ЧЕРНІВЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ЮРІЯ ФЕДЬКОВИЧА
ІНСТИТУТ ФІЗИКО-ТЕХНІЧНИХ ТА КОМП“ЮТЕРНИХ НАУК
КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП“ЮТЕРНИХ СИСТЕМ

Остапов С.Е., Жихаревич В.В., Добровольський Ю.Г.

СУЧАСНІ МЕТОДИ ТА ЗАСОБИ ЗАХИСТУ ІНФОРМАЦІЇ

МОНОГРАФІЯ

ЧЕРНІВЦІ — 2021

УДК 004.056(075.8)
ББК 32.973-0.18.10я73
О-76

Друкується за ухвалою Вченої ради Чернівецького національного університету імені Юрія Федьковича
(протокол №10 від 27 вересня 2021 року)

Рецензенти:
Доктор технічних наук, старший науковий співробітник Інституту метрології Купко О.Д.
Кандидат фізико-математичних наук, доцент Чернівецького торговельно-економічного інституту Київського торговельно-економічного університету Дрінь І.І.

Остапов С.Е., Жихаревич В.В., Добровольський Ю.Г.
Сучасні методи та засоби захисту інформації / С.Е.Остапов, В.В.Жихаревич, Ю.Г. Добровольський - Чернівці: ЧНУ, 2021. - 72 с.

У пропонованому виданні викладено навчальні матеріали до курсу «Сучасні методи та засоби захисту інформації», який читається на першому році аспірантури за освітньою програмою «Інженерія програмного забезпечення». Монографія систематизує напрацювання кафедри програмного забезпечення комп’ютерних систем Чернівецького національного університету імені Ю.Федьковича у галузі криптографічних засобів захисту інформації на основі клітинних автоматів.

Для студентів спеціальностей: 121 - «Інженерія програмного забезпечення», 122 - «Комп“ютерні науки» та «Інформатика» усіх форм навчання, інших спеціальностей, де вивчаються дисципліни захисту інформації, а також для самостійного опанування його основами.

ББК 32.973-0.18.10я73

Зміст

ВСТУП

Сьогодні ми перебуваємо на етапі побудови інформаційного суспільства. Наслідком такого стану є надзвичайно великі обсяги інформації, які передаються телекомунікаційними системами. Разом з відкритою інформацією стрімко зростають й об“єми конфіденційної інформації, що приводить до зростанням вимог до її захисту. Така тенденція простежується й в Україні, особливо після прийняття закону про захист персональних даних [1]. Однак, якщо на початку розвитку комп“ютерних систем засоби захисту використовувалися для файлів, що передаються мережею, то сьогодні на перший план виходить захист інформації у режимі реального часу.

В той же час в Україні давно увійшло у повсякдення використання смарт-карт, справа йде до електронних паспортів та електронного документообігу («Країна в смартфоні»). Ринок мобільних гаджетів насичений пристроями різного ґатунку, які також (може, навіть, більше за інших) потребують захисту даних, які передаються відкритими каналами зв“язку.

У будь-яких системах захисту як основний засіб використовуються криптографічні перетворення інформації. Це й генератори випадкових та псевдовипадкових послідовностей, функції хешування, і алгоритми симетричного та асиметричного шифрування. У цих перетвореннях завжди має бути знайдений компроміс між криптостійкістю (якістю захисту) та використаними ресурсами (швидкодією системи). Крім того, наявність кількох процесорів у сучасних гаджетах вимагає від розробників криптоалгоритмів такої архітектури, щоби роботу цих засобів можна було легко розпаралелювати.

Одним з підходів, які задовольняють усі зазначені вимоги, є клітинні автомати [2].

Розробка та дослідження криптографічних застосувань на основі клітинних автоматів (КА) останніми роками стають усе популярнішими.

Вперше висновок про можливість використання КА у криптографічних застосуваннях висловив Стівен Вольфрам [3].

Невдовзі з’явилися дослідження, спрямовані на доведення можливості реалізації конкретних застосувань [4],[5],[6],[7]. Пізніше з’явилися дослідження, що описують конкретні криптографічні алгоритми на основі КА. Перш за все, потрібно відзначити праці С.К.Росошека зі співробітниками [8] та Kumar K.J. [9], де описано симетричні системи шифрування на основі двовимірних КА; роботу Б.М. Сухініна [10], де описано розробку апаратної реалізації на FPGA швидкісного генератора двійкових псевдовипадкових

послідовностей з використанням КА; статтю [11], присвячену розробці функції хешування з використанням елементарних правил КА.

Кафедра програмного забезпечення комп“ютерних систем Чернівецького національного університету імені Юрія Федьковича також провадить наукові дослідження та розробки в галузі криптографічних засобів захисту на основі КА. Опубліковано цілий ряд статей, де подано результати розробки криптографічних генераторів псевдовипадкових бінарних послідовностей, функцій хешування та симетричних криптоалгоритмів. Ця монографія — своєрідний підсумок роботи кафедри у галузі сучасних методів захисту інформації.

Основи теорії клітинних автоматів

Клітинний автомат — двобічно нескінченна стрічка, кожна комірка якої може перебувати у певному стані. Множину станів позначимо через S, а стан комірки з номером i як $s[i]$. Спочатку усі комірки перебувають у стані $B \in S$ окрім комірок з номерами від 1 до n. Комірка з номером i, де $1 \le i \le n$ перебуває у стані x_i, де x — вхідне слово.

Правила роботи КА такі: задано число d і функція $f: S^{2d+1} \to S$. За один крок усі клітини змінюють свій стан за таким правилом: новий стан клітини i визначається $f(s[i-d], s[i-d+1], \ldots, s[i+d-1], s[i+d])$. Якщо клітина з номером 0 переходить до стану Y, то автомат допускає слово x.

Основні дослідження науковців в галузі КА спрямовані на побудові різного роду алгоритмів з використанням КА, а також знаходження початкових станів та правил міжклітинної взаємодії, оптимальних для тої чи іншої задачі.

Клітинний автомат, який ґрунтується на класичних правилах, повинен задовольняти такі критерії:

- зміна вмісту усіх клітин відбувається одночасно, в іншому випадку на кінцевий стан КА суттєво впливав би порядок взаємодії клітин;
- решітка однорідна, однак на практиці завжди розглядаються скінченні структури, що неодмінно призводить до граничних ефектів; такі проблеми, як правило, вирішуються замиканням решітки КА в замкнену структуру або іншими граничними умовами;
- взаємодія, як правило, локальна, хоча існують випадки, коли взаємодіють клітини, які знаходяться на деякій відстані одна від одної;
- множина станів клітини скінченна, оскільки новий стан КА має визначатися скінченною кількістю операцій.

Усім клітинним автоматам притаманні деякі властивості, загальні для всіх.

Стани клітин. Кожного моменту часу клітина набуває одного стану з усієї множини станів. Залежно від правил міжклітинної взаємодії КА може наути наступного стану. На практиці для цього використовують клітини з необхідною для заданої задачі алгебраїчною структурою, наприклад, лінійні КА.

Геометрія. Скінченне або нескінченне число клітин у просторі довільної розмірності можуть розташовуватися різноманітним чином. Динамічні КА можуть змінювати свою геометрію з часом. Існують також неоднорідні КА, геометрія яких різна у різних частинах простору.

Сусідство. Поняття сусідства є основним для КА, від якого залежить практично уся робота. Більшою мірою сусідство визначається геометрією КА. Існують яскраво визначені геометрії сусідства, однак, така важлива властивість не обмежується лише класичними підходами до понять сусідства. Для різних задач можлива зміна сусідів та підходів до сусідства.

Правила взаємодії. Стан КА змінюється в залежності від правил міжклітинної взаємодії. Правила взаємодії можуть бути однаковими для усіх клітин КА протягом усього часу. Якщо це не так, КА називаються неоднорідними. Також правила взаємодії можуть змінюватися в часі за певним правилом або мати випадкову природу.

Згідно з класифікацією С.Вольфрама [12] КА можна розподілити по таких класах:

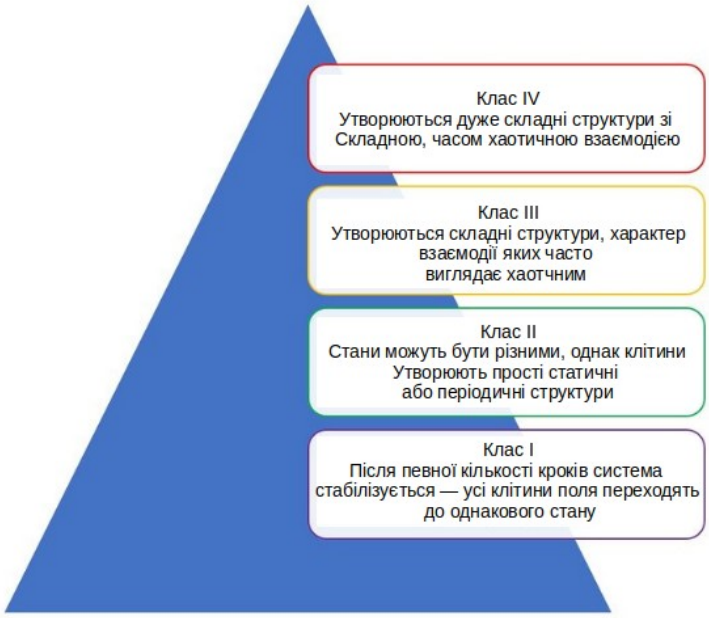


Рисунок 1 — Класифікація клітинних автоматів за С.Вольфрамом

Існують й інші класифікації КА. Наприклад, на рис. 2 подану класифікацію за Д.Епштейном [13].

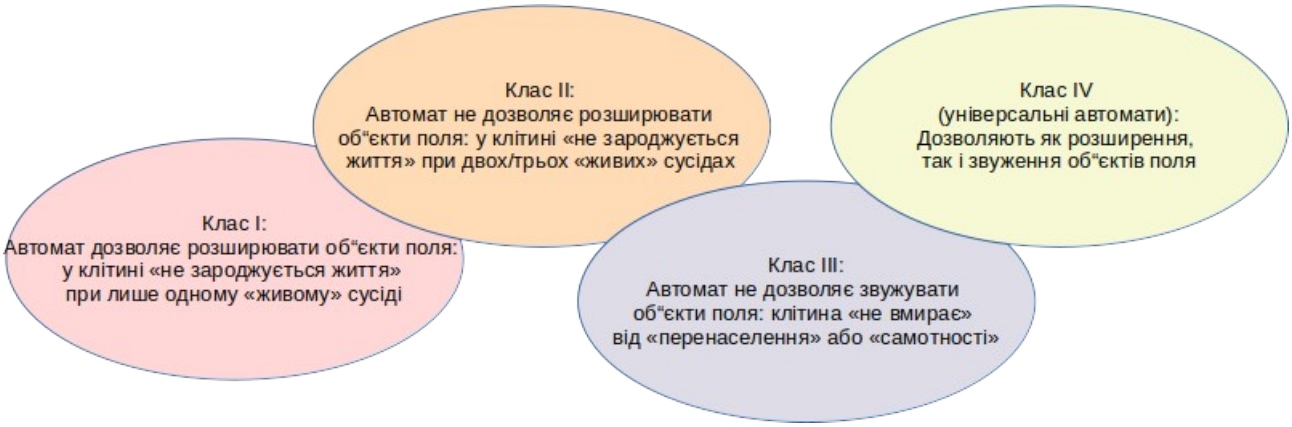


Рисунок 2 — Класифікація клітинних автоматів за Д.Епштейном

Однак, ця класифікація також мала серйозні проблеми та не дуже задовольняла призначенню. Існують й інші класифікації, які уточнюють та модифікують існуючі класифікаційні системи КА задля рішення багатьох їхніх проблем.

Одновимірні клітинні автомати

Елементарний клітинний автомат – це найпростіший можливий варіант КА, він працює в одновимірному просторі (стрічці), його клітинки можуть мати лише два стани (0 або 1), а правило визначення наступного стану клітинки визначається лише її поточним станом та станом її двох найближчих сусідів. Для найменування таких КА використовують так званий код Вольфрама — систему йменування, запропоновану С.Вольфрамом у 1983 році. Ця система ґрунтується на спостереженні, що таблиця, яка визначає новий стан кожної комірки у КА, як функція станів її сусідніх комірок, може інтерпретуватися як число з k-цифр у S-арній позиційній системі числення, де S — кількість станів, яку може мати кожна комірка у КА, $k=S^{2n+1}$ — кількість конфігурацій сусідніх комірок, а n — радіус околу комірки.

Наприклад, кожна можлива конфігурація поточної клітинки та її сусідів записується в такому порядку: 111, 110, 101, 100, 011, 010, 001, 000. Далі виконується “взаємодія” клітинок, а її результат, тобто наступний стан клітинки, записується під кожною конфігурацією. Вісім можливих станів клітинки розглядаються як 8-бітове число від 0 до 255, утворюючи назву відповідного правила.

Кожна можлива конфігурація поточної клітинки та її сусідів записується в такому порядку: 111, 110, 101, 100, 011, 010, 001, 000. Потім під кожною конфігурацією записується наступний стан клітинки, в результаті 8 отриманих

бітів нового стану формують десяткове число від 0 до 255 у двійковому представленні. Це число і буде правилом КА. Якщо значення «1» подати чорним квадратиком, а «0» - білим, то графічно це буде виглядати так, як подано на рис. 3 для правила 90.

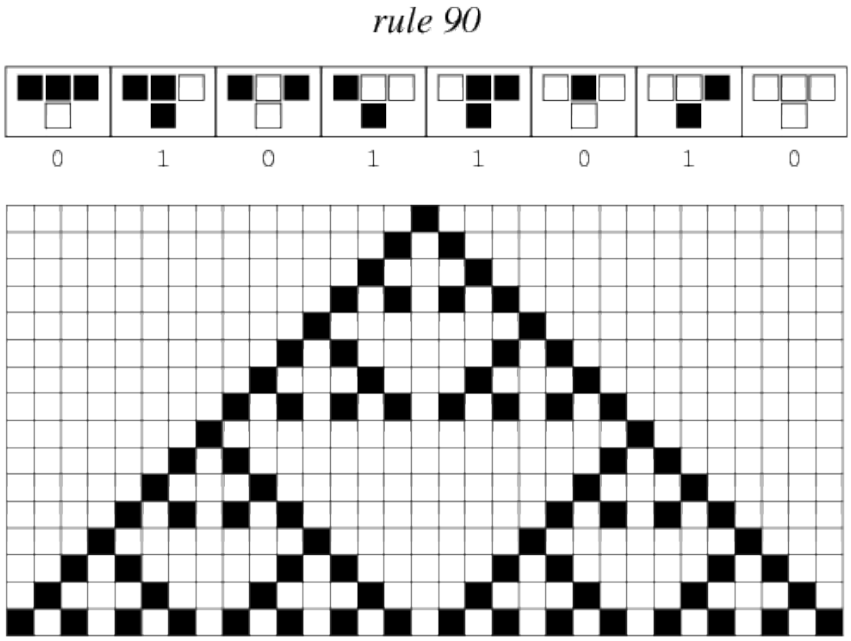


Рисунок 3 — Графічне зображення правила 90 та серветка Серпінського для нього

Можна побудувати аналогічні графічні зображення для інших правил, наприклад, для популярного правила 30, яке найбільше використовується у криптографічних перетвореннях (рис.4).

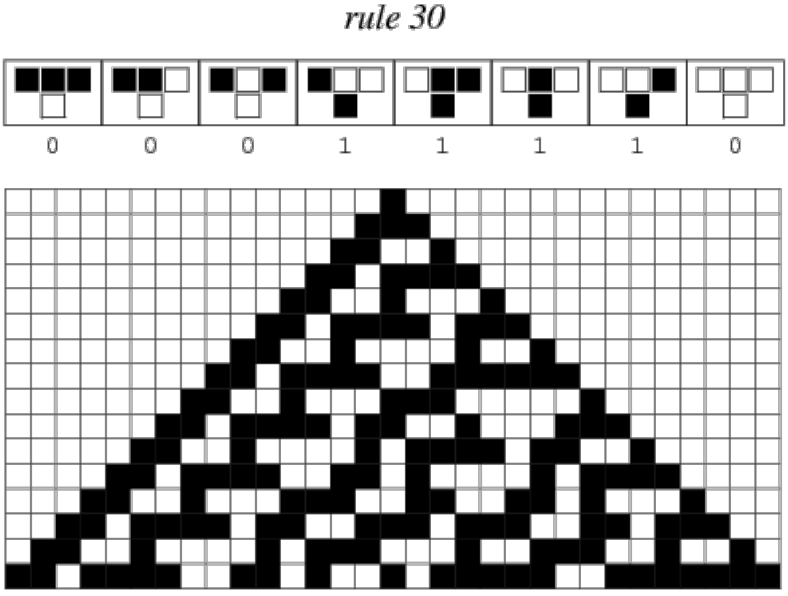


Рисунок 4 — Графічне зображення правила 30 та серветка Серпінського для нього

Якщо за стартовий стан КА з (2*n*+1) клітин взяти усі нулі, а центральну комірку виставити в “1”, а потім з такого стану здійснити 16 циклів перетворень

(16 епох), зображуючи “1” та “0” чорними та білими квадратиками відповідно, то отримаємо так звану “серветку Серпінського”, яка демонструє еволюцію КА згідно того чи іншого правила взаємодії. Такі серветки стали вже стандартом для елементарних КА. Саме їх і зображено на рис.3-4 для правил 90 та 30 відповідно. Детальніше познайомитися з елементарними правилами для одновимірних КА та відповідними графічними зображення можна у [14].

Двовимірні клітинні автомати

У двовимірному випадку клітинно-автоматне поле описується двовимірним масивом. У такому випадку кожна клітина має вісім сусідів. Для уникнення граничних ефектів поле, як і в одновимірному випадку, згортається в тор. Тоді можна записати таке співвідношення для усіх клітин КА:

$$y'[i,j]=F(y[i,j],y[i-1,j],y[i-1,j+1],y[i,j+1],y[i+1,j+1],y[i+1,j],y[i+1,j-1],y[i,j-1],y[i-1,j])$$

Таке сусідство називають околом Мура [15]. Графічно його можна зобразити наступним чином:

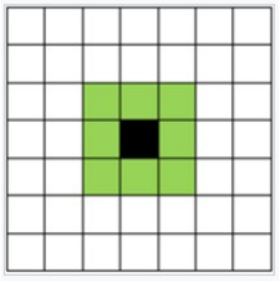


Рисунок 5 — Графічне зображення сусідства за Муром. Зеленим кольором виділено вісім сусідів центральної клітини

Інакшим чином, якщо клітина $y[i,j]$ має лиш чотири сусіда, з якими вона взаємодіє, то для довільної клітини ми можемо записати:

$$y'[i,j]=F(y[i,j],y[i-1,j],y[i,j+1],y[i+1,j],y[i,j-1])$$

Таке сусідство називають околом фон Неймана [16]:

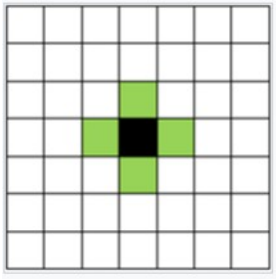


Рисунок 6 — Графічне зображення сусідства за фон Нейманом. Зеленим кольором позначено найближчих сусідів центральної клітинки

Для двовимірних КА такі моделі сусідства — найбільш популярні й використовуються буквально усюди.

Одним з найяскравіших прикладів двовимірних КА є гра “Життя” [17]. Автомат, що моделює цю гру, складається з квадратного клітинно-автоматного поля. Клітини, що складають КА, можуть знаходитися у двох станах: 1 (“живий”) та 0 (“мертвий”). Функція переходів реалізує такі умови:

- якщо деяка клітина “мертва” (тобто знаходиться у стані 0), то вона може перейти до стану 1 (“жива”) при умові, що вона має три “живих” сусіди;
- якщо деяка клітина “жива”, то вона залишається “жити” лише коли вона має два або три “живих” сусіди; інакше — вона “помре”.

Метою гри є спостереження за розвитком популяції клітин при різних початкових умовах.

Прийнято вважати, що прості КА з сусідством за фон Нейманом можуть демонструвати більшу різноманітність розвитку, ніж КА з більшим числом сусідів. Можливо, це пояснюється принципово іншим способом передавання інформації діагональним сусідам.

Можна, однак, значно покращити опис динамічних явищ за допомогою клітинних автоматів, якщо дозволити так званий асинхронний режим, тобто режим, коли взаємодія відбувається не одночасно. Такий режим роботи КА дуже нагадуватиме реальну взаємодію у багатьох динамічних середовищах. Такі КА будемо називати асинхронними. За допомогою такого підходу можна описати багато реальних динамічних об’єктів.

Метод асинхронних неперервних клітинних автоматів

Основна відмінність цього методу полягає в особливостях організації правил локальних взаємодій. Обчислення стану двох клітин у наступний момент часу нагадує процес числового розв’язку диференціальних рівнянь явними методами. У зв’язку з цим, деякі явні схеми розв’язку диференціальних рівнянь іноді називають неперервними КА [18]. Існують також КА-моделі, для яких правила локальних взаємодій записуються як звичайні диференціальні рівняння: моделі класу КА-ЗДР [19].

Розглянемо метод асинхронних КА, який буде використовуватися далі [20]. Структуру КА подано на рисунку 7.

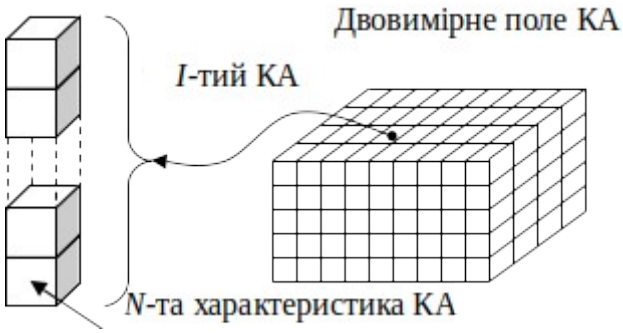


Рисунок 7 — Структура клітинного автомата

Клітинний автомат являє собою сукупність деяких характеристик, які, загалом, набувають неперервних дійсних значень. Позначимо через c_j^i величину j -ї характеристики *i*-го КА. Кожна характеристика може мати певний фізичний зміст, наприклад, концентрація деякої речовини, напруженість поля, енергетична характеристика тощо. Очевидно, що загальна кількість таких характеристик *N* безпосередньо пов’язана зі складністю системи, що моделюється. Можна стверджувати, що *N* — кількість параметрів мікрооб’єктів, які необхідно враховувати при моделюванні певної системи.

Процес моделювання в такому випадку відбуватиметься згідно такого алгоритму:

- На КА-полі випадковим чином визначаються координати деякої клітини c^i, причому усі клітини КА-поля є рівноймовірними з точки зору їхнього вибору;
- Випадковим чином обирається сусідня клітина. Тут можуть використовуватися зовсім різні схеми сусідства: Мура, фон Неймана та інші.
- Дві обрані таким чином клітини взаємодіють між собою, тобто з використанням початкових (в момент часу *t*) значень їхніх характеристик та правил взаємодії обчислюються значення цих характеристик в наступний момент часу *t*+1.

Відзначимо, що така схема випадкового визначення локалізації взаємодіючих клітин дозволяє при її реалізації на комп’ютері використовувати одну “часову площину”, тобто один масив для збереження інформації про усе КА-поле, замість двох, що зекономить трохи ресурсів та підвищить швидкість роботи алгоритму. Окрім цього, така схема, на наш погляд, найбільш зручно описує реальні динамічні процеси, в яких локальна взаємодія відбувається саме асинхронно, невпорядковано і незалежно.

Правила КА-взаємодії можна записати у вигляді системи ітераційних функцій такого вигляду:

$$\begin{cases} c_j^i = F_j^i(c_1^i, c_2^i, \ldots, c_N^i, c_1^k, c_2^k, \ldots, c_N^k); \\ c_j^k = F_j^k(c_1^k, c_2^k, \ldots, c_N^k, c_1^k, c_2^k, \ldots, c_N^k); \end{cases} \quad (1)$$

де j=1,2,...,N. Явний вигляд функції F у формулі (1) залежить від природи конкретної системи, що моделюється, і напряму пов’язаний із можливістю декомпозиції складних процесів взаємодії на елементарні акти. Це, у свою чергу, означає, що в загальному випадку, треба визначати функцію F як суперпозицію елементарних актів взаємодії певного фізичного змісту. Наприклад, це може бути зміна деякої характеристики, градієнтне чи дифузійне зміщення, перетворення однієї характеристики у іншу тощо. Інтенсивність різних елементарних актів взаємодії є різною, що можна враховувати за допомогою деяких параметрів — “коефіцієнтів активності”.

Моделювання еволюційних процесів у системах виконувалося з врахуванням таких міркувань. Системи, що мають здатність до самоорганізації та еволюції, мають бути, перш за все, відкритими. Відкритість системи передбачає вільний обмін речовиною та енергією з оточуючим середовищем, причому фізичні властивості частинок можуть бути зовсім різними. Таким чином, при моделюванні таких систем необхідно передбачити можливість випадкового “виникнення” частинок з новими властивостями. Оскільки модель поведінки частинок у методі асинхронних КА повністю описується глобальними правилами взаємодії (1), то виникає необхідність у модифікації цього методу деяким алгоритмом, який описує локальні та змінний характер міжклітинних взаємодій, інтенсивність яких можна регулювати за допомогою коефіцієнтів активності. Наприклад, глобальне правило взаємодії може містити функцію переміщення частинок, але якщо відповідний коефіцієнт активності деякої клітини набуде нульового значення, частинка буде залишатися у спокої. Можливість зміни локальних правил взаємодії може бути надана хаотичною зміною цих коефіцієнтів активності клітин.

Для демонстрації можливості КА-моделювання на основі класу неперервних асинхронних КА розглянемо декілька відомих моделей: зміну деякої величини за експоненційним законом, явище теплопереносу, дифузії та інтерференції, а також поведінку дискретних систем на основі гри Конвея “Життя”.

Експоненційна залежність

Як відомо, процес експоненційної зміни деякої величини описується таким диференціальним рівнянням: $\frac{d}{dt} p(t) = c \cdot p(t)$ з початковими умовами

$p(0) = p_0$, розв'язок якого має вигляд: $p(t) = p_0 \exp(c \cdot t)$. Якщо $p_0 = 1$, $c = 1.2$, то графік функції *p*(*t*) буде мати вигляд, як на рисунку 8:

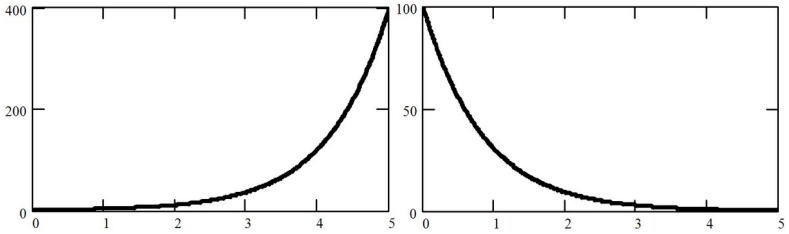


Рисунок 8 — Результати моделювання експоненційної залежності методом неперервних асинхронних КА

У випадку КА-моделювання такого процесу на двовимірному полі 100x100, система (1) буде мати вигляд ітераційної функції $c_1^i = c_1^i + 0.01 \cdot c_1^i$, де $i = 1,2$ - індекси двох клітин, що взаємодіють. Зростання параметра c_1 з часом повністю узгоджується з графіком на рисунку 8 (ліворуч), якщо за *t*=5 прийняти 3 млн. міжклітинних взаємодій. Аналогічним чином можна описати й процес експоненційного зменшення, зображений на рисунку 8 праворуч.

Явище теплопереносу

Як відомо, розподіл температури у стаціонарному режимі для двовимірного випадку описується рівнянням Лапласа такого вигляду: $\frac{\partial^2 T(x,y)}{\partial x^2} + \frac{\partial^2 T(x,y)}{\partial y^2} = 0$. На рисунку 9 ліворуч подано розв'язок цього рівняння для пластини розміром *DxD* з такими граничними умовами: на периметрі пластини $T(x,y) = 0^\circ C$, $T(0.5D, 0.5D) = -100^\circ C$, $T(0.9D, 0.5D) = 100^\circ C$, $T(0.75D, 0.75D) = 100^\circ C$, $T(0.75D, 0.25D) = 100^\circ C$. Діапазон температур між ізолініями — $10^\circ C$. Для моделювання явища методом асинхронних неперервних КА система (1) перетворюється в ітераційне рівняння $c_1^i = c_1^i + 0.5 \cdot s \cdot (c_1^2 - c_1^1)$, де $i = 1,2$, $s = +1$ при $i = 1$, а $s = -1$ при $i = 2$. Саме результати такого моделювання й подано на рисунку 9 праворуч.

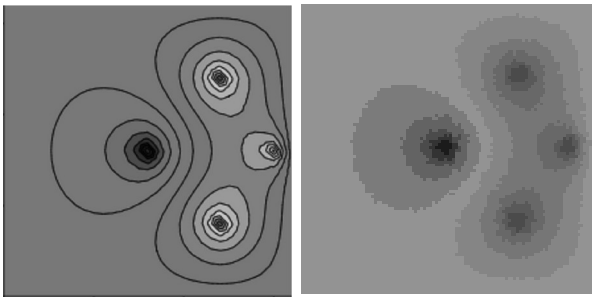


Рисунок 9 — Моделювання явища теплопереносу. Ліворуч — ізолінії розв’язку рівняння Лапласа; праворуч — результати моделювання процесу КА-методом з параметрами, вказаними у тексті

Моделювання дифузії

Процес дифузії також може описуватися рівнянням Лапласа, яке у нестаціонарному режимі для двовимірного випадку має такий вигляд:

$$D \cdot \left(\frac{\partial^2 C(x,y,t)}{\partial x^2} + \frac{\partial^2 C(x,y,t)}{\partial y^2} \right) = C(x,y,t) \qquad (2)$$

тут $C(x,y,t)$ - концентрація речовини в точці x, y в момент часу t, D — коефіцієнт дифузії.

У випадку КА-моделювання можна розглядати неперервну та дискретну моделі дифузії. Для неперервної моделі з максимальним коефіцієнтом дифузії система (1) буде мати вигляд, що повністю відповідає випадку теплопереносу: $c_1^i = c_1^i + 0.5 \cdot s \cdot \left(c_1^2 - c_1^1\right)$, причому множник 0,5 визначає коефіцієнт дифузії і змінюється від 0 до 0,5. Якщо прирівняти його до одиниці, то отримаємо функцію, що описує дискретну дифузію $c_1^i = c_1^i + s \cdot \left(c_1^2 - c_1^1\right)$. У цьому випадку коефіцієнт дифузії повинен визначатися деякою ймовірнісною функцією, яка набуває значень 0 чи 1 з деякою ймовірністю.

На рис. 10 подано картину дифузійної динаміки на полі клітин 100x100. Ліворуч — початковий стан дифундуючих частинок, в центрі - стан дифундуючих частинок у випадку неперервної, а праворуч — дискретної дифузії у момент часу, що відповідає 3 млн міжклітинних взаємодій. Граничні умови такі: $C(x,y,t)=0$ при $x=0$ та $x=100$ для будь-якого y та t.

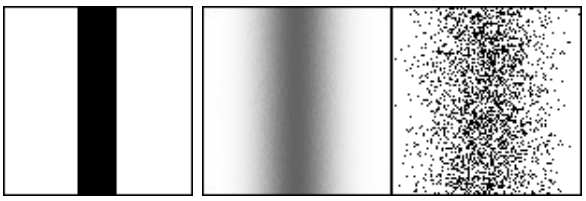


Рисунок 10 — Клітинно-автоматне моделювання дифузійної динаміки

Невелике доповнення цієї моделі дозволяє виконати КА-моделювання дендритного росту, результати якого подано на рис. 11. Тут розміри КА-поля — 200x200. Для моделювання такого процесу необхідно увести додатковий параметр, який має враховувати переміщення дифундуючих частинок. У випадку КА-моделювання такого процесу система (1) буде мати вигляд:

$$\begin{cases} c_1^i = c_1^i + s \cdot \left(c_1^2 - c_1^1\right) \cdot \left(1 - c_2^1\right) \cdot \left(1 - c_2^2\right) \\ c_2^i = c_2^i + c_2^{i+s} \cdot \left(1 - c_2^i\right) \cdot c_1^i \end{cases} \quad (3)$$

Ця система описує 2-шарову структуру двовимірного КА-поля. Шар C_1 відповідає за наявність/відсутність дискретної частинки, C_2 — за можливість пересування частинок по КА-полю.

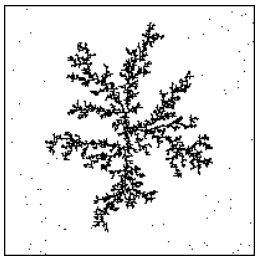


Рисунок 11 — Результати КА-моделювання дендритного росту за рахунок дифундуючих дискретних частинок

Моделювання явища інтерференції

Рис. 12 демонструє результати КА-моделювання інтерференції двох хвиль. Ліворуч подано результати інтерференції двох синусоїдальних коливань. Праворуч — результат КА-моделювання інтерференції. Відстань між джерелами хвиль — 20, гранична умова у джерелах коливань: $c_1^j = c_1^k = \sin(t)$, де j та k — індекси джерел; $t = t + 0.0000005$ на кожному черговому кроці взаємодій між клітинами.

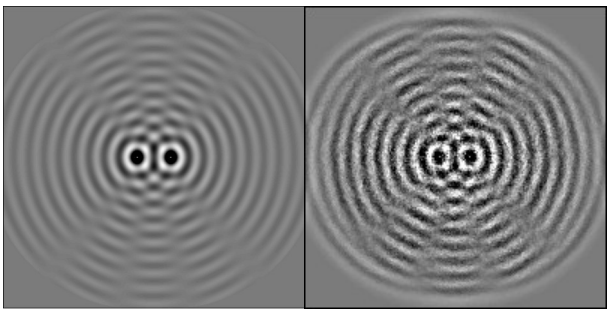


Рисунок 12 — Результати моделювання двохвильової інтерференції (ліворуч); праворуч — результати КА-моделювання інтерференції

Моделювання дискретних систем на прикладі гри Конвея “Життя”

Продемонструємо можливості методу неперервних асинхронних КА при моделюванні динаміки дискретних систем на прикладі відомої гри Конвея “Життя”, і виконаємо моделювання періодичної поведінки утворення “турбіна”, поданого на рис. 13.

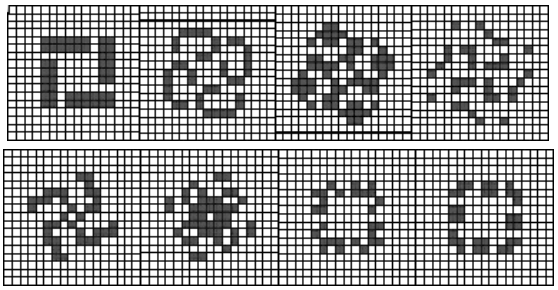


Рисунок 13 — Періодичні структури, які утворює “турбіна”

У випадку КА-моделюваня цього процесу методом неперервних асинхронних КА система (1) матиме вигляд:

$$\begin{cases} c_1^i = c_1^i + c_1^i \cdot F1\left(c_2^i\right) \cdot P\left(c_3^i\right) + \left(1 - c_1^i\right) \cdot F2\left(c_2^i\right) \cdot P\left(c_3^i\right) \\ c_2^i = \left(c_2^i + c_1^{i+s}\right) \cdot \left(1 - P\left(c_3^i\right)\right) \\ c_3^i = \left(c_3^i + 1\right) \cdot \left(1 - P\left(c_3^i\right)\right) + 0.5 \cdot s \cdot \left(c_3^2 - c_3^1\right) \cdot D\left(c_3^1\right) \end{cases}$$

Утворена система описує 3-шарову структуру двовимірного поля КА. Шар C_1 описує ознаку жива/нежива клітина; C_2 — шар обчислення кількості сусідів, C_3 — шар тимчасової синхронізації. Функції у цій системі мають такий зміст: $F1\left(c_2^i\right)$ - функція загибелі клітини (при $round\left(8 \cdot c_2^i / t_{пор}\right) = 2$) чи $round\left(8 \cdot c_2^i / t_{пор}\right) = 3$; функція $F1\left(c_2^i\right) = 0$ та $F1\left(c_2^i\right) = -1$ в усіх інших випадках; $F2\left(c_2^i\right)$ - функція народження клітини (при $round\left(8 \cdot c_2^i / t_{пор}\right) = 3$ функція $F2\left(c_2^i\right) = 1$ та $F2\left(c_2^i\right) = 0$ - в усіх інших випадках); $P\left(c_3^i\right)$ - порогова функція, що визначає момент зміни стану клітини ($P\left(c_3^i\right) = 1$ при $c_3^i \geq t_{пор}$, $P\left(c_3^i\right) = 0$ при $c_3^i < t_{пор}$); $D\left(c_3^1\right)$ - функція, що забезпечує часову синхронізацію у різних точках КА-поля в

моменти, віддалені від порогового значення ($D(c_3^1)=1$ при $t_{пор}\cdot 1/4 < c_3^1 < t_{пор}\cdot 3/4$, $D(c_3^1)=0$ в усіх інших випадках). На відміну від попередніх прикладів, коли використовувалася 4-клітинна схема сусідства, у цьому випадку використовувалася 8-клітинна схема. Фрагменти асинхронної покрокової (56 кроків) зміни початкового вигляду “турбіни” подані на рис. 14.

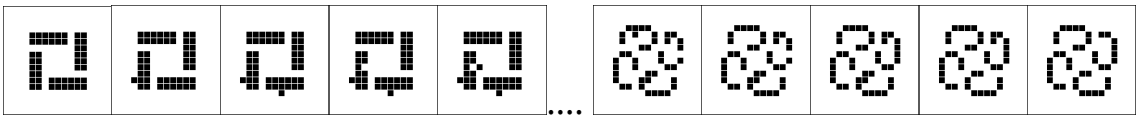


Рисунок 14 — Переходи утворення “турбіна” за кожні 56 кроків моделювання

Моделювання явищ самоорганізації та еволюції систем

Розглянемо модель деякої системи, яка є модифікацією класичної системи “хижак — жертва”. Суть модифікації полягає у розширенні функціональних можливостей взаємодії. Для прикладу наведемо таку аналогію: нехай “жертвою” у системі буде ґрунтова вода, а “хижаком” — особини, яким для виживання необхідно споживати воду. Нехай рівень ґрунтових вод самовільно зростає. Хай існує також градієнтне протікання вод. Для споживачів води передбачимо дві можливості: можливість хаотичного пересування по полю у пошуках води та можливість “вкопуватися” у ґрунт. Визначимо також таку властивість ґрунту як самовільне вирівнювання, тобто чим глибше “вкопується” споживач, тим інтенсивніше вирівнюється ґрунт.

Проаналізуємо цю модель. При повній відсутності споживачів води та будь-якому рівні ґрунтових вод, вони з часом піднімуться на поверхню ґрунту. З появою споживачів води почнеться процес її поглинання і, відповідно, інтенсивного збільшення їхньої кількості. Незалежно від того, якими функціями володіють споживачі, їхня кількість буде збільшуватися. Однак, при інтенсивному поглинанні води буде зростати й кількість загиблих споживачів. Разом з цими процесами буде також відбуватися процес конкуренції, суть якого полягає у більшій спроможності “вижити” для тих споживачів, які можуть заглиблюватися у ґрунт, оскільки ті, які просто пересуваються, врешті решт опиняться на поверхні, де кількість “їжі” буде недостатньою для збільшення популяції. Якщо на початковій стадії моделювання визначити лише споживачів, що пересуваються, то динаміка системи буде мати автоколивний характер, аналогічний до класичної системи “жертва — хижак”. Якщо ми “вмикаємо” процес конкуренції, то серед споживачів виживуть лиш ті, у яких переважає

властивість заглиблення у ґрунт, і система набуває стаціонарного стану у вигляді “комірчастої” структури, зображеної на рис. 15.

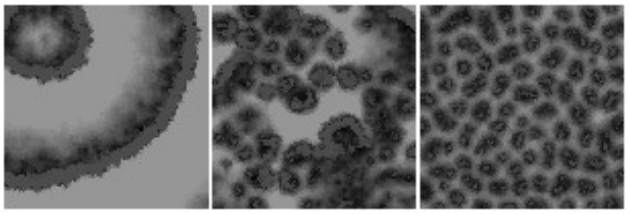


Рисунок 15 — Еволюція хвилеподібної структури у комірчасту

Функції взаємодії подано системою (4):

$$\begin{cases} c_1^i = c_1^i + k_{11}\cdot c_1^i + k_{12}\cdot s\cdot\left(c_1^2 - c_1^1\right) + k_{13}\cdot c_1^i\cdot c_4^i\cdot\left(c_1^{\text{и}} - c_1^i\right)\times \\ \quad \times\left[1 - 1/\left(1+\exp\left(10\cdot\left(c_1^i - c_1^{пор}\right)\right)\right)\right] \\ c_2^i = c_2^i + k_{21}\cdot\left(c_2^{\text{и}} - c_2^i\right) + k_{22}\cdot s\cdot\left(c_2^2 - c_2^1\right) + k_{23}\cdot c_1^i\cdot c_2^i + \\ \quad + k_{24}\cdot c_2^i\cdot\left|c_4^2 + c_2^2 - c_4^1 - c_2^1\right| \\ c_3^i = c_3^i + k_{31}\cdot\left(c_3^{\text{и}} - c_3^i\right) + k_{32}\cdot s\cdot\left(c_3^2 - c_3^1\right) + k_{33}\cdot\left(c_2^i - c_3^i\right)/ \\ \quad /\left(1+\exp\left(10\cdot\left(c_2^i - c_3^i\right)\right)\right) + k_{34}\cdot c_3^i\cdot\left|c_4^2 + c_2^2 - c_4^1 - c_2^1\right| \\ c_4^i = c_4^i + k_{41}\cdot s\cdot\left(c_4^2 + c_2^2 - c_4^1 - c_2^1\right) + k_{42}\cdot c_4^i\cdot c_1^i + k_{43}\times \\ \quad \times\left(c_2^i - c_3^i\right)/\left(1+\exp\left(10\cdot\left(c_2^i - c_3^i\right)\right)\right) \end{cases}, \quad (4)$$

Тут $i = 1,2$ - індекси двох клітин, що взаємодіють; $s = +1$ при i=1, $s = -1$, якщо i=2. Система (4) описує 4-шарову структуру КА-поля. Шар C_1 описує концентрацію споживачів води, C_2 — рівень ґрунту, C_3 — рівень ґрунтових вод, C_4 — рівень поверхневих вод. k_{ij} - коефіцієнти активності, які набувають таких значень: $k_{11} = -0.1$ - інтенсивність самовільного зменшення концентрації споживачів; k_{12} - інтенсивність градієнтного пересування споживачів; $k_{13} = 0.0001$ - інтенсивність збільшення кількості споживачів за рахунок живлення поверхневими водами; $c_1^{\text{и}} = 100$ - гранична концентрація; $c_1^{пор} = 10$ - порогове значення концентрації, при якому відбувається інтенсивне збільшення; $k_{21} = 0.005$ - інтенсивність градієнтного переміщення ґрунтових вод; $k_{33} = 1$ - інтенсивність зменшення ґрунтових вод за рахунок їхнього виходу на поверхню; $k_{34} = -0.01$ - інтенсивність зменшення рівня ґрунтових вод за рахунок ерозії ґрунту; $k_{41} = 0.5$ - інтенсивність переміщення поверхневих вод; $k_{42} = -0.005$ - інтенсивність зменшення рівня поверхневих вод за рахунок споживання; $k_{43} = -1$ - інтенсивність збільшення рівня поверхневих вод за рахунок перетворення ґрунтових.

Коефіцієнти активності k_{12} та k_{23} змінюються хаотичним чином. Ця зміна обмежується деяким законом збереження “трудової активності”: частинка або зміщується, або заглиблюється у ґрунт. Позначимо через ξ випадкову величину, що змінюється від 0 до 1. Тоді коефіцієнти активності k_{12} та k_{23}

можна подати у вигляді: $k_{12}=0.5\cdot\xi$, $k_{23}=0.01\cdot(1-\xi)$. Будемо вважати, що при зменшенні концентрації споживачів ξ набуває ненульового значення, наприклад, пропорційно відношенню різниці концентрації споживачів “до” та “після” взаємодії до максимально можливої у двох клітинах, що взаємодіють. Якщо ж концентрація споживачів збільшується, то ξ=0. Зрозуміло, що значення ξ у кожній клітині КА-поля може бути різним, що якраз і приводить до конкуренції споживачів та, як це видно з рис. 15, найбільше “виживають” частинки, для яких $\xi\to 0$.

Система (4) враховує також ерозію ґрунту при зміщенні поверхневих вод. Оскільки поверхневі води зміщуються вздовж градієнта поверхні ґрунту і розмивають його, виникнення випадкового заглиблення призведе до самовільної зміни його розмірів за рахунок взаємодії таких механізмів як: збільшення поверхневого потоку води, розмиття ґрунту потоком води, самовільне вирівнювання ґрунту. Результат моделювання подібної ситуації подано на рис. 16.

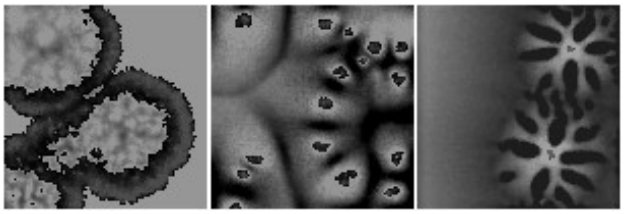


Рисунок 16 — Еволюція хвилеподібної структури у розгалужену

Функції взаємодії для цього випадку визначаються тією ж системою (4), але значення коефіцієнтів активності дещо інші: $k_{11}=-0.3$, $k_{12}=0.5\cdot\xi$, $k_{13}=0.0001$, $k_{21}=0.005$, $k_{22}=0.03$, $k_{23}=0.01\cdot(1-\xi)$, $k_{24}=-0.007$ $k_{31}=0.015$, $k_{32}=0.05$, $k_{33}=1$, $k_{34}=-0.007$, $k_{41}=0.5$, $k_{42}=-0.0003$, $k_{43}=-1$.

Як видно з рис.16, у процесі конкуренції споживачів, “виживають” лише ті, які переважно заглиблюються у ґрунт, але у найбільш вигідних умовах для “виживання” перебувають ті споживачі, навколо яких утворилася розгалужена структура припливу живильного середовища.

Розглянемо подані результати моделювання і, зокрема, побудову функцій взаємодії. Загальний вигляд системи (4) обумовлено особливостями взаємодії, актуальними для цієї системи. Нагадаємо, що основна мета цих обчислень полягає в отриманні самоорганізованих розгалужених структур. У таких системах повинні взаємодіяти як мінімум два параметри структури, які схематично показані на рисунку 17. Ця схема демонструє необхідні умови для виникнення самоорганізованих структур: система повинна бути відкритою і

далекою від стану термодинамічної рівноваги. На схемі 17 зображено взаємодію в системі як позитивні та негативні зворотні зв’язки, які, у загальному випадку, бувають нелінійними.

Рисунок 17 — Взаємодія у самоорганізованій системі

У даній моделі як параметр-джерело виступає деяке живильне середовище, яка може самовільно збільшуватись при відсутності споживачів, які, у свою чергу, виступають як параметр-стік. Зростання живильного середовища (Д↑) призводить до збільшення споживачів (С↑), що, у свою чергу, приведе до зменшення середовища (Д↓), і як наслідок, до зменшення кількості споживачів (С↓). Цей алгоритм наведено на рисунку 18, який описує процес саморегуляції утворених структур.

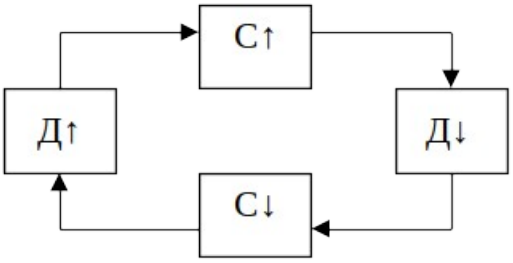


Рисунок 18 — Схема алгоритму саморегуляції структур

У випадку хвилеподібної структури Д — живильне середовище, С — середовище споживачів. У випадку розгалуженої структури Д — інтенсивність градієнтного стоку живильного середовища до комірки (пропорційна віддалі між комірками), С — кількість нерухомих споживачів (пропорційна розміру комірки).

Описані алгоритми реалізуються функціями взаємодії (4). Явний вигляд цих функцій визначається суперпозицією елементарних актів взаємодії, які описують основні залежності у системі. Таким чином, функції (4) можуть бути подані у наступному вигляді:

$$\begin{cases} c_1 = c_1 + F_1(c_1) + F_2(c_1) + F_4(c_1, c_4) \\ c_2 = c_2 + F_3(c_2) + F_2(c_2) + F_5(c_1, c_2) + F_6(c_2, c_4) \\ c_3 = c_3 + F_3(c_3) + F_2(c_3) + F_7(c_2, c_3) + F_6(c_2, c_3, c_4) \\ c_4 = c_4 + F_8(c_2, c_4) + F_5(c_1, c_4) + F_7(c_2, c_3), \end{cases} \quad (5)$$

де F_i - функції, що описують елементарні акти взаємодії; F_1 - самовільне зменшення; F_2 - дифузійне переміщення; F_3 - самовільне зростання до насичення; F_4 - зростання до насичення за рахунок поглинання; F_5 - зменшення за рахунок споживання; F_6 - зменшення за рахунок переносу; F_7 - перетворення однієї характеристики в іншу; F_8 - переміщення вздовж градієнту.

Значення деякої функції F_j визначається похідною за часом від відповідної часової залежності деякої характеристики $c_k = c_k(F_1,\ldots,F_i,F_j,\ldots,F_N,t)$ при умові, що $F_i = 0$ для $\forall i \neq j$. Для отримання явного вигляду функцію F_j апроксимують деяким алгебраїчним виразом, який враховує клітинно-автоматний механізм взаємодії. Інтенсивність різних елементарних актів взаємодії (тобто внесок відповідних функцій) є різною, що враховується уведенням деяких вагових коефіцієнтів (коефіцієнтів активності). Значення цих коефіцієнтів разом з параметрами функцій F_i підбиралися емпіричним шляхом, виходячи з умови досягнення мети моделювання, - отримання відповідних структур, поданих на рисунках 15-16.

Взагалі, питання про можливість безпосереднього визначення правил взаємодії КА, виходячи з результуючої картини, досі не вирішено. Однією з причин, що ускладнюють визначення правил взаємодії, є багатофакторність моделі, коли довільний параметри системи залежить від різних параметрів функцій F_i та різних вагових коефіцієнтів. Таким чином, процедура знаходження явного вигляду функцій взаємодії, а також значення їх параметрів, є напівемпіричною задачею, а успіх моделювання залежить від розуміння системи, що моделюється, виділення основних її складових та визначення основних взаємозв’язків між ними.

Таким чином, можна зробити висновок про те, що розроблена схема реалізації методу асинхронних неперервних КА є досить зручним способом моделювання явищ просторової динаміки загалом і процесів самоорганізації та еволюції різного роду систем, зокрема. Основна перевага такого методу полягає у тому, що стає непотрібним процес усереднення станів деякої множини сусідніх КА для отримання неперервного значення певного параметра системи, що моделюється.

Особливістю цього методу є подання КА як векторів однакової довжини, компоненти якого є сукупністю певних характеристик чи параметрів системи, які набувають неперервних значень. У зв’язку з цим, функції міжклітинної взаємодії подаються у вигляді системи ітераційних функцій, явний вигляд яких залежить від природи конкретної системи і безпосередньо пов’язаний з

можливістю декомпозиції складних процесів взаємодії на елементарні акти. Такий підхід, коли функція взаємодії подається у вигляді суперпозиції елементарних актів, що мають певний фізичний зміст, є досить зручним. Ще однією особливістю описаного методу є те, що для моделювання еволюційної динаміки його метод асинхронних неперервних КА доповнено алгоритмом, який дозволяє будувати різні правила взаємодії на різних ділянках КА-поля, і до того ж, хаотично їх змінювати. Такий підхід зумовлено необхідністю моделювання відкритих систем, які вільно обмінюються речовиною з оточуючим середовищем, причому фізичні властивості частинок можуть мати різноманітний характер. У процесі моделювання, крім того, відбуваються локальні мікровзаємодії між КА (частинками), в результаті яких їхня концентрація може змінюватися. Такий спосіб взаємодії при моделюванні складних відкритих нерівноважних систем може породжувати на макрорівні конкуруючі самоорганізовані структури та демонструвати у такий спосіб еволюційну динаміку таких систем.

Криптографія на основі клітинних автоматів

Розглянемо тепер використання КА у криптографічних застосуваннях: при розробці генераторів бінарних псевдовипадкових послідовностей, криптографічних функцій хешування та блокових шифрів.

Перш за все, оцінимо можливість застосування одновимірних КА для проєктування потокових шифрів. Зрозуміло, що основою довільного потокового шифру є генератор бінарної послідовності, випадковий чи псевдовипадковий. Ми будемо досліджувати псевдовипадкові бінарні послідовності, отримані при використанні КА.

Генератори псевдовипадкових послідовностей на основі КА

Елементарні одновимірні КА описано та класифіковано С.Вольфрамом у книзі «A New Kind of Science» [12]. Що стосується міжклітинної взаємодії, то усього було виділено 256 правил, які поділяються на декілька класів, кожен з яких формується з так званих статистично еквівалентних правил, тобто таких, що можуть бути отримані одне з одного застосуванням простих логічних перетворень: кон’юнктивного, рефлексивного та кон’юнктивно-рефлексивного. У такий спосіб, наприклад, правила «30», «86», «135» та «148» потрапляють до одного класу. Прийнято вважати [3], що цей клас, і зокрема, правило “30” за статистичними характеристиками може бути досить якісним генератором псевдовипадкових бінарних послідовностей. Тому для досліджень було обрано

усі правила, що складають з правилом “30” один клас, а також деякі інші правила для порівняння (див. Табл. 1)

Таблиця 1

Досліджені правила міжклітинної взаємодії КА

№	Правила	Логична форма	Арифметична форма
1	«22»	$b' = a \oplus abc \oplus b \oplus c$	$b' = ((a+b+c+abc) \bmod 2)$
2	«30»	$b' = a \oplus (b\ c)$	$b' = ((a+b+c+bc) \bmod 2)$
3	«54»	$b' = (ac) \oplus b$	$b' = ((a+b+c+ac) \bmod 2)$
4	«73»	$b' = \overline{a \wedge c \vee a \oplus b \oplus c}$	$b' = ((1+a+b+c+ac+abc) \bmod 2)$
5	«86»	$b' = (a\ \ b) \oplus c$	$b' = ((a+b+ab+c) \bmod 2)$
6	«135»	$b' = 1a \oplus b\ c$	$b' = ((1+a+bc) \bmod 2)$
7	«149»	$b' = ab \oplus c\ 1$	$b' = ((1+ab+c) \bmod 2)$
8	«150»	$b' = a \oplus b \oplus c$	$b' = ((a+b+c) \bmod 2)$
9	«158»	$b' = a \oplus b \oplus cb\ c$	$b' = ((a+b+c+bc+abc) \bmod 2)$

З аналізу таблиці видно, що правило “86” може бути отримано з правила “30” циклічною перестановкою; правило 2135” є інверсією правила “30”, а “149” — інверсія правила “86”. Це додатковий аргумент на користь того, що вказані правила належать до одного класу.

Першим кроком дослідження статистичних характеристик згаданих правил було просте їх вивчення за допомогою статистичного пакету NIST STS. Деякі отримані результати подано на рисунку 19 [21].

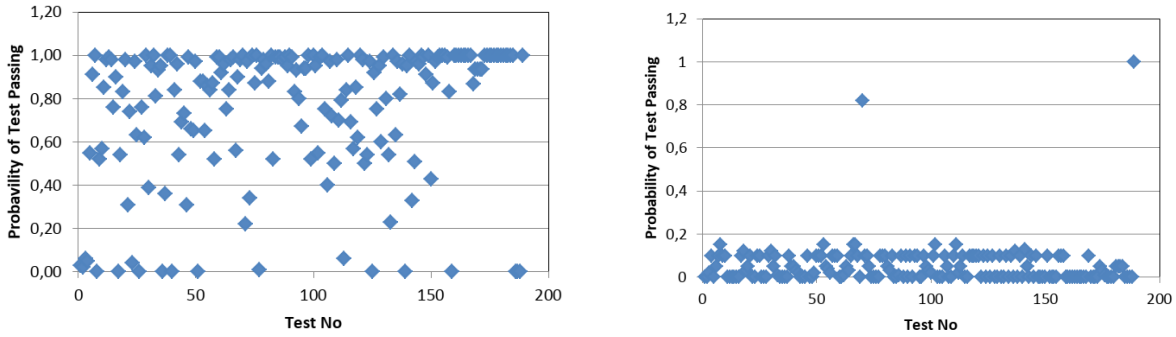


Рисунок 19 — Статистичні портрети генераторів на правилах “30” (ліворуч) та “22” (праворуч).

Як видно з рисунку, портрети генераторів суттєво відрізняються. Генератор на основі правила “22” не пройшов практично усі тести NIST STS, відповідно, застосовувати його для подальшої розробки криптографічно стійких генераторів безперспективно. Аналогічні результати отримано для правил «54», «73», «150» та «158». Зовсім інші результати отримано для правил «30», «86», «135» та «149», статистика яких дозволяє припустити, що з певними модифікаціями їх можна застосовувати у наших дослідженнях.

Як бачимо, для побудови якісного генератора псевдовипадкових бінарних послідовностей використання “чистого” правила «30», «86», «135» та «149» недостатньо. Усі ці правила мають бути модифіковані. Як модифікацію ми обрали додаткове комбінування клітин масиву для виводу. Тут можливі два варіанти: комбінація на основі простих логічних операцій кількох фіксованих клітин масиву та псевдовипадковий вибір клітин для виводу.

У першому випадку ми виводили результат додавання за модулем два вмісту трьох клітин:

$$M_{\text{out}} = \mathrm{M}[w] \oplus M[v] \oplus M[t]; \tag{6}$$

де w, v, t — номери клітин для виводу. У другому випадку використовувалася формула, яка давала номер клітини для виводу її вмісту:

$$w = \{i+3+\sum_{k=0}^{\log_2 n-1} c_{(i+3+k)} 2^k\} mod\, n. \tag{7}$$

де c – значення i+3+k клітини, n – загальна кількість клітин в автоматі; w – номер клітини для виводу.

Статистичні характеристики послідовностей, що їх генерували КА з такими модифікаціями, подано на рисунку 20 [22].

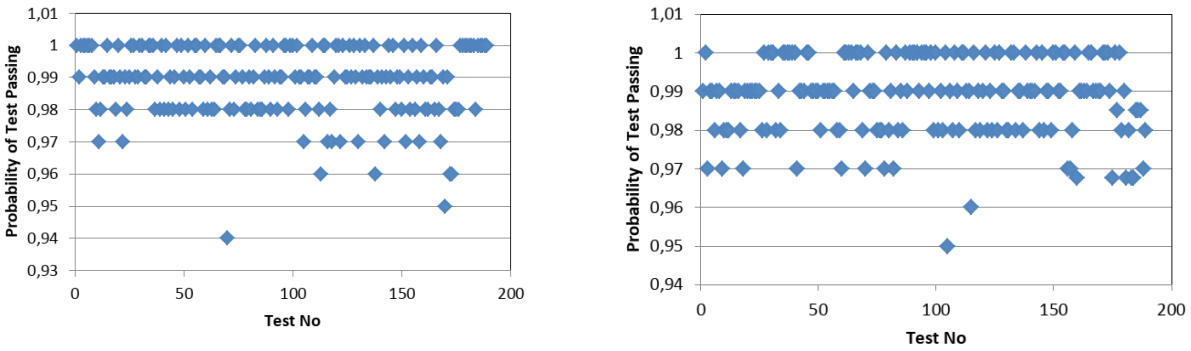


Рисунок 20 — Результати статистичного тестування генераторів на основі правила “30” з виводом за формулою (6) — ліворуч, за формулою (7) — праворуч

Як видно з рисунку, статистичні портрети досліджених генераторів значно покращилися. Для виводу за формулою (6) один тест проходиться на рівні 0,94, один — 0,95, три — на рівні 0,96, одинадцять — на рівні 0,97. Решта тестів NIST STS проходяться на рівні 0,98 і вище. Для виводу за формулою (7) результати ще кращі: один тест — на рівні 0,95, один — 0,96 і 14 — на рівні 0,97. Решта — на рівні 0,98 і вище. Отримані результати дозволяють зробити висновок про хорошу криптографічну стійкість таких генераторів. Аналогічні результати отримано для інших правил з цієї групи: «86», «135» та «149» (див. Рис.21) [23].

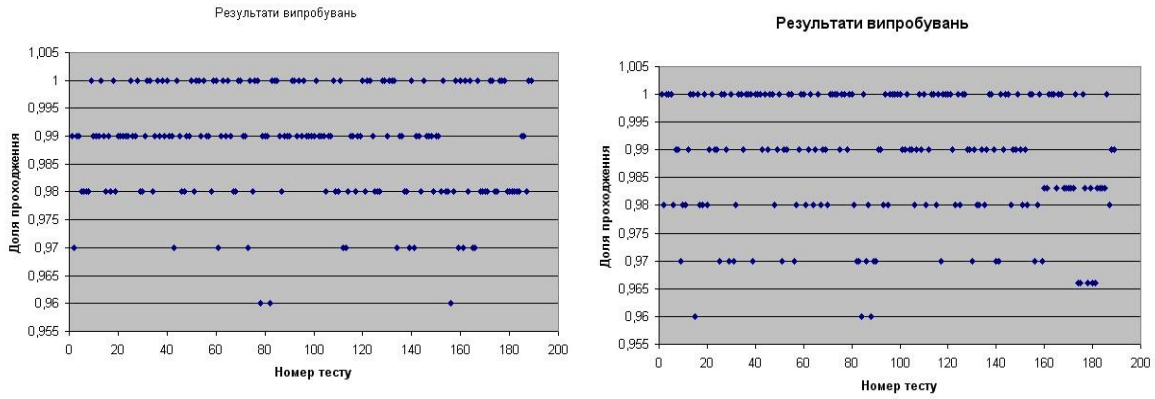


Рисунок 21 — Результати статистичного тестування генераторів на основі правил “86” (ліворуч) та “149” (праворуч)

В усіх випадках як стартовий стан одновимірного масиву довжиною 256 бітів приймався такий, коли 128-й біт містив “1”, а решта — “0”. З цього стану виконувалося 250 етапів взаємодії, після чогго вважалося, що генератор знаходиться у стартовому стані.

Наступним кроком було з’ясувати, наскільки можна покращити статистичні характеристики генератора, якщо використовувати комбінації елементарних правил міжклітинної взаємодії. Як показано у роботі [23], найкращі результати демонструють комбінації з правил, що належать до одного класу, причому самі правила також мають демонструвати гарні статистичні характеристики. Таким чином, ми маємо використовувати комбінації правил «30», «86», «135», «149». Використання інших правил на дає потрібного результату.

Разом з цим, ми дослідили вплив різних логічних операцій, за допомогою яких об’єднувалися різні правила. Показано, що об’єднання правил за допомогою операцій XOR та OR дає найкращий ефект, а використання логічної операції AND суттєво погіршує статистичні характеристики генератора.

Останньою спробою комбінування елементарних правил міжклітинної взаємодії була комбінація усіх правил з класу правила “30” у такий спосіб:

$$R_{comb} = (R_{30} \oplus R_{86} \oplus 1) \oplus (R_{30} \oplus R_{135} \oplus 1) \oplus (R_{30} \oplus R_{149} \oplus 1). \qquad (8)$$

На рисунку 22 подано статистичний портрет генератора, спроектованого за формулою (8). Як бачимо з рисунку, лиш 11 тестів NIST STS пройдено на рівні 0,97, решта 178 — на рівні 0,98 та вище. Таким чином, запропонований варіант комбінування правил міжклітинної взаємодії за (8) дозволяє

проектувати генератори псевдовипадкових бінарних послідовностей з гарними статистичними характеристиками.

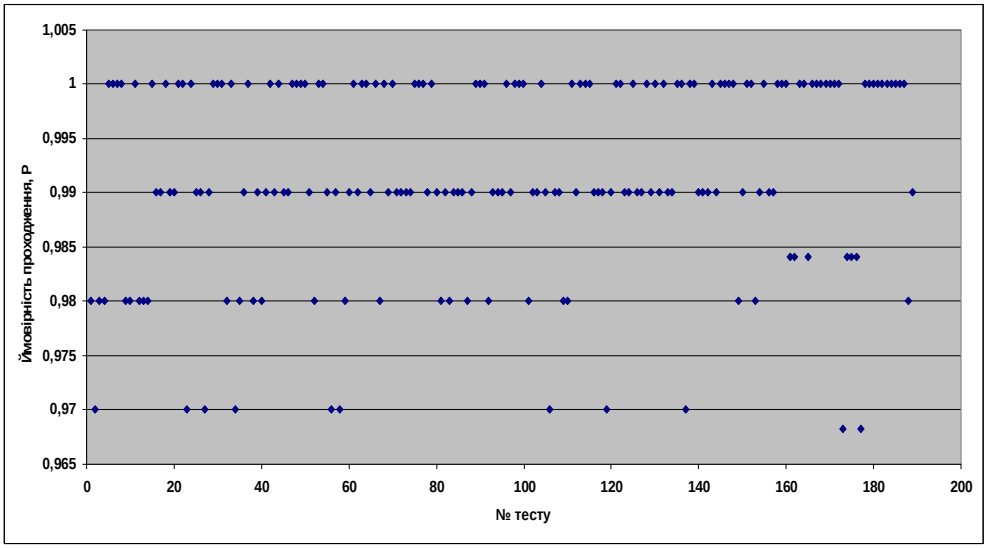


Рисунок 22 — Статистичний портрет генератора на основі (8).

Зрозуміло, що використання комбінацій типу (8) призводить до зменшення швидкодії генераторів. І, хоча втрати часу невеликі, вони будуть суттєво впливати на генерування потоку при використанні генератора у реальному часі. Тому першочерговим завданням є розробка швидкого власного алгоритму міжклітинної взаємодії, що демонстрував би характеристики, близькі до отриманих за допомогою комбінації елементарних правил.

Генератор на основі власного правила взаємодії

Розроблене правило міжклітинної взаємодії грунтується на взаємодії клітин, номера яких обираються згідно спеціального псевдовипадкового алгоритму. Для його роботи використовуються два масиви: один — робочий, а другий служить для визначення номерів клітин, що взаємодіють. Спрощений 8-бітовий варіант таких масивів подано на рисунку 23.

На рис.23а) подано стартове заповнення масивів. Вгорі — робочий масив, унизу — масив адрес клітин. Темні клітини робочого масиву символізують “1”, світлі — “0”. У нижньому масиві містяться десяткові числа — номери клітин (від 0 до 7), розміщені випадковим чином. На початку роботи вказівник масиву адрес встановлюється на його початок (рис.23б). Нульова комірка масиву містить число “5”. Це означає, що результат взаємодії буде записано до клітини з номером 5 (працюємо з лівим стовпчиком). Адреса однієї клітини, що взаємодіє, береться з наступної комірки масиву адрес (у даному випадку це комірка з номером 1, її підкреслено і виділено жирним шрифтом). На рисунку

23 там міститься “7”. Це значить, що взаємодіяти буде 7-ма клітина. Номер другої клітини, що взаємодіє, визначається безпосередньо за клітинами робочого масиву: номер клітини початку відліку задається положенням вказівника масиву адрес (тобто відраховуємо від клітини номер 1) і зчитуємо вміст 1-ї, 2-ї та 3-ї клітин.

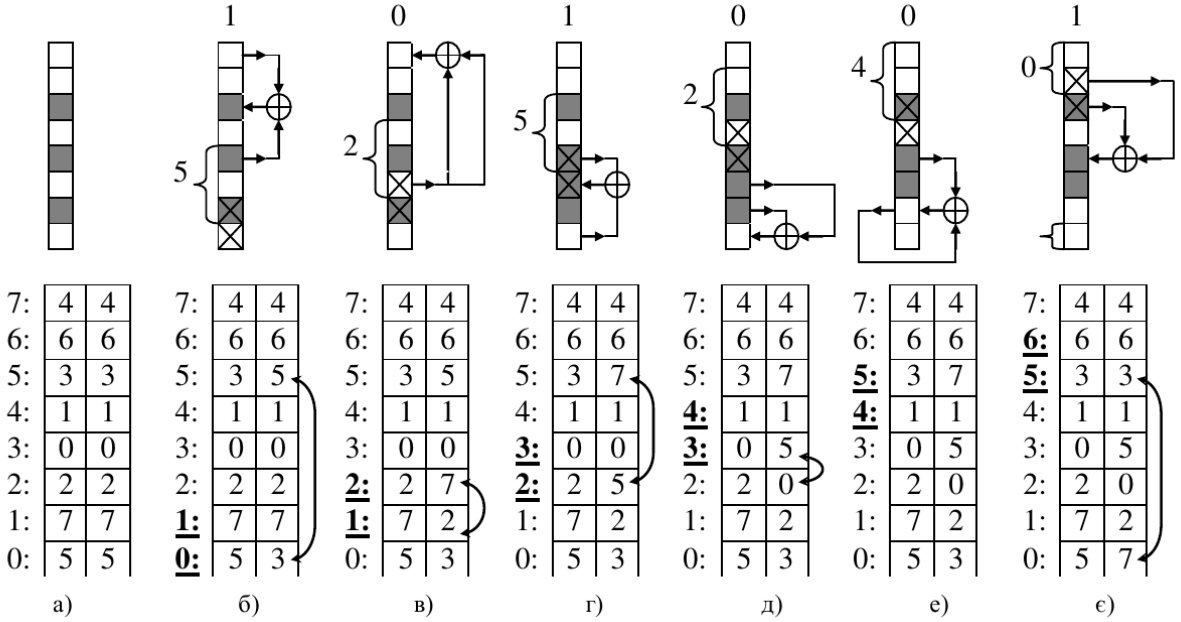


Рисунок 23 — Ілюстрація роботи розробленого правила взаємодії

Отримуємо $101_2 = 5_{10}$. П’ятий елемент масиву адрес містить “3”. Отже, взаємодіяти будуть 7-ма і 3-тя клітинки робочого масиву, а результат буде записуватися у 7-му клітину. Оскільки у нашому прикладі вказані клітинки містять “1”і “0”, то результатом взаємодії буде логічна “1”, що й вказано на рисунку 23 угорі. Для кращого розсіювання після акту взаємодії елемент масиву адрес, на якому знаходиться вказівник, і елемент, номер якого визначали, міняються місцями. У нашому випадку це 0-й та 5-й елементи масиву адрес, що й вказано ни рисунку 23б) стрілочкою.

Після перестановки вказівник масиву адрес пересувається на наступний елемент. Ці дії повторюються, поки вказівник не досягне кінця масиву, після чого стовпчики масиву адрес міняються місцями, вказівник встановлюється знову на нульовий елемент, і процес повторюється з самого початку.

На рисунку 23 а)-є) показано усі описані операції від нульового до п’ятого елемента масиву адрес.

Формалізм такого алгоритму досить простий. Правило взаємодії можна записати у вигляді:

$$C'_{(A[i,j])} = C_{(A[a,1])} \oplus C_{(A[i+1,1])},\ A[i,2] \Leftrightarrow A[a,2];\ i = (i+1) \bmod n; \qquad (9)$$

де $a=\left(i+1+\sum_{k=0}^{(\log_2 n)-1} c_{(i+1+k)} 2^{(\log_2 n-1-k)}\right) mod\, n$ - адрес, сформована зі значень адресних клітинок; n – загальна кількість клітин.

Як бачимо, алгоритм досить простий, що сприятиме високій швидкодії генератора.

Розроблений алгоритм було реалізовано у програмному коді і протестовано його статистичні характеристики. Особливості генератора такі:

- Довжина робочого масиву — 256 клітин;
- Взаємодія полягає у додаванні за модулем два вмісту трьох клітин (а не двох, як це показано на рисунку 23), що покращує статистичні характеристики послідовності, що генерується. Третьою клітиною служить та, куди записується результат.
- Стартовий стан генератора — псевдовипадковий, хоча статистичні характеристики таких генераторів практично не залежать від стартового стану. В подальшому ми використовували кількасот холостих циклів з певного початкового стану.

Результати статистичного тестування у NIST STS послідовності, що її генерує такий генератор, подані на рисунку 24.

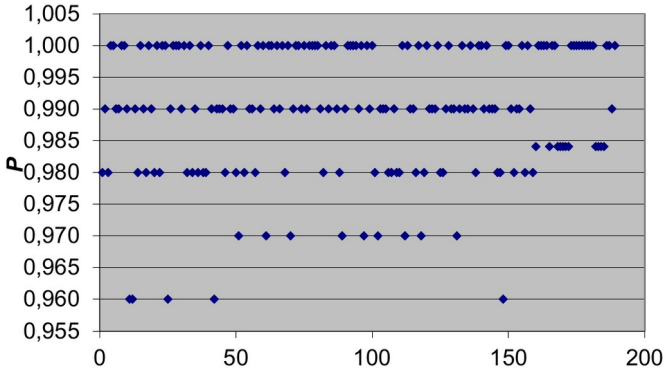


Рисунок 24 — Статистичний портрет генератора за правилом взаємодії (9)

Як видно з рисунку, 5 тестів виконуються на рівні 0,96, дев’ять — на рівні 0,97, а решта — 0,98 та вище, що свідчить про непогані статистичні характеристики розробленого генератора.

Дослідження лінійної складності генераторів на КА

Взагалі кажучи, відомо [24], що генератори бінарного ключового потоку характеризуються високими значеннями лінійної складності. Більше того, аналіз тестування розроблених нами генераторів за допомогою NIST STS демонструє, що усі вони проходять тести лінійної складності, які входять до складу пакету, на рівні не гірше 0,98. За основу тестування лінійної складності у NIST взято алгоритм Берлекемпа Мессі [25], однак, значення, що отримуються з NIST STS, суттєво відрізняються від прогнозованих у [24].

Для з’ясування цього факту ми виконали власні дослідження лінійної складності.

За основу було взято алгоритм Берлекемпа-Мессі, який обчислює найменшу довжину лінійного регістру зсуву, що здатний повторити запропоновану для аналізу послідовність. Найкращим вважається результат, коли довжина такого регістру приблизно дорівнює половині довжини послідовності.

Результати досліджень подано на рисунку 25. Для кожного правила було проведено по 10 дослідів.

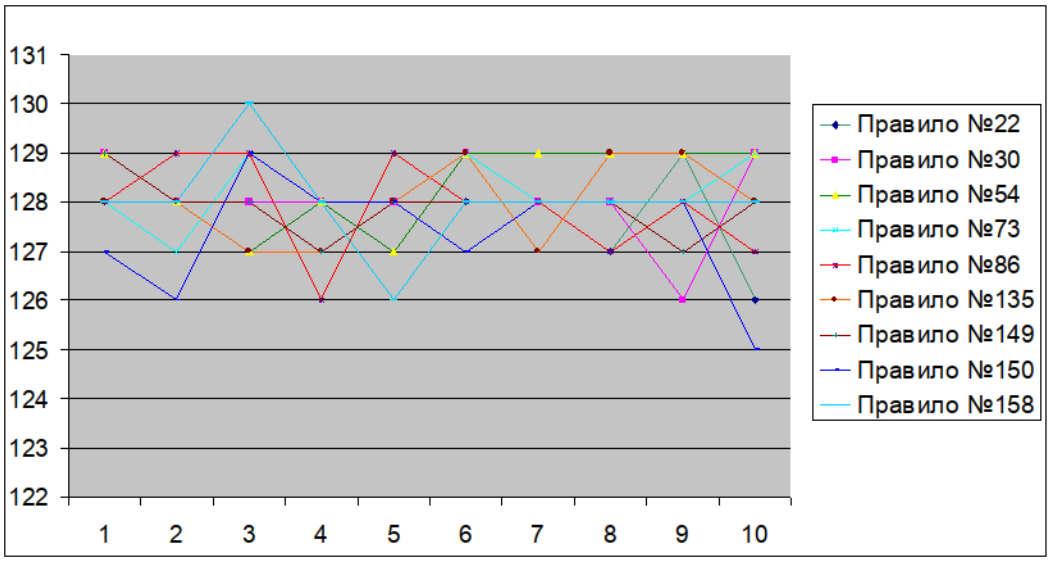


Рисунок 25 — Результати обчислення лінійної складності для 256-бітових послідовностей на основі різних правил міжклітинної взаємодії. По горизонталі — номер досліду, по вертикалі — значення лінійної складності.

З рисунку видно, що усі отримані значення лінійної складності зосереджені в околі числа 128, тобто половини довжини послідовності. Аналогічні результати отримані для послідовностей, довжинами 2560 та 25600 бітів.

Таким чином, запропоновані генератори псевдовипадкових бінарних послідовностей на КА, що й показали наші дослідження, демонструють не лише гарні статистичні характеристики, а й високі значення лінійної складності. Такі факти дозволяють рекомендувати генератори бінарних псевдовипадкових послідовностей на основі КА для використання у системах захищеного обміну даними.

Тепер розглянемо швидкодію досліджених генераторів.

Усі досліджені генератори було програмно реалізовано та досліджено їх швидкодію. В усіх випадках досліджувалася як абсолютна швидкість генерування, так і швидкість генерування файлу в 100 МБ для статистичних тестів.

Абсолютні значення швидкості генерування для різних генераторів коливаються в діапазоні 48 ÷250 Мб/с. Звичайно, комбінування, особливо зовнішнє, негативно впливає на швидкодію, отже найшвидшими виявляються генератори на елементарних правилах, для яких в багатопотоковому варіанті досягнуто швидкість 250 Мб/с. Такої швидкості ми досягли при п'яти потоках на звичайному ноутбуці. Код було написано на Java. Звичайно, при використанні більш швидких мов програмування, наприклад C або Assembler дозволять ще покращити характеристики генераторів.

Система захищеного обміну даними

Як приклад практичного застосування розробленого генератора з правилом міжклітинної взаємодії за формулою (9) нами було розроблено спеціалізоване програмне забезпечення — систему захищеного обміну даними. Програма дозволяє користувачам обмінюватися інформацією у реальному часі. Це може бути захищений голосовий чи текстовий чат, обмін файлами залежно від бажань користувачів. Розробка має клієнт-серверну архітектуру, що дозволяє її використовувати в однорангових мережах [26].

Узагальнена блок-схема та інтерфейс системи подані на рисунках 26-27.

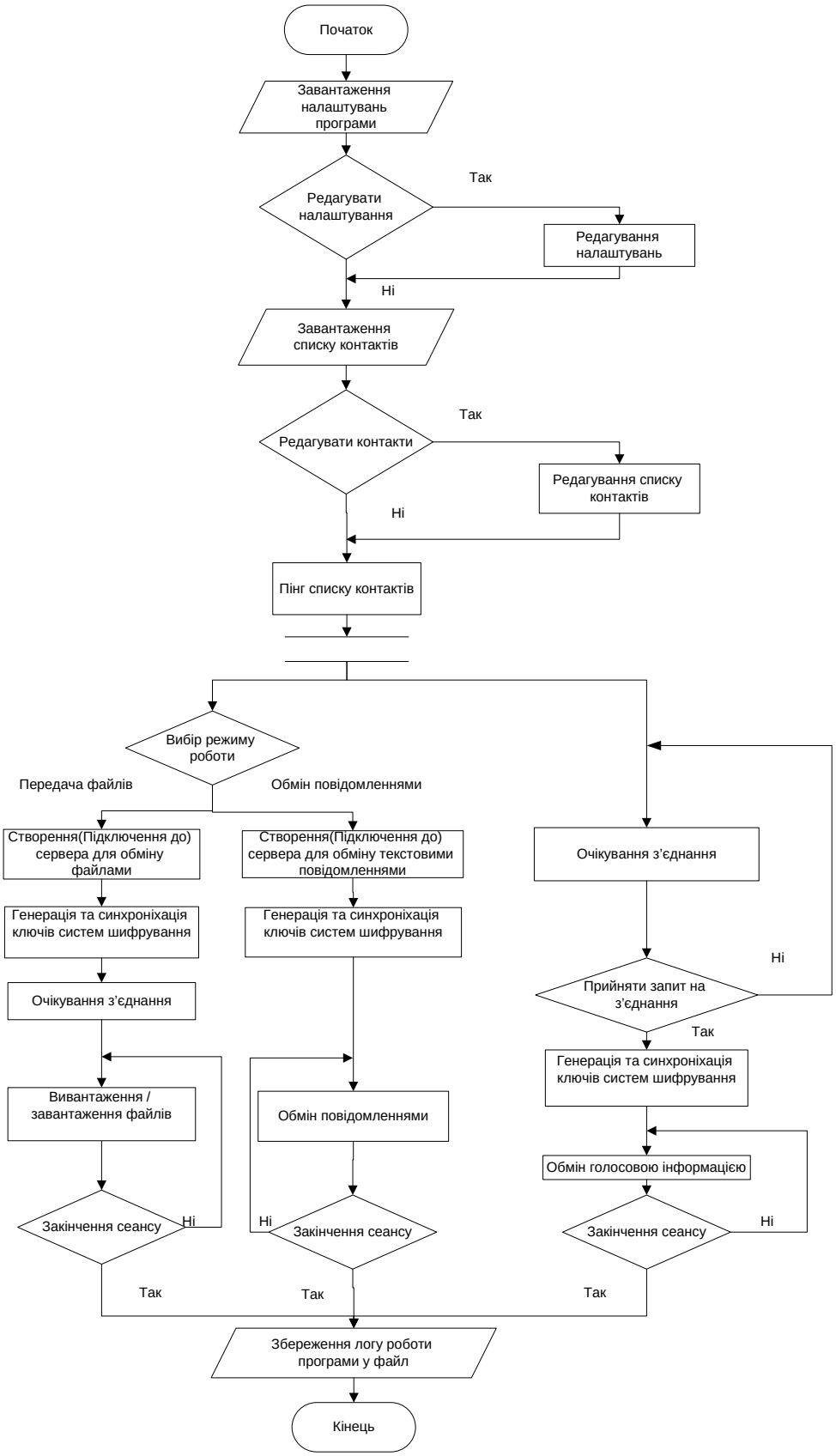


Рисунок 26 — Узагальнена блок-схема системи захищеного обміну даними

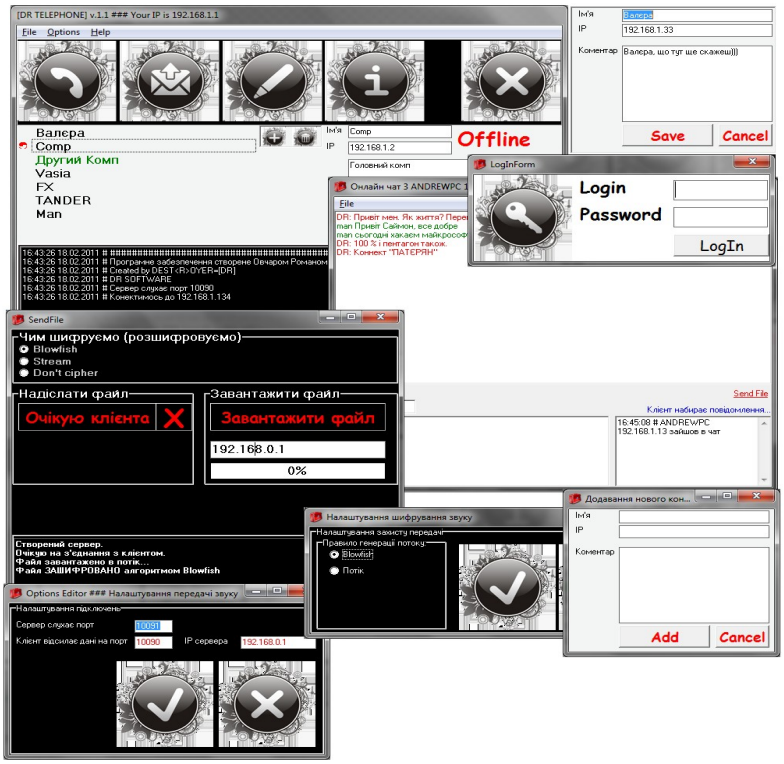


Рисунок 27 — Скрін-шоти інтерфейсу розробленої системи захищеного обміну даними

При з’єднанні абонентів виконується генерування та синхронізація ключів шифрування за алгоритмом Діффі-Хеллмана-Меркля. Отриманий ключ використовується для ініціалізації системи потокового шифрування на основі КА або для створення сеансового ключа симетричної системи шифрування за алгоритмом Blowfish залежно від налаштувань програми.

Як додаткова перевірка стійкості шифрування даних, було виконано статистичні тести вхідного файлу (100-МБ файл у форматі MS Word) до й після шифрування потоковим шифром на КА. Результати подано на рисунку 28.

Як бачимо з рисунку, вхідний файл не проходить практично жодного тесту NIST STS, тоді як зашифрований задовольняє усі статистичні тести на рівні не гірше 0,965, що говорить про достатню криптографічну стійкість розробленого потокового шифру на КА.

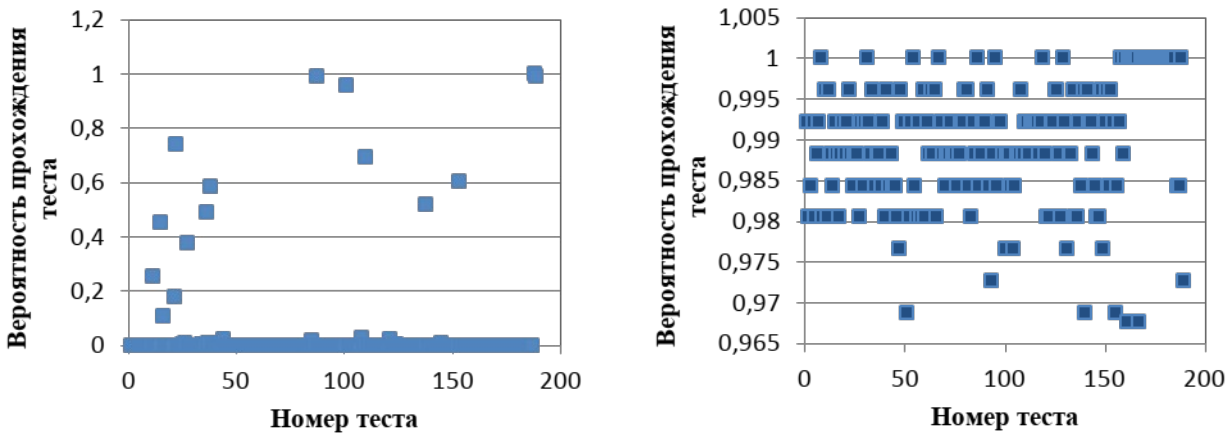


Рисунок 28 — Результати статистичного тестування файлу MS Word до (ліворуч) та після (праворуч) потокового шифрування

Що стосується швидкодії, то її цілком достатньо, щоби здійснювати шифрування аудіопотоку в реальному часі на звичайних ноутбуках.

Криптографічна функція хешування на основі КА

Криптографічну функцію хешування Keccak (SHA-3) було розроблено групою співробітників, до якої входив Joan Daemen — один з авторів блокового шифру Rijndael, більше відомого як AES. Зараз це діючий стандарт ISO під назвою SHA-3.

Відомою особливістю функції хешування Keccak є оригінальна архітектура, яка носить назву “криптографічної губки” (див.рис.29).

Вважається, що запропонована архітектура має безколізійний характер.

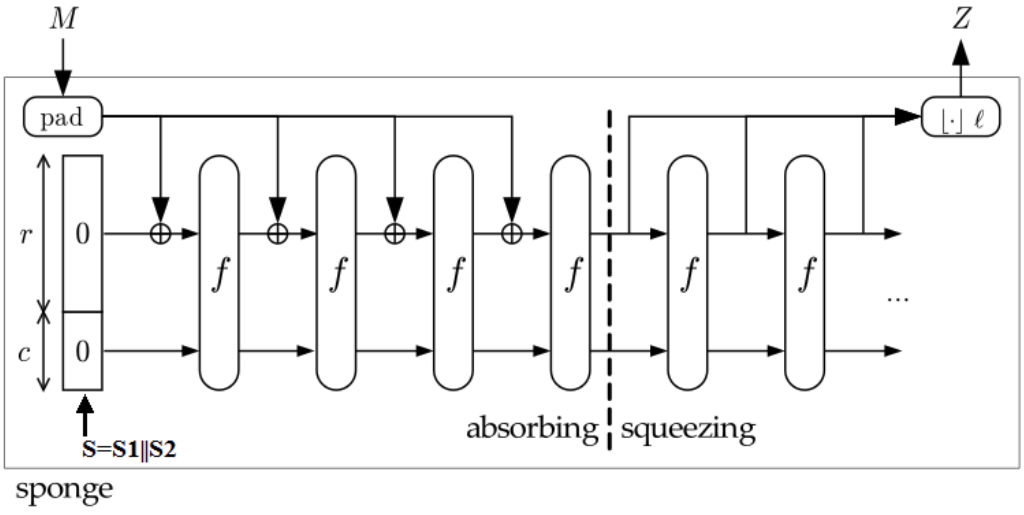


Рисунок 29 — Архітектура “криптографічна губка”, запропонована для SHA-3

Основною особливістю структури є те, що блок вхідного повідомлення (його розмір набагато менший, ніж $r||c$) додається до початкового стану ($r||c$) і обробляється функцією перестановки $f()$. Ця функція «розмиває» статистичні характеристики вхідного повідомлення по всьому стану ($r||c$). Оскільки стиснення не використовується, такий процес буде вільним від колізій. Процес повторюється до закінчення вхідного повідомлення. Цей режим отримав назву «поглинання» (absorbing). Наступний режим називається «віджиманням» (squeezing). Звідси і назва всієї структури: «криптографічна губка». Процес «віджимання» повторюється, поки не буде створено хеш-образ потрібної довжини. Як видно з рис. 29, процес перестановки з усім ($r||c$)-станом продовжується й під час віджимання, що робить знаходження статистичних зв'язків між вхідним повідомленням і хеш-образом майже неможливим.

Зважаючи на цей факт, ми використали цю структуру для конструювання власної криптографічної функції хешування на основі КА.

Функція перестановки може бути довільною. Єдиною вимогою є те, щоби вона добре перемішувала біти стану.

Ми спробували розробити функцію перемішування на основі КА. Вона складається з декількох кроків.

Крок 1. Додавання вхідного повідомлення до довжини, кратної 512 бітів.

Крок 2. Генерування значень c і r загальної довжини 1600 бітів. Довжина вектору c складає 1024 біти, і кожен його біт є логічною „1”. Решта бітів (576 бітів) заповнюються логічними нулями.

Крок 3. Отриманий вихідний стан вважається клітинним автоматом і обробляється протягом 200 порожніх циклів за елементарним правилом КА „86” (R86).

Крок 4. Виконується етап поглинання. Поточний блок вхідного повідомлення (довжина 512 бітів) додається за модулем 2 з бітами стану ($r||c$): $RC = RC$ XOR $M_i[512]$.

Крок 5. Виконується 30 раундів функції перестановки:

- $RC = R86(RC)$: стан RC обробляється правилом КА „86”;
- $RC' = R150(RC)$: стан обробляється правилом КА „150”;
- $RC >>> 31$: 31-бітовий циклічний зсув RC-стану праворуч;
- $RC' = RC$ XOR RC';
- $RC = R150(RC')$.

Етапи 4 і 5 продовжуються до завершення блоків вхідних повідомлень.

Крок 6. Виконується етап віджимання. Щоразу після закінчення 30-раундової функції перестановки виводяться наступні 64 біти хеш-образу. Отже, довжина хеш -образу може бути довільною, кратною 64 біти.

Для перевірки лавинного ефекту розробленої хеш-функції ми реалізували її на Java.

Лавинний ефект перевіряли за допомогою 64-, 128-, 256-, 512- і 1024-розрядних хеш-образів. Для цього тесту ми використовували текстовий файл. Один раз файл використовувався «*як є*», а інший - з додаванням після останнього символу «.» (крапки). Навіть така незначна зміна повідомлення повинна привести до іншого хешу (якщо лавинний ефект добре реалізований). Результати для 256-розрядної версії хеш-образу такі:

• оригінальний текст без крапок:

0xbed39b7aae3694fe2d4a57df88327589b6670c68dc9c13d391166865234cfcf

• змінений текст з крапкою:

0x2a1a2666518c676a28e587450dccf62019880580090135f590d7640ff337382

Як бачите, хеш-образи дуже різні. Такі ж результати були отримані і в усіх інших випадках.

Ці результати показують, що ми можемо використовувати клітинні автомати в функціях перестановки з хорошим лавинним ефектом. Такі функції можна успішно використовувати в алгоритмах криптографічного хешування.

Блокові шифри на основі КА

Спробуємо використати КА для побудови блокових шифрів.

WBC-шифр

Для початку розглянемо блоковий шифр, який отримав назву WBC (White&Blue Containers). В цьому шифрі використовують тривимірні КА та прості логічні операції.

Вихідні принципи дизайну

Алгоритм призначено для реалізації та виконання на ЕОМ.

З огляду на це:

1. Дизайн повинен передбачати роботу з двійковими даними;
2. Структура алгоритму повинна відповідати принципам, прийнятим в сучасних архітектурах ЕОМ. Отже, робота повинна відбуватись або з бітами, або з одиницями даних кратними розрядності машинного байту;
3. Обрана структура повинна бути придатною до відносно легкої адаптації для різних типів ЕОМ, в тому числі малопотужних.

В процесі проектування архітектури алгоритму було експериментально доведено, що швидкодія операцій з бітами в сучасних мікропроцесорах є на порядок повільнішою за операції із байтами, словами (*word*, 2B) чи подвійними словами (*dword*, 2B), отже основною одиницею даних було обрано машинний байт.

Крім цього, до дизайну архітектури висувалися наступні вимоги:

1 Алгоритм повинен забезпечувати гранулярну обробку даних невеличкими порціями. Передбачалось, що це значною мірою вплине на розсіювальні характеристики, що пізніше було підтверджено експериментально;

2 Структура алгоритму повинна бути максимально гнучкою до змін об’ємів блоку вхідних даних, тобто алгоритм повинен за мінімальних змін забезпечувати обробку блоків будь-яких розмірів;

3 Алгоритм повинен бути незалежним від гранулярності і однаково успішно працювати на контейнерах даних із декількох байт, декількох десятків або сотень байт;

4 Алгоритм повинен забезпечувати надійний рівень дифузії в тому числі за рахунок зміни послідовності операцій для власного виконання.

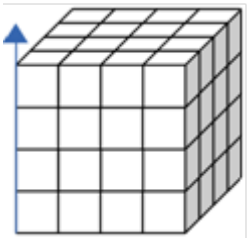


Рисунок 30 — Приклад контейнеру для шифру WBC

Для ілюстрації ідеї, покладеної в основу даного алгоритму, використаємо *контейнер* у формі кубу розмірністю 4x4x4, який складається з 64 комірок рівномірно розподілених в даному кубі (див. рис. 30). Саме такий контейнер використано для програмної реалізації даного алгоритму.

За рівень контейнера візьмемо усі комірки кубу, які мають однакову координату Y. Тоді будемо мати чотири накладених рівні, що містять шістнадцять комірок кожний. Нумерація комірок виконується знизу, починаючи з нуля.

В межах одного рівня нумерація комірок така сама і також починається з нуля (див. рис. 31).

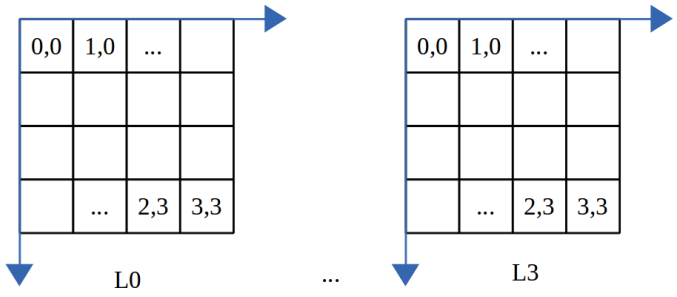


Рисунок 31 — Приклад нумерації комірок у шифрі WBC

Оскільки мінімальною одиницею адресації обрано байт, то дані розподіляються по комірках побайтно. Таким чином, розмірність блоку даних складає 64B, або 512b. Довжина ключа дорівнює довжині блоку даних

Схема обробки рівнів

Для кожного рівня визначається порядок його обробки алгоритмом, так звана *карта рівня*, яка являє собою послідовність координат комірок, згідно з якою алгоритм повинен обробити комірки контейнеру (див. рис. 32).

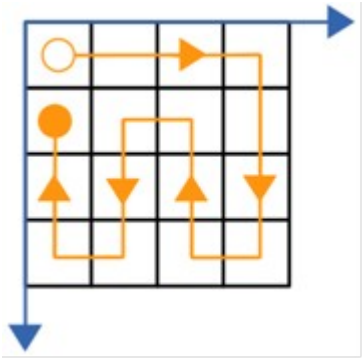


Рисунок 32 — Порядок обходу комірок алгоритмом шифру WBC

Призначення *карт рівнів* полягає у початковому збільшенні ентропії алгоритму за рахунок видозмінювання послідовності операцій для різних ділянок контейнеру. Таким чином зменшується детерміністична складова алгоритму, оскільки з вихідної шифрограми (потенційно) неможливо визначити карту рівня, а отже і шлях проходження алгоритму по контейнеру. В певній мірі, карти рівнів можна вважати аналогами S-блоків в сучасних блокових шифрах, хоча принцип їх дій і різний, але вони ведуть до схожих результатів.

Для кожного рівня задається лише одна карта, яка обирається з наперед визначеного переліку. Кількість координат в межах однієї карти рівня строго не задано: повторний прохід алгоритму по вже відвіданим коміркам є допустимим. Таке рішення дозволяє приховати від криптоаналітика кількість операцій, які

буде виконано алгоритмом, навіть за умови відритого доступу до самого алгоритму.

Перелік карт рівнів, його розмірність та розмірність самих карт формується на етапі реалізації алгоритму і не задається дизайном.

Алгоритм вибору карти із послідовності наступний: обчислюється побайтна сума ключів всіх комірок в порядку від 0-ї комірки до 15-ї, після чого за рівнем закріплюється певна карта.

Схема обробки комірок

Кожна із комірок кожного рівня містить два набори даних: частину вхідних даних, що підлягають криптографічному перетворенню та частину ключа, на якому буде здійснено шифрування. Розподіл даних та ключа відбувається шляхом рівномірного заповнення комірок контейнеру починаючи з нульової комірки нульового рівня і закінчуючи п’ятнадцятою коміркою останнього рівня.

Всі комірки контейнеру поділяються на два класи: *W* та *B*. Комірками класу *W* вважаються такі, старший біт даних яких містить одиницю, а комірки типу *B* — це ті, старший біт даних яких містить нуль.

Виконання алгоритму починається з комірки, на яку першою вказує карта 0-го рівня і триває доти, доки не вичерпаються координати карти останнього рівня. Таким чином забезпечується прохід алгоритму по всьому контейнеру, а як наслідок – дифузія базується на основі даних всього вхідного блоку. Всього таких проходів відбувається три. Дане число підібрано експериментально на основі даних аналізу статистичних характеристик.

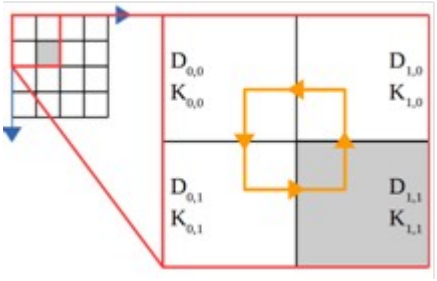


Рисунок 33 — Порядок обробки комірок у шифрі WBC

Обробка комірок відбувається послідовно.

Алгоритм розроблено у такий спосіб, щоби операції з будь-якою із комірок впливали на вміст трьох найближчих сусідів. Зокрема, за рахунок цього забезпечується дифузія.

Які саме три комірки буде змінено, залежить від класу комірки, що обробляється в даний момент. Приклад схеми обробки комірок показано на рис 33.

Алгоритмом визначено 8 операцій: по 4 на кожен клас комірок. Для всіх них алгоритм однаковий, відрізняється лише напрям впливу на сусідні клітинки. Практично доведено, що цього достатньо для забезпечення задовільних показників частотних характеристик.

Виконання операцій з комірками типу W виконуються на тому рівні, де ця комірка розташована. Розповсюдження впливу цієї операції на сусідні комірки виконується у такий спосіб: якщо другий старший біт дорівнює одиниці, — операцію буде виконано «вгору» (на наступний рівень), інакше — «вниз» (на попередній), якщо третій біт даних дорівнює одиниці — операцію буде проведено «ліворуч», інакше — «праворуч». Схематично послідовність виконання одного із варіантів наведено на рис. 34.

Виконання операцій з комірками типу B відбуваються у двох сусідніх рівнях, причому який з двох сусідів обирається — залежить від вмісту самої комірки: якщо другий старший біт дорівнює одиниці, — взаємодія буде з вищим рівнем, інакше — з нижчим рівнем; якщо третій старший біт даних дорівнює одиниці, — операцію буде проведено «ліворуч», інакше — «праворуч» (див. рис. 34).

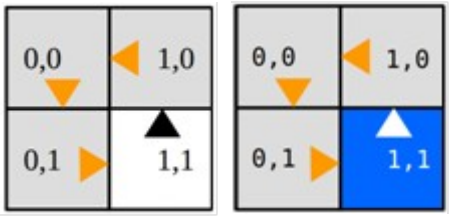


Рисунок 34 — Схеми обробки комірок: класу W[1,1] (ліворуч) та B[1,1] (праворуч)

Простір контейнеру замкнено. Це значить, що якщо обробляється крайня ліва комірка, то суміжною їй зліва вважатиметься крайня права комірка цього ж рівня. Аналогічно, якщо обробляється крайня права комірка суміжною їй справа вважатиметься крайня ліва комірка цього ж рівня. Аналогічним чином вирішується ситуація обробки 0-го рівня і останнього: для 0-го рівня суміжний зверху — 1-й, суміжний знизу — останній, для останнього суміжний зверху — 0-й, суміжний знизу — передостанній.

Алгоритм обробки комірок

Алгоритм обробки комірки виконує такі операції:

1), $K_0 = \overline{K_0 \oplus K_1}$ де K_0 — ключ поточної комірки, K_1 — ключ наступної комірки. Інвертований XOR — використовується для нівелювання слабких ключів.

2) Циклічний зсув ліворуч на п’ять бітів ряду поточної комірки.

3) $D_0 = D_0 \oplus K_0$, де D_0 — дані поточної комірки;

4) $K_1 = K_1 \oplus K_2$ (1-а операція не використовує інверсію);

5) Циклічний зсув ряду в, якому знаходиться перша комірка на 5 бітів ліворуч;

6) $D_1 = D_1 \oplus K_1$;

7) $K_2 = \overline{K_2 \oplus K_3}$;

8) Циклічний зсув ліворуч на сім бітів ряду поточної комірки 2;

9) $D_2 = D_2 \oplus K_2$;

10) $K_3 = K_3 \oplus K_3$ (3-я операція не використовує інверсію);

11) Циклічний зсув ряду в, якому знаходиться комірка 3 на **7** бітів вліво;

12) $D_3 = D_3 \oplus K_3$.

Оскільки спосіб обробки комірок залежить від даних, які, в свою чергу, змінюються після кожної операції, то детерміністична складова даного алгоритму в порівнянні із класичними блоковими шифрами є досить низькою.

Отримані результати

Отриманий в результаті блоковий шифр має такі основні характеристики:

- Оскільки в алгоритмі використовуються операції XOR, він симетричний відносно шифрування/розшифрування, тобто в обох випадках використовується один і той самий набір операцій.
- Алгоритм не засновується ні на петлі Фейстеля, ні на SP-мережі.
- Алгоритм працює з байтами. Це дозволяє застосувати ряд оптимізацій на сучасних платформах. Пропонована архітектура може бути легко адаптована до поширених апаратних архітектур, наприклад, оптимізована під 64-бітову архітектуру за рахунок мінімальної зміни контейнера.
- Довжина ключа та ємність блоку змінні і можуть варіюватись від 1-го байту і до безмежності за рахунок зміни контейнера. При цьому немає необхідності видозмінювати сам алгоритм. Він гарантовано залишиться працездатним.
- Розсіювальні характеристики задовольняють тестам NIST.
- Режим виконання за замовчуванням подібний до режиму cbc для класичних шифрів і здатний на одному і тому ж ключі і без повторної ініціалізації на монотонні дані видавати не монотонний результат.
- Швидкість шифрування/розшифрування однопотокової програмної реалізації на процесорі Intel Core i5 650 становить близько 100 Mbit/сек.

Статистичні характеристики розробленого шифру

Нами було досліджено статистичні характеристики програмної реалізації розробленого блокового шифру.

Результати статистичного тестування за доаомогою NIST STS подано на рисунку 35.

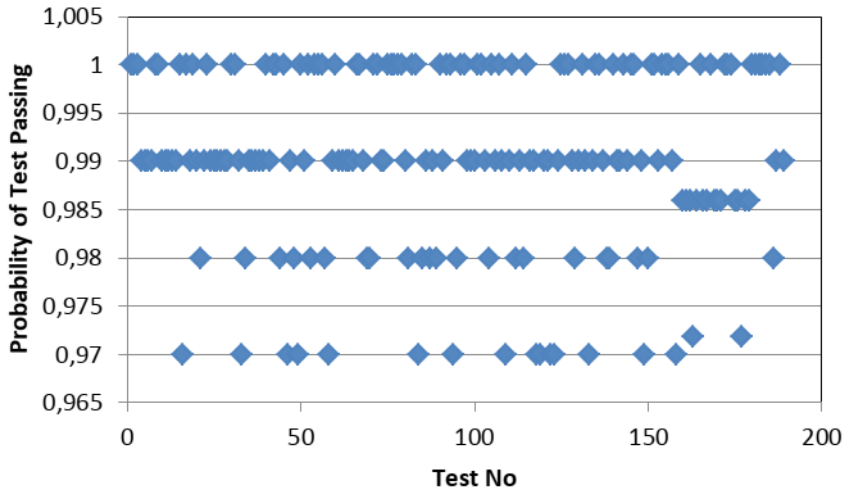


Рисунок 35 — Статистичний портрет блокового шифру WBC

Як видно з рисунку, розроблений шифр демонструє досить хороші статистичні характеристики: пройдено усі тестиNIST STS - 189 (100%); на рівні 1.00 - 69 (35,9%); на рівні 0,99 - 66 (34,9%); на рівні 0,98 - 36 (19%); на рівні 0,97 - 18 (10%). Таким чином, перевірений шифр пройшов 171 тест з 189 на рівні не менше 0,98 (90,5%). Ми вважаємо, що це дуже хороший результат.

Тести на дифузію та конфузію також демонструють хороші характеристики: навіть невеликі зміни введеного тексту або ключа шифрування призводять до зміни щонайменше половини бітів зашифрованого повідомлення.

Блоковий шифр на основі тривимірних КА

Наступним застосуванням КА для побудови блокових шифрів є тривимірний КА, який використовує прості правила міжклітинної взаємодії. У шифрі використовуються тривимірні КА як для обробки вхідного блоку, так і ключа.

Основні параметри розробленого шифру такі:

- розмір блоку – 64 байти (512 бітів);

- розмір ключа – 64 байти (512 бітів);
- кількість раундів – до 15;
- за один раунд обробляється цілий блок.

Зазначена довжина блоку та ключа є оптимальною для сучасних потреб криптостійкості. Оптимальна кількість раундів буде встановлена після проведення статистичних тестів.

Проектування SP-мережі з тривимірним блоком

Після визначення основних параметрів можна переходити до реалізації архітектури. В нашому випадку масив блоку даних буде тривимірним. Використання масивів як контейнера для байтів даних є цілком обґрунтованим рішенням через простоту реалізації, швидкодію і можливість доступу по індексу, це дає масивам перевагу над списками.

Якщо розглядати мову Java, то в ній є клас ArrayList, що реалізує список на основі масиву та надає різні додаткові методи колекцій (наприклад додати новий елемент не в кінець списку, а в зазначене індексом місце, така дія зсуває всі елементи від зазначеного індексу вправо, звільняючи простір для нового елемента), також в такому випадку фіксована кількість елементів необов’язкова, що робить ArrayList динамічною структурою.

Проте для потреб розробленого шифру зазначені властивості списку не надають жодних функціональних переваг, тому що розмір блоку даних визначається на етапі його ініціалізації, залишаючись в подальшому незмінним. Також для різних маніпуляцій над блоком, а особливо для запису в різні комірки, індексний доступ масиву дозволяє напряму зазначити комірку, в яку буде записано нове значення, без зайвих операцій зсуву та видалення. Варто зазначити що ще однією перевагою масивів є легший синтаксис і простіше створення багатовимірного масиву.

Отже, для представлення блоку даних використано простий тривимірний масив байтів розміром 4x4x4 (64 байти). Всі операції перетворення будуть працювати з цим блоком.

Дослідження та реалізація елементарних клітинних автоматів

Тепер настала черга розглянути роль елементарних КА в алгоритмі. Потрібно визначити яким чином вони будуть реалізовуватися в коді. Існує

варіант проходитися по кожній комірці масиву покоління КА, дивитися значення її сусідів, і за допомогою якої-небудь реалізованої в коді таблиці відповідності знаходити значення наступного стану. Хоч цей спосіб звучить, можливо, як просте рішення, насправді він надзвичайно незручний і громіздкий.

Набагато краще використовувати булеве представлення правил елементарних КА. Для цього застосовується мінімальна (містить найменше змінних) диз'юнктивна нормальна форма, тобто кожне правило представлене диз'юнкцією (логічне АБО) кон'юнкцій (логічне І) значень станів (або їх логічних заперечень) поточної клітинки КА та її сусідів [27]. Така форма представлення коротка і зручна, а згадані логічні операції виконуються на комп'ютерах швидко.

Проте варто пам'ятати, що майбутній шифр працюватиме не з окремими бітами, а з байтами. А комірки елементарних КА можуть приймати лише стани 0 та 1, тобто по суті вони є бітами. В такому випадку кожен байт клітинки та її сусідів повинен бути записаний в двійковому представленні, і потім правило переходу буде застосовуватися для кожної групи бітів.

Знову ж таки, рішення просте – більшість мов програмування, включаючи Java, підтримують побітові логічні операції, тобто кожна змінна, до якої застосовуються логічні І, АБО, НЕ (в мові Java це оператори &, | та ~ відповідно), буде переведена в двійковий формат, потім операція опрацює кожні окремі біти, і поверне остаточний результат, в мові Java це буде ціле число (integer).

Таким чином, питання реалізації правил елементарних клітинних автоматів було вирішено.

Проектування тривимірного клітинного автомата на основі елементарних КА

Далі поставлено завдання побудувати тривимірний клітинний автомат, визначивши його стани, сусідство, та правило переходу. Якщо елементарні КА представлені у вигляді одновимірної стрічки, то в нашому випадку буде куб. Відповідно у комірки зростає кількість найближчих сусідів – з 2 до 26. Тоді, маючи правила переходу елементарних КА, можна сформувати такий тривимірний варіант: так як у комірки 26 сусідів, їх можна розділити на 13 пар так, щоб кожні два сусіди знаходилися по протилежні сторони від клітинки. Таким чином кожна пара сусідів разом з клітинкою по центру буде по суті окремим елементарним клітинним автоматом, тоді правило переходу для

тривимірного КА буде представлено набором із 13 правил елементарних клітинних автоматів. Отже всього така структура буде мати $256^{13}=$ правил переходу.

Для спрощення розрахунків нами замість 26 обрано 6 сусідів – три пари, по одній на кожну вісь (X, Y, Z), отже виходить 3 елементарних КА. В такому випадку КА матиме $=2^{24}=16\ 777\ 216$ правил. Також КА працює в кінцевому просторі, тому ми замикаємо його краї – для комірки на верхньому краю сусідами будуть комірки з нижнього краю і т.д.

Варто зазначити, що так як елементарні КА мають правила, які в тій чи іншій мірі еквівалентні, то і в розробленому КА будуть еквівалентні правила. А це означає, що із 256 правил елементарних КА варто вибрати три найкращі з точки зору криптографічних перетворень.

Ефективність визначалась в першу чергу булевим представленням правила. Бралися до уваги ті, які включають в себе всі три змінні (клітинка і два сусіди) і мають найдовший запис, в такому випадку наступний стан клітинки залежить від складнішого перетворення. Також обрані три правила повинні бути не еквівалентними між собою [27]. Зважаючи на ці критерії, було обрано правила 22, 105 та 150.

Остаточний варіант розробленого тривимірного клітинного автомата виглядає так:

– стан однієї комірки виражається не бітом а байтом;
– комірка залежить від шести сусідів – два по осі X, два по осі Y і два по осі Z;
– кожна пара сусідів і комірка утворюють окремий елементарний КА;
– сформовані три елементарні КА будуть використовувати правила: по осі X правило 105, по осі Y правило 22, по осі Z правило 150;
– новий стан клітинки визначається після застосування до неї всіх трьох правил в такому порядку: X – Y – Z.

Визначення функцій перетворення в шифрі та роль клітинного автомата

Тепер, коли реалізація блоку даних в алгоритмі визначена, і клітинний автомат спроектовано, потрібно описати інші функції, які застосовуватимуться при шифруванні.

SubBytes. Дана функція SP-мережі є S-скринькою, тобто вона замінює при шифруванні байти блоку даних на відповідні байти з таблиці заміни, а при розшифруванні, за допомогою оберненої таблиці, однозначно замінює байти на

ті які були перед підстановкою. Звісно що S-скриньки для алгоритму шифрування повинні бути правильно спроектовані, щоб заміна позитивно впливала на криптостійкість та була однозначною. Тому в розробці використано S-скриньки алгоритму Rijndael, надійність яких є підтвердженою. В їх основі афінне перетворення над скінченним полем, що застосовується по формулі для кожного байту, але також S-скриньки зручно представляти таблицями простої заміни (рисунки 36 та 37). Наприклад в нас є байт в шістнадцятковому представленні – 3b. Для того щоб його замінити, потрібно знайти старшу половину нашого байта (3) в рядку таблиці, а молодшу (b) – в стовпці. Таким чином байт замінюється на e2. Для зворотної заміни проводимо з байтом e2 таку саму процедуру, але в оберненій таблиці.

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

Рисунок 36 – Пряма таблиця заміни

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

Рисунок 37 – Обернена таблиця заміни

RotateCubeMatrices. З назви цієї функції можна зрозуміти, що вона здійснює перестановку, тобто є P-скринькою. По суті вона є адаптацією функції shiftRows алгоритму Rijndael під тривимірний блок даних. В оригіналі функція зсуває кожен рядок матриці блоку вліво в залежності від номера рядка (нульовий рядок не зсувається, перший зсувається на одиницю і т.д.). У випадку тривимірного блоку відбувається поворот чотирьох матриць куба на 90 градусів вправо певну кількість разів, в залежності від індексу матриці в кубі (нульова матриця не повертається, перша повертається на 90 градусів, друга – на 180, третя – на 270). Рисунки 38 і 39 наочно демонструють виконання описаної функції на прикладі кубика Рубика 4x4x4.

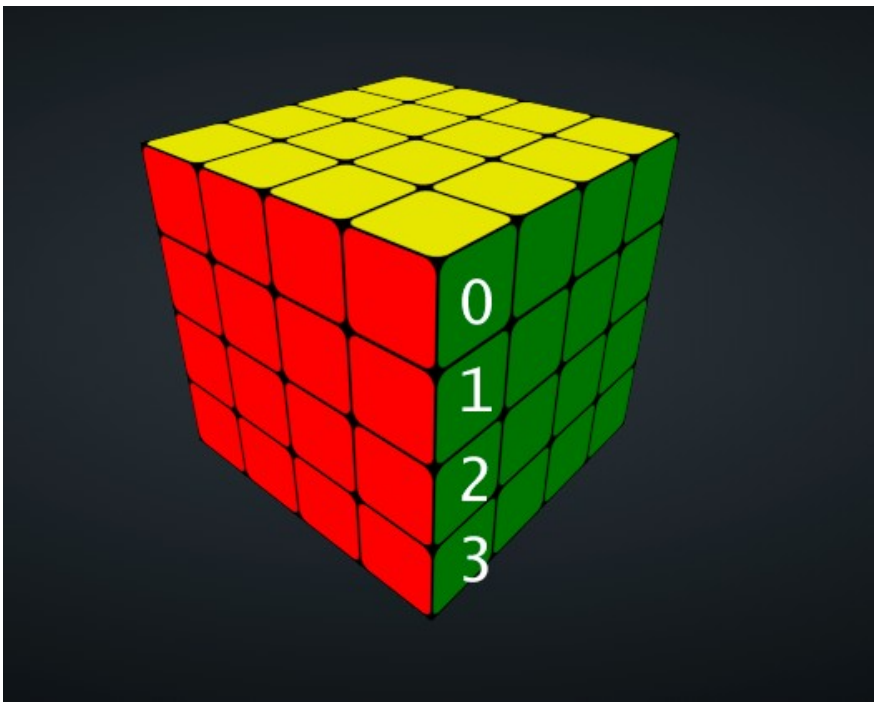


Рисунок 38 – Представлення блоку у вигляді кубика Рубика 4x4x4, цифри позначають індекс матриці у кубі

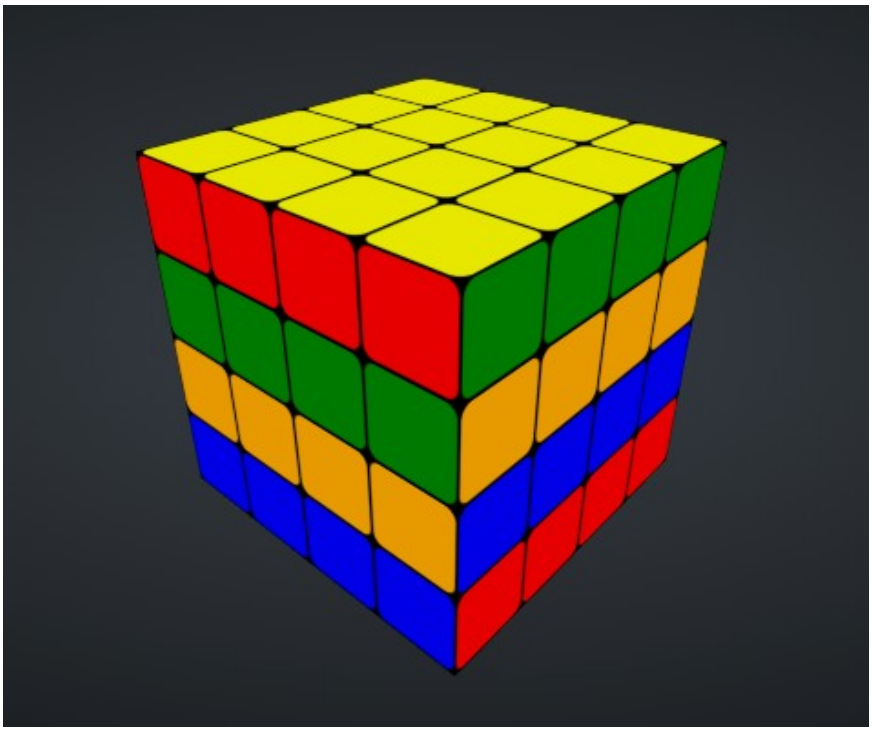


Рисунок 39 – Представлення блоку після виконання функції rotateCubeMatrices, кожну матрицю повернуто вправо на кількість разів, яку зазначено індексом

TransformDataCube. Дана функція відповідає за оборотне перетворення блоку даних. В ідеалі необхідно було б застосувати тут розроблений тривимірний клітинний автомат в якості перетворення, проте для цього потрібно дослідити можливість побудови однозначно оберненого КА, що не гарантовано. А в алгоритмі шифрування на основі SP-мережі оборотність будь-яких функцій перетворення є обов’язковою для однозначного розшифрування, тому в основі цієї функції вирішено використати просту операцію XOR. Проте нічого не заважає застосувати концепцію сусідства з КА – було вирішено взяти 8 сусідів, які лежать на чотирьох діагоналях куба (див. рисунок 40), в такому випадку у нас виходить 4 елементарних КА. Правила переходу однакові для всіх – XOR поточної комірки та її сусідів між собою, порядок виконання автоматів не принциповий, так як XOR є комутативною операцією.

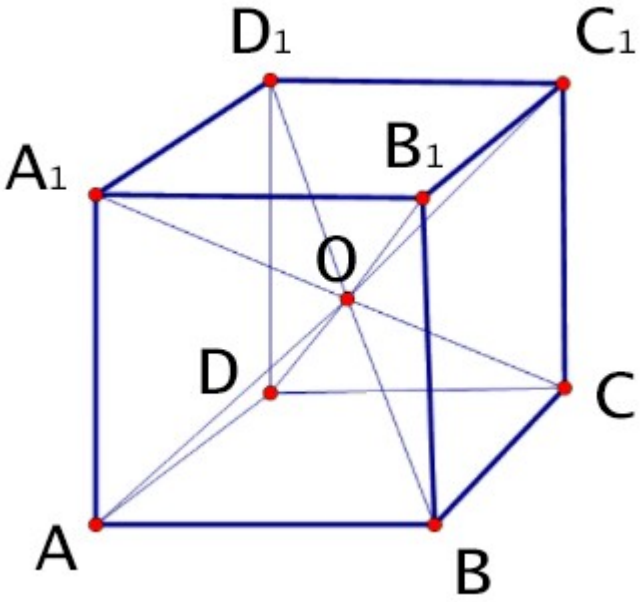


Рисунок 40 – Представлення функції transformDataCube, буквою O позначена поточна комірка, над якою відбувається перетворення, обрані сусіди знаходяться на краях діагоналей.

XorRoundKey. Операція підмішування раундового ключа, простий XOR байтів блоку даних та ключа, результат записується в блок даних.

Всі вище перелічені функції є перетвореннями, що застосовуються до блоку даних під час шифрування. Як було зазначено вище, функції SP-мережі повинні мати обернений варіант, який застосовується при розшифруванні. В даному випадку це функції invSubBytes (обернена підстановка) та invRotateCubeMatrices (обернена перестановка). Функції xorRoundKey та

transformDataCube є оберненими самі до себе через властивість операції XOR, єдина різниця це подача ключів до xorRoundKey в зворотньому порядку.

Тепер варто з’ясувати ще один важливий аспект – функцію розгортання ключа. І саме тут найкраще застосовується розроблений тривимірний клітинний автомат. Ця операція не потребує зворотності, просто потрібно згенерувати стільки ключів із заданого, скільки в алгоритмі раундів. Для цього раундовий ключ, так само як і блок, буде тривимірним масивом. Для зручності раундові ключі ініціалізуються всі одразу перед шифруванням і зберігаються у списку, на кожному раунді шифрування обирається потрібний ключ. Так як вони зберігаються окремо, після переходу до наступного блоку їх не потрібно перегенеровувати. Функція в якій реалізовано розгортання ключа за допомогою тривимірного КА називається transformKeyCube, одне її виконання робить перехід на наступну генерацію КА, замінюючи блок раундового ключа новим. Потім цей ключ додається в список. Дана операція повторюється стільки, скільки потрібно раундових ключів.

Об’єднання розроблених частин алгоритму в єдину систему

Після завершення проектування функцій алгоритму потрібно визначити в якому порядку вони будуть застосовуватися під час шифрування. Спроектований порядок шифрування зображено на рисунку 41.

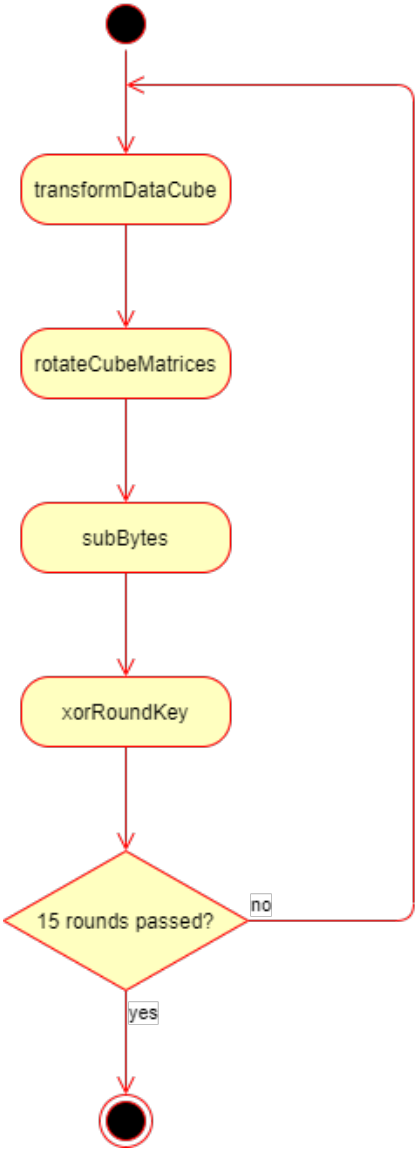


Рисунок 41 – Порядок шифрування одного блоку.

Відповідно порядок розшифрування буде оберненим, із застосуванням зворотних функцій перетворення та зворотнім порядком подання ключів в алгоритм (рисунок 42).

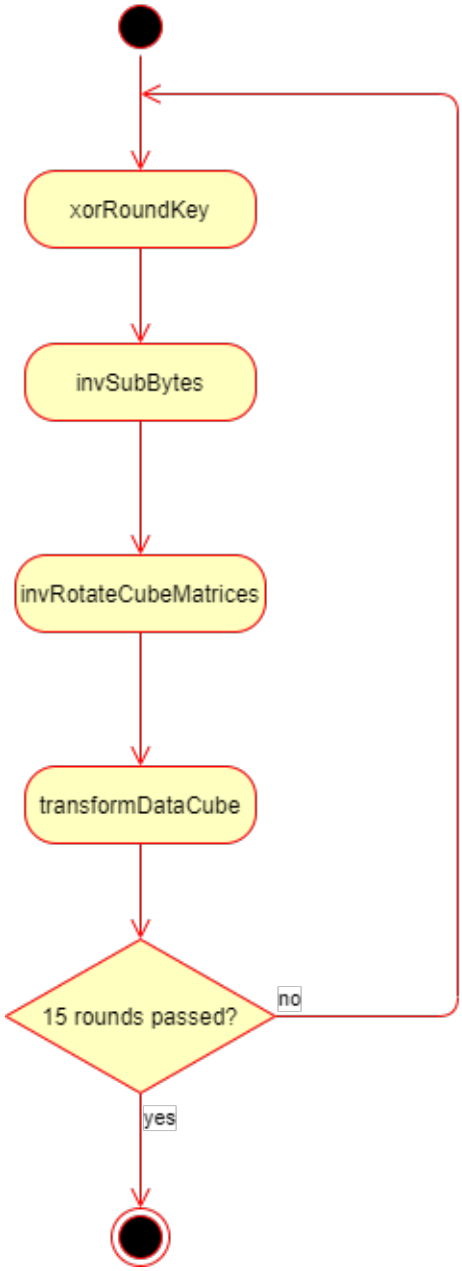


Рисунок 42 – Порядок розшифрування одного блоку

Варто зазначити, що для реалізації всіх цих головних функцій було розроблено декілька допоміжних.

Порядок використання та приклад роботи системи

Система симетричного блокового шифрування розроблена у вигляді бібліотеки підпрограм, тому що її основне призначення – це застосування в інших програмних додатках, які потребують використання шифрів для захисту даних. Бібліотека складається з двох пакетів: core та util, перший містить основні java-класи системи, другий – допоміжні (рисунок 43).

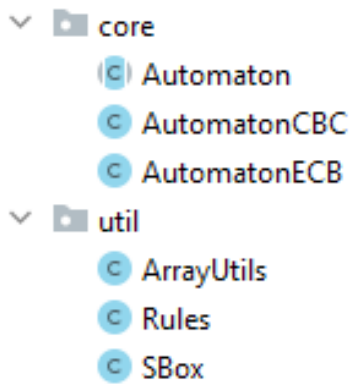


Рисунок 43 – Структура бібліотеки розробленої криптосистеми

Робота з системою відбувається через основні класи. Вона проходить в такому порядку:

1 За допомогою фабричних методів класу Automaton ми ініціалізуємо об’єкт криптосистеми. Кожен метод повертає свій тип об’єкту, який реалізує певний режим блокового шифрування (ECB або CBC). В параметр фабричного методу записується 64-байтовий ключ за допомогою якого система шифруватиме дані. Ключ бажано формувати за допомогою криптостійких випадкових генераторів, в мові Java такий реалізовано класом SecureRandom. На рисунку 44 показано ініціалізацію криптосистеми в режимі ECB.

```
byte[] key = new byte[64];
SecureRandom sr = new SecureRandom();
sr.nextBytes(key);
AutomatonECB cs = Automaton.createECB(key);
```

Рисунок 44 – Ініціалізація криптосистеми

2 Після ініціалізації, система готова до роботи. Тепер, для шифрування просто потрібно викликати метод encrypt і його параметром задати відкрите повідомлення. Для режимів, що потребують вектор ініціалізації (як от CBC), вектор записується другим параметром. В результаті метод нам поверне шифротекст. Якщо потрібно застосувати розшифрування, то порядок такий самий, лише викликається функція decrypt. Наведемо приклад роботи криптосистеми, зашифрувавши деякий відкритий текст на випадковому ключі і потім вивівши результати шифрування та розшифрування у консоль (рисунки 45 і 46).

```
String plaintext = "Це приклад відкритого повідомлення для шифрування";
System.out.println("plaintext:");
System.out.println(plaintext);

byte[] cipher = cs.encrypt(plaintext.getBytes(StandardCharsets.UTF_8));
System.out.println("ciphertext:");
System.out.println(new String(cipher, StandardCharsets.UTF_8));

byte [] decrypted = cs.decrypt(cipher);
System.out.println("decrypted:");
System.out.println(new String(decrypted, StandardCharsets.UTF_8));
```

Рисунок 45 – Застосування шифрування та розшифрування

```
plaintext:
Це приклад відкритого повідомлення для шифрування
ciphertext:
E◆◆◆⦿◆▯▯2◆◆9◆◆I▯◆◆{◆T◆◆l◆◆▯◆{◆$Y:◆8◆▯▯◆M$◆     K◆▯◆&◆f◆◆◆1◆%◆▯◆▯◆◆◆◆▯◆▯{G◆o◆j
◆b#◆(◆{▯$◆H◆↯◆◆▯◆3?◆◆◆◆l◆7▯◆◆{◆◆a◆`<◆◆X0◆Z◆◆
decrypted:
Це приклад відкритого повідомлення для шифрування
```

Рисунок 46 – Результат виконання розробленого шифру

Варто зазначити, що хоч дана реалізація криптосистеми виконана мовою Java, її можна реалізувати на будь-якій поширеній мові програмування. А так як це є бібліотека підпрограм для розробників, то єдиною вимогою до апаратного забезпечення є підтримка інструментів розробки для тієї чи іншої мови програмування, в нашому випадку для Java. Розроблена у випускній роботі криптосистема працює без застосування сторонніх бібліотек, тому для її використання не потрібно підключати зовнішні залежності.

Тестування роботи програмного засобу

Для перевірки коректності роботи розробленої криптосистеми є зміст лише протестувати роботу його функцій. Так як зовнішня взаємодія системи відбувається лише під час ініціалізації та виклику функцій шифрування і розшифрування, потрібно перевіряти лише параметри, які подаються в систему. Тип цих параметрів визначено статично, тому, все що треба, – це продумати деякі виключні ситуації, логіку опрацювання яких визначатиме користувач шифру, наприклад задання ключа неправильного розміру під час ініціалізації (рисунок 47).

```
public static AutomatonECB createECB(byte[] key) throws InvalidKeyException {
    if (key.length != 64) {
        throw new InvalidKeyException(String
                .format("Provided key has incorrect length (%d), must be 64 bytes", key.length));
    }
```

Рисунок 47 – Виключна ситуація, що виникає при введенні неправильного ключа в алгоритм

Але і функціональне тестування може обійтись лише перевіркою синтаксису і аналізом виводу результату роботи криптосистеми. Структура алгоритму шифрування досить ієрархічна, загальні функції викликають точніші, і все це по суті зводиться до проходження по байтах масиву і маніпуляцій над ними. Найпростіші перетворення (такі як правила елементарних клітинних автоматів) та складніші послідовні перетворення на їхній основі просто потребують уважності під час реалізації. Правильність роботи інших функцій, що впливають на шифрування, можна прослідкувати за результатом роботи всієї криптосистеми – наприклад помилки функцій з класу ArrayUtils, що зустрічались в ході розробки, одразу було виявлено і виправлено завдяки виключним ситуаціям під час виконання. Коректність роботи операцій pad, unpad та інших, які можуть мати помилки в логіці, перевіряється по виводу шифрування та розшифрування. Безпосередньо протестувати потрібно було лише функцію rotateMatrix, бо вона досить комплексна. Для цього потрібно просто створити тестову матрицю 4x4, вивести її, потім застосувати до неї rotateMatrix та вивести знову, після цього порівняти результати. На рисунку 48 показано що функція успішно пройшла цей невеликий тест.

```
1, 2, 3, 4,
5, 6, 7, 8,
9, 10, 11, 12,
13, 14, 15, 16,

13, 9, 5, 1,
14, 10, 6, 2,
15, 11, 7, 3,
16, 12, 8, 4,
```

Рисунок 48 – Тест правильності роботи функції rotateMatrix класу ArrayUtils

Тестування криптостійкості розробленого алгоритму шифрування

Останнім і одним з найважливіших завдань, є дослідження надійності розробленої криптосистеми. Так як максимальне ускладнення можливості отримання відкритого повідомлення з шифротексту є ключовою задачею алгоритму шифрування, важливо щоб результат роботи розробленого алгоритму був максимально випадковим. Дослідження цієї міри випадковості буде проведено за допомогою вже згаданого в третьому пункті першого розділу випускної роботи пакету статистичних тестів NIST STS.

Порядок тестування буде таким: потрібно згенерувати як мінімум 12,5 мільйонів байтів відкритого тексту, тому що статистичний пакет працює з послідовностями довжиною мінімум у 100 мільйонів бітів; далі криптоалгоритм буде обробляти згенерований текст, використовуючи один ключ, різну кількість раундів (від 1 до 15), кожна отримана послідовність шифротексту буде подаватись у статистичний пакет для тестування; після отримання всіх результатів буде досліджено як змінюється результат проходження тестів протягом 15 раундів і буде зроблено висновки про остаточну криптостійкість розробленого алгоритму та чи є обрана кількість раундів доцільною.

NIST STS містить 16 загальних статистичних тестів, під час виконання яких обчислюється 188 значень імовірності *p*, їх можна розглядати як результат роботи окремих тестів [28].

В таблиці 2 наведено опис кожного тесту, включаючи значення статистичного показника, що отримується, та дефект, для виявлення якого застосовується тест.

Таблиця 2 – Опис статистичних тестів пакету NIST STS [25]

№	Статистичний тест	Статистичний показник	Дефект, що виявляється
1.	Частотний (монобітовий тест)	Нормалізована абсолютна сума значень елементів послідовності	Надто багато нулів або одиниць у послідовності
2.	Частотний тест (в середині блоку)	Міра узгодженості кількості одиниць, що спостерігаються з тим, що очікується теоретично.	Локалізовані відхилення частоти появи одиниць в блоці від ідеального значення 0,5
3.	Перевірка накопичених сум	Максимальне відхилення значень накопиченої суми елементів послідовності від початкової точки відліку (точка 0)	Велика кількість одиниць або нулів на початку або наприкінці двійкової послідовності
4.	Перевірка серій	Загальна кількість серій на усій довжині послідовності	Надто швидка або надто повільна зміна знаку у ході

			генерації послідовності
5.	Перевірка максимальної довжини серії у блоці.	Міра узгодженості значень максимальної довжини, що спостерігаються, із значенням, що очікується теоретично	Відхилення від теоретичного закону розподілення максимальних довжин серій одиниць.
6.	Перевірка рангу двійкової матриці	Міра узгодженості значення рангів різного порядку, що спостерігаються, із значенням, що очікується теоретично	Відхилення емпіричного закону розподілення значень рангів матриць від теоретичного, що вказує на залежність символів у послідовності.
7.	Спектральний аналіз на основі дискретного перетворення Фур‘є	Нормалізована різниця кількості частотних компонент, що спостерігаються із тією, що очікується, які перевищують 95% рівень порогу.	Виявлення періодичних складових (трендів) у двійковій послідовності.
8.	Перевірка шаблонів, Що перекриваються	Міра узгодженості кількості шаблонів, що перекриваються, у послідовності із теоретичним значенням.	Велика кількість m-бітових серій із одиниць у послідовності.
9.	Універсальний тест Маурера	Сума логарифму відстані між однобітовими шаблонами.	Можливість стиснення послідовності.
10	Ентропійний тест	Міра узгодженості значення ентропії джерела із тим, що теоретично очікується для випадкового джерела.	Нерівномірність розподілення m-бітових слів у послідовності (регулярність властивостей джерела)
11	Перевірка випадкових відхилень	Міра узгодженості кількості візитів при випадковому блуканні в заданий стан в середині циклу із тим, що очікується теоретично.	Відхилення від теоретичного закону розподілення візитів у конкретний стан при випадковому блуканні.
12	Перевірка випадкових відхилень (варіант)	Загальна кількість візитів при випадковому блуканні.	Відхилення від теоретично очікуємої загальної кількості візитів при випадковому блуканні у заданий стан.
13	Послідовний тест	Міра узгодженості кількості усіх варіантів m-бітових шаблонів, що	Нерівномірність розподілення m-бітових

		зустрілись, із тією, що очікується теоретично.	слів у послідовності.
14.	Перевірка стиснення згідно алгоритму Лемпеля-Зіва	Кількість різних слів у послідовності.	Великий ступінь стиснення послідовності, що тестується порівняно із ступенем стиснення, що очікується для випадкової послідовності.
15.	Перевірка шаблонів, що не перекриваються	Міра узгодженості кількості неперіодичних шаблонів у послідовності із теоретичним значенням.	Велика кількість заданих неперіодичних шаблонів у послідовності.
16	Перевірка лінійної складності	Міра узгодженості кількості подій, що полягають у появі фіксованої довжини еквівалентного ЛРР для заданого блоку із теоретичним.	Відхилення емпіричного розподілу довжин еквівалентних ЛРР для послідовностей фіксованої довжини від теоретичного закону розподілення для випадкової послідовності, що вказує на недостатню складність послідовності, що тестується.

Оцінка отриманих результатів проводиться таким чином – NIST STS розділяє подану послідовність зі 100 млн бітів на 100 рівних підпослідовностей, кожна з них проходить всі 188 тестів. Вважається, що повна послідовність пройшла конкретний тест, якщо хоча би 96 із 100 її підпослідовностей пройшли його (це значення називається пропорцією). Отже чим більше значення пропорції послідовності, тим краща її випадковість.

Отримані результати дослідження зручно представити в табличному вигляді. Таким чином для всіх 15 досліджень буде показано скільки тестів з яким рівнем пропорції було пройдено, а також її середнє значення (таблиця 3).

Таблиця 3 – Результати проведення досліджень

Пропорція, % Раунди, к-ть		100	99	98	97	96	95	<95	Середнє значення
1	Кількість	71	67	39	7	3	1	-	0.99027
2		73	58	39	8	10	-	-	0.98936
3		60	64	44	18	-	1	1	0.98846
4		65	68	31	16	3	4	1	0.98840
5		76	60	39	12	1	-	-	0.99053
6		72	56	40	14	5	-	1	0.98909

7		67	59	39	12	10	1	-	0.98840
8		71	72	40	5	-	-	-	0.99112
9		75	60	36	11	3	3	-	0.98979
10		77	66	29	13	2	1	-	0.99064
11		75	55	35	11	11	1	-	0.98899
12		69	60	40	13	4	2	-	0.98909
13		83	49	39	11	6	-	-	0.99021
14		86	50	38	13	1	-	-	0.99101
15		61	64	39	16	3	5	-	0.98793

Додатково можна побудувати графік зміни середнього значення пропорції із збільшенням кількості раундів (рисунок 49).

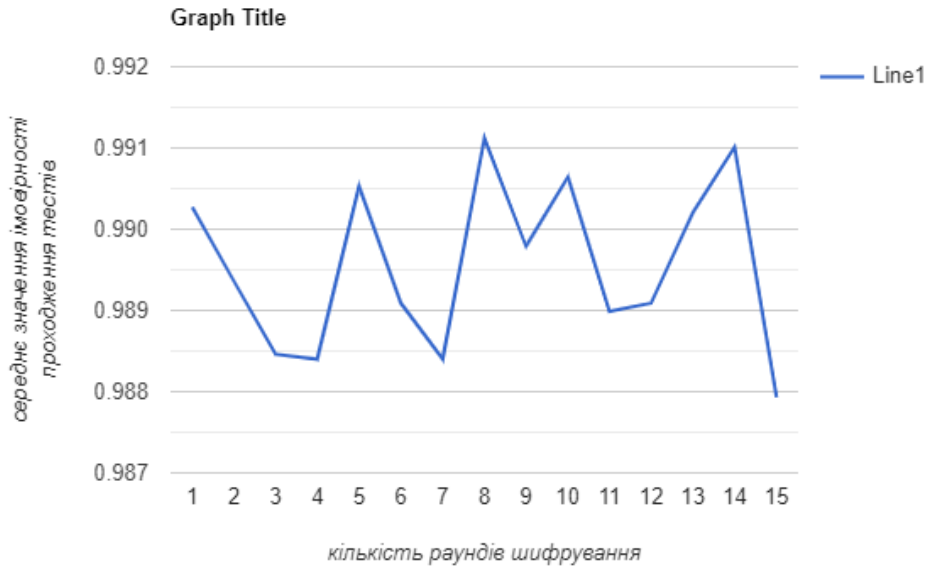


Рисунок 49 – Графік залежності середнього значення пропорції від кількості раундів

Проаналізувавши результати дослідження, можна сказати що кількість раундів неоднорідно впливає на міру випадковості шифротексту. По графіку видно, що під час 15 раундів шифрування найкраща криптостійкість буде досягнута на 8 і 14 раундах. На 15 раунді криптостійкість падає і провалюються декілька тестів. Тому якщо подібна тенденція буде зберігатися і для інших комбінацій відкритого тексту та ключа – варто змінити кількість раундів на 8 або 14. Але загалом – розроблений алгоритм в його теперішньому вигляді демонструє непогані результати.

Наступним етапом, етапом вдосконалення розробленого симетричного криптоалгоритму буде конструювання таблиць заміни на основі клітинних

автоматів, які будуть залежними від ключа та дослідження їх криптографічної стійкості, а також модифікація алгоритму обробки блоку тексту, який у цьому варіанті побудований на основі операції XOR. Планується використання зворотних правил клітинних автоматів.

Використана література

1. Закон України "Про захист персональних даних".Сайт Верховної ради України. -https://zakon.rada.gov.ua/laws/show/2297-17. - (дата звернення: 24.07.2021 р.).

2. Клітинний автомат. Вікіпедія: Вільна енциклопедія. -https://uk.wikipedia.org/wiki/%D0%9A%D0%BB%D1%96%D1%82%D0%B8%D0%BD%D0%BD%D0%B8%D0%B9_%D0%B0%D0%B2%D1%82%D0%BE%D0%BC%D0%B0%D1%82. - (дата звернення: 24.07.2021 р.).

3.Wolfram S.Random sequence generation by cellular automata.Advances in Applied Mathematics. - 1986. -Vol. 7. - pp. 123-164.

4.Nandi S., B. K. Kar, and P. P. Chaudhuri.Theory and applications of cellular automata in cryptography.IEEE Trans. Computers. - 1994. - 43(12). - pp. 1346–1357.

5.Mazoyer J. and Terrier V. Signals in one-dimensional cellular automata.Theoretical Computer Science. - 1999. - 217(1). - pp.53–80.

6.Mihaljevic M., Y. Zheng, and H. Imai. A cellular automaton based fast one-way hash function suitable for hardware implementation.Public Key Cryptography. -1998. - volume 1431. - pp. 217–234.

7.Seredynski F., Bouvry P., Zomaya A.V. Cellular automata computations and secret key cryptography. Parallel Computing. - 2004. - 30. - pp. 753-766.

8.Росошек С.К., Боровков А.А., Евсютин О.О. Криптосистемы клеточных автоматов. Прикладная дискретная математика. - 2008. - 1(1). - С. 43-49.

9.Jegadish Kumar K.J., Kesava Reddy K., Salivahanan S. Novel and Efficient Cellular Automata Based Symmetric Key Encryption Algorithm for Wireless Sensor Networks.International Journal of Computer Applications. - 2011. - Vol. 13, № 4. - pp. 30-37.

10. Сухинин Б. М. Высокоскоростные генераторы псевдослучайных последовательностей на основе клеточных автоматов. Прикладная дискретная математика. - 2010. -№ 2(8). - С. 34-41.

11. Norziana Jamil, Ramlan Mahmood, Muhammad Reza Z’aba. A New Cryptographic Hash Function Based on Cellular Automata Rules 30, 134 and Omega-Flip Network. 2012 International Conference on Information and Computer networks (ICICN 2012). - 2012. - Vol. 27. - pp. 163-169.

12. Wolfram, S. A New Kind of Science. - Wolfram Media, Inc. - 2002, 532 P.

13. Classification of Cellular Automata. Wolfram's Classification of Cellular Automata. - http://www.ics.uci.edu/~eppstein/ca/wolfram.html. - (дата звернення 28.07.2021 р.).
14. Wolfram Alfa.Wolfram Alfa. - https://www.wolframalpha.com/. - (дата звернення: 28.07.2021).
15. Окіл Мура. Вікіпедія: Вільна енциклопедія. -https://uk.wikipedia.org/wiki/%D0%9E%D0%BA%D1%96%D0%BB_%D0%9C%D1%83%D1%80%D0%B0. - (дата звернення: 02.08.2021).
16. Окіл фон Неймана. Вікіпедія: Вільна енциклопедія. -https://uk.wikipedia.org/wiki/%D0%9E%D0%BA%D1%96%D0%BB_%D1%84%D0%BE%D0%BD_%D0%9D%D0%B5%D0%B9%D0%BC%D0%B0%D0%BD%D0%B0. - (дата звернення: 02.08.2021).
17. Гра Дж.Конвея "Життя". Вікіпедія: Вільна енциклопедія. -https://uk.wikipedia.org/wiki/%D0%96%D0%B8%D1%82%D1%82%D1%8F_(%D0%B3%D1%80%D0%B0). - (дата звернення: 02.08.2021).
18. Бандман О.Л. Клеточно-автоматные модели пространственной динамики. Системная информатика. - 2005. - в. 10. - С. 57-113.
19. Ванаг В.К. Исследование пространственно распределённых динамических систем методами вероятностного клеточного автомата. Успехи физических наук. - 1999. - Том 169, № 5. - С. 481-505.
20.Жихаревич, В., Остапов С. Моделирование процессов самоорганизации и эволюции систем методом непрерывных асинхронных клеточных автоматов. Computing. - 2009. - Т. 8, вип. 3. - С. 61-69.
21.Валь Л.О., Жихаревич В.В., Остапов С.Е. Розробка і дослідження криптостійкого генератора бінарної послідовності на основі клітинних автоматів. Науковий вісник Чернівецького університету. Комп’ютерні системи та компоненти. - 2009. - Вип. 479. - С.147-150.
22.Валь Л.О., Жихаревич В.В.,Остапов С.Е. Застосування клітинних автоматів для генерування бінарних псевдовипадкових послідовностей. Науковий вісник Чернівецького університету. Комп’ютерні системи та компоненти. - 2010. - Т.1, вип. 1. - С.67-72.
23.Чайка Л.О., Жихаревич В.В., Остапов С.Е. Криптостійкий генератор псевдовипадкової послідовності на основі клітинних автоматів. Вісник Київського національного університету імені Тараса Шевченка. Серія фізико-математичні науки. - 2011. - Вип. 1. - С.215-219.
24. Кренгель Е.И. Исследование и разработка новых классов псевдослучайных последовательностей и устройств их генерации для систем с кодовым

разделением каналов.Електронний ресурс. - http://1-ebook.com/asu-peredacha-dannih/lineynaya-slojnost.html. -(дата звернення 31.08.2021).
25. Потій О.В., Лєншин А.В., Ізбенко Ю.А. Методика статистичного тестування NIST STS та математичне обґрунтування тестів. -Технічний звіт ІІТ – 001-2004. – Інститут інформаційних технологій. - 2004, 62 С.
26. В.В. Жихаревич, Чайка Л.О., Камінська О.В., Овчар Р.І., Остапов С.Е. Система захищеного обміну даними на основі клітинних автоматів. Науковий вісник Чернівецького університету. Комп’ютерні системи та компоненти. - 2011. - Т.2. – Вип. 1. - С.15-20.
27. Wolfram S. Tables of Cellular Automaton Properties. - [Електронний ресурс]. - Режим доступу: https://content.wolfram.com/uploads/sites/34/2020/07/cellular-automaton properties.pdf.
28. Статистические тесты NIST STS.Википедия - Электронная энциклопедия. - https://ru.wikipedia.org/wiki/Статистические_тесты_NIST. - (дата звернення: 02.09.2021 р.).

ДОДАТОК А

Інструкція роботи з пакетом статистичного тестування NIST STS

Пакет статистичного тестування NIST STS (National Institute of Standards and Technologies STatistical Suite) було розроблено для тестування статистичних характеристик шифрів, які брали участь у міжнародному конкурсі симетричних шифрів, що був оголошений у 1997 році. Однією з мов участі в цьому конкурсі було успішне проходження шифрами-конкурсантами усіх тестів цього пакету. З того часу він став стандартом статистичного тестування криптографічних примітивів, особливо генераторів випадкових та псевдовипадкових бінарних послідовностей. Використовуємо його й ми для визначення статистичних характеристик розроблених криптографічних примітивів на КА.

Як визначити, наскільки якісний шифр або генератор псевдовипадкових послідовностей ми розробили? За якими критеріями це можна зробити?

Однією з основних характеристик будь-якого криптографічного програмного забезпечення (блокового шифру, генератора псевдовипадкових послідовностей тощо) є його статистичні характеристики. Результат шифрування/генерування випадкової послідовності повинен мінімально відрізнятися від випадкового числа.

Для визначення цієї різниці криптографічна спільнота використовує пакет статистичного тестування, розроблений NIST перед конкурсом AES для тестування генераторів псевдовипадкових послідовностей. З того часу він став стандартом для оцінки статистичних характеристик шифрів, генераторів та інших криптографічних примітивів.

Пакет містить 16 статистичних тестів. В залежності від вхідних параметрів обчислюється 189 значень імовірності *P*, які можна розглядати як результат роботи окремих тестів. У табл. 1 приводяться зібрані дані по усіх тестах із вказівкою кількості значень, що обчислюються, імовірності *P*, фізичного змісту статистики тесту і дефекту, на виявлення якого спрямовано тест.

Таблиця 1

Опис статистичних тестів пакету NIST STS [25]

№ з/п	Статистичний тест	Статистика тесту c(S)	Дефект, що виявляється
1	Частотний (монобітний тест)	Нормалізована абсолютна сума значень елементів	Надто багато нулів або одиниць

		послідовності	у послідовності
2	Частотний тест (в середині блоку)	Міра узгодженості кількості одиниць, що спостерігаються із тим, що очікується теоретично.	Локалізовані відхилення частоти появи одиниць в блоці від ідеального значення ½
3	Перевірка накопичених сум	Максимальне відхилення значень накопиченої суми елементів послідовності від початкової точки відліку (точка 0)	Велика кількість одиниць або нулів на початку або наприкінці двійкової послідовності
4	Перевірка серій	Загальна кількість серій на усій довжині послідовності	Надто швидка або надто повільна зміна знака у ході генерації послідовності
5	Перевірка максимальної довжини серії у блоці.	Міра узгодженості значень максимальної довжини, що спостерігаються, із значенням, що очікується теоретично	Відхилення від теоретичного закону розподілення максимальних довжин серій одиниць.
6	Перевірка рангу двійкової матриці	Міра узгодженості значення рангів різного порядку, що спостерігаються, із значенням, що очікується теоретично	Відхилення емпіричного закону розподілення значень рангів матриць від теоретичного, що вказує на залежність символів у послідовності.
7	Спектральний аналіз на основі дискретного перетворення Фур‘є	Нормалізована різниця кількості частотних компонент, що спостерігаються із тією, що очікується, які перевищують 95% рівень порогу.	Виявлення періодичних складових (трендів) у двійковій послідовності.
8	Перевірка шаблонів, що перекриваються	Міра узгодженості кількості шаблонів, що перекриваються, у послідовності із теоретичним значенням.	Велика кількість m- бітних серій із одиниць у послідовності.
9	Універсальний тест Маурера	Сума логарифму відстані між l –бітними шаблонами.	Можливість стиснення послідовності.
10	Ентропійний тест	Міра узгодженості значення ентропії джерела	Нерівномірність розподілення m- бітних слів у послідовності

		із тим, що теоретично очікується для випадкового джерела.	(регулярність властивостей джерела)
11	Перевірка випадкових відхилень	Міра узгодженості кількості візитів при випадковому блуканні в заданий стан в середині циклу із тим, що очікується теоретично	Відхилення від теоретичного закону розподілення візитів у конкретний стан при випадковому блуканні
12	Перевірка випадкових відхилень (варіант)	Загальна кількість візитів при випадковому блуканні	Відхилення від теоретично очікуємої загальної кількості візитів при випадковому блуканні у заданий стан.
13	Послідовний тест	Міра узгодженості кількості усіх варіантів m-бітних шаблонів, що зустрілись, із тією, що очікується теоретично.	Нерівномірність розподілення m- бітних слів у послідовності.
14	Перевірка стиснення згідно алгоритму Лемпеля-Зіва	Кількість різних слів у послідовності	Великий ступінь стиснення послідовності, що тестується порівняно із ступенем стиснення, що очікується для випадкової послідовності.
15	Перевірка шаблонів, що не перекри-ваються	Міра узгодженості кількості неперіодичних шаблонів у послідовності із теоретичним значенням.	Велика кількість заданих неперіодичних шаблонів у послідовності.
16	Перевірка лінійної складності	Міра узгодженості кількості подій, що полягають у появі фіксованої довжини еквівалентного ЛРР для заданого блоку із теоретичним.	Відхилення емпіричного розподілу довжин еквівалентних ЛРР для послідовностей фіксованої довжини від теоретичного закону розподілення для випадкової послідовності, що вказує на недостатню складність послідовності, що тестується.

Таким чином у результаті тестування двійкової послідовності формується вектор значень імовірності 2^{24}. Аналіз складових P_i даного вектору дозволяє вказати на конкретні дефекти випадковості послідовності, що тестується.

Нижче коротко описано основні правила використання цього пакету для тестування статистичних характеристик різних послідовностей.

Для запуску графічної оболонки тестів використовують файл NIST_UI.exe. Як видно з назви файлу, використовується версія для ОС Windows. Для ОС Linux пакет також існує, але нам не вдалося знайти його готовим, скажімо у вигляді *.deb-файлу, а лише у бінарному вигляді. Тому ми рекомендуємо використовувати саме windiws-версію цього пакету.

Після його запуску на ерані виникне головна форма пакету, яка дозволяє керувати усіма його можливостями (рис.1).

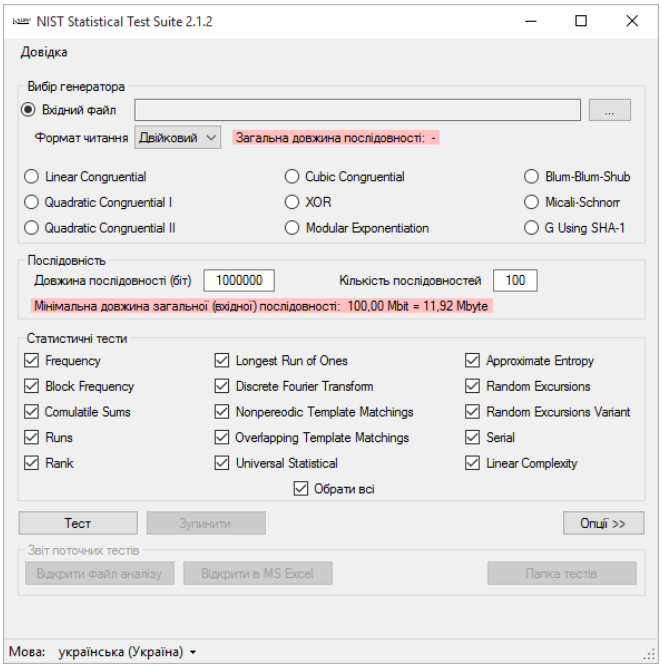


Рисунок 1: Головне вікно програми

Тут ми можемо вибрати генератор: або вхідний файл, або один з вбудованих генераторів для порівняння. Нас цікавить вхідний файл, тому ми натискаємо кнопку вибору файлів:

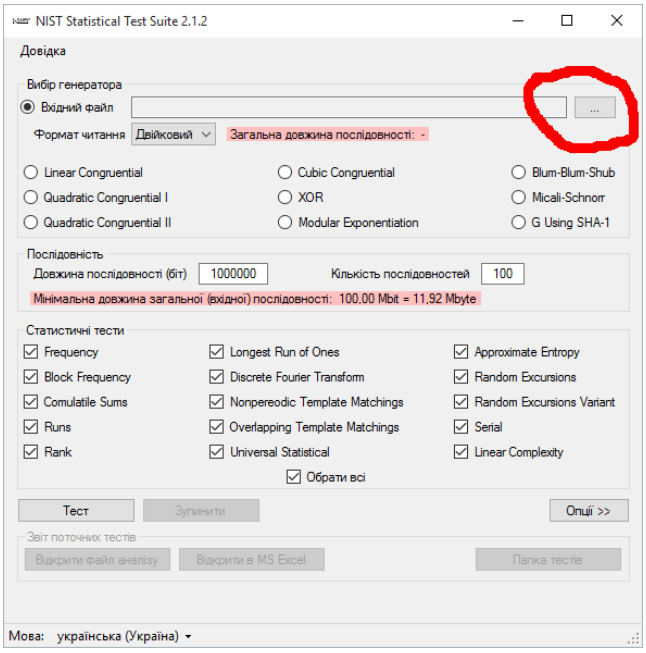


Рисунок 2: Кнопка вибору файлів

Якщо ми завантажили файл, згенерований шифром або генереатором послідовностей, який ми хочемо дослідити, система автоматично визначить, який він (двійковий або текстовий), яка його довжина та на скільки послідовностей його можна розбити. В результаті ми побачимо картинку, зображену на рис.3:

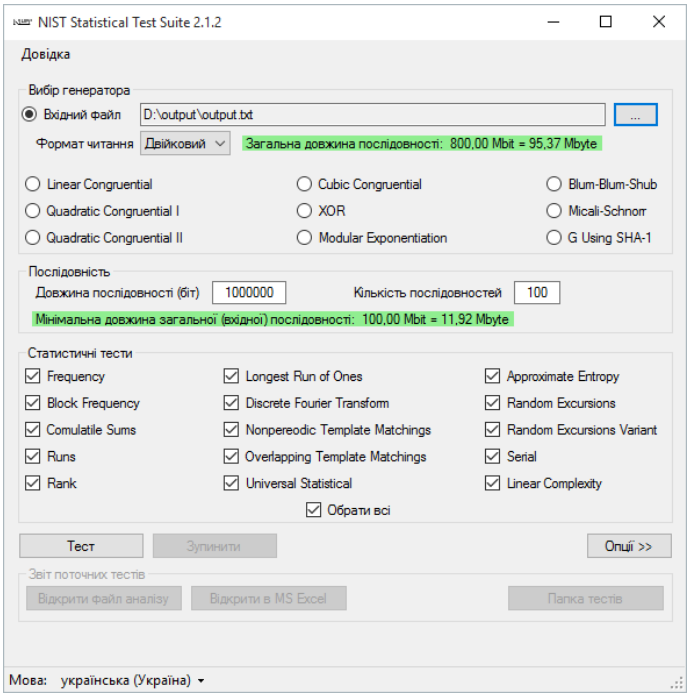


Рисунок 3: Система готова до тестування

Якщо повідомлення програми зафарбовано зеленим кольором, як на рисунку 3, система може протестувати вказаний файл. Якщо ж вони зафарбовані червоним — файл непридатний для тестування. В цьому випадку треба або вибрати інший файл, або перегенерувати його.

Для початку тестування натискаємо кнопку “Тест”. На формі з’явиться ProgressBar, який показує прогрес процесу тестування. По закінчення тестування система видасть повідомлення, як на рис.4.

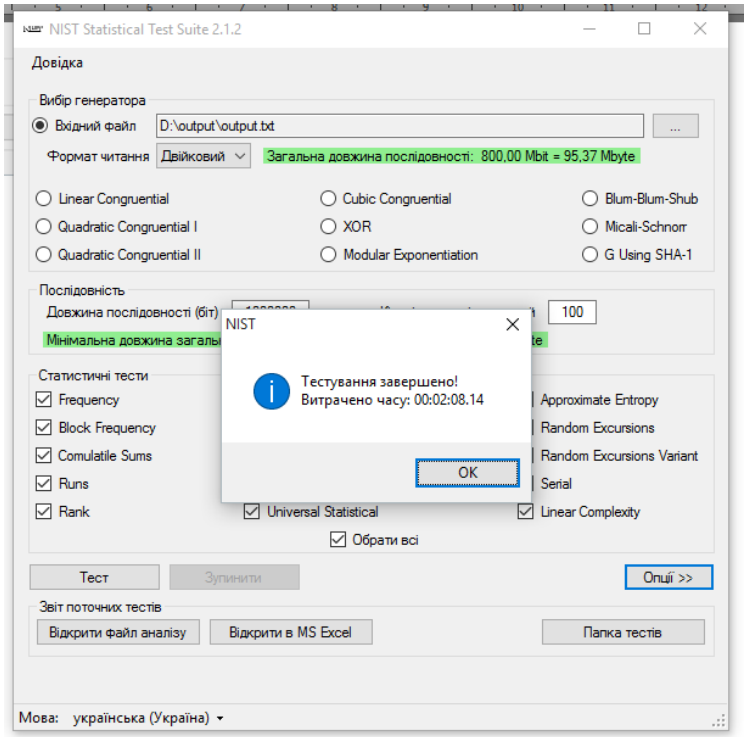


Рисунок 4: Повідомлення про завершення тестування

Тепер ми можемо подивитися на результати тестування. Це можливо зробити у два способи. Перший — експортувати результати в MS Excel. Тоді результати виглядатимуть як на рис.5.

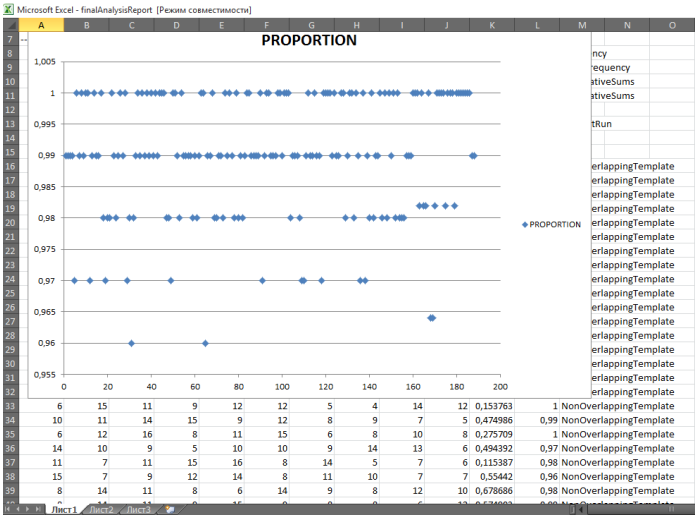
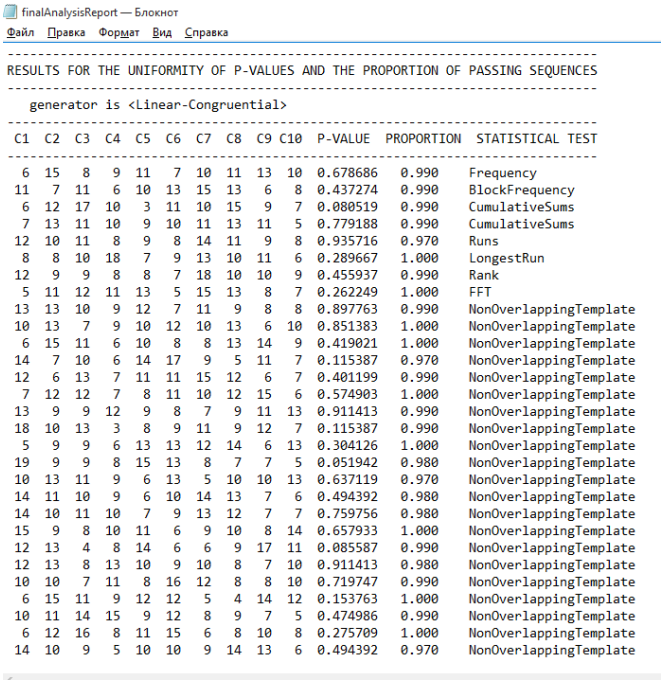


Рисунок 5: Результати тестування у MS Excel

Діаграму можна вставити безпосередньо у текст звіту або зберегти на диску. Якщо ж ми не хочемо використати вбудований експорт, можна отримати текстовий файл результатів тестування, вигляд якого подано на рис.6.



finalAnalysisReport — Блокнот

Файл Правка Формат Вид Справка

```
------------------------------------------------------------------------------
RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES
------------------------------------------------------------------------------
   generator is <Linear-Congruential>
------------------------------------------------------------------------------
 C1  C2  C3  C4  C5  C6  C7  C8  C9 C10  P-VALUE  PROPORTION  STATISTICAL TEST
------------------------------------------------------------------------------
  6  15   8   9  11   7  10  11  13  10  0.678686   0.990    Frequency
 11   7  11   6  10  13  15  13   6   8  0.437274   0.990    BlockFrequency
  6  12  17  10   3  11  10  15   9   7  0.080519   0.990    CumulativeSums
  7  13  11  10   9  10  11  13  11   5  0.779188   0.990    CumulativeSums
 12  10  11   8   9   8  14  11   9   8  0.935716   0.970    Runs
  8   8  10  18   7   9  13  10  11   6  0.289667   1.000    LongestRun
 12   9   9   8   8   7  18  10  10   9  0.455937   0.990    Rank
  5  11  12  11  13   5  15  13   8   7  0.262249   1.000    FFT
 13  13  10   9  12   7  11   9   8   8  0.897763   0.990    NonOverlappingTemplate
 10  13   7   9  10  12  10  13   6  10  0.851383   1.000    NonOverlappingTemplate
  6  15  11   6  10   8   8  13  14   9  0.419021   1.000    NonOverlappingTemplate
 14   7  10   6  14  17   9   5  11   7  0.115387   0.970    NonOverlappingTemplate
 12   6  13   7  11  11  15  12   6   7  0.401199   0.990    NonOverlappingTemplate
  7  12  12   7   8  11  10  12  15   6  0.574903   1.000    NonOverlappingTemplate
 13   9   9  12   9   8   7   9  11  13  0.911413   0.990    NonOverlappingTemplate
 18  10  13   3   8   9  11   9  12   7  0.115387   0.990    NonOverlappingTemplate
  5   9   9   6  13  13  12  14   6  13  0.304126   1.000    NonOverlappingTemplate
 19   9   9   8  15  13   8   7   7   5  0.051942   0.980    NonOverlappingTemplate
 10  13  11   9   6  13   5  10  10  13  0.637119   0.970    NonOverlappingTemplate
 14  11  10   9   6  10  14  13   7   6  0.494392   0.980    NonOverlappingTemplate
 14  10  11  10   7   9  13  12   7   7  0.759756   0.980    NonOverlappingTemplate
 15   9   8  10  11   6   9  10   8  14  0.657933   1.000    NonOverlappingTemplate
 12  13   4   8  14   6   6   9  17  11  0.085587   0.990    NonOverlappingTemplate
 12  13   8  13  10   9  10   8   7  10  0.911413   0.980    NonOverlappingTemplate
 10  10   7  11   8  16  12   8   8  10  0.719747   0.990    NonOverlappingTemplate
  6  15  11   9  12  12   5   4  14  12  0.153763   1.000    NonOverlappingTemplate
 10  11  14  15   9  12   8   9   7   5  0.474986   0.990    NonOverlappingTemplate
  6  12  16   8  11  15   6   8  10   8  0.275709   1.000    NonOverlappingTemplate
 14  10   9   5  10  10   9  14  13   6  0.494392   0.970    NonOverlappingTemplate
```

Рисунок 6: Результати тестування у текстовому вигляді

Як оцінити отримані результати? NIST STS розділяє вхідний файл (якщо його довжина дозволяє) на 100 однакових підпослідовностей по 1 млн бітів кожна. Тому довжина вхідного файлу повинна бути 100Мб (12,5 МБ). До кожної з цих підпослідовностей застосовуються усі 189 статистичних тестів. Вважається, що повна послідовність пройшла конкретний тест, якщо хоча би 96 зі 100 підпослідовностей його пройшли. Тому найкращим вважається той шифр/генератор, вихідна послідовність якого пройшла усі тести з максимальною пропорцією.

Часто результати тестування показують як табличку, де вказано, скільки тестів NIST пройдено на якому рівні (див. Табл.2):

Таблиця 2. Результати порівняльного тестування 5 шифрів у табличному вигляді.

Ймовірність проходження тестів, %	Шифр1	Шифр2	Шифр3	Шифр4	Шифр5
	Кількість тестів, які пройшли тестування (%)				
100	81 (43%)	89 (47,1%)	129 (68,3%)	80 (42,5%)	70 (37%)

99	58 (31%)	65 (34,4%)	16 (8,5%)	57 (30,5%)	60 (32%)
98	36 (19%)	26 (14%)	23 (12,2%)	34 (18%)	43 (23%)
97	12 (6,5%)	6 (3,2%)	8 (4,2%)	13 (7%)	13 (7%)
96	1 (0,5%)	2 (1%)	2 (1%)	2 (1%)	2 (1%)
95	-	-	3 (1,6%)	2 (1%)	-
< 95	-	-	6 (3,2%)	-	-
Середнє значення	0,98571 4286	0,98714 2857	0,981904 762	0,98507 9365	0,984497 354

Ще одним простим методом порівняльної оцінки шифрів є середнє значення ймовірності проходження тестів. Дуже наближено можна вважати, що шифр/генератор, який має найбільше середнє значення ймовірності проходження тестів, має найкращі статистичні характеристики. З такої точки зору (в середньому) найкращим можна вважати Шифр 1, оскільки його середнє значення найбільше.