Всеукраїнській конкурс студентських наукових робіт
з природничих, технічних та гуманітарних наук

Спеціальність: Кібербезпека

Конкурсна робота

Вдосконалення методів передачі паролів через відкритий простір на основі урізноманітнення односторонніх функцій

Шифр роботи «Функція»

2020-2021 навчальний рік

Анотація

Питання захисту інформації в телекомунікаційних мережах дуже **актуальне**. Тому **актуальним** є також дослідження щодо вдосконалення алгоритмів шифрування для забезпечення захисту передачі інформації.

Мета роботи. Покращити якість процесу створення спільних ключів шифрування за допомогою відкритих каналів зв'язку в рамках алгоритму Діффі-Хелмана за допомогою створення односторонньої функції на основі клітинних автоматів з розширеними правилами поведінки.

Об'єкт досліджень – процес генерації ключів шифрування в умовах відкритих каналів зв'язку.

Предмет досліджень – створення нових односторонніх функцій в рамках алгоритму Діффі-Хелмана.

Завдання дослідження, які дозволяють досягти поставленої мети

1. Аналіз методів формування ключів шифрування для декількох агентів з використанням відкритих каналів зв'язку.
2. Аналіз можливостей вдосконалення односторонніх функцій за допомогою клітинних автоматів.
3. Формалізація та вдосконалення існуючих правил моделі клітинних автоматів.
4. Створення програмних інструментів шифрування інформації з використанням вдосконалених клітинних автоматів.
5. Створення методів статистичної оцінки працездатності клітинних автоматів.

Методи і підходи, що були використані: алгоритм Діффі-Хелмана, дискретна математика, теорія клітинних автоматів, математична статистика.

Наукова новизна результатів дослідження:

1. Вдосконалено алгоритм Діффі-Хелмана за рахунок створення односторонньої функції на базі клітинних автоматів з розширеним набором правил.

2. Автором вперше розширені правила клітинного автомату за напрямками базового правила зміни стану клітини, базового правила визначення околиці клітини, модифікованого правила визначення околиці клітини, правила смерті від віку клітин та багатокомпонентної системи клітин.

Практична значущість.

1. Створений алгоритм на базі клітинного автомата дозволяє гарантовано реалізувати ПЗ для генерації ключів шифрування як у вигляді програмного, так й у вигляді програмно-апаратного рішення для створення ключів для безпечної передачі інформації. Можливість створення програмно-апаратного рішення суттєво збільшує зламостійкість пристроїв, засобів, що виконують задачі постійної передачі інформації по відкритим каналам.

2. Програмне забезпечення для генерації ключів шифрування реалізоване на трьох алгоритмічних мовах C#, Python та MatLab, що дозволяє виконувати порівняльних аналіз результатів і свідомо змінювати мову реалізації при зміні особливостей постановки задачі.

3. В залежності від потреб користувачів шифрування в алгоритмі може бути ускладнено за допомогою легко змінних параметрів клітинного поля та поведінки клітин, що дозволить регулювати швидкість та надійність шифрування.

В роботі вдосконалений алгоритм Діффі-Хелмана в напрямку створення нового типу односторонніх функції на основі модифікованого клітинного автомату, керуючі параметри якого було розширено для забезпечення складнопередбачуваної поведінки. У роботі удосконалено відомий клітинний автомат таким чином, щоб дозволити створювати складнопередбачувані клітинні автомати, що можуть бути використані для генерації ключів шифрування інформації.

Ключові слова: алгоритм Діффі-Хелмана, клітинні автомати, одностороння функція.

Зміст

Анотація 2
Вступ 5
1. Аналіз існуючих розробок та постановка задачі 6
1.1. Класичний алгоритм Діффі-Хелмана 7
1.2. Класичні односторонні функції 8
1.3. Загальні поняття теорії клітинних автоматів 9
1.4. Постановка задачі 10
2. Вдосконалення алгоритму Діффі-Хелмана 12
2.1. Модифікація процедури для n агентів 12
2.2. Модифікація процесу формування початкових значень функції 13
2.3. Модифікація процесу обміну проміжними значеннями функції 13
3. Збільшення різноманітності односторонніх функції за рахунок вдосконалення правил клітинного автомату одної популяції 15
3.1. Вдосконалення базового правила зміни стану клітини 15
3.2. Введення правила замкненого простору 17
3.3. Базове правило визначення околиці клітини 17
3.4. Ускладнене правило визначення околиці клітини 18
3.5. Введення правила смерті від віку 19
3.6. Вдосконалення правил для реалізації багатопопуляційної моделі 21
4. Програмна реалізація модифікованого алгоритму Діффі-Хелмана 22
4.1. Структура програми генерації ключів шифрування 22
4.2. Апробація модифікованого алгоритму Діффі-Хелмана 22
4.3. Перевірка працездатності односторонньої функції на основі клітинного автомата 25
ВИСНОВКИ 30
Література 31
ДОДАТКИ 34

Вступ

В умовах постійного збільшення розрахункових можливостей комп'ютерів, а також зменшення їх вартості використання, все складніше забезпечити безпеку передачі та зберігання інформації в мережі Інтернет. Ведеться активне дослідження як і надійності методів генерації ключів шифрування [1], так і квантових суперкомп'ютерів, що можуть бути застосовані для паралельного виконання великої кількості складних операцій (наприклад для прямого перебору значень паролів до відповідних значень отриманих в результаті застосування хеш-функції).

Зазвичай паролі, що генеруються людьми не є достатньо надійними, про це свідчать дослідження [2]: досвідчений хакер здатен зламати 38% паролів за 4 години, 62% – за тиждень, 89% – за 5 тижнів.

Особливо гостро стоїть питання забезпечення безпечної передачі даних по відкритим каналам зв'язку, нерідко інформацію необхідно передавати постійно та в великих об'ємах, як от при передачі даних між банками [3, 4, 5] або іншими великими компаніями. Для цього зазвичай використовується алгоритм Діффі-Хеллмана [6], [7], що дозволяє створюється спільні ключі шифрування інформації. В його класичному варіанті для генерації ключів шифрування використовується **одностороння арифметична функція** з властивостями комутативності. Для дешифрування інформації зловмисник має знайти обернену функцію. В умовах, коли одностороння функція алгоритму хоча і є складною, але залишається арифметичною, підбір функції виглядає можливим та може бути виконаний за відносно невеликий проміжок часу (при наявності значних розрахункових потужностей, доступність яких росте з кожним днем). Тому збільшення зламостійкості односторонніх функцій є актуальною задачею.

1. Аналіз існуючих розробок та постановка задачі

Вважається, що поняття клітинний автомат вперше було введено Джоном фон Нейманом під час його роботи над теорією самовідтворюючихся систем. Більшість структур, що він досліджував були одновимірні та двовимірні. Найбільшу увагу автоматам приділяли у 60-х та 70-х роках, коли цю тему досліджували такі вчені, як Гордон Мур, Станіслав Улам, Едгар Кодд. Джон Конвей у 1970 році винайшов найвідоміший клітинний автомат – «Гра Життя». Він зацікавився проблемою, запропонованою Нейманом, оскільки хоча Нейману вдалося створити математичну модель самовідтворюючої машини, але вона працювала за дуже складними правилами. Конвей спростив ідеї Неймана і створив правила «Гри Життя» [8].

Створений автомат був досить стійким, тобто повне вимирання або повне заповнення поля маловірогідне. Так чи інакше через деякий проміжок часу автомат приходить до стану, коли залишаються або статичні, або циклічні сукупності фігур. Це робить класичний автомат «Гра Життя» неспроможним створювати достатньо непередбачувані структури клітин. Тому його можна використовувати лише як основу для більш складного автомату.

Особливістю теорії клітинних автоматів є те, що за допомогою неї може бути змодельована та описана велика кількість явищ [9, 10, 11, 12], наприклад для моделювання розповсюдження лісних пожеж [13], опису процесів квантового світу [14]. Але при цьому існує досить мало досліджень напряму пов’язану з використанням клітинних автоматів з поведінкою, що складно передбачити стороннім спостерігачем без доступу до керуючих параметрів. Одною із споріднених тематик є генерація псевдовипадкових чисел [15] за допомогою клітинних автоматів. Але проблемою клітинного автомату, що використовується в згаданій роботі [15] є те, що розглянутий автомат керується незмінними правилами, що аналогічно відомим хеш-функціям незмінні за своєю структурою можуть бути попередньо досліджені. У випадку з алгоритмом Діффі-Хелмана клітинний автомат має бути таким, щоб до нього не можна було застосувати брутфорс. Це створює невідповідність між властивостями вже

існуючих клітинних автоматів та потребами в більш надійному методі генерації спільних ключів шифрування в алгоритмі Діффі-Хелмана.

Мета роботи. Покращити якість процесу створення спільних ключів шифрування за допомогою відкритих каналів зв'язку в рамках алгоритму Діффі-Хелмана за допомогою створення односторонньої функції на основі клітинних автоматів з розширеними правилами поведінки.

1.1. Класичний алгоритм Діффі-Хелмана

В умовах, коли необхідно постійно змінювати паролі або ключі шифрування доступ до закритих каналів зв'язку зазвичай або відсутній, або потребує занадто багато ресурсів. Через це розробка та подальше використання засобу генерації ключів шифрування на кшталт алгоритму Діффі-Хелмана є доцільним.

Таким чином, якщо говорити про алгоритм Діффі-Хелмана, агенти мають одну можливість скористатися закритим каналом зв'язку, але усе наступне спілкування відбувається по відкритим каналам зв'язку.

Послідовність **алгоритму** виглядає наступним чином:

1. При першому зв'язку по закритому каналу зв'язку агенти визначають властивості односторонньої функції (в нашому випадку правила та керуючі початкові параметри клітинного автомату). В класичному варіанті алгоритму в якості функції виступає дискретний логарифм [7, 15].

2. Перший агент формує таємне слово $x1$ (в нашому випадку кількість ітерацій автомату). При цьому параметр $y0$ є початковим значенням функції (в нашому випадку початкове клітинне поле). Перший агент знаходить проміжне значення функції

$$y1 = f(y0, x1)$$

і через відкритий канал надсилає проміжне значення функції другому агенту.

3. Другий агент отримує проміжне значення функції $y1$ від першого агенту, формує своє таємне слово $x2$ та використовує ту ж саму функцію. Тепер другий агент знає новий ключ шифрування

$$y12 = f(y1, x2).$$

4. Далі агенти діють аналогічно (пп.2, 3), але тепер агенти міняються місцями:

$$y2 = f(y0, x2); \quad y21 = f(y2, x1).$$

Оскільки функція комутативна, то

$$y21 = f(y2, x1) = f(y1, x2) = y12.$$

Тепер обидва агенти знають новий ключ шифрування $y21$.

Ймовірні зловмисники прослуховують відкритий канал, знають проміжні значення функції $y1, y2$, але при цьому не знають нового паролю $y21$ та початкових значень $y0$.

1.2. Класичні односторонні функції

Однім із поширених варіантів односторонньої функції є дискретний логарифм [7, 15]. Розглянемо її використання для розглянутого вище алгоритму Діффі-Хелмана.

На початку агенти формують деякі числа g і p, які не є таємними і можливо відомі зловмиснику. Після цього агент 1 та агент 2 окремо формують таємні слова - дуже великі числа $x1$, $x2$ та кожен використовує їх для формування проміжних значень одностороннії функції.

$$y1 = g^{x1} mod\ p$$

$$y2 = g^{x2} mod\ p$$

$y1$ та $y2$ агенти надсилають відкритим каналом один одному і повторюють ту ж процедуру з тим, що отримали від колеги

$$y1^{x2} = g^{x1x2} mod\ p = y3,$$

$$y2^{x1} = g^{x2x1} mod\ p = y3.$$

Як і в розглянутому вище узагальненому алгоритмі обидва агенти відкритим каналом отримали інформацію, яка дозволила їм сформувати спільний таємний код $y3$. Зворотня функція ніби то не існує, але всі сили криптоаналітиків сьогодні кинуті на вирішення цієї проблеми, яка, щонайменше, добре формалізована. І це дає криптоаналітикам надію, що збільшення обчислювальних потужностей дозволить знайти зворотній дискретний

логарифм. Тому один з шляхів створення незламних односторонніх функцій є використання замість арифметичних (алгебраїчних) функцій алгоритмічних, які мають математичну формалізацію окремих елементів, але практично не мають математичної формалізації для всієї функції в цілому, наприклад алгоритмічні функції на базі клітинних автоматів.

1.3. Загальні поняття теорії клітинних автоматів

Усі клітинні автомати, що будуть розглядатися далі керуються наступними правилами:

1) Для роботи клітинного автомату задається початкове значення клітин клітинного поля та правил переходу клітин з одного стану в інший.

2) Клітинне поле представляє прямокутне поле клітин при цьому є замкненим та геометрично приймає форму Тору. Кількість клітин задається користувачем через кількості рядків та стовпчиків клітин.

3) Клітина може приймати одне з двох значень: 0 (мертва) або 1 (жива).

4) Всі клітини поля мають попередньо визначену *околицю* – множину клітин «сусідів», від значень яких залежить стан досліджуваної клітини.

5) На кожній ітерації клітинного автомату за допомогою правил переходу та станів клітин поля визначаються новий стан кожної з клітин.

У випадку класичного автомату «Гри Життя» Джона Конвея правила формуються більш конкретно:

1) Гра відбувається на нескінченному двовимірному клітинному полі, кожна клітина якого може приймати один з двох станів: жива, мертва.

2) Кожна клітина в своїй околиці має вісім сусідніх клітин.

3) Якщо клітина жива і має 2 або 3 живі сусідні клітини, то на наступній ітерація вона залишається живою. Інакше вона вмирає.

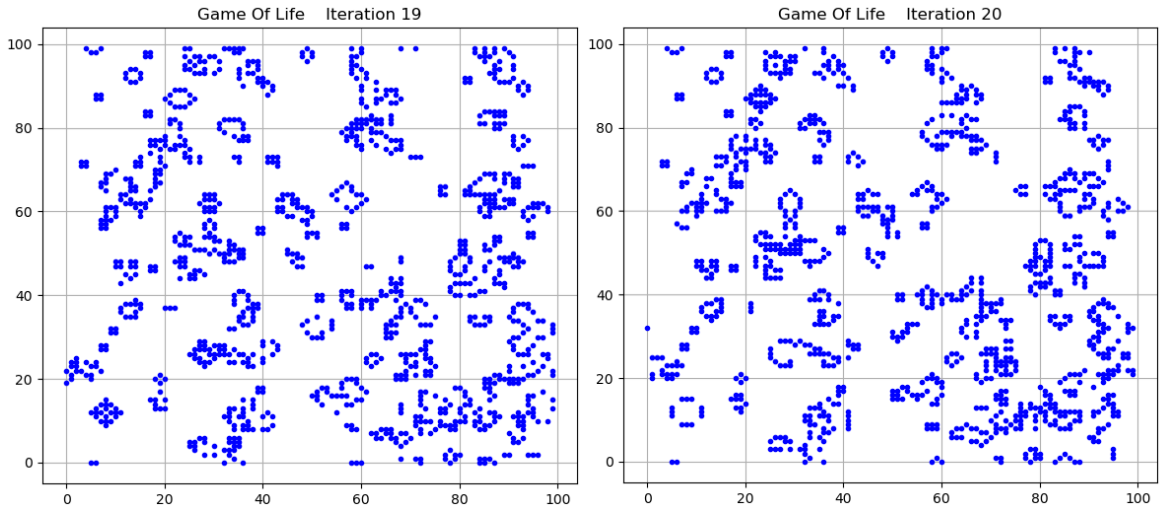
4) Якщо пуста(мертва) клітина має 3 живих сусідів, то на наступній ітерації вона стає живою. Інакше вона залишається мертвою.

Автомат завершує свою роботу, якщо:

- На полі не залишається жодної «живої» клітини.

- Клітинне поле на черговій ітерації співпадає з будь-яким іншим полем попередніх ітераціях.
- При черговому кроці жодна з клітин не змінює свого стану.

В подальшому для того, щоб проілюструвати роботу клітинних автоматів будуть показуватися дві послідовні ітерації автомату (Рис. 1). Недоліком наведеного клітинного автомату «Гра Життя» щодо створення односторонньої функції є те, що вже на початку клітини збираються в статичні та циклічні структури.



а) б)

Рис. 1. Ітерації класичної «Гри Життя». а) ітерація 19, б) ітерація 20.

1.4. Постановка задачі

Загальні умови

1. Потрібно створити метод, який дозволяє генерувати ключі шифрування, як завгодно часто, за допомогою лише відкритого каналу обміну інформацією.

2. На початку роботи алгоритму відбувається хоча б один таємний обмін інформацією щодо виду, властивостей, параметрів односторонньої функції та алгоритму взаємодії агентів при визначенні нового ключа шифрування.

3. Кількість агентів може бути довільною з урахуванням потрібної загальної кількості адрес взаємного обміну таємними повідомленнями.

4. У якості базового доцільного взяти алгоритм Діффі-Хелмана та модифікувати його.

Визначення процедури роботи односторонньої функції на основі клітинних автоматів.

По-перше, декомпозуємо задачу на 2 складові:

1. Способи перетворення результату роботи клітинного автомату на те, що може бути сприйняте, як результат роботи односторонньої функції (Хешування).

2. Модифікація властивостей клітинних автоматів, для урізноманітнення варіантів поведінки функції.

Властивості функції:

- Односторонність. Тобто можливість отримання значення функції на основі інформації про аргумент $y = f(x)$

і одночасна неможливість отримання значення аргументу на основі значення функції, точніше кажучи відсутність зворотної функції, яка би забезпечила знаходження аргументу для заданого значення функції $x = f^{-1}(y)$.

- Комутативність. Якщо позначити дію функції знаком «×», то властивість комутативності групових операцій ознає $x1 \times x2 = x2 \times x1$.

На прикладі функції це може бути

$$f(x1 \times x2) = f(x2 \times x1),$$

$$f(x1 + x2) = f(x2 + x1).$$

$$f(x1, x2) = f(x2, x1)$$

Урізноманітнення параметрів та варіантів поведінки односторонньої функції на основі клітинних автоматів

має відбутись на основі розширення набору правил класичної «Гри Життя».

2. Вдосконалення алгоритму Діффі-Хелмана

2.1. Модифікація процедури для n агентів

Нехай потрібно згенерувати єдиний пароль для n агентів. Алгоритм у такому випадку виглядатиме аналогічно, за винятком деяких особливостей:

1. В перший сеанс зв'язку таємним каналом агенти так само, як у випадку двох агентів, домовляються про параметри функції перетворення $f(x)$ та супутні дані. Усе інше спілкування відбувається через відкриті канали зв'язку.

2. Коли виникає потреба зміни ключ шифрування, всі агенти i формують таємні слова $xi, \quad i = \overline{1,n}$ та розраховують проміжні значення функції першого рівня

$$yi^{(1)} = f(y0, xi), \;\; i = \overline{1,n}.$$

Значення $yi^{(1)}$ кожен агент i відкритим каналом циклічно надсилає агенту $i + 1$. Циклічність значить, що агент n надсилає проміжні значення 1 рівня агенту 1. Якщо сказати узагальнено, то номер наступного агента дорівнює залишку від ділення $i + 1$ за модулем n.

$$r = (i + 1) \;\; mod \;\; n.$$

3. Тепер кожен агент r на кожному новому кроці k опрацьовує інформацію від попередніх агентів $yi^{(k-1)}$. Для цього він кожного разу використовує своє таємне слово xr. Тільки проміжне значення функції $yi^{(k-1)}$ кожного разу змінюється. Нове проміжне значення дорівнює

$$yr^{(k)} = f\left(yi^{(k-1)}, xr\right), \;\; i = \overline{1,n}.$$

4. На кроці n кожен агент отримає проміжне значення функції, яке він за допомогою свого таємного слова перетворить на новий спільний ключ шифрування.

$$yr^{(n)} = f\left(yi^{(n-1)}, xr\right), \;\; i = \overline{1,n}.$$

Під час генерації ключів шифрування агенти співпрацюють лише з двома агентами: з відправником проміжного значення функції та агентом-приймачем. Інформація про інших агентів недоступна.

<u>Ймовірні зловмисники</u>, як ї в двоагентному випадку, прослуховують відкритий канал, знають проміжні значення функції від всіх агентів, але при

цьому не знають ані нового ключа шифрування, ані таємних слів від окремих агентів.

Заплутування інформації щодо кількості агентів

Для збереження в таємниці інформації щодо кількості агентів, в роботі пропонується, щоб один з агентів «грав» замість декількох агентів. Це створить враження, що агентів більше, ніж їх є насправді. Але у той же час фальшивих агентів не має бути занадто багато, оскільки в такому разі зловмисник отримує більше інформації про проміжні значення функції, за допомогою чого може легше підібрати параметри клітинного автомату.

2.2. Модифікація процесу формування початкових значень функції

В якості спільного **початкового значення функції** у модифікованому алгоритмі Діффі-Хелмана виступає **клітинне поле автомату**, що може бути розіслано в закодованому вигляді агентам під час першого таємного зв'язку або може надсилатись стеганографічним методом. Наприклад, на певному сайті в Інтернеті знаходиться певний набір звичайних піксельних картинок або фотографій. При таємній комунікації визначається, що в якості первинного поля обирається певна частина певної картинки, яка знаходиться на якомусь порядковому номері в галереї. В той же час в якості початкових даних для формування клітинного поля може також виступати текст.

У нашому випадку обиратися буде графічна картинка. Після цього біти кольорів та напівтонів можуть перетворюватись на певні бінарні або чисельні характеристики окремих клітин поля.

В окремих випадках, навіть **можна не приховувати**, яке саме графічне зображення використовується, оскільки обернену функцію за результатами проміжних функцій в цьому випадку визначити також буде практично неможливо.

2.3. Модифікація процесу обміну проміжними значеннями функції

У багатьох хеш-функціях для забезпечення додаткової надійності при формування хешу в функцію додається так звана «криптографічна сіль» [17, 18]. Це ускладнює процес підбору паролю, оскільки приходиться шукати ще й

значення «солі». У випадку клітинних автоматів необхідними даними, на основі яких зловмисник може підібрати ключ шифрування або параметри автомату, є проміжні клітинні поля. Для захисту цих даних автором **запропоновано використовувати аналог «криптографічної солі» - додаткове клітинне поле ідентичного використаного автоматом розміру**. Перед відправкою проміжного значення на поле автомату накладається поле «солі» таким чином, що стан клітини можна розрахувати за формулою:

$$f = c \wedge \overline{s}$$

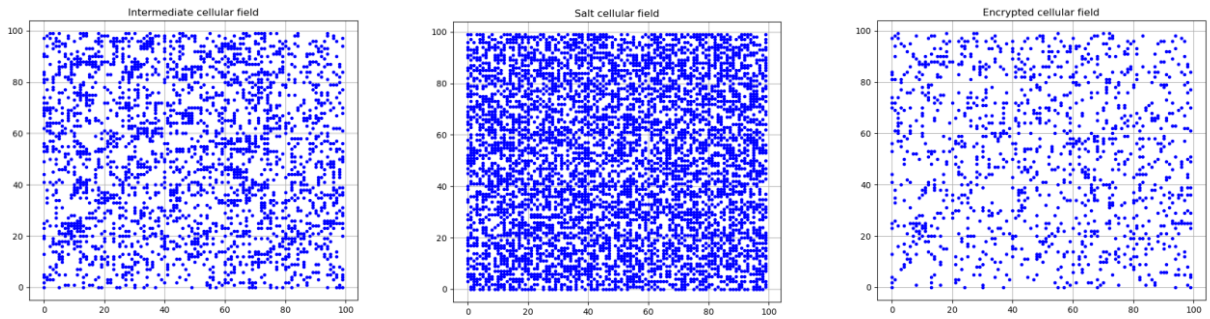
де c – значення стану клітини, s – відповідне значення стану клітини на полі «солі», f – зашифроване значення поля для відправки.

Для того, щоб дешифрувати проміжне клітинне поле агенту, що прийняв закодоване поле необхідно знати поле «солі» (воно формується підчас першої таємної зустрічі). Агент пропускає значення зашифрованого клітинного поля через функцію

$$c = f \wedge \overline{s} \vee s \wedge \overline{f}$$

де c – значення стану шуканої клітини, s – відповідне значення стану клітини на полі «солі», f – зашифроване значення поля від відправника.

Приклад зашифрованого поля (Рис. 2 в), поля «солі» (Рис. 2 б) та проміжного клітинного поля (Рис. 2 а) зображено на рис. 2.



а) б) в)

Рис. 2. Апробація шифрування а) проміжне клітинне поле, б) поле «солі», в) зашифроване поле.

3. Збільшення різноманітності односторонніх функції за рахунок вдосконалення правил клітинного автомату одної популяції

3.1. Вдосконалення базового правила зміни стану клітини

За основу було обрано двовимірний клітинний автомат Джона Конвея «Гра Життя», в якому правило зміни стану клітини задано статичними значеннями, що не можна змінювати. Вдосконалимо правило у напрямку розширення керування керуючими параметрами автомату. Обрахунок околиці виконується за правилом Мура [19], тобто околицею клітини вважаються 8 клітин навколо неї. Вважаймо, що попередньо необхідно задати дві множини значень: множину значень переходу в стан живої клітини D та множину значень збереження стану живої A. Пуста клітина змінює стан, якщо виразу приймає значення одиниці:

$$s_{t+1} = \bar{s}_t \wedge (n \in D) \vee s_t \wedge (n \in A) \quad (1)$$

де s_{t+1} – стан клітини на ітерації $t + 1$,

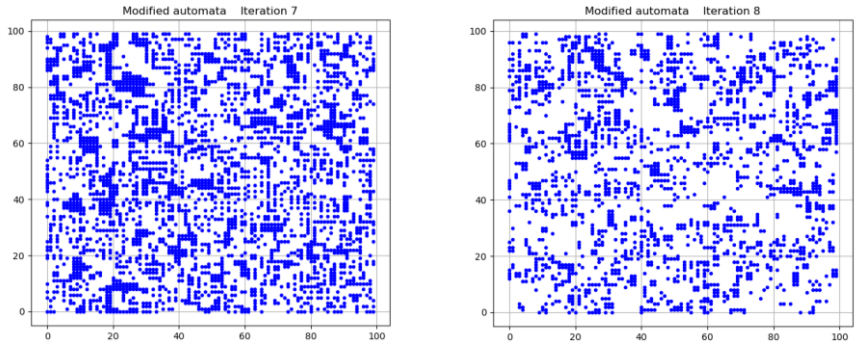
s_t – стан клітини на ітерації t,

n – кількість живих клітин в околиці клітини,

D – множина значень переходу в стан живої клітини,

A – множина значень збереження стану живої.

Таким чином було отримано базове правило зміни стану клітин, за допомогою якого можна отримати $2^{8(R+1)R}$ комбінацій правил поведінки клітин, де R – радіус взаємодії клітин. Працездатність моделі було перевірено на значеннях множин $D = \{0\}$ та $A = \{5, 7\}$ (Рис. 3).



а) б)

Рис. 3. Автомат з модифікованими базовим правилом зміни стану клітини. а) ітерація 7, б) ітерація 8.

При цьому у випадках, коли множина допустимих значень D та A складається зі значень, що можливо задати у вигляді одного проміжку значень, то загальне правило зводиться до окремого випадку, коли стан клітини на ітерації $t+1$, що залежить від кількості оточуючих клітин та власного стану на ітерації t, можна розрахувати за допомогою булевої функції дев'яти змінних, де перший параметр відповідає стану клітини на кроці t, а інші вісім відповідають стану сусідніх клітин. Клітина стає живою, якщо значення виразу дорівнює 1:

$$s_{t+1} = \bar{s}_t \wedge (d_l \leq n) \wedge (n \leq d_r) \vee s_t \wedge (a_l \leq n) \wedge (n \leq a_r), \quad (2)$$

де n – кількість живих клітин в околиці,

s_t – стан клітини на кроці t,

d_l – мінімальна кількість клітин в околиці для переходу в стан живої,

d_r – максимальна кількість клітин в околиці для переходу в стан живої,

a_l – мінімальна кількість клітин в околиці для зберігання стану живої,

a_r – максимальна кількість клітин в околиці для зберігання стану живої.

Якщо мінімальна кількість клітин в околиці для переходу в стан живої дорівнює мінімальна кількість клітин в околиці для зберігання стану живої та максимальна кількість клітин в околиці для переходу в стан живої дорівнює максимальна кількість клітин в околиці для зберігання стану живої ($d_l = a_l$ і $d_r = a_r$), формулу (2) можна спростити до наступної:

$$s_{t+1} = (d_l \leq n) \wedge (n \leq d_r) \quad (3)$$

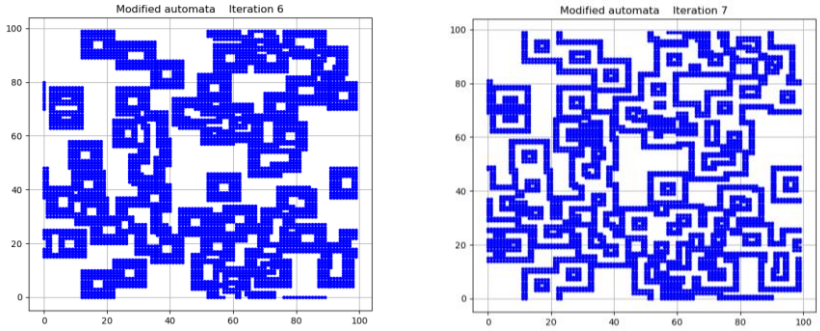
Під час написання програмного коду для введення правила народжуваності та продовження життя в алгоритм обрахунку станів поля клітинного автомату несподівано було знайдено додаткові переваги імітаційного (комп'ютерного) моделювання. Так у нотаціях MatLab та Python вираз (2) можна представити лаконічніше, ніж запропоновану булеву функцію:

$$s_{t+1} = !s_t * (d_l \leq n \leq d_r) + s_t * (a_l \leq n \leq a_r) \quad (4)$$

Тепер класичні правила для «Гри Життя» можна представити наступним чином:

$$s_{t+1} = !s_t * (3 \leq n \leq 3) + s_t * (2 \leq n \leq 3) \quad (5)$$

Працездатність залежності (2) була перевірена на прикладі клітинного автомату з параметрами $d_l = 1$, $d_r = 7$, $a_l = 1$, $a_r = 8$ (Рис. 4):



а) б)

Рис. 4. Ітерації модифікованого клітинного автомату. а) ітерація 6, б) ітерація 7.

Початкове розташування живих точок на полі обирається випадково за допомогою генератора псевдовипадкових чисел, або, як зазначено вище, початковим полем може бути і шматок початково вибраної картинки.

3.2. Введення правила замкненого простору

В класичному клітинному автоматі «Гри Життя» клітинне поле є нескінченним. У випадку обмежених розрахункових ресурсів клітинне поле має бути або чимось обмежене, або замкненим. В нашій ситуації було обрано другий варіант, оскільки тоді немає потреби вводити особливі правила для клітин на краях поля. Замикання відбувається шляхом з'єднання протилежних сторін та кутів. Така ситуація була б можлива, якби життєвий простір розміщався на торі. На програмному рівні замикання реалізується створенням навколо масиву, що складає основне поле автомату, додаткових частин поля, що відповідають протилежним краям основного поля. Це робить можливим керувати станом усіх клітин за допомогою єдиного правила.

3.3. Базове правило визначення околиці клітини

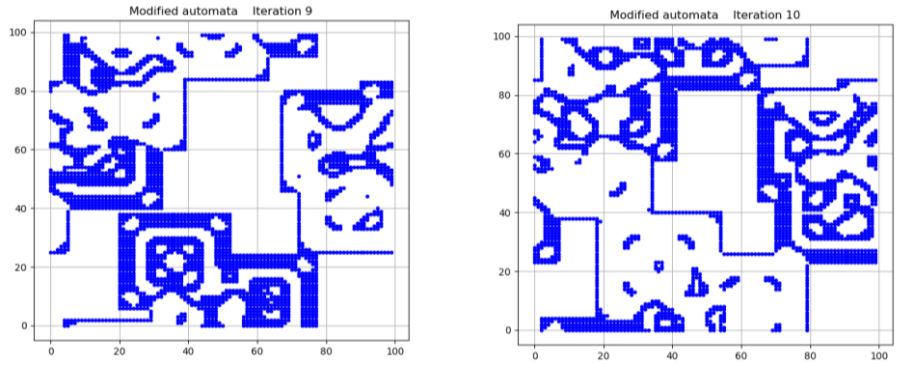
В класичному клітинному автоматі Джона Конвея «Гра Життя» стан кожної клітини залежить від станів оточуючих клітин в радіусі одної клітини ($R = 1$). Це накладає обмеження на кількість комбінацій можливих значень параметрів формули (1) для базового правила зміни станів клітин таким чином, що зловмисник в теорії може просто підібрати можливі вхідні правила з

обмеженого набору (просто проаналізувавши проміжкові клітинні поля, якими обмінюються агенти по відкритих лініях зв'язку) та наблизитися до отримання ключа шифрування. Це не є допустимим, тому було запропоновано ввести базове правило визначення околиці клітини, що дозволяє агентам встановлювати необхідну околицю клітини (радіус взаємодії). Наприклад, радіус $R = 2$ (дозволяє мати 24 сусідніх клітини), або $R = 3$ (максимум 48 сусідніх клітин). Для знаходження кількості сусідніх клітин при довільному радіусі виведемо формулу:

$$n = (2R + 1)^2 - 1,$$

де R – радіус внутрішньопопуляційної взаємодії.

Працездатність моделі зі зміненим радіусом взаємодії була перевірена на мові програмування MatLab на прикладі попереднього клітинного автомату з параметрами $d_l = 1$, $d_r = 7$, $a_l = 1$, $a_r = 8$ та з радіусом взаємодії $R = 2$ (Рис. 5). Впроваджене правило вирішує проблему протидії прямому перебору значень параметрів клітинного автомата, що робить односторонню функцію надійнішою.



а) б)

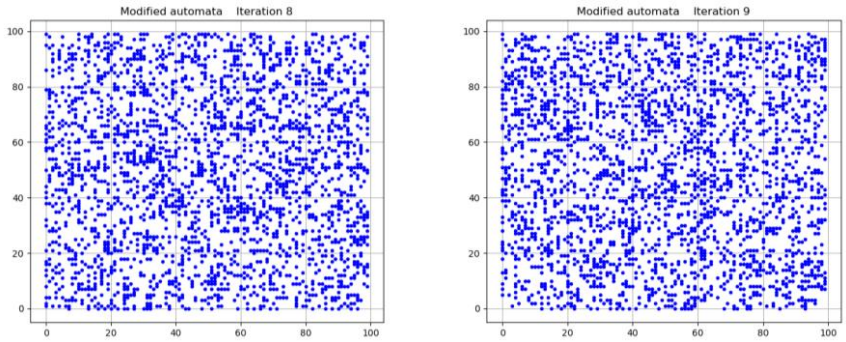
Рис. 5. Клітинний автомат з радіусом $R = 2$. а) ітерація 9, б) ітерація 10.

3.4. Ускладнене правило визначення околиці клітини

Базове правило визначення околиці клітини може бути ускладнене шляхом введення «мапи місцевості», що складає з себе набір значень радіусів взаємодії для кожної окремої клітини поля автомату. Значення для кожної клітини можуть бути задані випадково або користувачем. Це дозволить забезпечити додатковий рівень безпеки односторонньої функції на випадок, якщо зловмисник отримає інформації про початкове клітинне поле та значення параметрів базового

правила зміни стану клітини, оскільки за відсутності значень «мапи місцевості» клітинний автомат не зможе працювати. При цьому перебір усіх можливих «мап» не є доцільним, оскільки значення радіусу взаємодії може приймати значення будь-якого натурального числа в межах розміру клітинного поля.

Приклад роботи клітинного автомату з ускладненим правилом визначення околиці клітини зображено на Рис. 6.



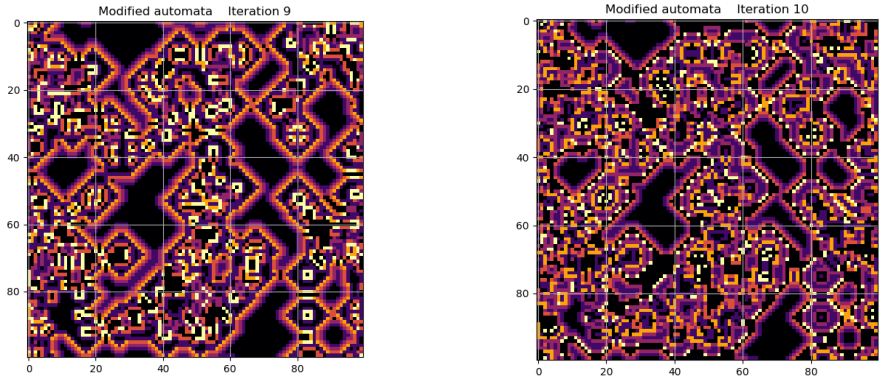
а) б)

Рис. 6. Клітинний автомат з модифікованим правилом визначення околиці клітини. а) ітерація 8, б) ітерація 9.

3.5. Введення правила смерті від віку

У досліджуваних автоматах, що використовувалися іншими авторами для моделювання процесів реального світу, утворення статичних структур клітин на полі можливо, що у нашому випадку може призвести до незадовільної роботи алгоритму, а саме утворення колізій клітинних полів(повного збігу клітинних полів). Зокрема дуже часто статичні або циклічні структури клітин утворюються в класичному автоматі «Гра Життя», на якому ми базувалися при розробці власних клітинних автоматів. Для виправлення цього недоліку було введено правила смерті від віку, що звучить наступним чином: з кожною ітерацією вік живої клітини зростає на одиницю; якщо вік клітини перевищує встановлене максимальне значення віку max_age, вона вмирає.

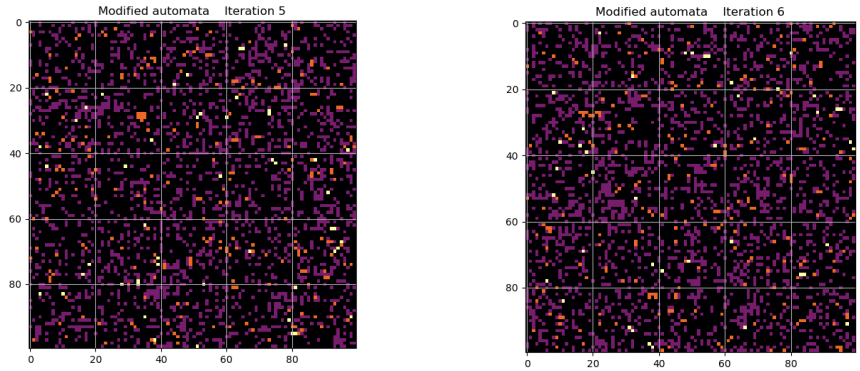
Наочно дію правила смерті від віку на клітинному полі можна побачити за допомогою розробленої моделі (Рис. 7), на якій колір клітини позначає вік клітини: чим світліші клітини, тим більше їх вік. Для контрастності клітинного поля фоновим кольором було зроблено чорний.



а) б)

Рис. 7. Клітинний автомат з правилом смерті від віку (реалізоване на Python). а) крок 9, б) крок 10.

Дане та модифіковане правило визначення околиці здатні співіснувати в одному клітинному автоматі, що робить односторонню функцію ще більш непередбачуваною. Приклад реалізації зображено на Рис. 8.



а) б)

Рис. 8. Клітинний автомат з правилом смерті від віку (реалізовано на Python). а) ітерація 5, б) ітерація 6.

Для клітинного автомату було встановлено максимально можливий вік $max_{age} = 3$.

Об’єднання правил зміни стану клітини та смерті від віку

В умовах постійної генерації ключів шифрування необхідно, щоб ключі генерувалися швидко. Для цього необхідно мати оптимізований алгоритм. В зв’язку з цим було вирішено об’єднати правила зміни стану клітини та смерті від віку в одне. Об’єднане правило може мати вигляд:

$$s_{t+1} = \left(\bar{s}_t \wedge (n \in D) \vee s_t \wedge (n \in A)\right) \wedge \left(m \leq max_{age}\right)$$

В перспективі таке об’єднання може зробити можливою реалізацію клітинного автомату на базі перепрограмуємих логічних матриць FPGA [20].

3.6. Вдосконалення правил для реалізації багатопопуляційної моделі

Найвідоміша багатопопуляційна модель – «Хижак-жертва». Такі моделі зазвичай використовуються для моделювання процесів реального світу [21]. В контексті наших досліджень сенс має те, що данні моделі складаються з клітинних автоматів зі складною поведінкою. Тому було вирішено впровадити правила поведінки автомату з декількома популяціями.

Спочатку введемо у багатопопуляційну модель можливі ролі популяцій: перша – «жертв», друга – «хижаків». Кожна з популяцій має свої окремі незалежні від іншої параметри виживання: визначення околиці R та зміни стану клітини A та D. Також автомат має мати правила взаємодії популяцій:

Правило 1. «Хижаки» їдять «жертв».

У клітині поля в якій одночасно присутні клітини популяції «хижаків» та «жертв» на наступній ітерації залишається лише клітина «хижаків. Реалізація правила зображено на рисунках Додатку А.

Правило 2. «Хижак має голод».

Введемо додатковий параметр для популяції хижаків – $food_{min}$, що позначає мінімальну необхідну кількість «жертв» в радіусі взаємодії «хижака» необхідну для продовження існування. Тобто будемо вважати, що якщо в межах околиці «хижака» недостатня кількість «жертв», він помирає від «Голоду». Дане правило введено для того, щоб урівняти шанси популяцій на виживання і таким чином збільшити непередбачуваність моделі.

Аналогічне правило можна ввести і для популяції «жертв», але це не є доцільним, оскільки тоді з’являється шанс повного вимирання популяції, що зробить односторонню функції недієздатною. Реалізація автомату з правилами поїдання «жертв» «хижаками» та голодом «хижаків» зображено на рисунках Додатку Б. В моделі з введеним правилом поїданням кроликів поле, на якому реалізується модель має білий колір, червоним позначаються клітини популяції «хижаків», а синім – клітини «жертв».

4. Програмна реалізація модифікованого алгоритму Діффі-Хелмана

4.1. Структура програми генерації ключів шифрування

Програма «CryptoAutomata» розроблена на мові програмування Python – мові програмування загального призначення. Модель (Рис. 9) складається з головної програми, що відповідає за управління підпрограмами та обробку результатів роботи клітинного автомату, і підпрограм спеціального призначення (Додаток В):

- Підпрограма підрахунку сусідів клітин nbrModified.
- Підпрограма обрахунку стану клітин на наступній ітерації nextStepModified.
- Підпрограма реалізації правила смерті від віку fatalityAge.
- Підпрограма визначення координат живих клітин на клітинному полі на кожній ітерації transArray.

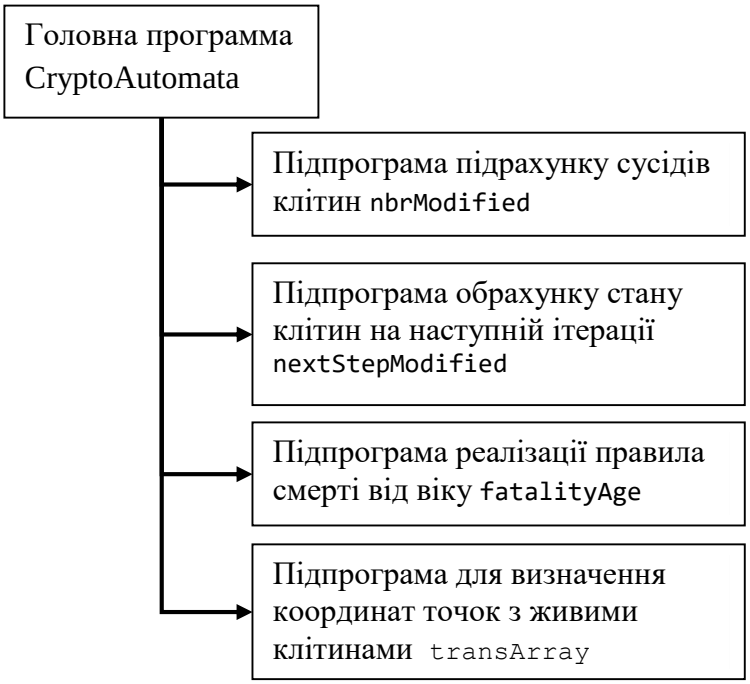


Рис.9. Структура програми «CryptoAutomata».

4.2. Апробація модифікованого алгоритму Діффі-Хелмана

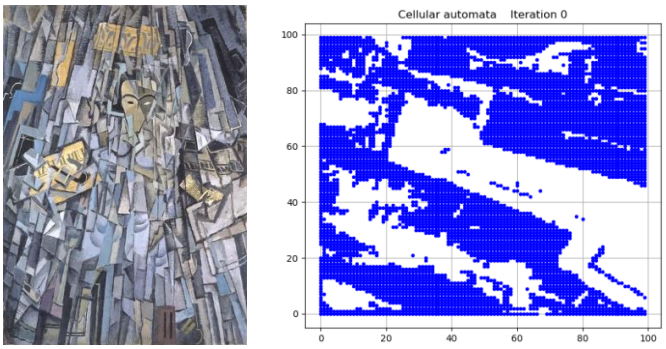
В якості односторонньої функції був використаний модифікований клітинний автомат з полем 100x100 з такими розширеними правилами:

- Базове правило зміни стану клітини.
- Правило замкненого простору.

- Ускладнене правило визначення околиці клітини
- Правило смерті від віку

Апробація роботи з великими клітинними полями.

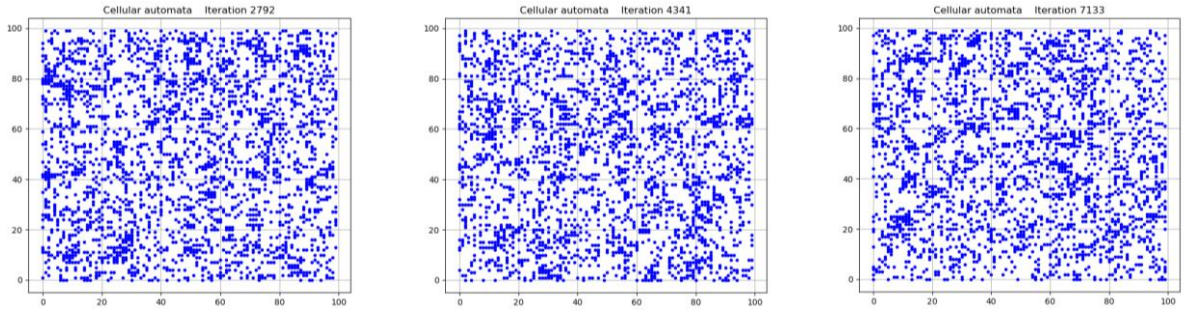
На основі випадкового графічного зображення (рис. 10. а) агенти формують однакове спільне первинне клітинне поле модифікованого автомату (рис. 10. б).



а) б)

Рис. 10. Початкові дані автомату. а) зображення для генерації первинного поля автомату, б) утворене первинне поле розміром 100x100.

Після цього кожен з агентів придумав своє таємне слово (2792 і 4341), яке відповідає кількості ітерацій автомату, знайшов свою проміжне поле та відправив її другому агенту (рис.11 а, б). Кодові слова відповідали кількості кроків, які мала би пройти програма від отриманої картинки. В першого агенту це 2792+4341=7133 кроків. В другого 4341+2792=7133 кроків. Отже обидва агенти різними шляхами отримали єдину кодову картинку №7133 (рис. 11 в).



а) б) в)

Рис. 11. Поле 100x100 в процесі застосування таємних слів. а) проміжне поле агенту 1; б) проміжне поле агенту 2; в) кінцеве поле обох агентів.

Далі отримане кінцеве клітинне поле може використовуватись в залежності від встановлених вимог до захисту безпосередньо у двійковому

вигляді (Додаток Г), безпосередньо у шестнадцятирічному (Додаток Д) або у вигляді хеша (рис.12).

3226332823302120272716282118292323192823292220282223193025232618272
4153014232225282522172221262628263124242824272121262223202425213020
3322272922293822351934352821333926302726131724201730242524303525 32

Рис. 12. Спільний ключ шифрування на основі клітинного поля у вигляді хеша.

В роботі було виконано також тестування обчислювальних витрат машинного часу. В якості мов програмування для комерційної реалізації було обрано Python та C#. Для тесту була використана звичайний побутовий комп'ютер Dell Inspiron з процесором Intel Core 3 Gen.8, тактовою частотою 2,1 – 2,3 ГГц, RAM 8 Gb, SSD 128 Gb.

В результаті тестування програми обрахунку ітерацій автомату на Python 32000 кроків автомату для поля 100x100 було обраховано протягом 15 хвилин 27 секунд. При цьому на C# з полем того ж самого розміру та з тими ж правилами автомату було обраховано за 54 секунди. Ця різниця у декілька разів пов'язано з особливостями мов програмування та підходами при розробці програм: програма на Python була написана за правилами імперативного програмування, а на C# за правилами об'єктно-орієнтованого програмування.

Без введення в процес генерації ключів шифрування поняття «солі» необхідною також була перевірка клітинних полів автомату на повторюваність, оскільки присутність колізій полів могла полегшити знаходження кінцевого клітинного поля зловмисником. На разі цієї проблеми немає, оскільки для функціонування автомату достатньо відсутності колізій на перших 100000 ітераціях автомату. При цьому доречним є питання достатніх розмірів клітинного поля автомату, оскільки чим менше поле, тим менше витрачається розрахункових ресурсів комп'ютерів.

Апробація роботи з малими клітинними полями.

В якості односторонньої функції був використаний клітинний автомат з полем 16x16 з такими ж розширеними правилами, як і в попередній апробації.

На основі випадкового графічного зображення (рис. 10) агенти сформували єдине первинне клітинне поле автомату (рис. 13).

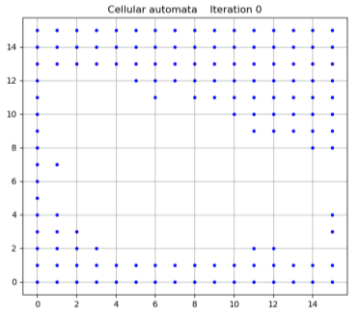
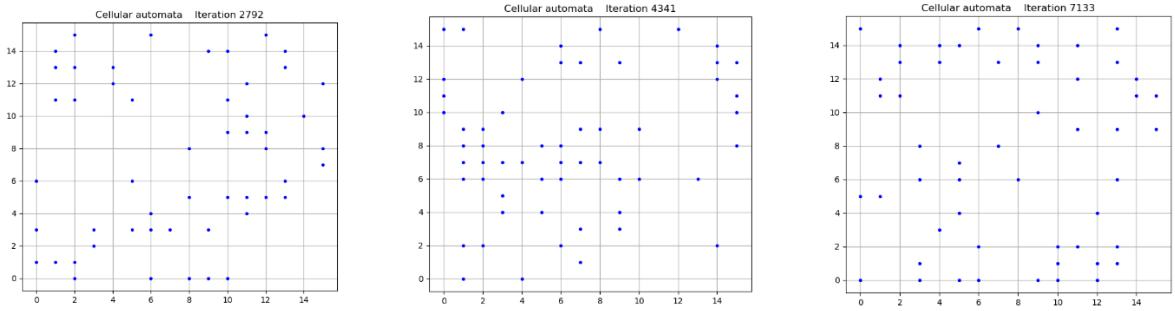


Рис.13. Створене первинне поле автомату розміром 16x16.

Після чого процедура генерації ключів була виконана аналогічно попередньому прикладу. В результаті використання тих самих ключів агенти отримали проміжні поля (Рис. 14 а, б) та спільне кінцеве поле (Рис. 14 в).

Апробація алгоритму показала, що використання клітинних автоматів з модифікованими правилами дозволяє створювати односторонні функції для алгоритму Діффі-Хелмана за умови коректного обрання правил та розміру поля автомату.



а) б) в)

Рис. 14. Клітинне поле 16x16 в процесі застосування таємних слів.

а) проміжне поле агенту 1; б) проміжне поле агенту 2; в) кінцеве поле обох агентів.

4.3. Перевірка працездатності односторонньої функції на основі клітинного автомата

Однією з проблем вибору клітинного автомату для використання в якості односторонньої функції була невизначеність надійності конкретного клітинного автомата [22, 23]. Це пов’язано з тим, що випадкове задання параметрів клітинного автомата не забезпечує його однозначну стійкість виживання: в деяких популяція поля закономірно вимирала, в деяких поле переповнювалося

через занадто високу виживаність, в деяких життєвий шлях клітин зациклювався. Данні випадки недопустимі, оскільки імовірний зловмисник має мати як можна менше інформації про досліджуваний клітинний автомат. Для досягнення цього необхідно, щоб одностороння функція була складнопередбачуваною, тобто жодна ітерація клітинного автомату не повторювала стани клітинного поля, що вже були на попередніх ітераціях.

Кожне клітинне поле теоретично може знаходитися в $N = 2^n$ станах, де n – кількість клітин на полі. А оскільки порівнювати 2^{n*m} ітерацій автомату (у нашому випадку 2^{10000}), де n – кількість рядків клітинного поля, m – кількість стовпців, не є ефективним було вирішено використовувати статистичні методи для приблизного прогнозування надійності клітинного автомату.

Для цього було побудовано залежність чисельності популяції від часу(Рис. 15).

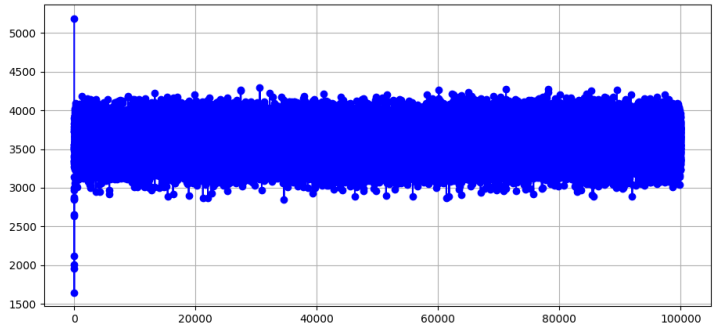


Рис. 15. Залежність чисельності популяції від часу.

На графіку залежності чисельності популяції від часу можна побачити, що на перших ітераціях клітинного автомату відбувається етап стабілізації, під час якого значення поступово доходить до значень, що знаходяться в проміжку між 2850 та 3350. Після цього наступає етап стабільності, в якому значення хаотично коливаються в межах від 2850 до 3350 клітин на полі. Таким чином проміжок можливих значень обмежується проміжком в 500 значень. Чим більше буде цей проміжок, тим кращим в теорії буде клітинний автомат, і відповідно одностороння функція.

Але величина проміжку, в якому мають знаходитися значення чисельності популяції не єдиний показник, що необхідний при оцінці клітинного автомату, також важливим є те наскільки часто зустрічаються конкретні значення. Для

отримання цієї інформації було побудовано графік розподілу чисельності популяції (Рис. 16). На цьому графіку можна побачити, що значення, що зустрічаються частіше всього знаходяться приблизно в середині зазначеного раніше проміжку. При цьому клітинний автомат може вважатися кращим за інший, якщо при тому ж проміжку можливих значень, вони розподіляються більш рівномірно.

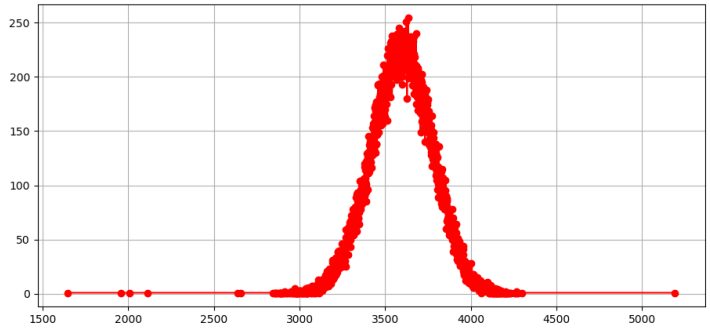


Рис. 16. Графік частотного розподілу чисельності популяції.

Також було побудовано графік імовірнісного розподілу чисельності популяції (Рис. 17). З цього графіку можна також робити додаткові висновки, але вже с точки зору імовірності зустріти конкретне значення під час роботи клітинного автомату.

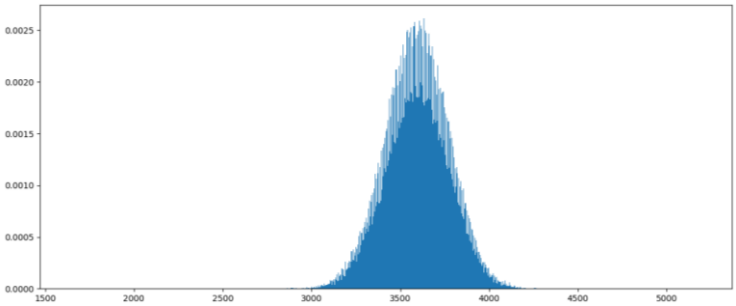


Рис. 17. Графік імовірнісного розподілу чисельності популяції.

Для аналізу клітинного автомату також доречним є пошук закономірностей у його поведінці, зокрема пошук залежності чисельності популяції від номеру ітерації автомату. В ідеальному випадку необхідно, щоб автомат зміг видати максимально близьку до 2^{10000} кількість неповторних клітинних полів, але при цьому вибір має бути непередбачуваним для зловмисника. Для перевірки залежності чисельності популяції від номеру ітерації було вирішено використовувати коефіцієнт кореляції Пірсона, що обраховується за формулою:

$$r_{xy} = \frac{\sum_{i=1}^{m}(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum_{i=1}^{m}(x_i-\bar{x})^2 \sum_{i=1}^{m}(y_i-\bar{y})^2}}, \text{ де}$$

x_i – і-й елемент вибірки x^m,

y_i – і-й елемент вибірки y^m,

$\bar{x}$ – середнє арифметичне вибірки x^m,

$\bar{y}$ – середнє арифметичне вибірки y^m.

Таким чином було отримано, що коефіцієнт кореляції Пірсона для цілої вибірки значень популяції клітинних полів: $r = 0.00032$. Це відповідає дуже слабкій кореляції між номером ітерації клітинного ітерації та чисельності його популяції. Також було проведено перевірку відсутності кореляції для проміжків різної довжини, а саме окремих проміжків довжиною в 10, 100, 1000(Рис. 19) та 10000(Рис. 18) ітерацій.

У випадку, коли кореляція обраховувалася для проміжків в 10000(Рис. 16) ітерацій було визначено, що кореляція є досить слабкою, на деяких проміжках (наприклад для проміжку між 10000 та 20000 ітераціями) навіть менша, ніж для всієї вибірки. Але також є і відхилення, що перевищують значення кореляції всієї вибірки в два рази. Це говорить, про те, що локальна кореляції може бути більшою, ніж для цілою вибірки.

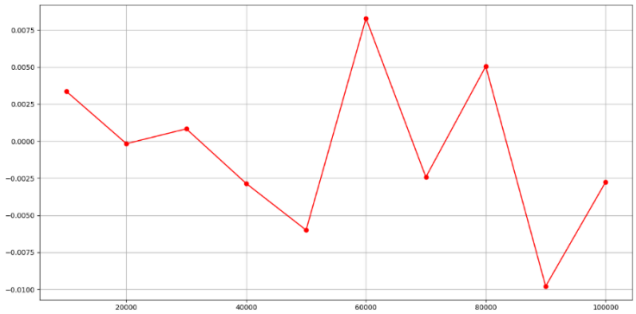


Рис. 18. Коефіцієнт кореляції Пірсона для 10 проміжків (1 проміжок – 10000 ітерацій).

У випадку, коли кореляція обраховувалася для проміжків в 1000(Рис. 19) ітерацій, кореляція на локальних проміжках хоча і збільшилася на два порядки, але все одно залишається змінюється від проміжку до проміжку досить сильно. Потрібно також зазначити, що коефіцієнт кореляції Пірсона для проміжку від 1 до 100 ітерації складає приблизно $r = 0.114$, що є великим відхиленням відносно

минулих значень. Це пов'язано лише з тим, що на початку роботи клітинного автомату відбувається етап стабілізація, під час якого чисельність популяції стабільно збільшується до моменту стабілізації. Тому зважати на це при оцінці роботи клітинного автомату звертати не треба.

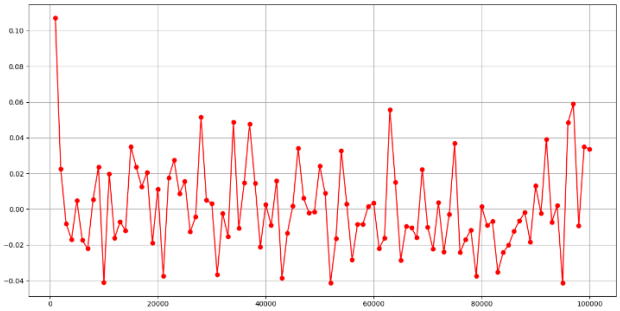


Рис. 19. Коефіцієнт кореляції Пірсона для 100 проміжків (1 проміжок – 1000 ітерацій).

Аналіз значень кореляції для проміжків в 100 та 10 ітерацій можна подивитися в Додатку З.

Для винесення висновку щодо якості клітинного автомату було побудовано зі значеннями мінімальних, середніх та максимальних значень коефіцієнту кореляції Пірсона для проміжків в 10, 100, 1000 та 10000 ітерацій (Рис. 20). Червоним позначено максимальні значення кореляції Пірсона для проміжків відповідної довжини, синім – середні значення, а зеленим мінімальні. Таким чином ми можемо зробити висновок, що хоча локально чисельність популяції клітинного поля може корелюватися з номером ітерації автомату, але в глобальному плані ці змінні не корелюються. На підставі цього можна вважати, що досліджуваний клітинний автомат є працездатним і його можна використовувати в якості односторонньої функції модифікованого алгоритму Діффі-Хелмана.

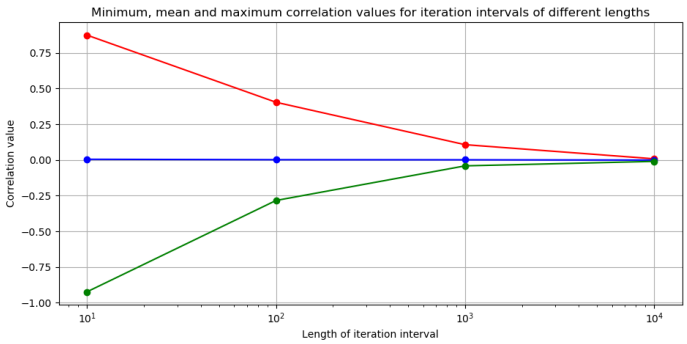


Рис. 20. Мінімальні, середні та максимальні значення кореляції.

ВИСНОВКИ

1. На основі отриманих наукових та прикладних результатів можна стверджувати, що мета роботи «покращити якість процесу створення спільних ключів шифрування за допомогою відкритих каналів зв'язку в рамках алгоритму Діффі-Хелмана за допомогою створення односторонньої функції на основі клітинних автоматів з розширеними правилами поведінки» досягнута.

2. В роботі вдосконалено алгоритм Діффі-Хелмана за рахунок створення односторонньої функції на базі клітинних автоматів з розширеними правилами.

3. Автором вперше розширено правила клітинного автомата за напрямками універсального визначення правил зміни стану клітини, визначення околиці клітини, смерті від віку та багатокомпонентної системи клітин.

4. Створений алгоритм з модифікованою односторонньою функцією на базі клітинного автомата може бути використаний для створення ключів шифрування та паролів для безпечної передачі інформації.

5. В залежності від потреб користувачів складність ключів шифрування може бути підвищена за допомогою зміни початкових параметрів автомату, що також дозволить керувати швидкістю генерації ключів та шифрування загалом.

6. Програмне забезпечення реалізовано на трьох мовах програмування C#, Python та MatLab, що дозволяє виконувати порівняльних аналіз результатів і свідомо обирати мову програмування окремих частин програмного забезпечення для оптимізації процесу генерації ключів шифрування.

7. Оскільки розробка хеш-функції не є метою дослідження для прозорості результатів у якості хеш-функції були використані суми значень в стовпчиках матриці клітинного поля, що відображає результат. Але навіть така проста хеш-функція при чисельному тестуванні не дала жодного повтору значень на 100 000 ітераціях модифікованого клітинного автомату.

8. Було розроблено методи статистичної перевірки односторонніх функцій на базі клітинних автоматів для обрання найоптимальнішої функції.

9. Напрямки подальших досліджень – розширення правил клітинного автомату для урізноманітнення варіантів поведінки односторонньої функції.

Література

1. Mitsuru Matsui Robert - Selected Areas in Cryptography / Zuccherato Henri Gilbert, Helena Handschuh // Security Analysis of SHA-256 and Sisters – August 14 – August 15, 2003, Ottawa, Canada, pp. 175 – 193. doi.org/10.1007/978-3-540-24654-1_13

2. Метт Уийр. DEFCON 17. Взлом 400 000 паролей, или как объяснить соседу по комнате, почему счет за электричество увеличился. Часть 1. [електронний ресурс] // Сайт Хабр - Режим доступу https://habr.com/ru/company/ua-hosting/blog/422731/

3. Petrov, P., Dimitrov, G., Ivanov, S. “A Comparative Study on WebSecurity Technologies UsedinIrish and FinnishBanks.” 18 International MultidisciplinaryScientificGeoconference SGEM 2018: Conference Proceedings, 2 - 8 July 2018, Albena, Bulgaria : Vol. 18. Informatics, Geoinformatics a. RemoteSensing. Iss. 2.1. Informatics, Sofia : STEF92 Technology Ltd., Vol. 18, 2018, Iss. 2.1, pp. 3 - 10.

4. Pavel Petrov, Stefan Krumovich, Nikola Nikolov, Georgi Dimitrov, and Vladimir Sulov. 2018. “Web Technologies Used in the Commercial Banks in Finland.” InProceedingsofthe 19th International Conference on Computer Systems and Technologies (CompSysTech'18), Boris Rachev and Angel Smrikarov (Eds.). ACM, New York, NY, USA, pp. 94-98. DOI: https://doi.org/10.1145/3274005.3274018. ISBN: 978-1-4503-6425-6

5. **Шифр «Функція»**; Georgi Dimitrov; Denys Berestov; Pepa Petrova; Igor Sinitcyn; Eugenia Kovatcheva; Ivan Garvanov; Iva Kostadinova One-way Function Based on Modified Cellular Automata in the Diffie-Hellman Algorithm for Big Data Exchange Tasks through Open Space. – DIGILIENCE 2020: Cyber Protection of Critical Infrastructures, Big Data and Artificial Intelligence. – Varna, September 30 – October 2, 2020. – pp.233-246. http://isij.eu/isij-47-digilience-2020-cyber-protection-critical-infrastructures-big-data-and-artificial

6. Whitfield Diffie and Martin E. Hellman. IEEE Transaction on Information Theory. Vol. IT-22, No.6, November 1976, pp.644-654.

7. Протокол Диффи-Хеллмана. [електронний ресурс] // Сайт Википедия - Режим доступу https://ru.wikipedia.org/wiki/Протокол_Диффи_—_Хеллмана.

8. Stephen Hawking, Leonard Mlodinow. The Grand Design. New York, NY, USA: Bantam Books, 2010, ст. 187–196.

9. G. Schneckenreither, N. Popper, F. Breitenecker. Methods for Cellular Automata and Evolution Systems in Modelling and Simulation. – 8th Vienna International Conferenceon Mathematical Modelling MATHMOD 2015, February 18 – February 20, 2015, Vienna, Austria, pp. 141 – 146. doi.org/10.1016/j.ifacol.2015.05.151

10. **Шифр «Функція»**, Oleksiy Bychkov, Denis Berestov Real life models with cellular automatons. - MSTIoE 2019-5. 5-th East European Conference on Mathematical Foundations and Software Technology of Internet of Everything. – Kyiv, January 19-20, 2019. - pp.19-21.

11. Denis Berestov, **Шифр «Функція»**, N.M. Samojlenko New Rules for Real Life Models with Cellular Automatons. - Всеукраїнська Науково-Технічна Конференція «Застосування Програмного Забезпечення В Інфокомунікаціних Технологіях». – Kyiv, February 5, 2020. – pp.4-5.

12. **Шифр «Функція»**, Берестов Деніс Дослідження систем моделювання розповсюдження інформації в соціумі на базі клітинних автоматів. - XX Всеукраїнська науково-технічна конференція молодих вчених, аспірантів та студентів. - Одеса, Квітень 21-22, 2020. – сс.225- 226.

13. J. Freire, C. DaCamara. Using cellular automata to simulate wildfire propagation and to assist in fire management. - Natural Hazards and Earth System Sciences (NHESS), January 22, 2019, Lisbon, Portugal, pp. 169 – 179.

14. Lorenzo Piroli, Ignacio Cirac. Quantum Cellular Automata, Tensor Networks, and Area Laws. - Physical Review Letters, 6 November, 2020, Danvers, USA, p.5. doi.org/10.1103/PhysRevLett.125.190402

15. Stepan Bilan. Formation Methods, Models, and Hardware Implementation of Pseudorandom Number Generators: Emerging Research and Opportunities.- (2017).- IGI Global, USA.- P. 301. DOI: 10.4018/978-1-5225-2773-2.

16. Мухачев В.А., Хорошко В.А. Методы практической криптографии. – К.: ООО «Полиграф-Консалтинг», 2005. – 215 с.

17. Philip MacKenzie, Thomas Shrimpton, Markus Jakobsson. Threshold Password-Authenticated Key Exchange. // Journal of Cryptology, Vol. 19, Issue 1, January 2006, pp. 27-66. doi.org/10.1007/s00145-005-0232-5

18. Jin Hong, Sunghwan Moon. A Comparison of Cryptanalytic Tradeoff Algorithms. // Journal of Cryptology, Vol. 26, Issue 4, October 2013, pp. 559-637. doi.org/10.1007/s00145-012-9128-3

19. Salcido A. Cellular Automata - Simplicity Behind Complexity / Alejandro Salcido // Equilibrium Properties of the Cellular Automata Models for Traffic Flow in a Single Lane – 2011, pp. 159 – 192.

20. **Шифр «Функція»**, Denis Berestov, Igor Sinitcyn Built-In Processor for Sharing Passwords Through the Open Information Space. - 2020 16-th International Conference Perspective Technologies and Methods in MEMS Design (MEMSTECH). Proceeding. - Lviv, April 22-26, 2020. - pp.40-44. ieeexplore.ieee.org/document/9109523

21. Salcido A. Cellular Automata - Simplicity Behind Complexity / Rosana Matta Jafelice, Patricia Nunes da Silva // Studies on Population Dynamics Using Cellular Automata – 2011, pp. 105 – 130.

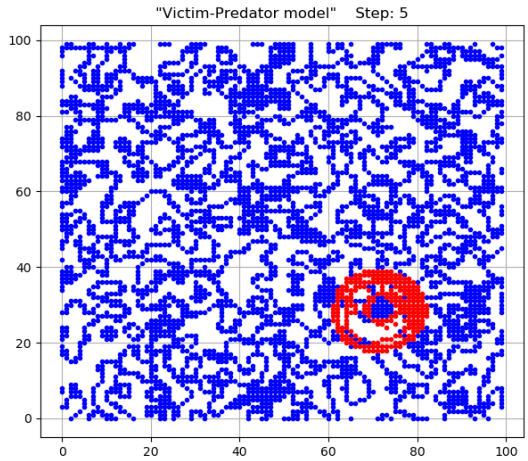
22. **Шифр «Функція»**, Тетяна Григор’єва. Вдосконалення алгоритму Діффі-Хелмана шляхом використання клітинних автоматів. – VII Всеукраїнської науково-практичної конференції студентів, аспірантів та молодих вчених «Об’єднані наукою: перспективи міждисциплінарних досліджень». – Київ, Листопад 12 – Листопад 13, 2020. – сс. 100 – 102.

23. **Шифр «Функція»**, Denis Berestov. Research of The Workability of One-Way Functions Based on Cellular Automata for The Modified Diffie-Helman Algorithm. - MSTIoE 2020-7. 7-th East European Conference on Mathematical Foundations and Software Technology of Internet of Everything. – Kyiv, December 22-23, 2020. - pp.22-23.

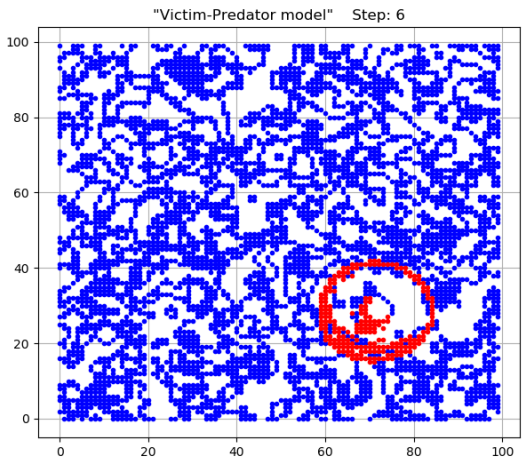
ДОДАТКИ

Додаток А

Ітерації автомату багатопопуляційної моделі з введеним правилом поїдання жертв



а)

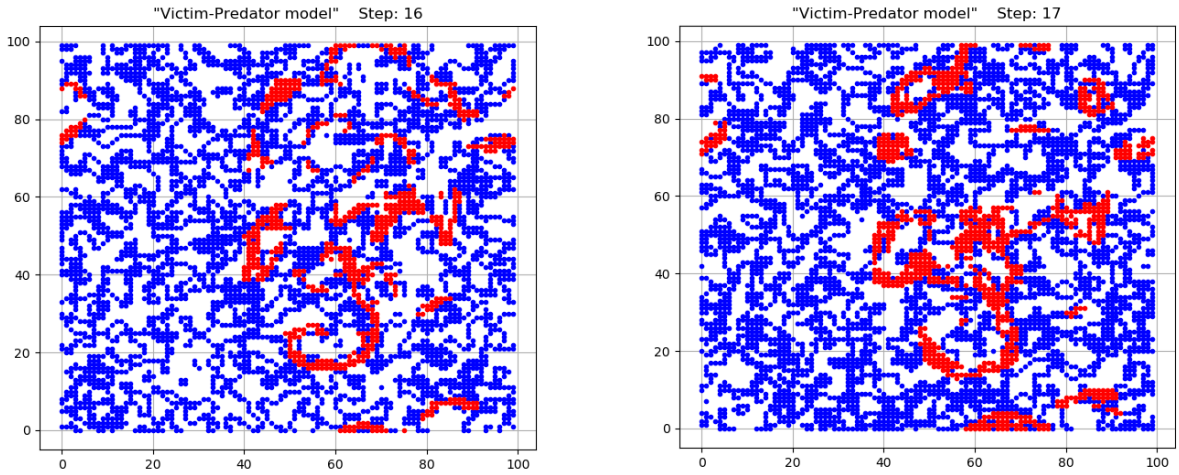


б)

Рис. А. Клітинний автомат із введеним правилом поїдання жертв хижаками (реалізовано на Python). а) ітерація 5, б) ітерація 6.

Додаток Б

Ітерації автомату багатопопуляційної моделі з введеними правилами поїдання жертв та голодом у хижаків



а) б)

Рис. Б. Клітинний автомат з голодом у хижаків і поїданням жертв хижаками. а) ітерація 21, б) ітерація 22.

Додаток В

Будова головної програми «CryptoAutomata»

Описаний алгоритм реалізований на мові Python.

```
import numpy as np
from matplotlib import pyplot as plt
import gofpack as gof
import cv2

columns = 100
rows = 100
max_step = 7134
max_age = 3
# Info about victim                maxRadius  randN
life_conditions_victim = np.array([2, 0.3])
A = [3, 14, 15, 16, 17, 18]
D = [10, 2, 3, 11, 12]
# Creation of arrays for victims
L_victim = np.zeros((columns, rows))
color_image = cv2.imread("cubism.jpg")
gray_image = cv2.cvtColor(color_image, cv2.COLOR_BGR2GRAY)
n_image = np.around(np.divide(gray_image, 255.0), decimals=1)
L2_victim = (n_image[0:columns, 0: rows] > life_conditions_victim[1]) * 1
# Creation of arrays for Radius
L_radius = np.random.rand(columns, rows)
L_radius = (L_radius > life_conditions_victim[1]) * 1 + 1
L_age = np.copy(L2_victim)
Sum_victim = np.zeros(max_step)
fig = plt.figure('Game Of Life', figsize=(7, 6))
for ii in range(max_step):
    L_victim = L2_victim
    NBA_victim = gof.nbrModified(L_victim, L_radius, life_conditions_victim[0])
    L2_victim = gof.nextStepModified(L_victim, NBA_victim, D, A)
    L2_victim = gof.fatalityAge(L_age, L2_victim, max_age)  # Death by old age for L2
    L_age = gof.ageL(L_age, L2_victim)  # Calculating the age of each cell in the array
L
    # Creation of arrays with x and y for plot
    Y_victim, X_victim, SumL_victim = gof.transarray(L_victim)
    Sum_victim[ii] = SumL_victim
    plt.title("Cellular automata    Iteration %i" % ii)
```

```
    plt.grid(True)
    # plt.imshow(L_age.T, cmap='inferno', alpha=1)
    print(ii)
    s = ''
    if ii == 7133:
        plt.plot(Y_victim, X_victim, ".b")
        plt.pause(0.1)
        for g in range(len(L_victim)):
            s += str(int(sum(L_victim[g])))
        print(s)
    if ii < max_step - 1:
        fig.clear()
steps = range(max_step)
plt.figure('Statistics', figsize=(11, 5))
plt.grid(True)
plt.plot(steps, Sum_victim, "-ob")
plt.show()
```

Додаток Г

Бітове представлення коду кінцевого клітинного поля

'1010110000000100000110010100000000000000010010000100010110000001
0000111101010000011011000001100000100000110000000000000000000010000
0010100100000000000000111000001011000011010000100010000000100010001
0000101001000000010000000100000000000000010000000100000001001101101
0111000100000000000010000000001010001000100000010011100000000000000
0100010000100000001110100100010000010000001101110000000000101000100
0
0001011000010010000000100101010010011001000000101000100001000010000
0000000000001000000000001011001000010000011100011001100000000000010000
0001000100000010000111000100000001100100000011110100010101001000000
0
0001110101010010011000001000000000100001000000001000011010100101000
0
0100000000010000000010000010001000000000000101101010000000000000000
0
0001100011000000010100000000000000010000100011001100110000010000000
0
0010001100111110100000000000000000010011001001000000000111110000100
0
0011000000100000001100000000000000000000011110110010000001100000000
0
0001000000000010000000100000000000110111111001000000000100000000000
0
1011010011100100000001111100011010000000101011001000110100100101000
0
1000101000000100000000010100000000010100011111000000000011101100000
0
0110011000001010010010001100000000011111010000000000000100110010101
0
0101111110000000100011111000000011101100000010000100001111111000000
0
1110000000101101010000000110000001010100000000000010011000000100011
0
1000000000000000000010001100000001001000000110000010000011000001010
0
1000100000001011111100001001011100000000001100001111110000000001100
0
0010000110100000001000000000000001000101000100000000100000000101000
0
0000011000101010100000011000111100100000001000000001100000100010000
0
1010000000010000000000000000000000000000000000011010010001000111010
0
0
0000000010000000011001100000000100100100010000000101101000000100000
0
0
0011000000001000010001011101100001000100010000000000100000000000100
0
0
1011010110000000111100110000000000100000000000000000000011000101001110

1100000001101100010000100001000000101010000000000100000110001010001 00
00000000000000000000110100010101000000000000100100011001000010100011
11100000000000000000100000010000000000010000010000000000100000000010001
10000011010000000000000000001000110100000000000100100011100100000010000
01000010000001000010000000000100011000000000000000000100000000110000 1100
000001101010001001011010000000100000010000001000100000000010000000000
10101100000110101010000000101100010001000010000001010110000000000000010
00000000000011101001111001000010000000000000000000000000000001001000000
10001000000001101010000000110001000000000000000000000101110110010000000000
01000110010010001100101010000101000000000000000000010000000000000101010001
01000000000000000001000000001100110001000010100010001001000000111 1100000
10000000000001000101000010001010000110100011110100110001010110100 0000
01000101000010010010000000001111010000000100000000000000001110000000000
01110000000000010100000001010000100000001010000000000000000100010010110101
10000100000011010101111010010011111000001000101001000101011001000 0110
00000001000000000000001000010000001100000101001100100010000000000000 11111
01000000000100110000000110011000000010000100111000011010001000000001010
10011000100000000101000100000000111110000000000100000010001001000010000
00110000000000000000101100111000000000111110000100010011110100100111100
00000100000000000000000100010000000100011110000000000000000100110100000000
01010100000000000000000000110011000000110100010000000000001000000001101 0100
01101010000001000010011000000000000011101001100000000001000000000100000 11
10100010010010111000000101000000000000010000001011000010101000000000001111
0010000101011010001010000110001111010000000001000000000000010010001010
000010000000000011000000000011000000100000000100100000000000000000000011110
1000000001110001000000010100000001000000000000000000000000011000101110 0100
000000000100000001001000000110111001000000000000000000000001100000100000000
01100100000001011010000000111010101111100000000000011000100000000010000
00100000000100000000000010110001000001000000000000100000000000101001000 00
01000100000001000010000000100100001000000001000010010110000000011000 0000

0100000000001100000011010000000001000000000000000000000000000010000
0000010011111100000001010000000010101010001100000100000000001000000
0000011000000001001100111000000101110000010000100001010000000000000
0000000001010100000001110000110000000001010110000000000000000000000
0000000100000000001010001011101000000101000100000010011001010100000
0000001000101001010000000000001000001001000000010001110001110000010
0010000000000000010000100100001010010010010000100001000100000001000
0110000000011000110110111100000110010011000001100100010000001100001
1101100000000100000001000000100001000100000010010001100001000011000
0011000000000000010001101000001000000011010101101000000000100000000
1000100001000000001000010000100000110000010001011100000000100011101
0000101100011000100010000001001000000000100000100000000010000010000
0000010001001110000000000011101100000000000100000010000011100000010
0100010000000001110001100000001100000000001111101000001011000001000
0000001001000000000001100011000110000000000001000010000000001000100
0000011111010000001100011111000000000000100000000000101001011101000
0000100001000010001000000101100000000000000010000000000100010100000
0111000000000000011100111010101000000100001001100001001000000000100
0100000100000000010000110000010000000000000000000001001000100001001
0000010111000001010110001000100000010011100011001000110110011010000
0000000000100001001000110000000000000000100001001100000111000001001
0000001000001000000000110010100000011000000001000000101111000011000
0010000000011000010110000110100000100011010000001000000110000010011
0100100000010000000110000000100000000100101010000000000100000000001
0001000000000000100011100000001000110101000000000101000001000000000
1000000011000100001000000000000001101000100010001001000000100010000
0000011000000000101100000000010000011000100000000110011100000000100
1100000000100100000100000000100000000000100000000000000000001100000
0000010100000100010000101100000001000000000100010001100001001000000
0001000100001100000000011000000100110010001110101001101010101000011

0001110000100111100000011001001100010000010010000010001001000000001
1000000011010000001000110010000001011000100100011010000100100000100
0
1000000010001000010101000010010110010000011000000010000001100000110
0
0101011000111000000010010010110001000010001010010011000010000001110
0
1101011100001010000000000000101000001100001101000010000111000001001
1
0001000000000111001000001111000000010100000000111110000100000001101
000
1011101110010000000000000100010001010000010001010101000011111000001
1
0011010000000010000001000111000001101010000000010000001100000000010
0
0010000000010000010011110000001100110001100000000000000001100000001
00
1010101000010001000001000010011100000001000000011111010100100010100
00
0100000000000001111000000100011000001010100000000010000000000110101
1000001111010001101100000011001110000000000100100000001001001100001
0
0001110110000000001010010010100000000100011011100000000001100111111
1000
0110010001100100010100001011011000010111000010110100000000000000100
0000010011110000000010100100000000000000100101001010010001000000000
0
0101000000010011000010000000000000000100011000000000001011111001000
0
0011100010001000100010100000110010000000011110110000000010100000000
0
0001010000000000000010000000011100000111011101101100100011001000100
0
0100000100100000100010110001111010011100010000010000000110000000000
0
0001010101100110010100100010100100100001001000000000000010101110010
0
1110000100000101010101011000001000000010000011000000000100111100000
00
0000101100001000100011000000000011010000010101110101010110110110011
1
1001001011000001000000000011000000000101011001000010111011001100000
000
0001010000100000001100010011100001011100100001000001000010000000000
0
0000110000100000001001010110000100000000001000000000100011000000000
0
0100010000001001000001000000110100100000010001001000001000000000100
0
0110101000001001010010011000000000011000000011000000100000000000010
0
0110000001000000000000000000000000010000000010011011000010001100000
0
0110000011100100010000000010100001011001000010010010000000000001000
0
0100000000000000100000000000100001110101001010000001000011001100100
0

1110000010001011001000001000000000000000010111000000000000110000000
0000000101000001000001000110101010001100010000010000000001000000000
0100000001000100000010000010100000001010000000100110000100000001110 01
1001000000001001111001010010110100010011000100000000000000000001000
1000000000000011001000000001100011010000000101000010010010101000100 1
0000100000000000100001100001000100001001001000000010101010000111000 0
0001100010000000011000000000010100000000001100000000000000000000001
0001100000100000000100101010000100010000001101100100010010000001000 0
0000000010000000000001000000000000000000100001110001100011100000000 0
0000100101011010000000000000000001101000110000001000001000000000000
0011010000000001000000010010010000000001010000000010000000100001011
0000000000000001010100011000000000000100000000000011010010100010000 1
0011000000000001010101000100000110100000000001000100110010001000000 0
0001000000000000001001000000111000001000000000000011010000010100000 0
0000001010000000001001000010000000001001000001010000000100001100000100
0000000011000000100110000101100100100101100000000100000101000000100 1
0010000001000010010001000000100001111000001000000000100000110000011 0
0000100010001000011000000000000000000000110000010000000110001010000 0
1000111000100100010000010100110011000001010000010101000000000000000
0110001010001011000011100100001001010000000000100000100001000100000 0
0000000100100010010001000010000010001100100010110010000001000000000
0100100000101100100000000010110000100000100010100001100000101010011 0
0001001001110000000011000000000011100001010000000000001001000000100 0
0000100000000000000010010100000000000000010110000011001011010111011 10
0000010000010000000100000000100001000000000000011010001100000010100 00
0000001010100011101100000000000000011001000010000100000010000100010 00
0001000001000110000001000000000000000010000100000100100011110001000 1
0000000100000000001000100010100011000000000000100111010000000000000 100
1010011010000110100000000011101001000010000001010000001000000100000 0
000100100'

Додаток Д

Шістнадцятирічне представлення коду кінцевого клітинного поля

AC041940004845821EA0D830418000020520007058684402221480808
00040204DAE200100511027000088407488206E005102C240952640A2108
4001002C841C66002022043880C81E8A9003AA4C10042010D4A08020104
4002D400003180A000423330400467D00004C9007C206040600001EC81800
20040400DF900400169C80F8D01591A4A114080500A3E007600CC1491803
E800132ABF011F01D81087F01C05A80C0A8004C08D00001180903041829
1017E12E0061F801843404001144020280 0C55031E4040608814020000000
D223A4010198092205A041060108BB08880100216B01E60040000629D80D
8842054010628800001A2A001232147C000204008200802306800023400 91C
8208410802300040618 0D44B404081100801583540588841580040 01D3C84
000000481101A81880002EC8008C91950A0002002A2800080CC4288907C1
0011422868F4C56808A12403D010003800E00502840A000896B08357A4F8
2291590C0200108305322003E804C06604270D101531014407C01022420600
059C03E113D278080008811E0009A00A80003306880101A8D4109800E980
2010744970500105854 01E42B450C7A01000914100300608090000 3D00E20
50200000C5C8008120DC80000C200C81680EAF800C402040400B10400400
A4088108121010960300801 81A0080000002009F80A015460800800C04CE0
5C1085000000A80E1802B0000000200A2E814409950004528004120238E04
40008485248422010C031B783260C88183B00808108812308606000 8D0406
AD0040110804210608B8047416311024010401040089C0076002041C08880
38C0600FA0B04004800C630008401120FA063E0020029742108440B000200
8A08E000E754084C2404082008608000048848 0B82B1104E32366A0004246
0002130704804100CA060105E1884030B0D046810304C9020301009500400
820023811A8050400501884001A222408800C02C01062019C041804820200
200006000A0885810088C240022180604C8EA6AA18384F03262090448030
1A04640B123424110110A84B20C040C18AC701258845261039AE14001418
6843826200E41E0280F840D177200088A08AA1F06680408E0D4020600840
209E0663000180954220 84E0207D48A08003C08C15004006B07A36067002

4049843B00A4A02370067F0C8C8A16C2E168000809E01480025291000A02
6100008C005F207111141900F6014002800100E0EED919108241163D38820
30002ACCA4524240015C9C20AAB04041804F0016111801A0AEAB6CF25
8200C02B217660028406270B9082100018404AC20040230008812081A4089
04008D4129300301810004C0800000201361180C1C88050B2124001080010
010EA5021991C116410000B800300028208D5188200800808810501409840
732013CA5A26200001100064031A0284951210010C221240550E03100C00
A0060000023040254220 6C8902001000800010E31C0012B5000068C082000
6802024802804021600 02A3000800694426002A8834008991002000481C100
0682800500908024140860801813 0B24B008281240848810F0401060C1110C
0000182031411C4882998282A0000C5161C84A0041088002448841191640 8
0090590058411430 54C24E01801C28004810100012800160CB5DC08202010
80068C0A0054760006421021102 08C0800042091E22020088A30009D00094
D0D0074840A0408024

Додаток З

Обрахунок значень кореляції для проміжків в 100 та 10 ітерацій

Для випадку, коли кореляція обраховувалася для проміжків в 100(Рис. З) ітерацій, ми продовжуємо спостерігати збільшення розкиду значень кореляції в локальних проміжках. При цьому якоїсь закономірності в розподілу значень не спостерігається.

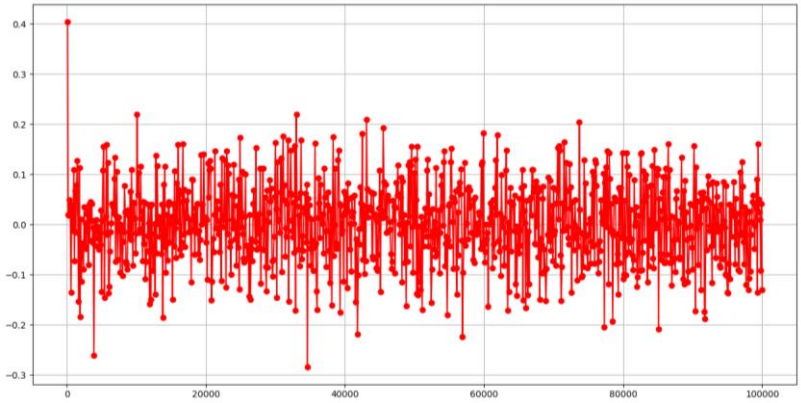


Рис. З. Коефіцієнт кореляції Пірсона для 1000 проміжків (1 проміжок – 100 ітерацій).

У випадку, коли кореляція обраховувалася для проміжків в 10(Рис. К) ітерацій, значення кореляції коливається ще в більших проміжках.

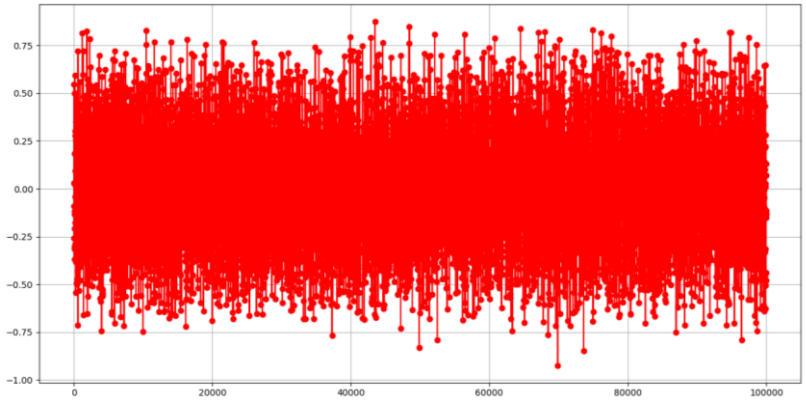


Рис. К. Коефіцієнт кореляції Пірсона для 10000 проміжків (1 проміжок – 10 ітерацій).