VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
KOMPIUTERIJOS KATEDRA

Magistro baigiamasis darbas

Chaoso teorijos taikymai duomenų šifravime

Atliko:
Lukas Vengelis parašas

Vadovas:
dr. Tadas Meškauskas

Vilnius
2017

Turinys

Santrauka

Šiame darbe aptariame pagrindines kriptografijos ir chaoso teorijos mokslų sąvokas, apibrėžiame jų sąsają. Apžvelgiame įvairius chaosu paremtus duomenų šifravimo algoritmus, įvertiname jų privalumus ir trūkumus. Toliau pristatome savo siūlomą chaosu paremtą šifravimo ir dešifravimo algoritmą, pateikiame jo praktinio realizavimo gaires bei įvertiname įvairius jo parametrus. Atliekame šio algoritmo spartos analizę ir palyginame jį su kitais chaotiniai algoritmais. Taip pat atliekame šifro kokybės analizę: įvertiname šifravimo procese naudojamos chaotinės skaičių eilutės atsitiktinumą bei šifruojamų duomenų sąryšį su jų šifru.

Summary

Applications of Chaos Theory to Cryptography

In this paper we describe main concepts of cryptography and chaos theory and establish a link between these two sciences. We describe multiple chaotic algorithms, and analyze their strengths and weaknesses.

Based on this analysis, we establish goals for a new chaotic cryptosystem:

1. It must be faster than other reviewed algorithms.
2. It must be safe.
3. It must be capable of being used practically - it should be possible to encrypted all data types.

After we establish our goals, we propose a new chaotic cryptosystem, describe its encryption and decryption algorithm, give and analyse all the required parameters for its practical realization. We then analyse its runtime by encrypting and decrypting various files and compare it to other chaotic algorithms.

Finally, we evaluate quality of the proposed cryptosysted by testing its main process - generation of chaotic digit sequences with statistical randomness tests and by examining correlation between plaintext and ciphertext.

We conclude that our proposed cryptosystem is superior to other reviewed chaotic cryptosystems - it is much faster, safer and can be used with any type of data. We also give guidelines for possible improvements.

Įvadas

Technologijoms aprėpiant vis daugiau gyvenimo sričių, daugėja procesų, kurie yra perkeliami į virtualią erdvę. Elektroninė bankininkystė, balsavimas internetu ar tiesiog nuotolinis įrangos valdymas - visi šie procesai turi būti saugūs. Tad galimybė greitai ir saugiai šifruoti perduodamus duomenis yra kaip niekad svarbi.

Egzistuojantys šifravimo algoritmai nuolat tobulinami - ieškoma spartesnių ir saugesnių būdų. Viena iš įmanomų krypčių - algoritmai, pasitelkiantys chaoso teoriją.

Literatūroje randama įvairių chaotinių algoritmų pavyzdžių, tačiau kol kas jie nėra naudojami praktikoje. Tam yra įvairių priežasčių, bet pagrindinės: saugumo spragos ir lėta sparta. Todėl mokslininkai nuolat bando sukurti chaosu paremtus algoritmus, kurie tokių spragų neturi.

Šio darbo tikslas - išanalizavus egzistuojančių chaotinių kriptografinių algoritmų privalumus ir trūkumus, pristatyti savo sukurtą chaotinę kriptosistemą. Iškeliame pagrindinius reikalavimus kriptosistemai:

- Siūlomos kriptosistemos šifravimo ir dešifravimo algoritmas turi būti ženkliai spartesnis nei kiti nagrinėti algoritmai.
- Algoritmas privalo būti saugus.
- Kriptosistema turi būti pritaikoma praktiniui panaudojimui - bet kokių duomenų failų šifravimui.

Šio darbo pradžioje apžvelgiame kriptografijos mokslą. Apibrėžiame įvairias šio mokslo sąvokas ir šifravimo metodų klasifikaciją. Toliau apibūdiname chaoso savoką, pateikiame chaotinių atvaizdžių pavyzdžių ir apibrėžiame chaoso vertinimo būdus. Susieję šiuos du mokslus, detaliai apibūdiname vieną iš pirmųjų chaotinių algoritmų - Baptistos algoritmą[2] ir jo modifikacijas, išskiriame pagrindinius jų privalumus ir trūkumus. Toliau pristatome ir išanalizuojame kiek kitokiais principais besiremiantį Formolo-Oliveira-Sobotkka algoritmą[4]. Didžioji šio darbo 1-3 skyrių paimta iš ta pačia temą atlikto mokslo tiriamojo darbo projekto [8].

Antroje darbo pusėje pristatome savo siūlomą chaotinę kriptosistemą, paremtą logistinio atvaizdžio generuojamomis chaotinėmis skaičių eilutėmis. Detaliai apibūdiname visus šios kriptosistemos šifravimo ir dešifravimo algoritmo žingsnius. Šią kriptosistemą pilnai realizuojame ir pristatome savo sukurtą šifravimo bei dešifravimo aplikaciją, parašytą Python(versija 3.4.5) programavimo kalba. Toliau apibūdiname ir pagrindžiame jos praktiniui realizavimui pasirinktus algoritmo parametrus bei naudotus įrankius.

Siūlomą kriptosistemą vertiname atlikdami jos spartos bei šifro kokybės analizę. Algoritmo spartą tiriame matuodami įvairių dydžių, formatų bei turinio failus, taip pat ją palyginame su kitų analizuotų chaotinių algoritmų rezultatais. Galiausiai atliekame algoritmo kokybinę analizę - atliekame 15 NIST[1] [7] statistinių testų, skirtų įvertinti duomenų sekų atsitiktinumą. Jais vertiname vieną iš esminių siūlomo algoritmo fragmentų - generuojamas chaotines skaičių eilutes. Galiausiai atliekame šifruojamo pranešimo ir jo šifro sąryšio analizę, ieškodami Pirsono koreliacijos koeficiento ir pateikiame išvadas.

Siūloma kriptosistema pateisina lūkesčius - nors jos algoritmas greičiu vis dar smarkiai atsilieka nuo populiariųjų praktikoje naudojamų algoritmų, tačiau yra žymiai spartesnis nei nagrinėti chaosu paremti algoritmai. Taip pat nenuvilia ir šifro kokybė - nerandame jokio sąryšio tarp šifruojamo pranešimo ir jo šifro.Galiausiai, mūsų siūlomos kriptosistemos šifravimo ir dešifravimo

[1] National Institue of Standards and Technology, liet. Nacionalinis Standartų ir Technologijų institutas

algoritmas nėra apribotas subjektyviai parinkta abecėle, todėl sukurta aplikacija galime užšifruoti bet kokio tipo duomenis. Todėl laikome, jog užsibrėžtus tikslus įgyvendinome.

Darbo pabaigoje pateikiame ateities tyrimų gaires.

1. Kriptografija

Kriptografija yra mokslas, tiriantis saugaus duomenų perdavimo metodus.Pagrindiniai duomenų apsaugos tikslai yra užtikrinti:

- Duomenų konfidencialumą.
- Duomenų vientisumą,
- Duomenų autentiškumą.
- Duomenų neišsižadėjimą.

Tam, kad šie tikslai būtų įgyvendinti, reikalinga saugi ir efektyvi kriptosistema.

1.1. Kriptosistema

Apibrėžiame pagrindinius kriptosistemos komponentus:

- Šifruojamas pranešimas M,
- Užšifruotas pranešimas C,
- Šifravimo raktas k_e,
- Dešifravimo raktas k_d,
- Šifravimo algoritmas $e(M, k_e)$,
- Dešifravimo algoritmas $d(C, k_d)$.

Šifravimo(dešifravimo) raktu vadiname tam tikrą parametrą arba parametrų rinkinį, nuo kurio priklauso šifravimo(dešifravimo) algoritmo rezultatas - užšifruotas(dešifruotas) pranešimas. Tada teksto šifravimas ir dešifravimas aprašomi kaip:

$$C = e(M, k_e),$$
$$M = d(C, k_d),$$

o kriptosistema vadiname visų įmanomų pranešimų bei raktų ir turimų šifravimo bei dešifravimo algoritmų rinkinį.

1.2. Šifravimo metodų klasifikacija

Kriptosistemos ir šifravimo algoritmus klasifikuojame pagal raktų perdavimo bei duomenų apdorojimo metodus.

1.2.1. Simetrinės ir asimetrinės sistemos

Kriptosistemos gali būti simetrinės ir asimetrinės. Simetrinių sistemų šifravimo bei dešifravimo algoritmai naudoja tą patį raktą. Tai privalumas, bet kartu ir trūkumas, nes turime rasti saugų būdą perduoti raktą pranešimo gavėjui. Asimetrinės sistemos naudoja du raktus - vieną šifravimui, kitą dešifravimui. Tam naudojame viešojo ir privataus rakto architektūrą, kai bet kuris asmuo gali užšifruoti pranešimą gavėjo pateiktu viešuoju raktu, tačiau tik pats gavėjas gali tą pranešimą dešifruoti panaudodamas tik jam žinomą privatųjį raktą.

1.2.2. Srautiniai ir blokiniai šifrai

Srautiniai šifrai - tai simetriniai šifrai kuriuose pranešimo simboliai tam tikros operacijos metu sujungiami su pseudo-atsitiktinio duomenų srauto simboliais. Praktikoje pranešimą ir srautą atvaizduojame bitais, tada tarp kiekvieno pranešimo ir tam tikro srauto bito atliekame XOR operaciją (kuri gražina 0 jei abu bitai vienodi ir 1 kitais atvejais) ir taip gauname šifruotą simbolį. Vieni iš žinomesnių srautinių algoritmų: RC4, A5/1, A5/2. Blokiniai šifrai vienu metu operuoja keliais bitais, t.y. fiksuotais bitų blokais. Paprastai šiuos blokus iteruojame tam tikrą skaičių kartų pagal tam tikrą taisyklę, tol kol gauname norimą šifrą. Žinomiausi blokiniai šifrai: DES, IDEA, RC5, AES.

1.3. Maišos funkcijos

Užšifravę duomenis,natūraliai norime gebėti juos dešifruoti. Visgi dažnai užtenka turėti tik vienos krypties šifravimo algoritmą, kitaip vadinamą maišos funkcija. Tokia funkcija turi daugybę panaudojimo sričių, pavyzdžiui ja užšifravę vartotojo slaptažodį, galime jį laikyti duomenų bazėje nepažeisdami vartotojo konfidencialumo, o vėliau vartotoją galime nesunkiai identifikuoti palyginę duomenų bazėje laikomo ir vartotojo iš naujo pateikto užšifruoto slaptažodžio reikšmes. Dar vienas maišos funkcijos panaudojimo būdas - duomenų pokyčių tikrinimas. Tai ypač svarbu sudėtingoms, daug skaičiavimų atliekančioms ir dideliais duomenų kiekiais manipuliuojančioms aplikacijoms. Užšifravę tam tikras duomenų dalis vieną skaitine reikšme, vėliau atlikę pokyčius su kitais duomenis, panaudoję maišos funkciją pastebime kurie duomenys nepasikeitė ir kurių skaičiavimų nereikia atlikti iš naujo ir taip sutaupome resursų.

1.4. Kriptoanalizė

Kriptoanalizės tikslas yra rasti spragų kriptosistemoje. Esminis to būdas yra bandymas nulaužti šifravimo algoritmą - gauti pranešimo turinį neturint rakto. Šifravimo algoritmai yra laikomi saugiais, jei jiems įveikti prireikia pakankamai ilgo laiko tarpo. Tad akivaizdu, jog sugeneruotas tikslas turi atrodyti kaip atsitiktinių simbolių seka, be jokios aiškios pasiskirstymo taisyklės.

1.4.1. Kriptoanalizės atakos

Vienas iš kriptoanalizės metodų yra šifravimo algoritmų atakos. Jų tikslas yra rasti kriptosistemos dešifravimo raktus bei šifruojamą pranešimą (jei jis nėra žinomas). Paprastai atakos yra skirstomos į šiuos 6 tipus:

- Pavienių šifrų (angl. ciphertext-only) ataka. Jos metu gauname užšifruotą pranešimą nežinodami jį atitinkančio šifruojamo pranešimo ir tada įvairiais metodais bandome rasti dešifravimo raktus. Ši ataka dažniausiai yra labai sudėtinga.

- Teksto-šifro porų (angl. known-plaintext) ataka. Ji nuo pavienių šifrų atakos skiriasi tuo, kad žinome ir turimo užšifruoto teksto pavyzdžio pradinį, neužšifruotą atitikmenį.

- Pasirinktų teksto-šifro porų (angl. chosen-plaintext) ataka. Tai ataka, kurios metu galime savo nuožiūra pasirinkti nagrinėjamą šifruojamą pranešimą ir tada gauti jo užšifruotą pavidalą.

- Adaptyvi pasirinktų teksto-šifro porų (angl. adaptive-chosen-plaintext) ataka. Tai išskirtinis teksto-šifro porų atakos atvejis, kai galime dinamiškai pasirinkti šifruojamus ir užšifruotus pranešimus ir kriptoanalizės metu keisti juos pagal savo poreikius.
- Pasirinktų šifrų ataka. Jos metu galime pasirinkti užšifruotą pranešimą ir gauti jo pradinį atitikmenį.
- Adaptyvi pasirinktų šifrų ataka. Tai pasirinktų šifrų atakos versija, kai kriptoanalizės metu galima dinamiškai keisti pasirinktą užšifruotą pranešimą.

2. Chaoso teorija

Chaoso teorija - matematikos sritis, tirianti labai jautrių pradinėms sąlygoms dinaminių sistemų elgseną. Iš pirmo žvilgsnio tokios sistemos atrodo atsitiktinės, tačiau iš tiesų jos yra deterministinės, t.y. žinodami pradines sąlygas, žinome ir tikslią sistemos reikšmę tam tikru momentu.

2.1. Chaotiniai atvaizdžiai

Toliau pateikiame keletą chaotinių atvaizdžių pavyzdžių.

2.1.1. Logistinis atvaizdis

Logistinis atvaizdis - chaotinio atvaizdžio pavyzdys. Jį aprašome kaip:

$$x_{n+1} = rx_n(1 - x_n),$$

kur $x \in [0, 1]$, $n \in N$, o dominančios r reikšmės yra intervale $(0, 4]$.

2.1.2. Palapinės atvaizdis.

Palapinės atvaizdis - dar vienas chaotinis atvaizdis. Jį aprašome kaip:

$$x_{n+1} = \begin{cases} \mu x_n & \text{,kai } x_n < 0.5 \\ \mu(1 - x_n) & \text{,kai } x_n \geqslant 0.5 \end{cases}, \tag{2.1}$$

čia $\mu \in [0, 2]$, $n \in N$, o $x_n \in [0, 1]$. Šiuo atveju mus domina atvaizdžio reikšmės, kai $\mu \in (1, 2]$.

2.1.3. Dvigubas logistinis atvaizdis

R. Agarwal ir M. Rani [6] 2009 m. atliko dviejų žingsnio logistinio atvaizdžio analizę. Šį atvaizdį aprašome kaip:

$$x_n = \beta_n F(x_{n-1}) + (1 - \beta_n)x_{n-1}, \tag{2.2}$$

kur atvaizdis F - 2.1.1 skyrelyje aprašytas logistinis atvaizdis. Čia $0 < \beta_n \leqslant 1$, $n \in N$, o logistinio atvaizdžio parametras r turi patekti į intervalą $[3.86, 4]$.

Savo darbe autoriai nagrinėja $\beta_n = \beta$ t.y. šį parametrą pasirenka kaip konstantą. Jų teigimu toks atvaizdis yra chaotiškas, kai $\beta \geqslant 0.64$.

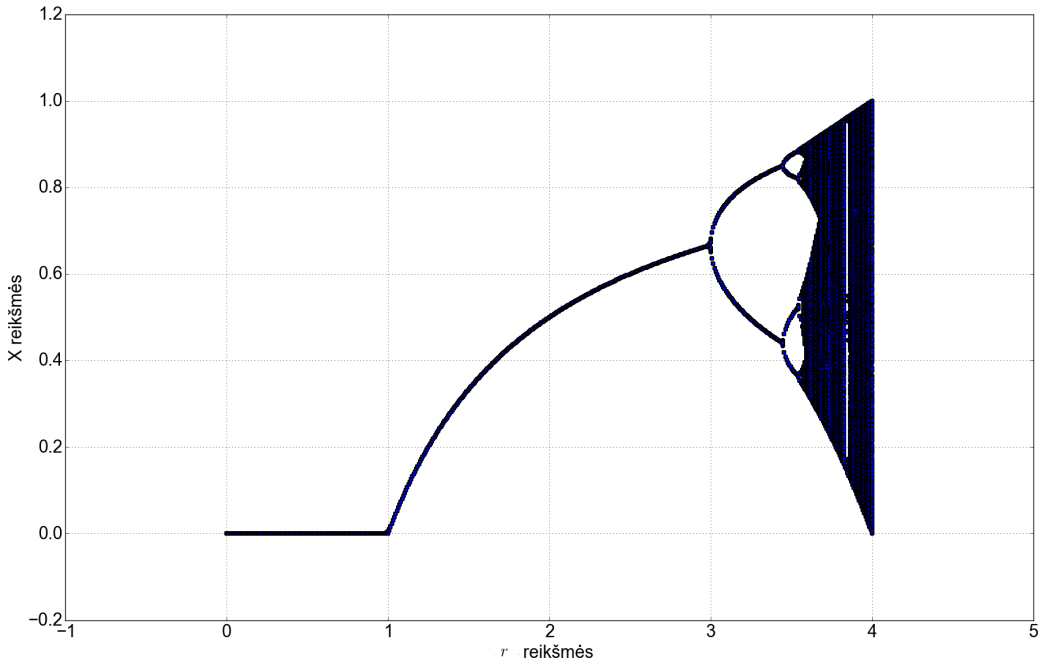
2.2. Chaoso įvertinimas

2.1 skyrelyje aprašydami chaotinių atvaizdžių pavyzdžius, paskelbėme kurios parametrų reikšmės mus domina tiriant chaotiškumą. Visgi, minėti atvaizdžiai nebūtinai bus chaotiški su tomis reikšmėmis. Taip pat kylą klausimas, kaip nustatyti su kokiais parametrais chaotiški bus kiti atvaizdžiai? Šiame skyrelyje pateikiame metodus, kuriuos naudodami galime rasti tinkamų parametrų intervalus.

2.2.1. Bifurkacijos diagrama

Atvaizdžių chaotiškumo priklausomybę nuo jų parametrų vaizdiniu būdu galime įvertinti brėždami bifurkacijos diagramą. Tai taškinė diagrama, vaizduojanti visus taškus, kuriuos aplankė nagrinėjamas atvaizdis. Jei tam tikras taškas nenuspalvintas, iteruojant atvaizdį, šis tokios reikšmės neįgijo. Norėdami būti tikri, jog mūsų atvaizdis elgiasi chaotiškai, turime vengti tokių parametrų, su kuriais diagramoje atsiranda nenuspalvintų taškų kišenės.

1 pav. pavaizduota logistinio , 2 pav. palapinės, o 3 pav. dviejų žingsnių logistinio atvaizdžių bifurkacijos diagramos.



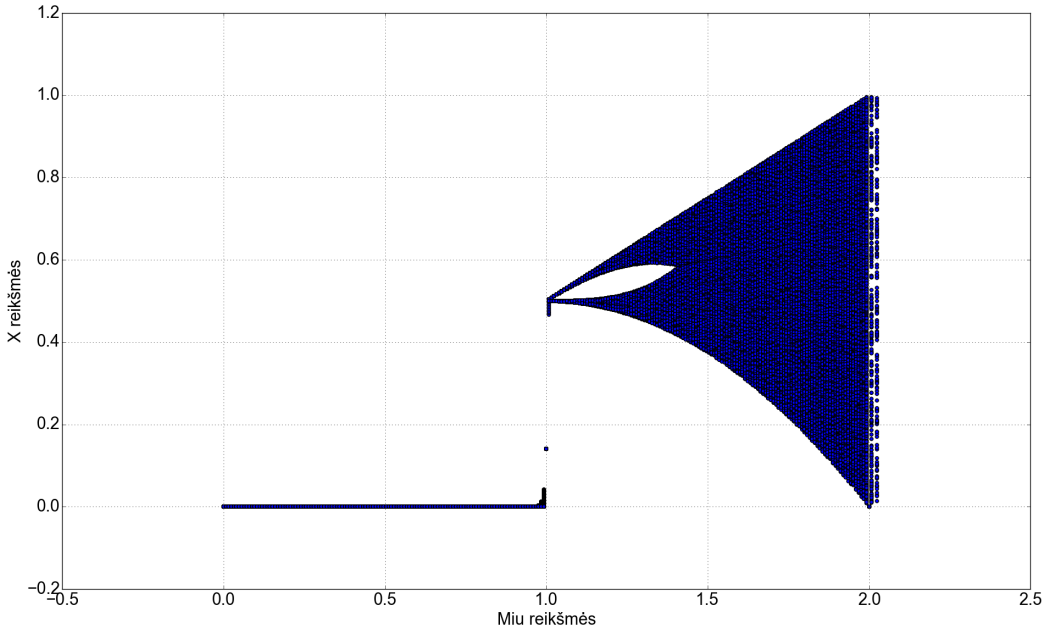
1 pav. Logistinio atvaizdžio bifurkacijos diagrama

2.2.2. Liapunovo eksponentė

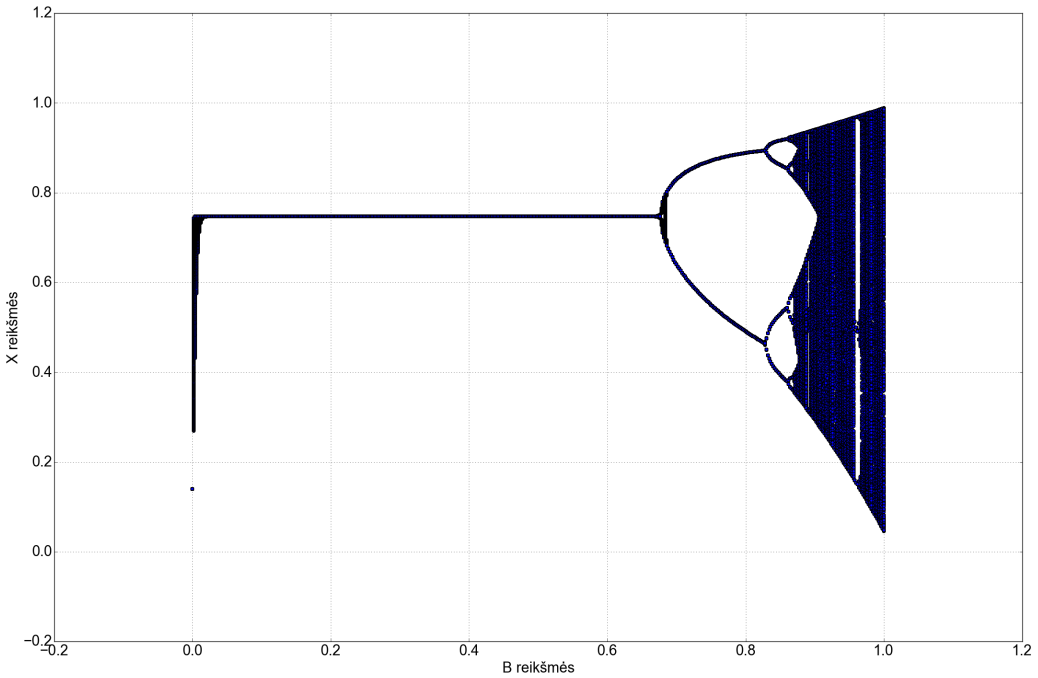
Kitas atvaizdžio chaotiškumo matas - Liapunovo eksponentė:

$$\lambda = \lim_{x\to\infty} \frac{1}{n} \sum_{i=0}^{n-1} |f'(X_i)|,$$

kur $f'(X_i)$ - chaotinės funkcijos išvestinė, o n - iteracijų skaičius. Atvaizdžius laikome chaotiškais, jei $\lambda > 0$.



2 pav. Palapinės atvaizdžio bifurkacijos diagrama



3 pav. Dviejų žingsnių logistinio atvaizdžio bifurkacijos diagrama. Čia parametras $R = 3.95345885$

3. Chaoso teorijos panaudojimas kriptografijoje

Pasinaudoję chaotinių atvaizdžių deterministiškumu ir jautria priklausomybe nuo pradiniu sąlygų, pradines sąlygas laikydami kriptosistemos raktais, chaotiškus atvaizdžius galime panaudoti kriptografijoje. Toliau pateikiame keletą chaotinių algoritmų pavyzdžių.

3.1. Baptistos algoritmas

Baptistos algoritmas [2] - vienas pirmųjų bandymų sukurti chaoso teorija paremtą duomenų šifravimo algoritmą. Jo idėja yra pasirinkus tam tikro dydžio abecėlę, suskaidyti visas įmanomas chaotinės funkcijos reikšmes į intervalus, atitinkančius po vieną tos abėcėlės reikšmę. Tada funkciją iteruojame tol, kol jos reikšmės patenka į šifruojamų simbolių intervalus, o šifru laikome tam reikalingų iteracijų skaičių n. Šiame algoritme naudojame chaotinį logistinį atvaizdį.

3.1.1. Algoritmo žingsniai

Šifravimas:

1. Pasirenkame chaotinės sistemos pradines sąlygas: parametrą r, pradinę reikšmę X_0, maksimalias chaotinio atvaizdžio reikšmes X_{min} ir X_{max} ir šifruojamų duomenų abėcėlės simbolių skaičių S. Taip pat, algoritmo stiprumui padidinti, Baptista rekomenduoja pasirinkti papildomus parametrus $\eta \in [0, 1)$ ir N_0. Maksimalus iteracijų skaičius turi neviršyti 65532.

2. Logistinio atvaizdžio reikšmių intervalą $[X_{min}, X_{max}]$ padaliname į S intervalų, ilgio $\epsilon = (X_{max} - X_{min})/S$, kiekvieną abėcėlės simbolį susiejame su tam tikru intervalu.

3. Pradine reikšme laikydami X_0, chaotinę funkciją iteruojame tol, kol jos reikšmės patenka į šifruojamo simbolio intervalą. Šifru laikome tam reikalingų iteracijų skaičių n. Tada iteracijų skaičių prilyginame nuliui, o paskutinę gautą funkcijos reikšmę laikome pradine funkcijos reikšme šifruojant sekantį simbolį.

4. 3 žingsnį kartojame tol, kol užšifruojame visą tekstą.

Tuo atveju, kai nurodome reikšmes N_0 ir η, trečią žingsnį modifikuojame:

- Jei nurodome N_0, funkciją būtinai iteruojame bent N_0 kartų, t.y. jei funkcijos reikšmė patenką į tinkamą intervalą per mažesnį iteracijų skaičių, jį ignoruojame ir procesą tęsiame toliau. Šio parametro dėka net ir žinant visus S simbolius ir atitinkamus jų intervalus, algoritmo neįmanoma lengvai įveikti. Šis parametras neturėtų būti labai didelis (pavyzdžiui, pats Baptista rinkosi $N_0 = 250$), nes kitu atveju gali smarkiai išaugti šifravimo laikas.

- Jei nurodome η, chaotinės funkcijos reikšmei patekus į tinkamą intervalą, generuojame atsitiktinį skaičių $\kappa \in (0, 1)$. Tada, jeigu $\kappa \geqslant \eta$, šifru laikome atitinkamą iteracijų skaičių n, jei ne, funkciją iteruojame toliau, kol tenkinama ši sąlyga. Šis parametras padaro šifrą labiau nenuspėjamu padidindamas didesnių n reikšmių skaičių, tačiau jį reikia rinktis atsargiai, nes jei η bus labai didelis, labai padidės ir šifravimo laikas.

Dešifravimas:

1. Nurodome raktus, t.y. visus šifravimo algoritmo pirmame žingsnyje naudotus parametrus, išskyrus η ir N_0.

2. Žingsnis atitinka šifravimo algoritmo antrąjį žingsnį.

3. Nuskaitome užšifruotą simbolį, kuris yra iteracijų skaičius n. Chaotinę funkciją iteruojame n kartų, surandame kuriam intervalui priklauso ta reikšmė ir koks simbolis tam intervalui priskirtas.

4. 3 žingsnį kartojame tol, kol dešifruojamas visas tekstas.

3.1.2. Algoritmo trūkumai

Curiac [3] nurodo keletą svarbių Baptistos kriptosistemos problemų:

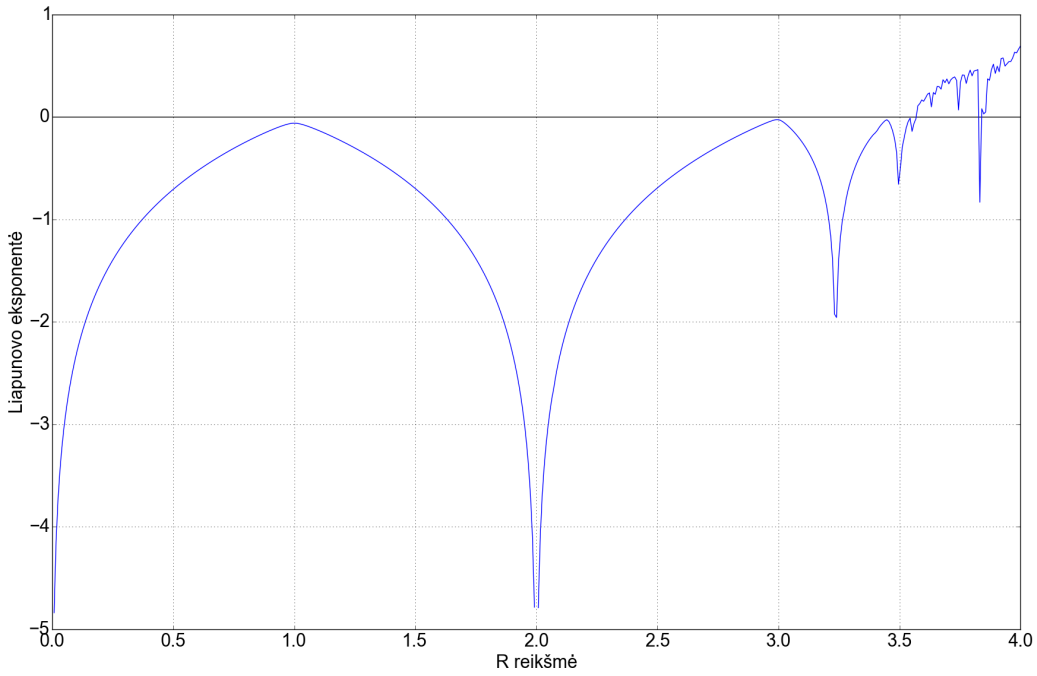
- **Greitis**. Chaotinė funkcija yra iteruojama daug kartų, šis procesas yra lėtas.
- **Padidėjęs užšifruoto failo dydis.** Kiekvieną simbolį reikia atvaizduoti iteracijų skaičiumi, tad užšifruotas tekstas natūraliai yra didesnis.
- **Netolygus užšifruotų simbolių pasiskirstymas.** Užšifravus tekstą, mažesnių n reikšmių būna kur kas daugiau nei didesnių.
- **Galimybė, jog neteisingai parinkus raktus algoritmas nesugebės užbaigti šifravimo proceso**. Baptista pristatydamas algoritmą, pabrėžė, kad šis maksimalus vienam simbloliui užšifruoti reikalingas iteracijų skaičius neturėtų viršyti 65532. Šis skaičius pasirinktas subjektyviai, po empirinių tyrimų pastebėjus, jog jį viršijus algoritmas veikia netaisyklingai ir nepabaigia darbo. Taip nutinka todėl, kad kai kuriomis raktų variacijomis inicializuotas logistinis atvaizdis nebegeneruoja chaotinių reikšmių ir šio nepatenką į šifruojamam simboliui priskirtą intervalą. Tokiu būdu šifravimo procesas niekada nesibaigia.
- **Logistinio atvaizdžio trūkumai.** Logistinis atvaizdis turi keletą trūkumų, sukeliančių problemų jį naudojant Baptistos algoritme. Pirma, jo reikšmės nėra tolygiai pasiskirsčiusios, tad kai kurie intervalai iteruojant bus aplankomi kur kas dažniau. Tai labai didelė saugumo spraga. Antra, ne su visais parametrais r atvaizdis yra pakankamai chaotiškas. Chaotiškumą pamatuojame paskaičiavę Liapunovo eksponentę. 6 pav. matome, kad tik su gana nedaug r reikšmių, logistinis atvaizdis yra pakankamai chaotiškas.

3.1.3. Baptistos algoritmo modifikacijų pavyzdžiai

Įvertinus Baptistos algoritmo trūkumus, sekė daugybė bandymų sukurti tobulesnį chaosu paremtą šifravimo algoritmą. Nemažai mokslininkų tai darė kurdami Baptistos algoritmo modifikacijas.

Wong metodas. Wong [9], įvertinęs, jog dvi didžiausios Baptistos algoritmo silpnybės yra netolygus šifro pasiskirstymas bei lėtas greitis, pasiūlė savo sprendimą šioms problemoms. Pirmiausiai, jis atsisakė atsitiktinio dydžio η kurį generuodavome kiekvieną kartą funkcijos reikšmei patekus į reikiamą intervalą. Wong jį pakeitė kitu atsitiktiniu dydžiu, generuojamu tik kartą vienam simboliui ir naudojamu tiesiog kaip dar vienu papildomu būtinų pradinių iteracijų skaičiumi. Taip autorius siekė sutaupyti šifravimo laiko. Kitas pasiūlymas buvo buvo išvis nenaudoti atsitiktinio dydžio generavimo, tačiau statinį abėcėlės masyvą pakeisti dinamine simbolių lentelė,kuri kiekvieną kartą užšifravus simbolį pasikeistų, sukeičiant i-tąjį ir j-tąjį elementus pagal formulę:

$$j = i + \left(\frac{x - x_{min}}{x_{max} - x_{min}} \right) \times N mod N, \tag{3.1}$$



4 pav. Logistinio atvaizdžio Liapunovo eksponentės reikšmės, kai r kinta intervale [0, 4], o $X_0 = 0.40621$

kur x - chaotinės funkcijos reikšmė, x_{min} ir x_{max} - logistinio atvaizdžio minimali ir maksimali reikšmės, o N - simbolių skaičius, lygus 256 ir atitinkantis ASCII lentelę. Galiausiai Wong pasiūlė šifruojamus simbolius skaidyti 4 bitų blokus. Tada, jei turime 256 simbolių abecėlę, ji sumažėja iki 16 simbolių, taip pat pradinį privalomą iteracijų skaičių galime pasirinkti kur kas mažesnį (pavyzdžiui autorius savo straipsnyje siūlo naudoti $N_0 = 10$). Rezultatas - gerokai sumažintas šifravimui sugaištas laikas ir gauto šifruoto failo dydis. Visgi ir šiame algoritme rasta didelių saugumo spragų, kurias aprašo Alvarez [1].

Patobulintas Wong metodas. D. Xiao, Xiaofeng Liao ir K.W. Wong [10]2006 m. aprašė patobulintą Wong metodą. Atsižvelgę į saugumo spragas autoriai pasiūlė naują simbolių keitimo dinaminėje lentelėje būdą:

$$j = (i + 3sk. \times 100 + 4sk. \times 10 + 5sk.) mod N, \tag{3.2}$$

kur i ir j yra keičiamų simbolių indeksai dinaminėje lentelėje, $3sk.$ - trečiasis logistinio atvaizdžio reikšmės x skaičius po kablelio, $4sk.$ - ketvirtasis, $5sk.$ -penktasis, N - abėcėlės dydis. Šis patobulintas simbolių keitimo metodas taip pat turi trūkumų - keitimo formulėje naudojamas pranešimo abecėlės dydis, todėl visada egzistuoja sąryšis tarp pranešimo ir jo šifro.

Dviejų žingsnių logistinio atvaizdžio metodas. Nitharwal ir Rani[5] modifikavo Baptistos algoritmą ir pasiūlė naudoti dviejų žingsnių logistinį atvaizdį minėtoms saugumo spragoms užkamšyti. Šis atvaizdis plačiau aprašomas [6]. Gautos kriptosistemos schema:

1. Pasirenkame 160 bitų raktą, du logistinius atvaizdžius, turinčius du žingsnius - vieną simbolių šifravimui, kitą simbolių lentelės atnaujinimui, taip pat jų parametrus $r_1, r_2 \in [3.86, 4]$.

2. Iš 160 bitų rakto sugeneruojame pradines atvaizdžių ir raktų atnaujinimo sąlygas.

3. Inicializuojame simbolių lentelę.

4. Duomenis šifruojame analogiškai Baptistos algoritmui, naudodami pirmąjį dviejų žingsnių logistinį atvaizdį.

5. Antrąjį dviejų žingsnių logistinį atvaizdį iteruojame tiek kartų, koks buvo prieš tai šifruojamo simbolio indeksas simbolių lentelėje. Sugeneruojame naujus raktus ir atnaujiname simbolių lentelę.

6. 4 ir 5 žingsnį kartojame tol, kol užšifruojami visi duomenys.

Ši sistema, kaip ir Baptistos, yra simetrinė, tad dešifravimo algoritmas yra analogiškas šifravimo procesui. Autorių teigimu, šis metodas smarkiai padidina algoritmo saugumą, tačiau jų pateikti duomenys rodo, kad algoritmas yra dar lėtesnis ir generuoja didesnius šifruotus failus nei Baptistos algoritmas. Taip pat realizuojant šį algoritmą, kyla problemų su nepakankamu dviejų žingsnių logistinio atvaizdžio chaotiškumu.

3.2. Formolo-Oliveira-Sobottka algoritmas

Formolo-Oliveira-Sobottka [4] 2010 metais pasiūlė naują chaosu ir simbolių ieškojimu paremtą šifravimą algoritmą.Autorių teigimu algoritmas yra daug greitesnis nei Baptistos tipo algoritmai ir yra statistiškai nepriklausomas nuo šifruojamų duomenų pasiskirstymo.

3.2.1. Algoritmo žingsniai

Šifravimas:

1. Pasirenkame 10^d dydžio abecėlę, kurią sudaro skaičiai nuo 0 iki $10^d - 1$.

2. Pagal taisyklę $\sqrt[p]{x_0}$ sugeneruojame didelio tikslumo skaičius (tikslumas priklauso nuo šifruojamų duomenų dydžio ar pasirinktų taisyklių). Iš gauto skaičiaus paėmę trupmeninę jo dalį, gauname šifravime skaičių eilutę, kurią panaudosime šifravime. x_0 laikome kriptosistemos raktu. Sistemos raktu galima būtų laikyti ir p, tačiau paprastumo dėlei tai bus mažas skaičius ir jo raktu nelaikysime.

3. Gautą skaičių eilutę padaliname į d dydžio blokus. Akivaizdu, jog tada galimų bloko variantų yra 10 kartų daugiau nei abecėlės simbolių. Pagal pasirinktą taisyklę, kiekvienam simboliui priskiriame po 10 tokių d-blokų.

4. Sukuriame adresų lentelę, kur viena eilutė atitinka vieną abecėlės simbolį. Tada tikriname, kokias pozicijas sugeneruotoje skaičių eilutėje užima tam tikriems simboliams priskirti blokai. Tomis pozicijomis ir užpildome atitinkamas adresų lentelės eilutes. Kiekvieną šifruojamos žinutės simbolį užšifruojame pagal taisyklę:

   dabartinė pozicija - buvusi pozicija,

   kur dabartinė pozicija- šifruojamo simbolio pozicija skaičių eilutėje, o buvusi pozicija - prieš tai užšifruoto pozicija simbolių lentelėje. Užšifravus simbolį, šis pašalinamas, šifruojant panaudota pozicija pašalinama iš adresų lentelės. Šifruojant pirmą simbolį, laikome kad buvusi pozicija yra lygi 0.

Dešifravimas:

1. Tokiu pat principu panaudoję raktą x_0 sugeneruojame skaičių eilutę.

2. Sukuriame adresų lentelę.

3. Suradę šifrą atitinkančias pozicijas, iš adresų lentelės gauname užšifruotus simbolius.

3.2.2. Algoritmo pranašumai ir trūkumai

Pagrindiniai tokio algoritmo pranašumai yra:

1. Milžiniška raktų aibė

2. Autorių teigimu - didelė sparta, ypač palyginus su kitais chaotiniais algoritmais.

Raktu gali būti bet koks teigiamas realusis skaičius, kurio šaknis yra iracionalus skaičius. Šiek tiek pakeitus x_0, visiškai pasikeičia gaunama skaičių eilutė - todėl šį algoritmą ir priskiriame chaotiniams algoritmams.

Algoritmo sparta, autorių teigimu, šifruojant 10 simbolių abecėlę, yra tik 4 kartus lėtesnė nei AES algoritmo, o dešifruojant šis algoritmas yra netgi 1.3 karto greitesnis nei AES. Taip pat autorių teigimu šiuo algoritmu gautas šifras yra nepriklausomas nuo šifruojamo pranešimo simbolių pasiskirstymo ir yra gana saugus.

Po atliktos algoritmo analizės ir ji praktinio atnalizavimo, pastebėjome nemažai trukumų:

1. Nelankstaus dydžio abecėlė

2. Smarkus sulėtėjimas padidinus abecėlę.

3. Neapibrėžtos šaknies eilutės radimo taisyklės esant labai dideliems duomenims

Kaip minėjome, šiame algoritme abėcėlė privalo būti 10^d dydžio. Tokia abėcėlė yra patogi šifruojant skaitmenis, bet yra sunkiai praktiškai panaudojama šifruojant įvairius neskaitinius simbolius. Tokiu atveju reikia kokiu nors būdų plėsti abecėlę kuri nebūtinai bus žinomo dydžio. Taip pat tada algoritmas smarkiai sulėtėja ir neprilygsta greičiui kurį pabrėžė autoriai.

4. Siūloma kriptosistema

Šiame skyriuje pristatome savo pasiūlytą ir realizuotą chaosu paremtą kriptosistemą. Pagrindiniai reikalavimai kuriuos iškėleme kriptosistemai ir jos šifravimo ir dešifravimo algoritmui:

1. Algoritmas turi būti ženkliai spartesnis nei kiti nagrinėti chaotiniai algoritmai.
2. Algoritmas turi būti saugus ir neturėti jau nagrinėtų algoritmų spragų.
3. Kriptosistema turi būti pritaikoma praktiškai, t.y. turime galėti užšifruoti bet kokio dydžio, abecėlės ar formato failus.

Pirmoji problema kyla dėl algoritmo prigimties - chaotinio atvaizdžio iteravimas, kuriuo remiasi tokio tipo algoritmai yra lėtas procesas. Šios problemos visiškai panaikinti neįmanoma, tad nusprendėme ją įveikti efektyviau išnaudodami iteravimo rezultatą - gautą skaičių, panaudodami 3.2 skyrelyje pristatyto Formolo-Oliveira-Sobottka algoritmo įdėją - iteravimą skaičių eilute. Antrąją problemą nusprendėme įveikti padidindami raktų aibę, taip pat sukurdami naują, skirtingai nei prieš tai nagrinėtuose algoritmuose, nuo šifruojamo pranešimo turinio nepriklausomą simbolių lentelės maišymo sistemą. Trečiąją problemą sprendėme atsisakydami subjektyviai parinktų abecėlių ir algoritme simboliu laikydami bitų bloką.

Tad siūlomos kriptosistemos šifravimo ir dešifravimo algoritmas artimas Formolo-Oliveira-Sobottka algoritmui, tačiau atlikome šiuos esminius pakeitimus:

1. Pakeitėme galimų šifruoti simbolių abecėlę: mūsų kriptosistema galima užšifruoti bet kokio tipo failą, nes šifruojamas elementas - 3 bitų blokas
2. Pakeitėme skaičių (rakto) eilutės radimo procesą atsisakydami lėto šaknies skaičiavimo - mūsų sistemoje rakto eilutę sudarys iteruoto logistinio atvaizdžio reikšmės
3. Atsisakėme adresų lentelės - šifruojamo bloko pozicija rakto eilutėje bus randama tiesiog iteruojant generuotos skaičių eilutės narius.
4. Sukurėme simbolių lentelės maišymo taisyklę, pagal kurią šifravimo proceso metu keisime simboliui priskirtus skaičių eilutės blokus.

Toliau šiame skyriuje pristatome kriptosistemos šifravimo ir dešifravimo algoritmą, apibūdiname kaip šią kriptosistemą realizuoti praktiškai bei pristatome savo sukurtą šifravimo ir dešifravimo aplikaciją.

4.1. Šifravimo ir dešifravimo algoritmas

Šiame skyrelyje apibrėžiame siūlomos kriptosistemos šifravimo ir dešifravimo algoritmą. Jis yra simetrinis, tad dešifravimo procesas iš esmės atkartoja šifravimo procesą iš kitos pusės. Algoritmas taip pat yra blokinis, pasirinktas bloko dydis - 3 bitai. Kriptosistema naudoja du logistinius atvaizdžius - vieną rakto eilutės generavimui, kitą dinaminės simbolių lentelės inicializavimui ir maišymui.

x_0 = 0.85545545236992

x_1 = 0.494178895896200

x_2 = 0.999001687307954

x_3 = 0.00398582196446416

x_4 = 0.0158660383030762

x_5 = 0.0624033348937784

.....

Skaičių eilutė:

94|17|88|95|89|62|00|99|00|16|87|30|79|54|03|98|58|21|96|44|64|16|15|86|60|38|30|30|76|26|24|03|33|48|93|77|84 ...

5 pav. Chaotinės skaičių eilutės generavimo pavyzdys.

4.1.1. Šifravimas

Algoritmo žingsniai:

1. **Raktų pasirinkimas**. Šifravimo algoritme naudojame du logistinius atvaizdžius. Pasirenkame jų pradines sąlygas r^p ir x_0^p (pozicijų eilutės atvaizdžiui) bei r^m ir x_0^m (maišymo atvaizdžiui). Tai kriptosistemos raktai.

2. **Pozicijų eilutės generavimas.** Naudodami pirmuosius raktus r^p ir x_0^p iteruojame pirmąjį logistinį atvaizdį. Išsaugome gauto skaičiaus trupmeninės dalies skaitmenis, pradedant nuo antrojo po kablelio. Šį procesą kartojame pasirinktą skaičių kartų. Gautą skaičių eilutę vadinsime pozicijų eilute. Šią eilutę padaliname į 2 skaičių ilgio blokus. Įmanomos 100 skirtingų bloko variacijų. Skaičių eilutės generavimo proceso pavyzdį pateikiame 5 pav.

3. **Simbolių lentelės inicializavimas.** Algoritmu šifruojame 3 bitų blokus, tad iš viso šifruojamų blokų variacijų yra 8. Tokius šifruojamus blokus vadinsime simboliais. Kiekvienam iš simbolių priskiriame po 12 ar 13 skirtingų pozicijų eilutės blokų tol, kol visi tokie blokai yra priskirti. Šį susiejimą vadiname simbolių lentele. Jos pavyzdį pateikiame 1 lentelėje.

4. **Simbolių lentelės maišymas.** Tokiu pat būdu kaip antrajame žingsnyje, tik šį kartą naudodami raktų porą r^m ir x_0^m , generuojame dar vieną pasirinkto dydžio skaičių eilutę. Ją vadinsime maišymo eilute. Šią eilutę padaliname į dviejų skaitmenų blokus ir tada ją iteruojame, išrinkdami po du blokus. Simbolių lentelėje patikriname, kuriems simboliui priskirti šie maišymo eilutės blokai ir:

    - Kai šie blokai priskirti skirtingiems simboliams, bei tiems simboliams priskirta po 13 blokų, sukeičiame simbolius kuriems priskirti išrinkti maišymo eilutės blokai.
    - Kai šie blokai priskirti skirtingiems simboliams, bei tiems simboliams priskirta po nevienodą skaičių blokų (vienam 12 kitam 13), atiduodame abu blokus eilutei su mažiau priskirtų blokų.

- Kai šie blokai priskirti tam pačiam simboliui, nieko neatliekame.

5. **Pirmojo simbolio šifravimas.** Šifruojame pirmąjį pranešimo simbolį. Iteruojame pozicijų eilutę tol, kol randame bloką, priskirtą šifruojam simboliui. Tada simbolio šifru laikoma bloko pozicija pozicijų eilutėje.

6. **Likusių simbolių šifravimas.** Šifruojame antrąjį pranešimo simbolį. Pradedant sekančia nei prieš tai užšifruoto simbolio pozicija(ją vadiname buvusia pozicija), iteruojame pozicijų eilutę tol, kol randame bloką, priskirtą šifruojam simboliui. To bloko poziciją vadiname dabartine pozicija. Tada simbolio šifras yra skaičius:

$$\textit{dabartinė pozicija} - \textit{buvusi pozicija}.$$

Kartojame šį procesą, kol užšifruojame visą pranešimą. Kiekvieną kartą šio proceso metu pasibaigus pozicijų eilutei, generuojame naują pozicijų eilutę, ją sujungiame su turėta pozicijų eilute ir dar kartą atliekame simbolių lentelės maišymą.

4.1.2. Dešifravimas

Kaip jau minėjome, kriptosistemos šifravimo algoritmas yra simetrinis, tad dešifravimo procesas yra beveik analogiškas šifravimo procesui - pirmi 4 žingsniai sutampa. Likusieji žingsniai:

5. **Pirmojo simbolio dešifravimas.** Dešifruojame pirmąjį užšifruoto pranešimo simbolį. Žinome, kad jis yra lygus bloko, simbolių lentelėje priskirto šifruojamui simboliui, pozicijai pozicijų eilutėje. Radę šį bloką, simboliu lentelės pagalba randame ir pirmąjį dešifruotą simbolį.

6. **Likusių simbolių dešifravimas.** Dešifruojame antrąjį užšifruoto pranešimo simbolį. Norėdami tai padaryti, turime rasti jo poziciją pozicijų eilutėje. Prieš tai užšifruoto simbolio poziciją vadinant buvusia pozicija, užšifruoto simbolio pozicija yra lygi skaičiui:

$$\textit{buvusi pozicija} + \textit{užšifruotas simbolis}.$$

Turėdami užšifruoto simbolio poziciją, simboliu lentelės pagalba randame antrąjį užšifruotą simbolį.

Kartojame šį procesą, kol dešifruojame visą pranešimą. Kiekvieną kartą šio proceso metu pasibaigus pozicijų eilutei, generuojame naują pozicijų eilutę, ją sujungiame su turėta pozicijų eilute ir dar kartą atliekame simbolių lentelės maišymą.

4.2. Praktinis kriptosistemos realizavimas

Siūlomą kriptosistemą realizavome praktiškai, sukurdami šifravimo ir dešifravimo aplikaciją, į kurią įtraukėme ir anksčiau nagrinėtus chaotinius Baptistos, Wong ir Patobulintus Wong algoritmus. Šiame skyrelyje pristatome šią aplikaciją bei apibūdiname algoritme naudotus parametrus.

1 lentelė. Simbolių lentelės pavyzdys

Šifruojamas blokas(simbolis)	Priskirti pozicijų blokai
000	0, 8, 16, 24, 32, 40, 48, 56, 64, 72, 80, 88, 96
001	1, 9, 17, 25, 33, 41, 49, 57, 65, 73, 81, 89, 97
010	2, 10, 18, 26, 34, 42, 50, 58, 66, 74, 82, 90, 98
011	3, 11, 19, 27, 35, 43, 51, 59, 67, 75, 83, 91, 99
100	4, 12, 20, 28, 36, 44, 52, 60, 68, 76, 84, 92
101	5, 13, 21, 29, 37, 45, 53, 61, 69, 77, 85, 93
110	6, 14, 22, 30, 38, 46, 54, 62, 70, 78, 86, 94
111	7, 15, 23, 31, 39, 47, 55, 63, 71, 79, 87, 95

4.2.1. Raktų aibės ir tikslumo pasirinkimas

Norėdami praktiškai realizuoti savo kriptosistemą, privalome apsibrėžti raktų aibes bei tikslumą (skaitmenų po kablelio skaičių), nes kaip minėjome 2.2 ir 3.1 skyreliuose, logistinis atvaizdis tik su tam tikrais parametrais generuoja iš tiesų chaotiškas reikšmes. 1 ir 4 pav. matome, kad atvaizdis yra iš tiesų chaotiškas nuo maždaug $r = 3.7$ su trumpu nechaotiškumo intervalu apie $r = 3.85$. Atlikę išsamesnius tyrimus, siekdami išvengti netinkamų reikšmių ir turėti vientisus intervalus, pasirenkame tokias raktų aibes:

- $x_0 \in (0, 1)$
- $r \in [3.86, 4)$

Taip pat nusprendėme, kad reikšmių, gautų pirmuoju 1000 iteracijų neįtrauksime į pozicijų ar maišymo eilutes. Galiausiai, pasirenkame 14 skaitmenų po kablelio tikslumą. Iš tiesų, pačiame algoritme operuojame 15 skaitmenų po kablelio reikšmėmis, tačiau vartotojas galės pasirinkti tik 14 skaitmenų kaip raktą. Visgi net ir su tokiais saugikliais egzistuoja maža tikimybė, jog pasirinktomis raktų poromis inicializuoti logistiniai atvaizdžiai negeneruos chaotinių reikšmių. Dėl to, prieš pradedant šifravimą atliekame raktų testą ir tik jį įveikus šifravimas bus leidžiamas. Testuojame visus raktus: r^p, x_0^p, r^m ir x_0^m. Testą parametrų porai atliekame tokiu būdu:

1. Pasirenkame parametrų porą r ir $x0$. Iš pradžių testuojame parametrą r. Sugeneruojame labai nežymiai nuo pasirinkto parametro besiskiriantį antrąjį parametrą:

$$r^1 = r + 0.00000000000001.$$

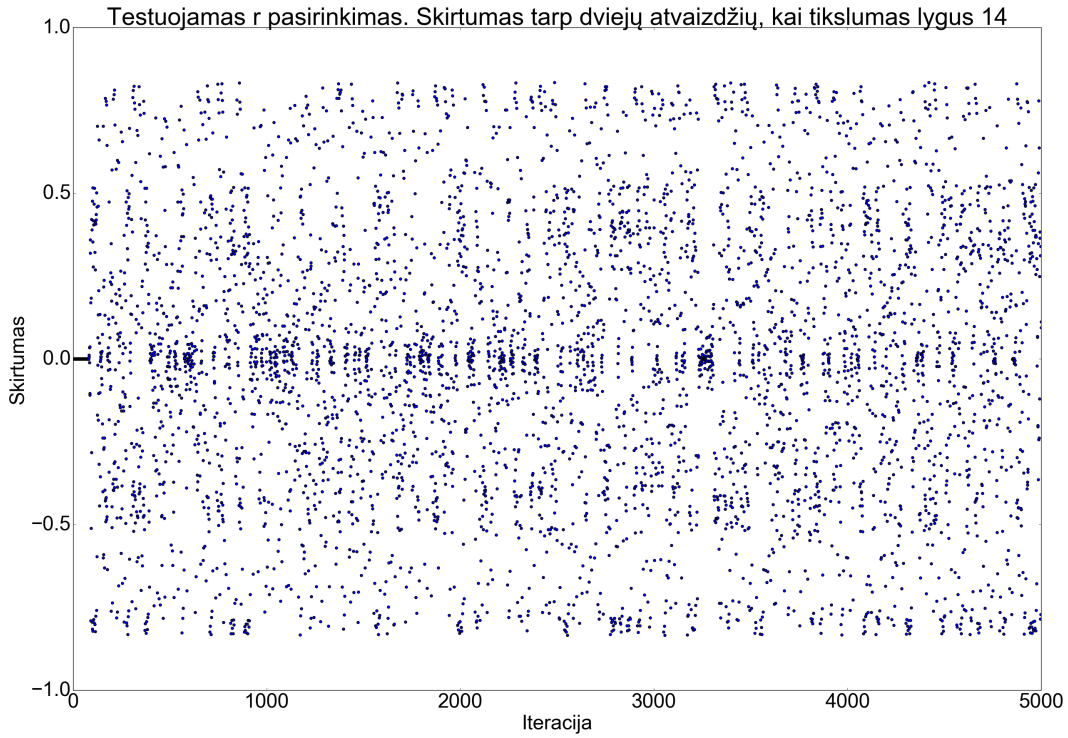
2. Lygiagrečiai generuojame du logistinius atvaizdžius, pirmame naudodami parametrus r ir $x0$, antrame r^1 ir $x0$. Skaičiuojame absoliučius skirtumus tarp jų sugeneruotų reikšmių.

3. Pakartojame 1 ir 2 žingsnį su parametru x_0.

4. Skaičiuojame Liapunovo eksponentę naudodami pasirinktus parametrus, logistinio atvaizdžio reikšmes nuo 1001 iteracijos.

5. Laikome, kad parametrai nėra tinkami jei:

    - 2 ir 3 žingsnyje skaičiuotų absoliučių skirtumų maksimumai bent vienam iš parametrų mažesni nei 0.1
    - 100 iš eilės absoliučių skirtumų skiriasi mažiau nei per 0.01



6 pav. Dviejų logistinių atvaizdžių, inicializuotų bendru $x_0 = 0.32456852254147$ bei skirtingais parametrais $r = 3.86$ ir $r^1 = 3.86000000000001$, reikšmių skirtumas skirtingų iteracijų metu. Atlikta 5000 iteracijų.

- Apskaičiuota Liapunovo eksponentė (pradedant nuo 1001 iteracijos iki 5000 iteracijos) yra mažesnė nei 0.05.

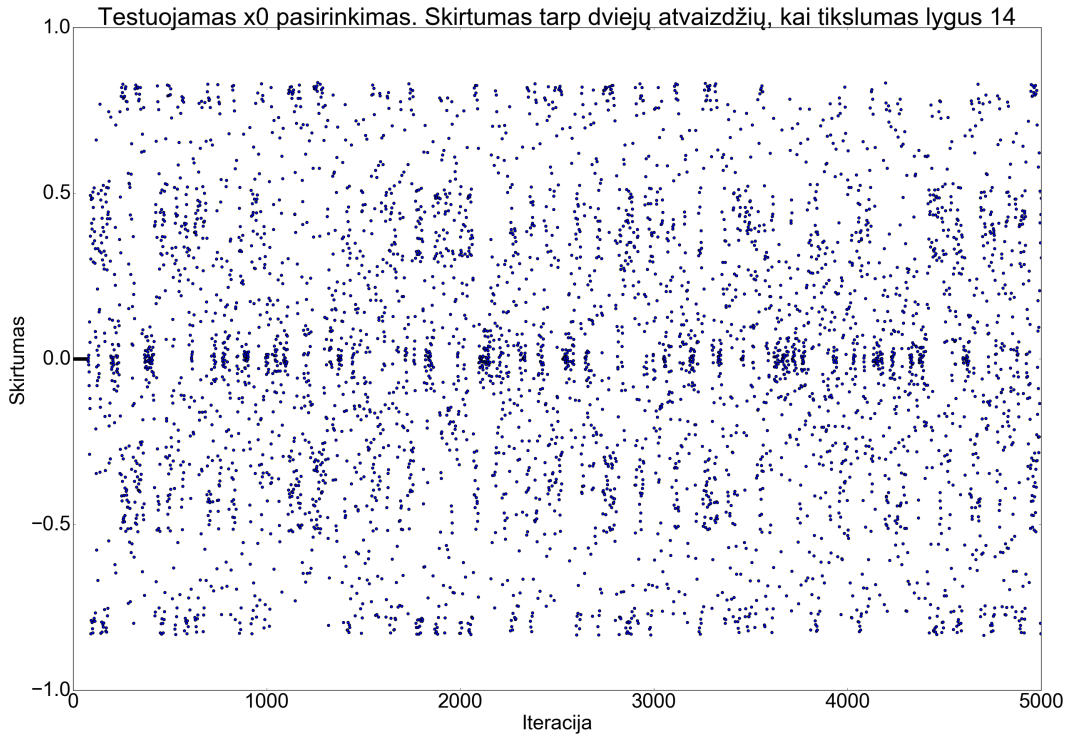
Šie skaičiai buvo pasirinkti subjektyviai, tačiau tokio testo pagalba parametrų pasirinkimas sugriežtinamas, tad šifro kokybė gali tik pagerėti.

6-11 pav. testavimo procesą pavaizdavome vizualiai, pateikdami (neabsoliučius) skirtumus tarp generuotų logistinių atvaizdžių reikšmių. 6 ir 7 pav. matome, kad raktų pora $r = 3.86$ ir $x_0 = 0.32456852254147$ inicializuotas logistinis atvaizdis generuoja vizualiai chaotines reikšmes r ir x_0 atžvilgiu.

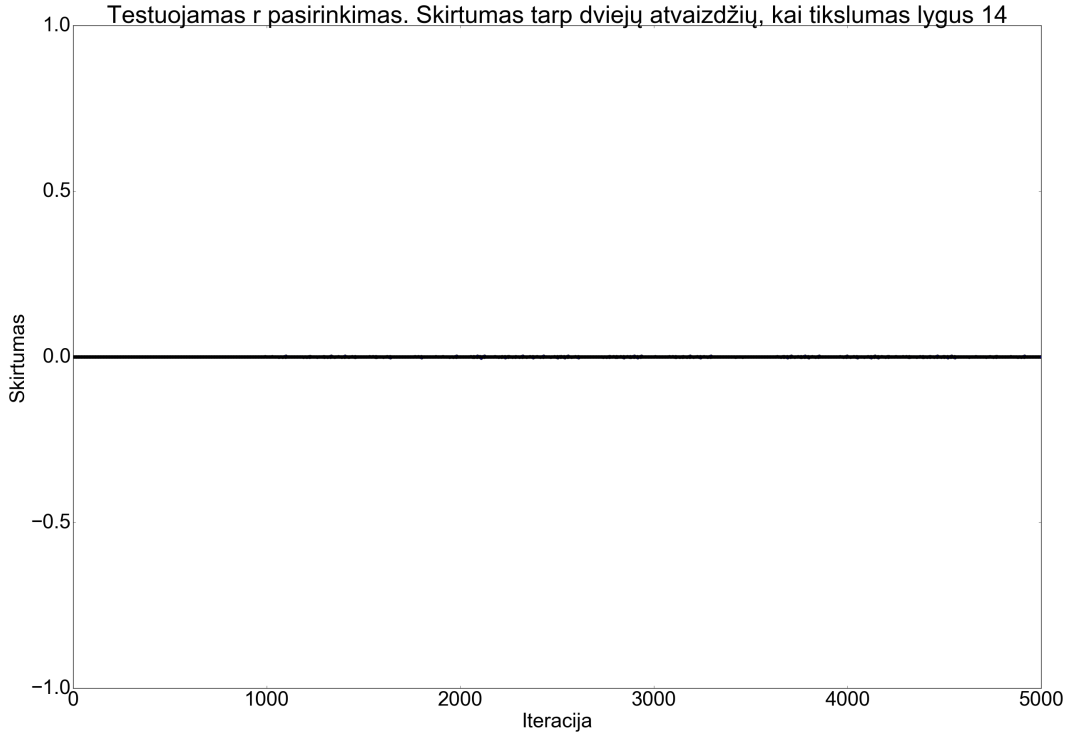
8-9 pav. matome, jog pasirinkę raktų porą
$r = 3.8504200082830047$ ir $x0 = 0.72444853956235$, gauname visiškai nechaotines reikšmes abiejų parametrų atžvilgiu.

Galiausiai 10 pav. matome, jog su parametrais $r = 3.984753$ ir $x_0 = 0.87184858736745$, atvaizdis iš pradžių yra chaotinis, tačiau vėliau r atžvilgiu nusistovi ties tam tikromis reikšmėmis, tad skirtumai su tam tikra paklaida kartojasi. 11 pav. matome, kad x_0 atžvilgiu skirtumai tampa gana artimi 0. Akvaizdu, kad tokiais parametrais inicializuoto atvaizdžio negalime laikyti chaotišku.

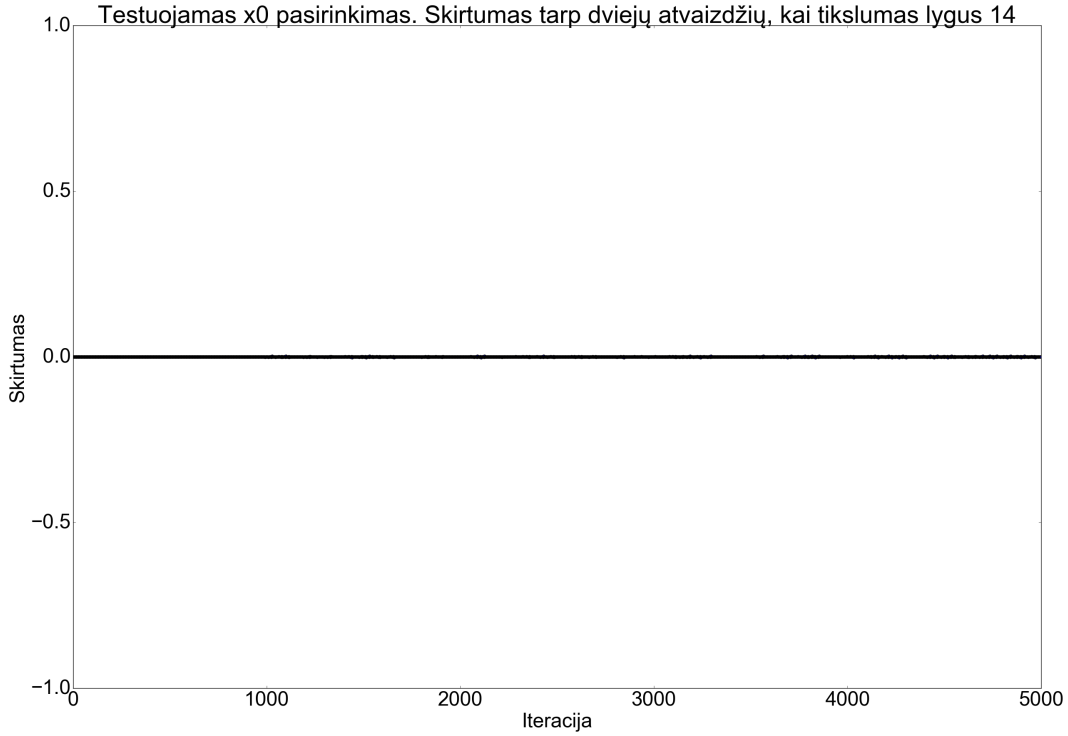
Šias prielaidas patvirtina minėtiems parametrams apskaičiuotos Liapunovo eksponentės reikšmės. Jas pateikiame 2 lentelėje. Pastebime, kad antruoju atveju, nors vizualiai parametrai nėra tinkami, gauta Liapunovo eksponentės reikšmė nors itin maža, bet yra teigiama. Būtent dėl tokių atveju, kai atvaizdis yra itin mažai chaotiškas, testuojant parametrus, Liapunovo eksponentė privalo būti nemažesnė nei 0.05



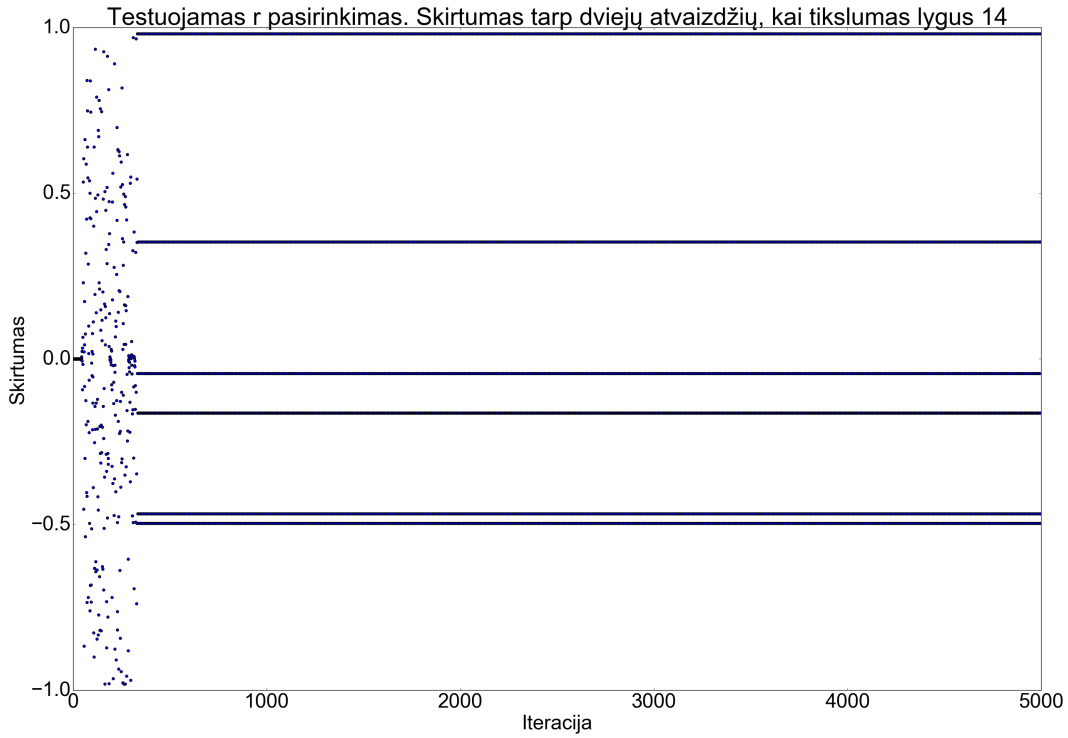
7 pav. Dviejų logistinių atvaizdžių, inicializuotų bendru $r = 3.86$ bei skirtingais parametrais $x_0 = 0.32456852254147$ ir $x_0^1 = 0.32456852254148$, reikšmių skirtumas skirtingų iteracijų metu. Atlikta 5000 iteracijų.



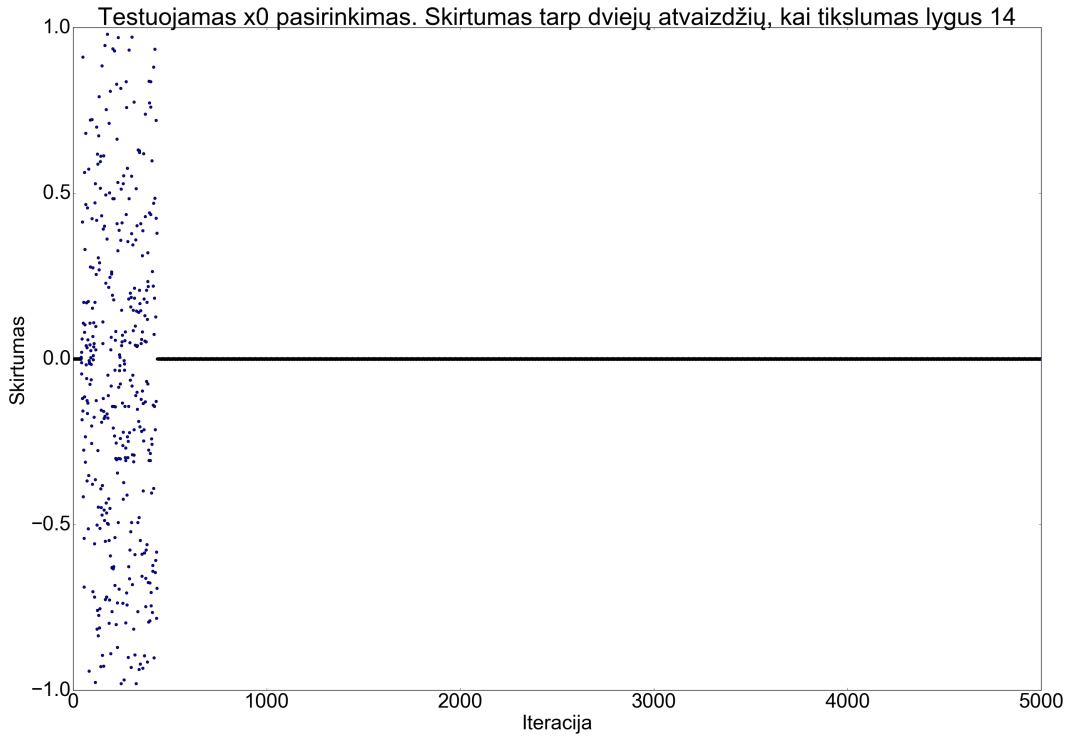
8 pav. Dviejų logistinių atvaizdžių, inicializuotų bendru $x_0 = 0.72444853956235$ bei skirtingais parametrais $r = 3.85042000828347$ ir $r^1 = 3.85042000828348$, reikšmių skirtumas skirtingų iteracijų metu. Atlikta 5000 iteracijų.



9 pav. Dviejų logistinių atvaizdžių, inicializuotų bendru $r = 3.85042000828347$ bei skirtingais parametrais $x_0 = 0.72444853956235$ ir $x_0^1 = 0.72444853956236$, reikšmių skirtumas. Atlikta 5000 iteracijų.



10 pav. Dviejų logistinių atvaizdžių, inicializuotų bendru $x_0 = 0.87184858736745$ bei skirtingais parametrais $r = 3.984753$ ir $r^1 = 3.98475300000001$ reikšmių skirtumas skirtingų iteracijų metu. Atlikta 5000 iteracijų.



11 pav. Dviejų logistinių atvaizdžių, inicializuotų bendru $r = 3.98475300000000$ bei skirtingais parametrais $x_0 = 0.87184858736745$ ir $x_0^1 = 0.87184858736745$ reikšmių skirtumas skirtingų iteracijų metu. Atlikta 5000 iteracijų.

2 lentelė. Logistinio atvaizdžio Liapunovo eksponentės reikšmės skirtingiems parametrams r ir x_0

r	x_0	Liapunovo eksponentės reikšmė
3.86000000000000	0.32456852254147	0.162989
3.85042000828347	0.72444853956235	0.008662
3.98475300000000	0.87184858736745	-0.009371

4.2.2. Kitų parametrų pasirinkimas

Realizuodami mūsų siūlomą kriptosistemą, turime pasirinkti ir kitus parametrus. Vienas iš tokių parametrų - skaičius, kiek kartų maišome simbolių lentelę. Atlikę bandymus nusprendėme, jog šis skaičius bus lygus 1000. Lentelę sumaišyti 1000 kartų užtrunka apie 0.002 sekundės, tad maišymas labai menkai sulėtina šifro veikimą, tačiau smarkiai pagerina jo kokybę.

3 lentelėje pateikiame pavyzdį, kaip simbolių (šifruojamų blokų) lentelė atrodo prieš maišymą ir po.

3 lentelė. Šifruojamiems blokams priskirti pozicijų blokai, prieš ir po maišymo. Naudoti parametrai: $r = 3.99123652316128$, $x_0 = 0.23649823615422$

Šifruojamas blokas	Priskirti pozicijų blokai prieš maišymą	Priskirti pozicijų blokai po maišymo
000	0, 8, 16, 24, 32, 40, 48, 56, 64, 72, 80, 88, 96	9, 10, 11, 12, 16, 28, 32, 37, 60, 61, 74, 79
001	1, 9, 17, 25, 33, 41, 49, 57, 65, 73, 81, 89, 97	2, 23, 31, 35, 38, 41, 54, 55, 57, 59, 71, 77, 81
010	2, 10, 18, 26, 34, 42, 50, 58, 66, 74, 82, 90, 98	3, 17, 20, 30, 49, 52, 66, 70, 76, 83, 89, 92, 97
011	3, 11, 19, 27, 35, 43, 51, 59, 67, 75, 83, 91, 99	4, 24, 33, 45, 46, 47, 56, 63, 67, 86, 91, 93
100	4, 12, 20, 28, 36, 44, 52, 60, 68, 76, 84, 92	5, 8, 18, 29, 42, 48, 58, 69, 84, 88, 90, 96
101	5, 13, 21, 29, 37, 45, 53, 61, 69, 77, 85, 93	6, 7, 15, 21, 25, 27, 40, 53, 64, 68, 72, 78, 82
110	6, 14, 22, 30, 38, 46, 54, 62, 70, 78, 86, 94	0, 1, 14, 34, 36, 39, 43, 62, 65, 87, 94, 98
111	7, 15, 23, 31, 39, 47, 55, 63, 71, 79, 87, 95	13, 19, 22, 26, 44, 50, 51, 73, 75, 80, 85, 95, 99

Kitas parametras kurį turime pasirinkti yra vienu kartu generuojamos pozicijų eilutės ilgis, tiksliau, skaičius kiek kartų logistinis atvaizdis bus iteruojamas randant pozicijų eilutę. 4 lentelėje matome bandymų rezultatus, iteruojant logistinį atvaizdį tam tikrą skaičių kartų. Pateikiame vidutinę trukmę ir greitį, kiekvienam iteracijų skaičiui atlikus po 100 bandymų. Kaip matome, greičiausias iš testuotų variantų yra 50000 logistinio atvaizdžio iteracijų pozicijų eilutės generavimas. Šį variantą naudojame savo kriptosistemoje.

4 lentelė. Logistinio atvaizdžio Liapunovo eksponentės reikšmės skirtingiems parametrams r ir x_0

Iteracijų skaičius	Trukmė	Iteracijos per sekundę
1000	0.001685 s	593472.81
10000	0.012856 s	777846.92
50000	0.061471 s	831391.68
100000	0.174925 s	571673.57
500000	2.666219 s	187531.48
1000000	10.028029 s	99720.49

4.2.3. Šifravimo ir dešifravimo aplikacija

Chaotinių kriptosistemų analizei sukūrėme šifravimo ir dešifravimo aplikaciją. Jį parašyta Python (3.4.5) kalba. Naudojame tokias bibliotekas:

1. "Numpy", failų nuskaitymui ir saugojimui, operacijoms su skaičiais ir masyvais.
2. "Struct", operacijoms su failais.
3. "Array", operacijoms su failais.
4. "Bitarray", failų nuskaitymui į dvejetaines eilutes ir operacijoms su jomis.
5. "Timeit", operacijų trukmės apskaičiavimui.
6. "Decimal", operacijoms su itin tiksliais skaičiais (šiuo atveju raktais).
7. "Easygui", grafinės sąsajos kūrimui.
8. "Cx_Freeze", vykdomojo failo kūrimui.

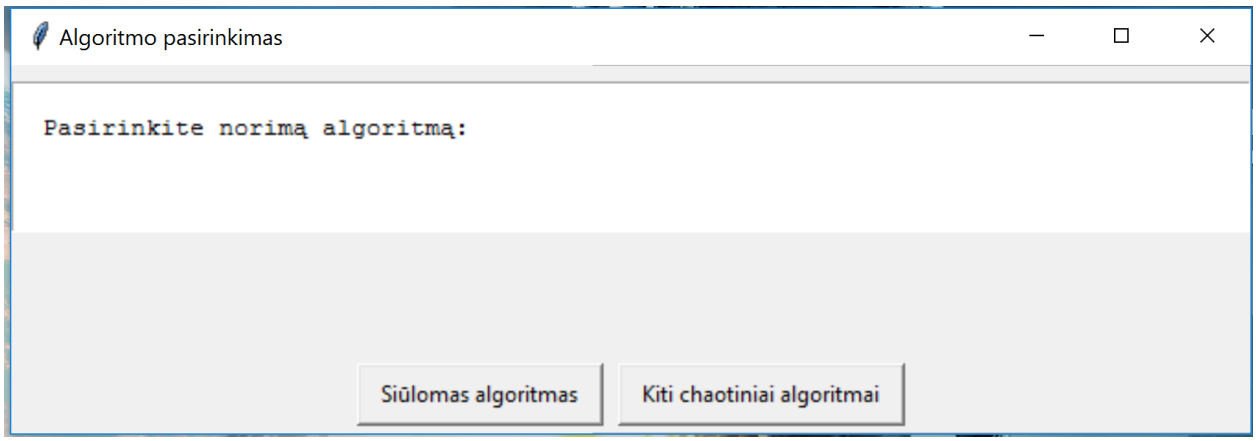
Šią aplikacija galima šifruoti ir dešifruoti duomenis, naudojant tokius algoritmus:

- Mūsų siūlomą,
- Baptistos,
- Wong 4 bitų,
- Wong 8 bitų,
- Patobulintą Wong 4 bitų,
- Patobulintą Wong 8 bitų.

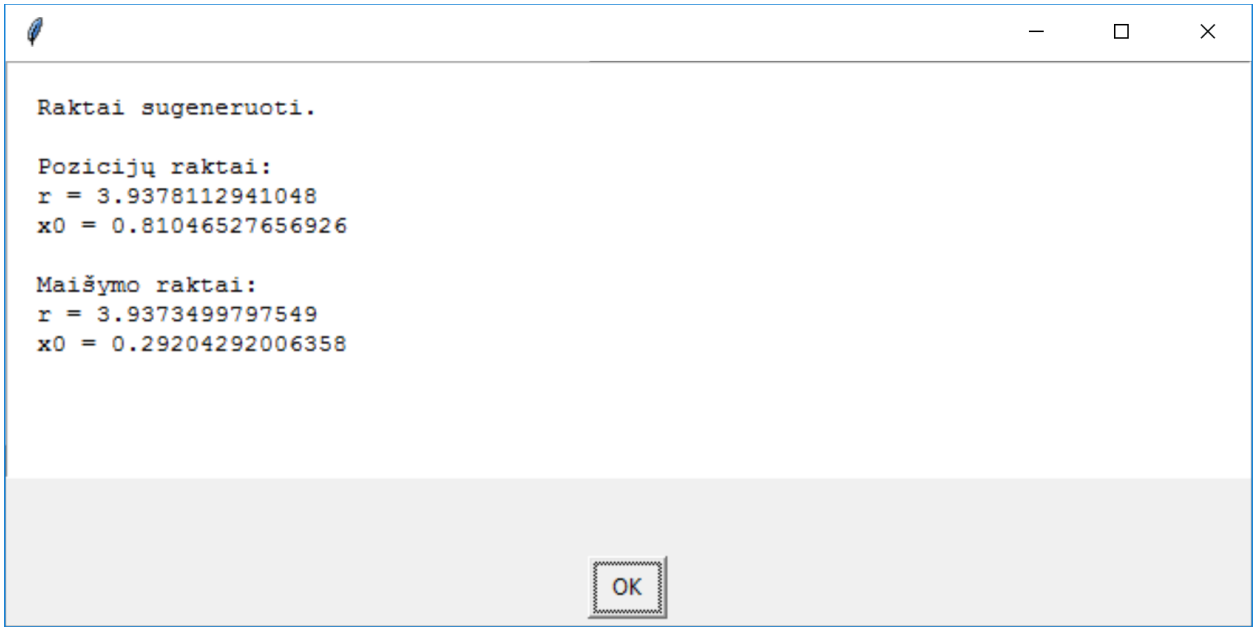
Šifruojant kitomis chaotinėmis kriptosistemomis, suteikiame galimybe rinktis iš dviejų atvaizdžių - logistinio ir palapinės. Originaliai šie algoritmai naudoja tik logistinį atvaizdį, tad šifro kokybė naudojant kitus atvaizdžius nėra žinoma.

Instrukcijos, norint failus šifruoti mūsų siūlomu algoritmu:

1. Paleiskite main.exe failą.
2. Pasirinkite "Siūlomas algoritmas".
3. Pasirinkite operaciją: šifravimą arba dešifravimą
4. Jei norite failą šifruoti, pasirinkite ar raktai turi būti sugeneruoti automatiškai, ar juos įvesite patys. Jei raktus įvesite patys, automatiškai bus patikrinama ar jie tinkami. Tuo atveju, kai raktai bus pripažinti netinkamai, turėsite įvesti kitus raktus kurie bus tikrinami iš naujo. Dešifruodami failą, privalote įvesti raktus, kuriuos naudojote jį užšifruodami.
5. Nurodykite failą, kurį šifruosite arba dešifruosite. Dešifruoti reikia šifravimo metu sugeneruotą dvejetainį encrypted_file.npy failą.



12 pav. Duomenų šifravimo algoritmo pasirinkimas



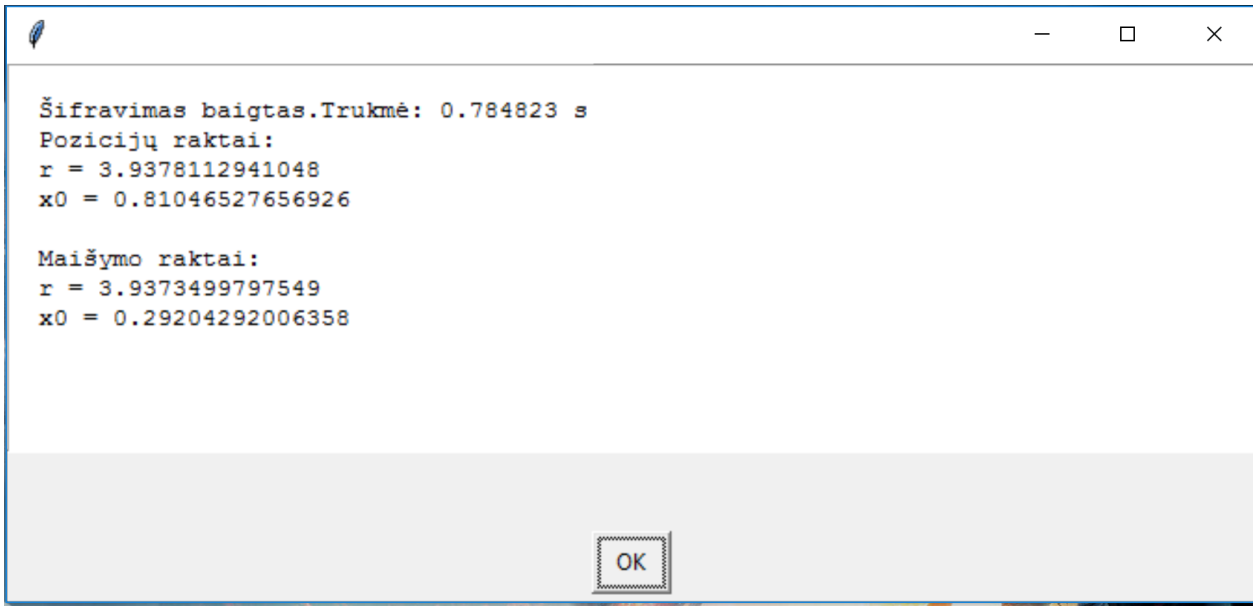
13 pav. Automatinis raktų generavimas

6. Nurodykite direktoriją, kurioje bus išsaugotas užšifruotas arba dešifruotas failas. Šifruojant programa sukurs encrypted_file.npy: dvejetainį failą, saugantį užšifruotą tekstą. Dešifruojant bus sukurtas tekstinis failas decrypted_file.txt.

7. Programa praneš apie prasidėjusį šifravimo arba dešifravimo procesą. Procesui pasibaigus apie tai praneš atsivėręs dar vienas programos langas. Pabaigę šifravimą galėsite pasirinkti, ar taip pat norite sugeneruoti ir tekstinį failą, tam kad šifrą matytumėte išreikštą suprantamu būdu.

12, 13 ir 14 pav. pateikiame šifravimo sukurta aplikacija proceso pavyzdžius.

Šifravimo kitais chaotiniais algoritmais instrukcijos pateiktos šio darbo autoriaus ta pačia tema atlikto Mokslo tiriamojo darbo projekto [8] 4.2 skyrelyje.



14 pav. Duomenų šifravimo siūloma kriptosistema pabaigos pranešimas

5. Siūlomos kriptosistemos vertinimas

Šiame skyriuje pristatome atliktus savo siūlomos kriptosistemos vertinimo testus. Analizei naudojame tokių parametrų kompiuterį:

- Procesorius: *Intel Core i7-4720HQ*
- Procesoriaus dažnis: *2.6 GHz*
- Operatyvioji atmintis (RAM): *8GB*

5.1. Kriptosistemos spartos tyrimas

Šiame skyrelyje pristatome įvairių atliekamų siūlomos kriptosistemos spartos tyrimų rezultatus, taip pat jos šifravimo ir dešifravimo spartą lyginame su kitais chaotiniais algoritmais. Visi testai atlekam minėtų parametrų kompiuteriu, kiekvieną algoritmą testuojanti po 10 kartų - rezultatuose pateikiame šių testų vidurkius.

5.1.1. Įvairių formatų failų šifravimas

Vienas iš mūsų siūlomos kriptosistemos privalumų yra tai, kad ši kriptosistema gali šifruoti visokių tipų failus ir neapsiriboti tik ASCII ar kokios nors kitos siauros abecėlės šifravimu. 5 lentelėje pateikiame 494 kb dydžio tekstinio, 48.3 Kb dydžio MP3 formato garsinio ir 103Kb dydžio JPG formato vaizdinio failo šifravimo ir dešifravimo trukmes ir spartas mūsų siūlomu algoritmu. Matome, kad šifravimo sparta svyruoja tarp maždaug 43.5-47.8 kilobaitų per sekundę, dešifravimo tarp 73 ir 83.2 kilobaitų per sekundę.

5 lentelė. Įvairaus dydžio tekstinio, garsinio ir vaizdinio failų šifravimo siūlomu algoritmu trukmės (s) ir spartos (Kb/s) palyginimas.

Failas	Dydis	Šifravimas		Dešifravimas	
		Trukmė	Sparta	Trukmė	Sparta
Tekstinis	494 Kb	10.346 s	47.796 Kb/s	6.619 s	74.634 Kb/s
Garsinis	48.3 Kb	1.108 s	43.592 Kb/s	0.662 s	72.96 Kb/s
Vaizdinis	103 Kb	2.270 s	45.374 Kb/s	1.2380 s	83.199 Kb/s

5.1.2. Siūlomo bei kitų chaotinių algoritmų spartos palyginimas

Vienas iš šio darbo tikslų buvo sukurti kriptosistemą, kuri būtų greitesnė už anksčiau nagrinėtas kitas chaotines kriptosistemas. Lyginame Baptistos, Wong 4 ir 8 bitų, Patobulintą Wong 4 ir 8 bitų bei mūsų siūlomą kriptosistemą. Mūsų siūlomą kriptosistemą inicializuojame tokiais raktais:

- $x_0^p = 0.84545545236985$
- $r^p = 3.99654844545$
- $x_0^m = 0.24545545236985$

- $r^m = 3.86446654844545$

Kitas chaotines kriptosistemas inicializuojame tokiais raktais:

- $X_{min} = 0.214$
- $X_{max} = 0.782$
- $X_0 = 0.5621$
- $r = 3.951152$
- $\mu = 1.902346$
- $\eta = 0.5$
- 4 bitų Wong algoritmų $N_0 = 10$
- 8 bitų Wong algoritmų $N_0 = 256$

6 lentelėje pateikiame 1.78 Kb, 7 lentelėje 17.2 Kb, 8 lentelėje 60.4 KB dydžio failų šifravimo ir dešifravimo mūsų siūloma bei kitomis chaotinėmis kriptosistemomis trukmes. Matome, kad vienas pirmųjų chaotinių algoritmų - Baptistos bei kiti 8 bitų algoritmai spartos prasme negali lygintis su 4 bitų algoritmais. Tuo tarpu 4 bitų Wong ir Patobulintas Wong algoritmai smarkiai lėtesni nei mūsų siūlomas algoritmai, tai puikiai matome 15-18 pav. : mūsų siūlomas algoritmas už saugiausią iš kitų algoritmų - Patobulintą Wong 4 bitų, 60.4 KB failo šifravimo metu greitesnis beveik 6 kartus, kai naudojamas logistinis atvaizdis ir apie 4.5 karto kai naudojamas palapinės atvaizdis. Dešifravimo metu naudojant logistinį atvaidį jis greitesnis daugiau nei 7.5 karto, o palapinės - daugiau nei 6 kartus. Primename, kad originaliai kiti chaotiniai algoritmai naudoja tik logistinį atvaizdį, tad korektiškiausias būtent lyginimas su šiuo atvaizdžiu gautais rezultatais.

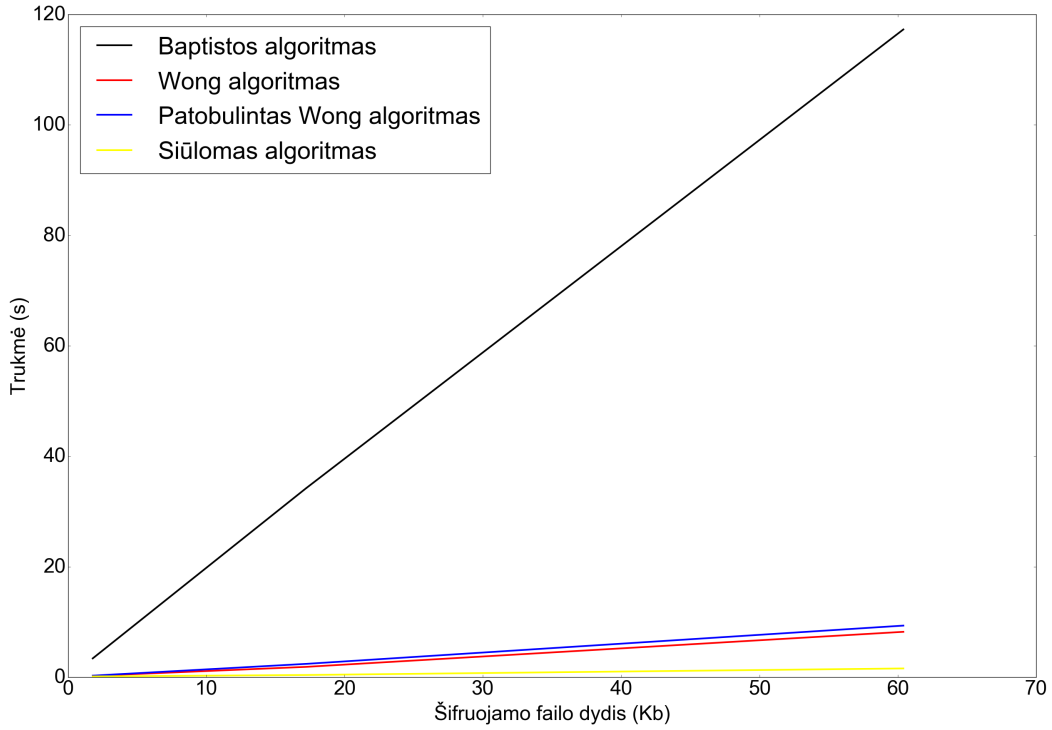
6 lentelė. Chaotinių šifravimo algoritmų trukmės palyginimas. Failo dydis 1.78 Kb

Algoritmas	Logistinis atvaizdis		Palapines atvaizdis	
	Šifravimas	Dešifravimas	Šifravimas	Dešifravimas
Baptistos	3.399 s	2.335 s	1.930s	1.567 s
Wong 4 bitu	0.242 s	0.191 s	0.208 s	0.159 s
Wong 8 bitu	1.489 s	1.356 s	1.369 s	1.023 s
Patobulintas Wong 4 bitu	0.275 s	0.234 s	0.214 s	0.185 s
Patobulintas Wong 8 bitu	1.833 s	1.367 s	1.124 s	1.121 s
Siūlomas algoritmas	0.132 s	0.112 s		

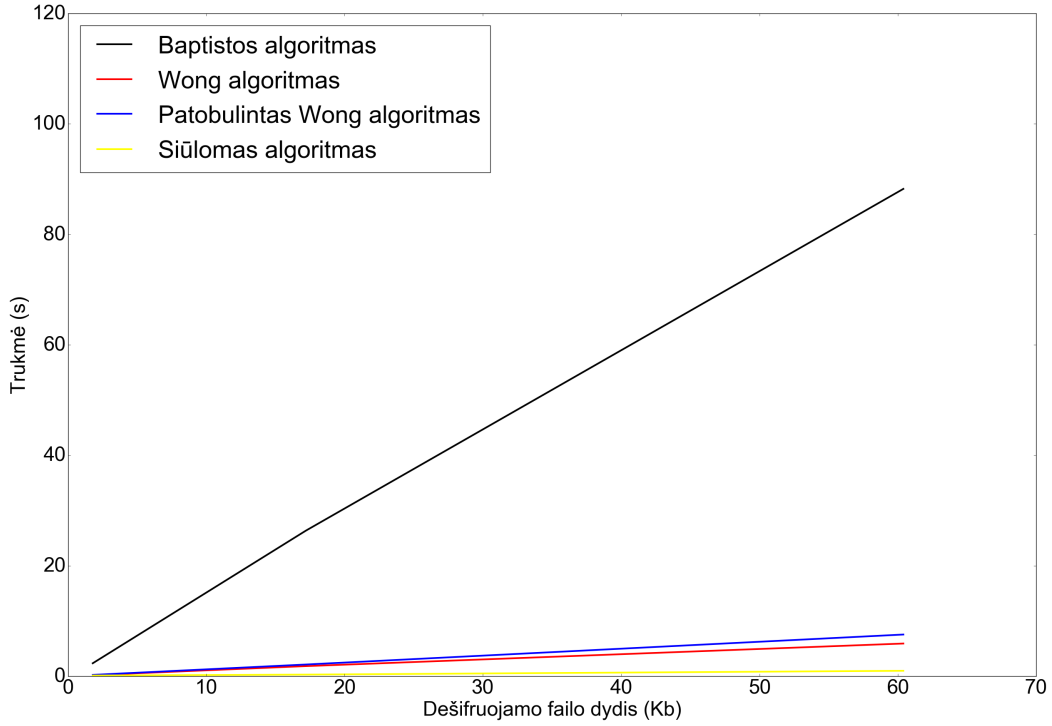
5.2. Kriptosistemos šifro kokybės tyrimas

Šiame skyrelyje tiriame siūlomos kriptosistemos šifro kokybę. Naudodami įvairius įrankius, atliekame tokius tyrimus:

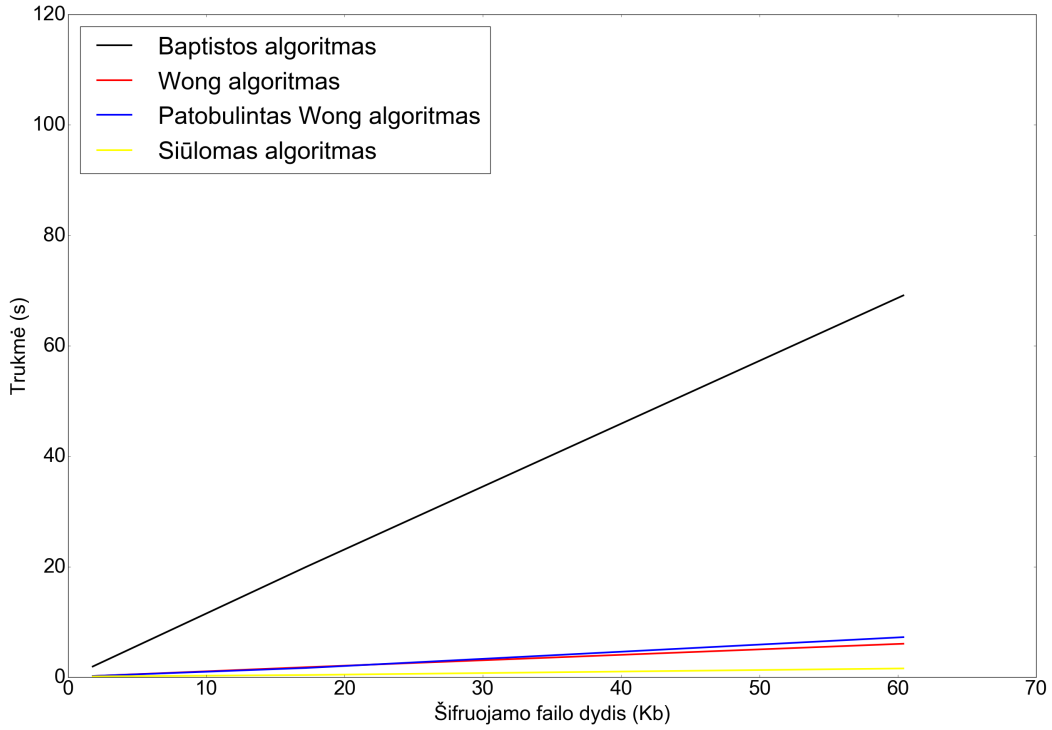
1. Generuotos skaičių eilutės (pozicijų ir maišymo) atsitiktinumo,
2. Šifro nepriklausomumo nuo šifruojamo failo.



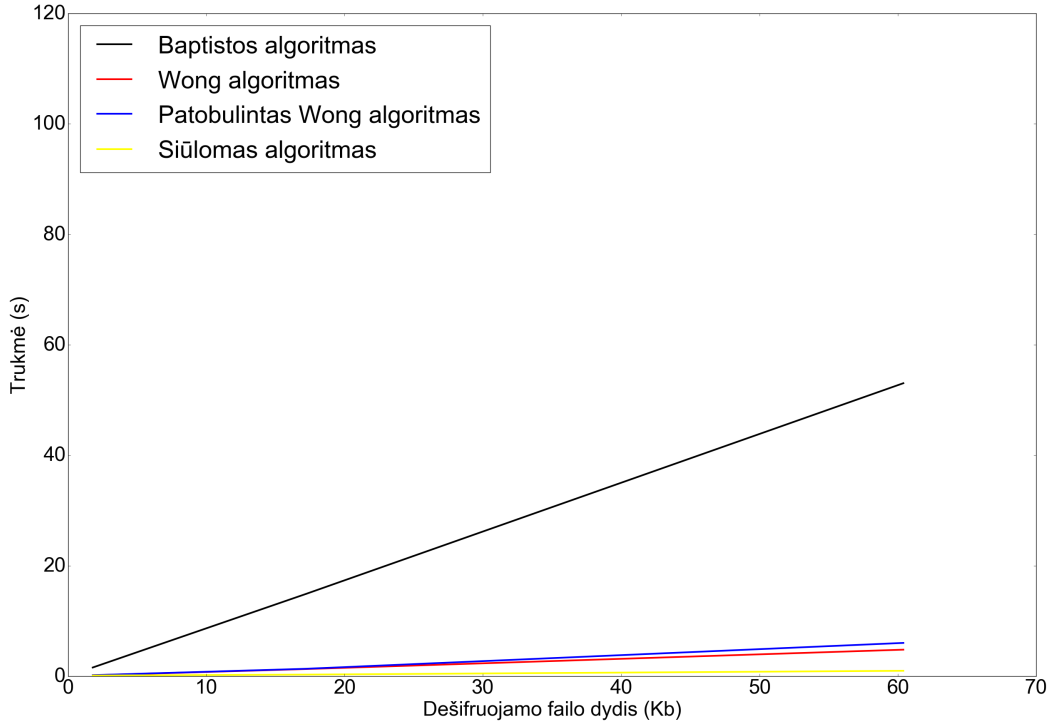
15 pav. Chaotinių algoritmų, naudojančių logistinį atvaizdį ir siūlomo algoritmo šifravimo trukmės palyginimas palyginimas.



16 pav. Chaotinių algoritmų, naudojančių logistinį atvaizdį ir siūlomo algoritmo dešifravimo trukmės palyginimas palyginimas.



17 pav. Chaotinių algoritmų, naudojančių palapinės atvaizdį ir siūlomo algoritmo šifravimo trukmės palyginimas.



18 pav. Chaotinių algoritmų, naudojančių logistinį atvaizdį ir siūlomo algoritmo dešifravimo trukmės palyginimas.

7 lentelė. Chaotinių šifravimo algoritmų trukmės palyginimas. Failo dydis 17.2 Kb

Algoritmas	Logistinis atvaizdis		Palapines atvaizdis	
	Šifravimas	Dešifravimas	Šifravimas	Dešifravimas
Baptistos	34.192 s	26.359 s	19.953 s	14.889 s
Wong 4 bitu	1.863 s	1.819 s	1.804 s	1.288 s
Wong 8 bitu	16.524 s	12.810 s	12.184 s	10.182 s
Patobulintas Wong 4 bitu	2.405 s	2.111 s	1.664 s	1.336 s
Patobulintas Wong 8 bitu	17.101 s	13.694 s	12.305 s	10.835 s
Siūlomas algoritmas	0.392 s	0.283 s		

8 lentelė. Chaotinių šifravimo algoritmų trukmė palyginimas. Failo dydis 60.4 Kb

Algoritmas	Logistinis atvaizdis		Palapines atvaizdis	
	Šifravimas	Dešifravimas	Šifravimas	Dešifravimas
Baptistos	117.281 s	88.231 s	69.125 s	53.043 s
Wong 4 bitu	8.210 s	5.908 s	6.053 s	4.797 s
Wong 8 bitu	60.737 s	47.828 s	48.681 s	37.105 s
Patobulintas Wong 4 bitu	9.335 s	7.538 s	7.244 s	6.020 s
Patobulintas Wong 8 bitu	63.207 s	51.030 s	47.664 s	38.481 s
Siūlomas algoritmas	1.578 s	0.973 s		

5.2.1. Skaičių eilutės tolygumo ir atsitiktinumo tyrimas

Generuojamos skaičių eilutės atsitiktinimui ištirti pasitelksime NIST statistinių testų rinkinį. Šiais testais galime ištirti bet kokias sekas, susidedančias iš 0 ir 1. Pasak testų rinkinio specifikacijų [7] testai laikosi tokių prielaidų tiriamų sekų atžvilgiu:

1. **Tolygumas:** bet kokiu momentu, generuojant atsitiktines ar pseudoatsitiktines dvejetaines sekas, tikimybė sugeneruoti 0 ir 1 yra vienoda, t.y. lygi 0.5. Laukiamas 0 (arba 1) skaičius yra $\frac{n}{2}$, kur n lygus sekos ilgiui.

2. **Vientisumas (angl. scalability)**: Bet kuris statistinis testas gali būti pritaikomas ir sekos posekiui. Tad seka laikoma atsitiktine tik tada, jei visi jos posekiai taip pat yra atsitiktiniai.

3. **Pilnumas:** Nėra korektiška atsitiktinės sekos generatorių vertinti tik vienu generuotos sekos pavyzdžiu, t.y. tik tam tikromis fiksuotomis pradinėmis sąlygomis.

NIST statistinių testų rinkinį sudaro 15 testų:

1. Dažnių (Monobitų) testas,
2. Dažnių testas posekiuose,
3. Kumuliacinių sumų testas,
4. Sekų testas,
5. Ilgiausių vienetų sekų blokuose testas,
6. Dvejetainių matricų rangų testas,
7. Diskrečios Furjė transformacijos (spektrinis) testas,

8. Nesutampančių šablonų testas,
9. Sutampančių šablonų testas,
10. Universalus Mauerio testas,
11. Apytikslės entropijos testas,
12. Atsitiktinių nuokrypių testas,
13. Atsitiktinių nuokrypių variantų testas,
14. Serijų testas,
15. Tiesinio sudėtingumo testas.

Dėl testų gausos, toliau kiekvieną testą apibūdinsime aukščiau pateiktu numeriu. Visų testų aprašymus galima rasti NIST pateiktame statistinio testų rinkinio dokumente [7].

Kiekvieno NIST testo rezultatas - sugeneruota *p-reikšmė* nagrinėjamai bitų sekai.Laikome, kad testuojama seka įveikė testą, kai:

$$\textit{p-reikšmė} \geqslant \alpha,$$

kur α yra pasirinktas reikšmingumo lygmuo. Reikšmingumo lygmeniu vertiname savo testo patikimumą. Pavyzdžiui, jei pasirenkame $\alpha = 0.01$, laikome, kad tikimybė, jog statistiniu testu priimta hipotezė nėra tik atsitiktinumas, yra 99%. NIST siūlo pasirinkti $0.01 \leqslant \alpha \leqslant 0.1$.

Trečioje NIST testų prielaidoje minėjome, jog nėra korektiška atsitiktinių skaičių generatorių testuoti tik turint viena jo generuotos sekos pavyzdį. Tad tą patį testą atliksim dideliam skaičiui sekų ir baigę testus, žinosime, kiek iš tų testuotų sekų juos įveikė bei skaičiuosime proporciją, lygia įveiktų ir iš viso atliktų testų skaičiui. Pasirinkę reikšmingumo lygmenį α, laikome,jog vidutinė tikimybė, kad seką įveikė testą, būtų iš intervalo:

$$\left[\hat{p} - 3\sqrt{\frac{\hat{p}(1-\hat{p})}{m}}; \hat{p} + 3\sqrt{\frac{\hat{p}(1-\hat{p})}{m}}\right],$$

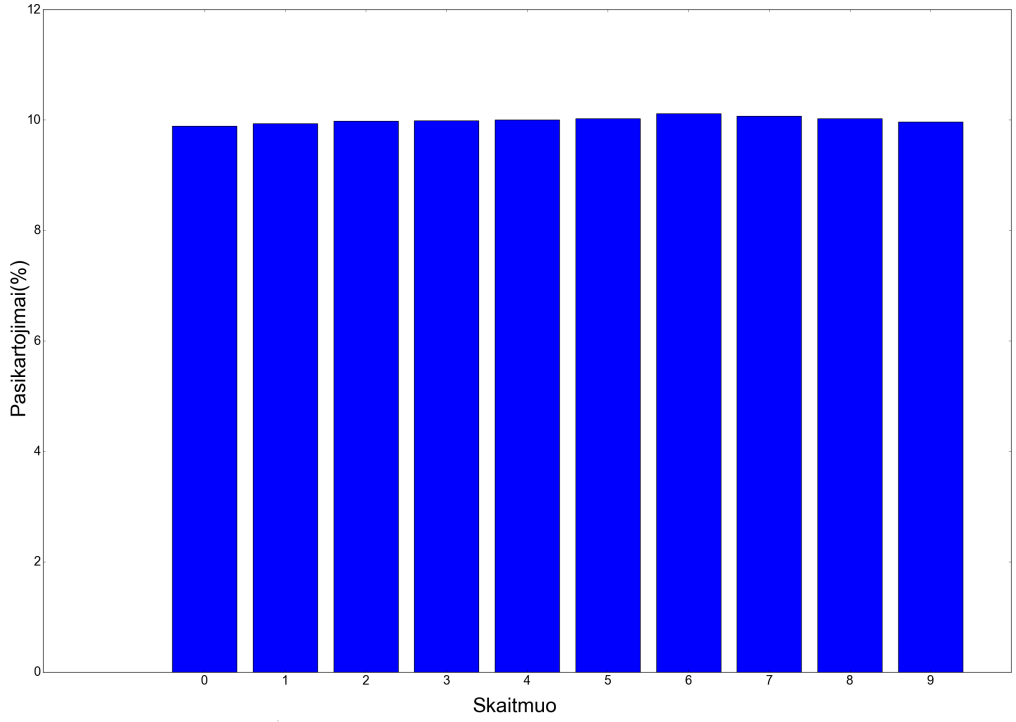
kur $\hat{p} = 1 - \alpha$, o m yra testuojamų sekų skaičius. Tad testą įveikusių sekų proporcija turi būti ne mažesnį nei šis skaičius.

Svarbiausias iš NIST testų yra pats pirmasis - dažnių testas. Jo metu tiriame 0 ir 1 dažnius sekoje. Jeigu kažkurį iš dviejų elementų sekoje sutinkame daug dažniau negu kitą, šio testo seka neįveikia, o neįveikus tokio testo, tikimybė jog seka nėra atsitiktinė yra beveik šimtaprocentinė.

NIST statistinių testų atlikimo žingsniai:

1. **Generuojame skaičių eilutę**. Siekdami laikytis trečiosios NIST testų prielaidos - pilnumo, atsitiktinai pasirenkame 100 skirtingų parametrų r ir x_0 porų, atitinkančių chaotiškumo reikalavimus. Tada kiekvienai tokiai porai sugeneruojame po skaičių eilutę (atlikdami po 50000 logistinio atvaizdžio iteracijų) ir šias eilutes sudedame į vieną.

2. **Paverčiame skaičių eilutę dvejetaine seka.** Kaip jau minėjome, NIST testų rinkinys skirtas tirti sekas susidedančias iš 0 ir 1, tuo tarpų mūsų seką sudaro skaitmenys nuo 0 iki 9. Tad mūsų turimą eilutę reikia paversti dvejetaine seka. Nepamirškime, kad šifruodami pozicijų eilutę suskaidome į blokus po 2 skaitmenis, tad ir mūsų turimą skaičių eilutę reikia padalinti į tokius blokus. Tada:



19 pav. NIST testuose naudotos skaičių eilutės pasiskirstymas.

(a) Turimus 2 skaitmenu blokus, t.y. skaičius nuo 0 iki 99 padaliname iš 100 ir padauginame iš 64.

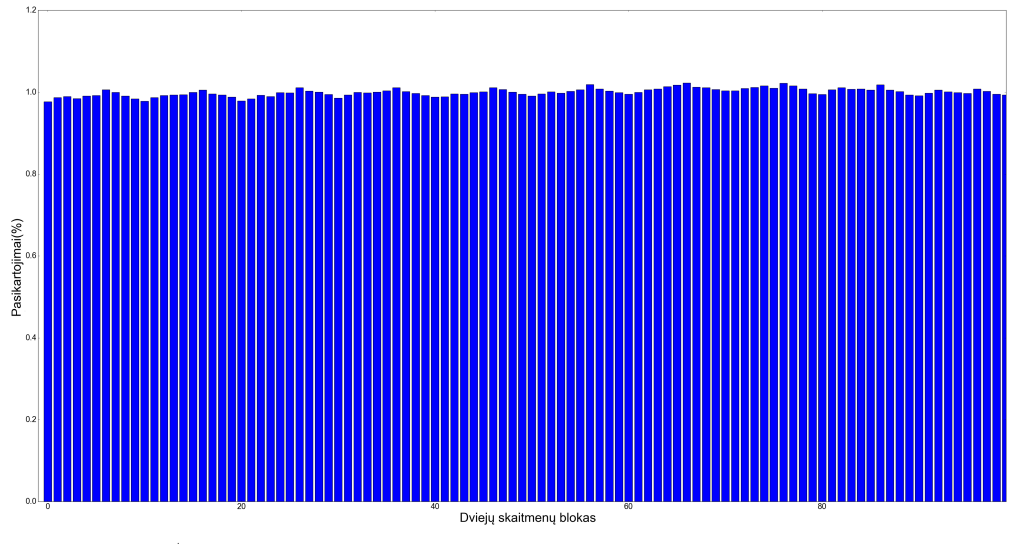
(b) Gautus skaičius suapvaliname. Tokiu būdu maksimalus imanomas gautos sekos skaičius yra 63, t.y. didžiausias 6 bitu skaičius.

(c) Visus skaičius pakeičiame į jų dvejetainį pavidalą - 6 bitų eilutes. Pavyzdžiui, tokiu atveju mažiausias skaičius - $0 = 000000$, o didžiausias $63 = 111111$.

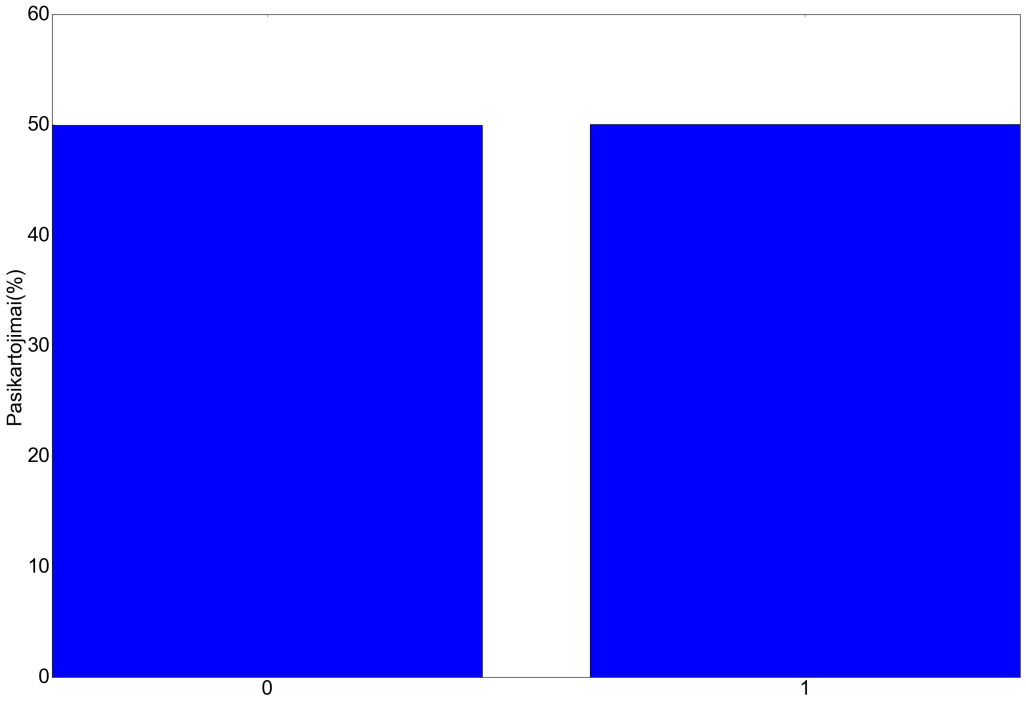
Tokiu budu gauname 1 ir 0 seką, kuri išliks tolygi, jeigu ji buvo tolygi savo dešimtainiame pavidale. Gauto dvejetainio failo dydis - 25.1 MB (210921120 bitai).

3. **Pasirenkame įvairius parametrus ir vykdome NIST testus.** Atsižvelgdami į NIST rekomendacijas, pasirenkame tokius testų parametrus:

   - Bitų kiekis: testams 1-5, 7, 8, 11 lygus 20000, likusiems lygus 1000000.
   - Dažnių testo posekiuose testo posekio ilgis lygus 128 bitams,
   - Nesutampančių šablonų testo bloko ilgis lygus 9 bitams,
   - Sutampančių šablonų testo bloko ilgis lygus 10 bitų,
   - Apytikslės entropijos testo bloko ilgis lygus 9 bitams,
   - Serijų testo bloko ilgis = 2 bitų,
   - Tiesinio sudėtingumo testo bloko ilgis = 5000 bitų,
   - Testuojamų sekų skaičius: testams 1-5, 7-8,11 lygus 1000, likusiems 100.



20 pav. NIST testuose naudotos skaičių eilutės skaitmenų pasiskirstymas, ją suskirsčius į dviejų skaičių posekius.



21 pav. NIST testuose naudotos skaičių eilutės pasiskirstymas, performatavus ją į dvejetainę seką.

19 pav. pateikiame pirmajame žingsnyje generuotos skaičių eilutės pasiskirstymą. Matome kad skaičiai 0-9 pasiskirstę gana tolygiai. 20 pav. pateikiame mus labiau dominantį skaičių eilutės, padalintos į dviejų skaičių blokus pasiskirtymą. Matome, kad ir šiuo atveju pasiskirstymas vizualiai atrodo gana tolygus. Galiausiai, 21 pav. matome eilutės, pagal apsibrėžtas taisykles paverstos 0 ir 1 seka, pasiskirstymą. Matome, kad 0 ir 1 sekoje yra beveik po lygiai.

9 lentelė. NIST testų rezultatai, kai $\alpha = 0.01, 0.05$. Taip - testas įveiktas, Ne - neįveiktas.

Testas	Įveikti testai(%)	Rezultatas	
		$\alpha = 0.01$	$\alpha = 0.05$
1. Dažnių (Monobitų)	99	Taip	Taip
2. Dažnių posekiuose	99.6	Taip	Taip
3. Kumuliacinių sumų	99.2	Taip	Taip
4. Sekų	96.8	Ne	Taip
5. Ilgiausių vienetų sekų blokuose	95.2	Ne	Taip
6. Dvejetainių matricų rangų	98	Taip	Taip
7. Diskrečios Furjė transformacijos	98.5	Taip	Taip
8. Nesutampančių šablonų	97.72	Ne	Taip
9. Sutampančių šablonų	7	Ne	Ne
10. Universalus Mauerio	61	Ne	Ne
11. Apytikslės entropijos	66.8	Ne	Ne
12. Atsitiktinių nuokrypių	97.22	Taip	Taip
13. Atsitiktinių nuokrypių variantų	98.85	Taip	Taip
14. Serijų	97.05	Ne	Taip
15. Tiesinio sudėtingumo	97	Ne	Taip

Kai kurie iš statistinių testų sistemos buvo kartojami ne vieną kartą, jų rezultatuose pateikiame įveiktų testų proporcijų vidurkius. Pagal turimus parametrus ir apsibrėžtas taisykles, laikome, kad su reikšmingumo lygmeniu $\alpha = 0.01$ sekos įveikia testus:

- 1-5, 7, 8, 11, kai įveiktų testų santykis didesnis nei 0.981,
- 6, 9, 10, 14-15, kai įveiktų testų santykis didesnis nei 0.929.
- 12,13, kai įveiktų testų santykis didesnis nei 0.94

Su reikšmingumo lygmeniu $\alpha = 0.05$ sekos įveikia testus:

- 1-5, 7, 8, 11, kai įveiktų testų santykis didesnis nei 0.96,
- 6, 9, 10, 14,15, kai įveiktų testų santykis didesnis nei 0.885,
- 12,13, kai įveiktų testų santykis didesnis nei 0.841

NIST statistinių testų rezultatus pateikiame 9 lentelėje . Su $\alpha = 0.01$, tirta generuota skaičių eilutė, paversta į dvejetaines sekas, įveikia 7 iš 15 testų. Su $\alpha = 0.05$, šis skaičius padidėja iki 12 testų. Taigi, laikyti skaičių eilutės generavimo proceso visiškai atsitiktiniu negalime. Visgi, šio darbo tikslas nėra sukurti atsitiktinių skaičių generatorių - chaotinis skaičių eilutės generavimas yra tik dalis šifravimo proceso.

Būtent netobulas skaičių eilutės atsitiktininumas buvo pagrindine motyvacija sukurti simbolių lentelės maišymo procesą. Primename, kad jis taip pat paremtas skaičių eilutės generavimu, tad

šifravimo metu netobulu, bet itin aukštu atsitiktinumu keičiami simboliai kurie priskirti tam tikram pozicijų eilutės blokui. Tad kriptoanalizės metu, netgi jei ir pavyktų atsekti tam tikrą sąryši tarp simbolių ir jų šifravimui reikalingų iteracijų skaičiaus, šio sąryšio sumaišius lentelę nebeliktų. Tai daro statistinę šifro analizę praktiškai neįmanoma.

5.2.2. Šifro nepriklausomumas nuo šifruojamo pranešimo

Prisiminkime, kad 3 bitų simbolių blokus, turėdami sugeneruotą pozicijų eilutę užšifruojame skaičiumi:

$$\textit{dabartinė pozicija} - \textit{buvusi pozicija},$$

t.y. iteracijų skaičiumi, reikalingu rasti patį pirmą šifruojamam simboliui priskirtą bloką, pozicijų eilutę iteruojant nuo sekančios nei prieš tai šifruojant naudoto bloko pozicijos. Akivaizdu, kad dėl to gauto šifro pasiskirstymas nebus tolygus ir bus artimesnis normaliajam skirstiniui. Šifro kokybei tai žalos nedaro, nes nors tam tikri iteracijų skaičiai šifre ir pasirodys dažniau, tačiau dėl itin aukšto pozicijų eilutės atsitiktinumo bei simbolių lentelės maišymo, visi simboliai turės beveik tokią pačią tikimybę būti užšifruotais tuo pačiu skaičiumi. Taigi, šifras turi būti nepriklausomas nuo šifruojamo pranešimo turinio. Tai, kad tokios priklausomybės nėra, galime įrodyti rasdami Pirsono koreliacijos koeficientą r:

$$r = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \overline{y})^2}},$$

kur x_i ir y_i , $i = 1, ..., n$ yra dviejų aibių tarp kurių priklausomybės ieškome elementai, $\overline{x}$ ir $\overline{y}$ -šių aibių vidurkiai, o n - duomenų skaičius šiose aibėse. Gautas koreliacijos koeficiento r (toliau vadinsime r_{kor}, tam kad jo nemaišytume su raktais r) reikšmes interpretuojame taip:

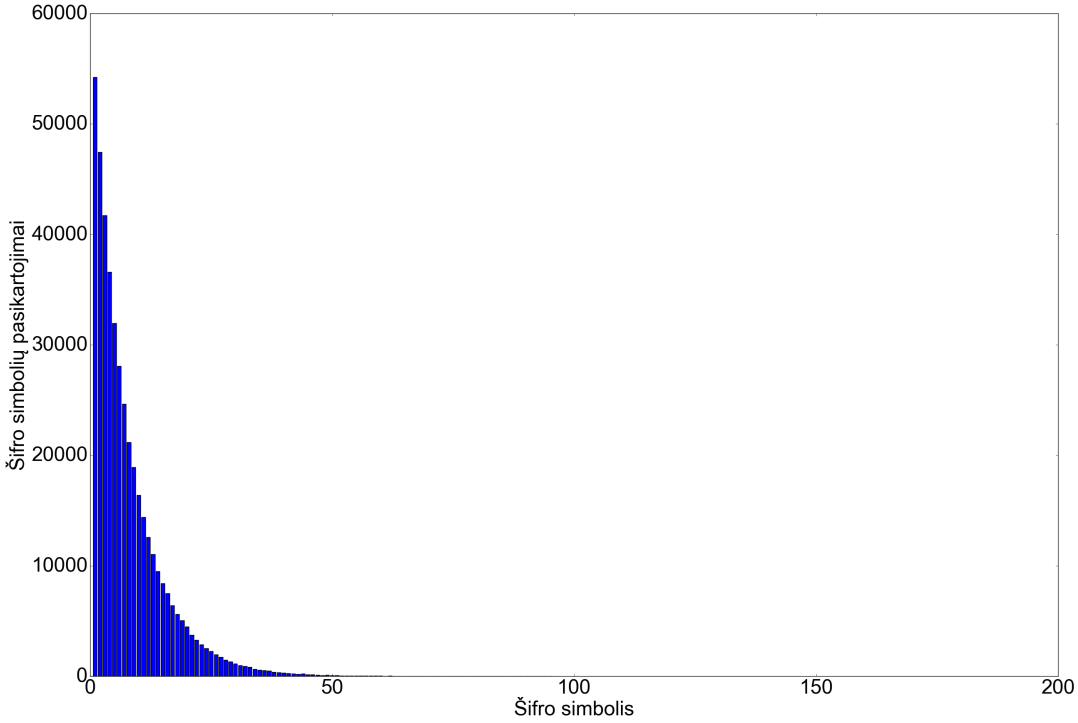
- $r_{kor} \in [0, 0.3]$ - labai silpna koreliacija,
- $r_{kor} \in (0.3, 0.5]$ - silpna koreliacija,
- $r_{kor} \in (0.5, 0.7]$ - vidutinė koreliacija,
- $r_{kor} \in (0.7, 0.9]$ - stipri koreliacija,
- $r_{kor} \in (0.9, 0.1]$ - labai stipri koreliacija.

Koreliacijos koeficientas taip pat gali įgyti ir neigiamas reikšmes $r_{kor} \in [-1, 0)$. Tokiu atveju reikšmes reikia interpretuoti tokiais pačiais intervalai, tik koreliacija tokiu atveju bus ne tiesioginė, o atvirkštinė.

Tiriame tris failus:

1. 159 Kb dydžio tekstinį failą, kurį sudaro tekstas anglų kalba ir įvairūs unikodo simboliai,
2. 488 kb dydžio tekstinį failą, kurį sudaro vien tik A raidės,
3. 1 Mbb dydžio dvejetainį failą, kurį sudaro atsitiktinai generuoti bitai,

bei užšifruotas jų versijas. Šifravimo procesu metu naudojome tokius raktus:



22 pav. 159 Kb tekstinio failo, kurį sudaro tekstas anglų kalba ir įvairūs unikodo simboliai, šifro pasiskirstymas

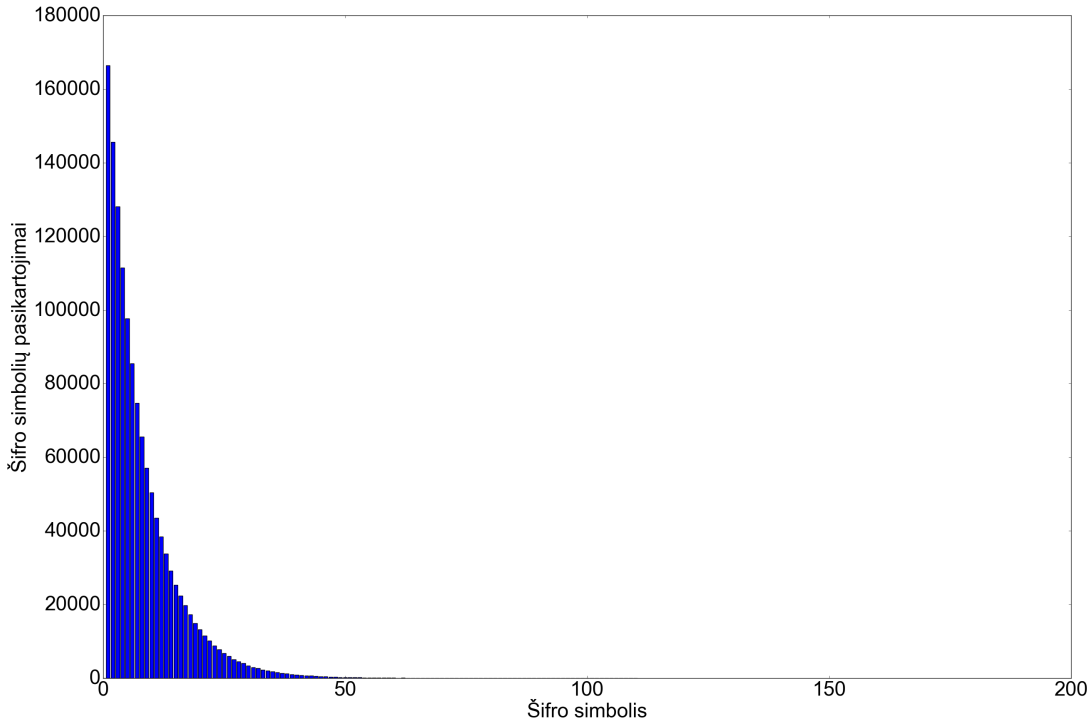
- $x_0^p = 0.84545545236985$
- $r^p = 3.99654844545$
- $x_0^m = 0.24545545236985$
- $r^m = 3.86446654844545$

22-24 pav. matome šių failų šifrų pasiskirstymus. Šifruojamų failų sudėtis kardinaliai skiriasi, tačiau gautų užšifruotų failų pasiskirstymas itin panašus.

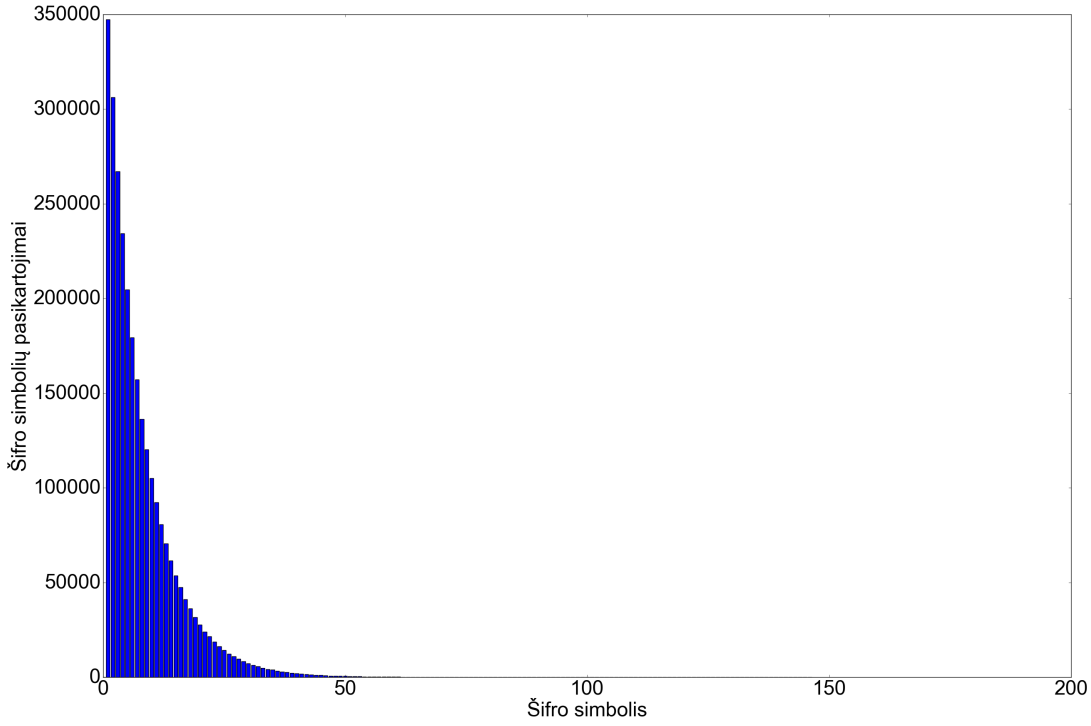
10 lentelėje pateikiame apskaičiuotus Pirsono koreliacijos koeficientus nagrinėtiems failams ir jų šifruotoms versijoms. Matome, kad gauti koeficientai yra itin maži ir tad sąryšis tarp failų ir jų šifro yra labai silpnas. Todėl teigiame, kad šifras yra nepriklausomas nuo šifruojamo failo.

10 lentelė. Pirsono koreliacijos koeficientai, paskaičiuoti nagrinėtiems failams ir jų šifruotoms versijoms.

Failas	Pirsono koreliacijos koeficientas
Tekstinis 159 kb	0.000138
Tekstinis A raidžių, 488 kb	0.006246
Dvejetainis	-0.002069



23 pav. 488 Kb tekstinio failo, kurį sudaro vien tik A raidės, šifro pasiskirstymas



24 pav. 1 Mb dvejetainio failo, kurį sudaro atsitiktinai generuoti bitai, šifro pasiskirstymas

Išvados ir rekomendacijos

Šiame darbe apibrėžiame pagrindines kriptografijos mokslo bei chaoso teorijos sąvokas ir galimą ryšį tarp jų. Pristatome jau egzistuojančių chaotinių algoritmų pavyzdžius ir pabrėžiame jų privalumus ir trūkumus. Remdamiesi atlikta analize, pasiūlome savo chaotinę kriptosistemą bei ją realizuojame praktiškai.

3 skyriuje pristatome 5 chaotinius algoritmus:

- Baptistos,
- Wong,
- Patobulintą Wong,
- Dviejų žingsnio logistinio atvaizdžio,
- Formolo-Oliveira-Sobottka.

Atlikę jų analizę pateikiame pagrindinius jų trūkumus:

- Lėtumą, egzistuojantį dėl itin daug iteracijų reikalaujančios algoritmo prigimties,
- Saugumo spragas, kylančias dėl logistinio atvaizdžio netolygumo, nekorektiško simbolių lentelės maišymo ir mažos raktų aibės.
- Mažas leistinas šifruojamo pranešimo abecėles, nepritaikomas praktiškai.
- Kai kurių algoritmų sudėtingą praktinį realizavimą.

4 skyriuje pateikiame savo siūlomos kriptosistemos šifravimo bei dešifravimo algoritmą ir praktinio realizavimo taisykles. Kriptosistema tenkina prieš pradedant darbą išsikeltą reikalavimą - ją galime panaudoti praktiškai, nes jos šifravimo algoritmas nenaudoja subjektyviai parinktos abecėlės - naudojami bitų blokai, tad šifruoti galime bet kokio tipo duomenis.

Atlikę siūlomo algoritmo spartos analizę, galime teigti, kad jis yra ženkliai spartesnis nei kiti nagrinėti algoritmai ir šifravimo ir dešifravimo proceso metu. Detalesnius rezultatus ir išvadas pateikiame 5.1 skyrelyje.

Siūlomu šifravimo algoritmu gautas šifras yra kokybiškas. Tokias išvadas priėmame po 5.2 skyrelyje atliktų tyrimų. Algoritme naudojamų chaotinių skaičių sekų negalime vadinti visiškai atsitiktinėmis, nes jos neįveikia visų NIST statistinių testų, tačiau kadangi naudojamos dvi tokios sekos - pozicijų ir maišymo eilutės generavimo metu, laikome, kad atsitiktinumas yra pakankamas. Tą patvirtina ir atliktas sąryšio tarp šifruojamo pranešimo turinio bei gauto šifro testas - tokio sąryšio nerandame.

Ateities tyrimų gairės

Chaotiniai šifravimo algoritmai yra dar gana neatrasta kriptografijos sritis. Mūsų siūlomas algoritmas yra bandymas judėti šia linkme, efektyviau išnaudojant chaotiniais atvaizdžiai generuojamas reikšmes. Visgi algoritmas vis dar yra smarkiai lėtesnis nei šiuo metu praktikoje naudojami algoritmai.

Vienas iš būdų šį procesą pagreitinti - absoliučiai atsitiktinės chaotinės skaičių eilutės generavimas. Kaip pabrėžiame darbe, siūlomu algoritmu generuota eilutės neįveikė visų statistinių atsitiktinumų testų, tad teko vykdyti nors ir daug nekainuojančią, bet papildomą operaciją - simbolių lentelės maišymą. Akivaizdu, kad jeigu chaotinė eilutė iškart būtų sugeneruojama visiškai atsitiktinai, tokios operacijos nereiktų. Darbo metu mums to nepavyko padaryti, tačiau neatmetame, kad tai įmanoma atrinkus tam tobulesnius logistinio atvaizdžio parametrus, ar radus papildomų iteracijų atžvilgiu nebrangų būdą manipuliuoti skaičių eilutės reikšmėmis, taip, kad ji taptų visiškai atsitiktine.

Literatūros šaltiniai

[1] G. Alvarez, F. Montoya, M. Romera, and G. Pastor. Cryptanalysis of dynamic look-up table based chaotic cryptosystems. *Physics Letters A*, 326:211–218, 2004.

[2] M. S. Baptista. Cryptography with chaos. *Physics Letters A*, 240:50–74, 1998.

[3] D. I. Curiac, D. Iercan, O. Dranga, and F. Dragan. Chaos-based cryptography: End of the road? *Emerging Security Information, Systems, and Technologies, 2007. SecureWare 2007. The International Conference on*, pages 74–75, 2007.

[4] Daniel Formolo, Luiz P. De Oliveira, and Marcelo Sobotkka. A competitive searching-based chaotic cipher. *International Journal of Modern Physics C*, 21(11):1377–1390, 2010.

[5] Balram Nitharwal, Mamta Rani, and Hukam Chand Saini. Improving security of the baptista's cryptosystem using two-step logistic map. *International Journal of Computer Network and Information Security*, 7(5):2062—2066, 2009.

[6] Mamta Rani and Rashi Agarwal. A new experimental approach to study the stability of logistic map. *Chaos, Solitons, Fractal*, 41(4):34–40, 2015.

[7] Andrew Rukhin, Juan Soto, Nechvatal James, Miles Smid, Elaine Barker, Stefan Leigh, Levenson Mark, Vangel Mark, David Banks, Alan Heckert, James Dray, and San Vo. A statistical test suite for random and pseudorandom number generators for cryptographic applications. *Special Publication*, 800(22), 2010.

[8] Lukas Vengelis. Chaoso teorijos taikymai duomenų šifravime. *Mokslo tiriamojo darbo projektas*, 2016.

[9] Kwok-Wo Wong. A fast chaotic cryptographic scheme with dynamic look-up table. *Physics Letters A*, 298(4):238–242, 2002.

[10] Kwok-Wo Wong, Di Xiao, and Xiaofeng Liao. Improving the security of a dynamic look-up table based chaotic cryptosystem. *Circuits and Systems II: Express Briefs*, 53(6):502–506, 2006.