da je $P \geq 0.01$, možemo smatrati da je binarna sekvenca slučajna.

4.3.1.4 Šema 1 – Entropija

Ovaj test nam pokazuje učestanost svih mogućih preklapajućih n-bitnih uzoraka u binarnoj sekvenci.

Parametar	Rezultat testa
Monobit	0.9999514960491492
Bigram	0.9998170231269219
Trigram	0.9996763562902342
4x4 Matrica	0.9997360661781672

Tabela 5 – Šema 1-Ispitivanje entropije

Vrednosti entropije dobijene ovim testom i prikazane u tabeli 5, zadovoljavaju zahteve NIST-a u pogledu korišćenja slučajnih binarnih sekvenci za potrebe generisanja kriptoloških ključeva.

4.3.2 Statistička analiza šema 2 (NIST)

4.3.2.1 Šema 2 - Ispitivanje učestanosti u nizu

Ovaj test ima za zadatak da ispita odnos jedinica i nula u binarnoj sekvenci. Potrebno je da ovaj odnos bude što više jednak, odnosno da broj jedinica i nula u nizu bude približno isti.

Parametar	Rezultat testa
P	0.16758664483002872
Broj nula	5069
Broj jedinica	4931

Tabela 6 – Šema 2- Ispitivanje učestanosti u nizu

Nakon testiranja dobijena je vrednost P i s obzirom da je $P \geq 0.01$, možemo smatrati da je binarna sekvenca slučajna.

4.3.2.2 Šema 2 - Serijski test

Ovaj test se koristi za određivanje uzastopnog ponavljanja nula i jedinica i mogućeg preklapanja n-bitnog niza u binarnoj sekvenci. U tabeli 7 su prikazani rezultati test nad ispitivanom binarnom sekvencom.

Parametar	Rezultat testa
000	1221
001	1254
010	1288
011	1234
100	1255
101	1268
110	1234
111	1246
P₁	0.785770983156482
P₂	0.6326739983548884

Tabela 7 – Šema 2-Rezultati serijskog testa

Nakon testiranja dobijena je vrednost P i s obzirom da je $P \geq 0.01$, možemo smatrati da je binarna sekvenca slučajna.

4.3.1.3 Šema 1 – Ispitivanje uzastopnih ponavljanja

Ovaj test se sprovodi radi utvrđivanja ukupnog broja ponavljanja jedinica i nula u posmatranoj binarnoj sekvenci.

Parametar	Rezultat testa
pi	0.5003
tau	0.02
P	0.36810092044129983

Tabela 8 – Šema 2-Ispitivanje uzastopnih ponavljanja

Nakon testiranja dobijena je vrednost P i s obzirom da je $P \geq 0.01$, možemo smatrati da je binarna sekvenca slučajna.

4.3.1.4 Šema 1 – Entropija

Ovaj test nam pokazuje učestanost svih mogućih preklapajućih n-bitnih uzoraka u binarnoj sekvenci.

Parametar	Rezultat testa
Monobit	0.999999740314877
Bigram	0.9999723396496509
Trigram	0.9999485181072572
4x4 Matrica	0.9996870304832884

Tabela 9 – Šema 2-Ispitivanje entropije

Vrednosti entropije dobijene ovim testom i prikazane u tabeli 9, zadovoljavaju zahteve NIST-a u pogledu korišćenja slučajnih binarnih sekvenci za potrebe generisanja kriptoloških ključeva.

4.3.3 Razlike performansi šeme 1 i šeme 2

Iz priloženih rezultata može se zaključiti da Šema 2 generiše kvalitetnije binarne sekvence i da je postdigitalna obrada popravila karakteristike niza. Izdvojena entropijska vrednost veća je u Šemi 2. Bolja je i raspoređenost bigrama i trigrama. Povećana entropija monobita je uzrok kompresije u procesu kodovanja

4.4 Prikaz funkcionalnosti razvijene Java aplikacije

4.4.1 Primeri koda

Klasa DESEncrypter – vrši šifrovanje ulazne binarne sekvence:

```
package org.randomix.api.encrypt;

/**
 *
 * @author Milutin Djukanovic
 */
import java.security.InvalidAlgorithmParameterException;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.security.spec.AlgorithmParameterSpec;

import java.security.Security;
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.SecretKey;
import javax.crypto.spec.IvParameterSpec;

public class DesEncrypter {

    private Cipher ecipher;
```

```
public DesEncrypter(byte[] initVector) throws NoSuchAlgorithmException,
NoSuchPaddingException, InvalidKeyException, InvalidAlgorithmParameterException {

    Security.addProvider(new org.bouncycastle.jce.provider.BouncyCastleProvider());
    AlgorithmParameterSpec algParamSpec = new IvParameterSpec(initVector);
    SecretKey key = KeyGenerator.getInstance("DES").generateKey();

    ecipher = Cipher.getInstance("DES/CFB/PKCS5Padding");
    ecipher.init(Cipher.ENCRYPT_MODE, key, algParamSpec);
}

private byte[] encrypt(byte[] data) throws Exception {
    return ecipher.doFinal(data);
}

public byte[] getEncrypted(byte[] data) throws Exception {
    return encrypt(data);
}
}
```

Klasa GPSPongerDbConnection – Generisanje konekcije na bazu GPSPonger.com:

```
package org.randomix.api.dao;

import java.sql.Connection;
import javax.annotation.Resource;
import javax.naming.*;
import javax.sql.*;

/**
 *
 * @author Milutin Djukanovic
 */
public class GPSPongerDbConnection {

    @Resource(name="jdbc/randomix")
    private DataSource ds = null;

    /* Metod za generisanje DataSource objekta */
    public GPSPongerDbConnection() throws Exception{

        this.generateDS();
    }
}
```

```
    }

    private void generateDS(){

        try {
            Context ctx = new InitialContext();
            ds = (DataSource)ctx.lookup("java:comp/env/jdbc/randomix");
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }

    /* Metod za generisanje objekta konekcije */
    private Connection MySqlConnection(){

        Connection conn = null;
        try {
            conn = ds.getConnection();
        }
        catch (Exception e) {
            e.printStackTrace();
        }

        return conn;
    }

    /* Getter objekta konekcije */
    public Connection getMysqlConn(){
        return MySqlConnection();
    }
}
```

Klasa ExternalDataModel – učitavanje podataka iz baze, konverzija i kompresovanje:

```
package org.randomix.api.model;

import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.util.LinkedList;
import java.util.Queue;
import java.util.zip.Deflater;
import javax.xml.bind.DatatypeConverter;
import org.apache.commons.lang3.ArrayUtils;
```

```
/**
 *
 * @author Milutin Djukanovic
 */
public class ExternalDataModel extends MainModel {

    private static Queue<Byte> seedBuffer = new LinkedList<Byte>();
    private byte[] compressedData;
    private byte[] sourceData;

    public Boolean loadSeeds() {

        try {
            // citanje podataka iz baze GPSMongera
            this.openConnection("CALL z_sel_data()");

            String completeRowData, la, lo;

            //Konverzija podataka izvora u binarni string
            while (this.rst.next()) {
                la = this.rst.getString("latitude");
                lo = this.rst.getString("longitude");
                if (la.length() < 10 || lo.length() < 10) {
                    continue;
                }
                completeRowData = la.split("\\.")[1].substring(2);
                completeRowData += lo.split("\\.")[1].substring(2);

                long d = Long.parseLong(completeRowData);

                String s;
                if (Long.toHexString(d).length() % 2 == 1) {
                    s = "0" + Long.toHexString(d);
                } else {
                    s = Long.toHexString(d);
                }

                if(s.length()>6){
                    s = s.substring(2);
                }

                byte[] b = DatatypeConverter.parseHexBinary(s);
                this.sourceData = ArrayUtils.addAll(this.sourceData, b);
            }
            // Kompresija binarnog stringa
```

```
        Deflater compressor = new Deflater();
        compressor.setLevel(Deflater.BEST_COMPRESSION);

        compressor.setInput(this.sourceData);
        compressor.finish();

        ByteArrayOutputStream bos = new
ByteArrayOutputStream(this.sourceData.length);

        byte[] buf = new byte[1024];
        while (!compressor.finished()) {
            int count = compressor.deflate(buf);
            bos.write(buf, 0, count);
        }
        try {
            bos.close();
        } catch (IOException e) {
        }

        this.compressedData = bos.toByteArray();

    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        this.close();
    }

    for (byte x : compressedData) {
        ExternalDataModel.seedBuffer.add(x);
    }

    return true;
}

public static Queue<Byte> getSeedBuffer() {
    return seedBuffer;
}

}
```

5. Zaključak

Zadatak u ovom radu je bio realizovati kvalitetan izvor slučajnih brojeva što je i potvrđeno primenjenim testovima na rezultate generatora. Izvor je realizovan na osnovu ideje o korišćenju geografskih položaja vozila u pokretu. Rezultati pokazuju da je dobijen dovoljan kvalitet slučajnih binarnih sekvenci za primenu u kriptografiji, posebno za generisanje kriptografskih ključeva što je i bio cilj ovog rada.

Prednosti ovog rešenja u odnosu na postojeća ogledaju se u tome da za izvor nije korišćen hardver koji iziskuje dodatna sredstva za realizaciju konverzije fizičke pojave u binarne sekvence, već je korišćeno postojeće rešenje druge namene. Sistem se lako implementira u postojeće kriptološke sisteme i može biti integrisan u istom hardverskom okruženju što povećava bezbednost već postojećeg kriptološkog sistema.

Korisnici ovog rešenja mogu biti svi oni koji imaju potrebe za slučajnim brojevima. Pored upotrebe u kriptografiji, generisane slučajne sekvence se mogu koristiti i za kompjuterske igre, u umetničkom izražavanju i u simulacijama. Širok je spektar upotrebe sekvenci koje generiše ovaj izvor.

Predlog za budući rad ogleda se u poboljšanju i uvođenju novih tehnika postobrade sekvenci sa izvora kao i poboljšanje kodovanja. Radi povećanja entropije izvora potrebno je uvesti još neki parametar kretanja vozila kao na primer brzina kretanja ili pravac.

6. Literatura

1. Veinović Mladen i Adamović Saša, (2013) *Kriptologija 1*, prvo izdanje, Univerzitet Singidunum, Beograd, ISBN: 978-86-7912-469-2
2. Adamović Saša, (2015), *Zaštita informacionih sistema*, prvo izdanje, Univerzitet Singidunum, Beograd, ISBN: 978-86-7912-589-7
3. Milosavljević Milan i Adamović Saša, (2013), *Osnovi teorije informacija i kodovanja*, prvo izdanje, Univerzitet Singidunum, Beograd, ISBN: 978-86-7912-506-4
4. William Stallings, (2014), *Cryptography and Network Security – Principles and Practice*, šesto izdanje, Pearson Education, Inc., USA
5. Henk C.A. van Tilborg, (2005), *Encyclopedia of cryptography and security*, Eindhoven University of Technology, Holandija
6. Paul Glasserman, (2003), *Monte Carlo Methods in Financial Engineering*, prvo izdanje, Springer-Verlag, New York, USA, ISBN: 978-0-387-00451-8
7. Markus Dichtl, (2007), *Bad and good ways of post-processing biased physical random numbers*, [na mreži],
<https://www.iacr.org/archive/fse2007/45930135/45930135.pdf>
8. Çetin Kaya Koç, (2009), *Cryptographic Engineering*, Springer-Verlag, New York, USA, ISBN: 978-0-387-71816-3

9. Tatović Milomir, Adamović Saša i Milosavljević Milan, (2015), *FlyBit – Generator slučajnih brojeva*, [na mreži], <http://dx.doi.org/10.15308/Synthesis-2015-107-111>
10. John Walker, *How HotBits Works*, [na mreži], <https://www.fourmilab.ch/hotbits/how3.html>
11. Mads Haar, *Introduction to Randomness and Random Numbers*, Random.org [na mreži], <https://www.random.org/randomness/>
12. S. S. Keller, (2005), *NIST-recommended random number generator based on ANSI X9.31 Appendix A.2.4 using the 3-key triple DES and AES algorithms*, NIST Computer Security Division, [na mreži], <http://csrc.nist.gov/groups/STM/cavp/documents/rng/931rngext.pdf>
13. Weiss Jason, (2004), *Java cryptography extensions*, Elsevier Inc., ISBN: 0-12-742751-1
14. Avi Wigderson, (2009), *Randomness extractors – applications and constructions*, [na mreži], <http://www.math.ias.edu/~avi/PUBLICATIONS/Wigderson2009.pdf>
15. Von Neumann J., (1951), *Various techniques used in connection with random digits*, Applied Mathematics Series, br. 12, [na mreži], <https://dornsifecms.usc.edu/assets/sites/520/docs/VonNeumann-ams12p36-38.pdf>
16. C. Floudas i P. Pardalos, (2001) *Encyclopedia of Optimisation*, Springer Science & Business Media, USA
17. Physical sources of randomness, [na mreži], <http://www.randomnumbers.info/content/Generating.htm>

