Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації

Комплексна дипломна робота на тему:

«Засіб потокового шифрування на основі латинських квадратів. Частина 2. Операційний блок.»

Виконав: студент 4 курсу групи 1БКС-20б
спеціальності 125 Кібербезпека
_____ Богдан ЗАГИРНЯК
Керівник: к. ф.-м. н., доцент каф. ЗІ
_____ Галина ШЕЛЕПАЛО
«17» червня 2024 р.
Рецензент: к. т. н., доц., доц. каф. ПЗ
_____ Олександр ХОШАБА
«17» червня 2024 р.

Допущено до захисту
Завідувач кафедри ЗІ
д. т. н., проф.
_____ Володимир ЛУЖЕЦЬКИЙ
«17» червня 2024 р.

Вінниця ВНТУ – 2024 року

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації
Рівень вищої освіти І (бакалаврський)
Галузь знань – 12 Інформаційні технології
Спеціальність – 125 Кібербезпека
Освітньо-професійна програма – Кібербезпека критичних систем

ЗАТВЕРДЖУЮ
Зав. кафедри ЗІ, д. т. н., проф.
Володимир ЛУЖЕЦЬКИЙ
12 березня 2024 року

З А В Д А Н Н Я
НА КОМПЛЕКСНУ БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Загирняку Богдану Дмитровичу

1. Тема роботи: "Засіб потокового шифрування на основі латинських квадратів. Частина 2. Операційний блок.",
керівник роботи: Шелепало Галина Василівна к.ф.-м.н., доцент, затверджені наказом ректора ВНТУ від 11 березня 2024 року №80.
2. Строк подання студентом роботи 17 червня 2024 р.
3. Вихідні дані до роботи:
   - приклади використання квазігруп у криптографії;
   - сучасні алгоритми шифрування на основі латинських квадратів;
   - вимоги до засобів LW-криптографії.
4. Зміст текстової частини: Вступ. 1 Аналіз інформаційних джерел. 2 Метод шифрування на основі латинських квадратів. 3 Розробка програмного засобу потокового шифрування. Висновки. Список використаних джерел. Додатки.
5. Перелік ілюстративного матеріалу: Таблиці Келі 4-го порядку для виконання операції шифрування з властивостями схрещеної оборотності (плакат А4). Тотальносиметрична таблиця Келі 4-го порядку для генератора гами (плакат А4). Схема алгоритму шифрування (плакат А4). Таблиці Келі 4-го порядку для виконання операції розшифрування (плакат А4). Структура засобу шифрування (плакат А4). Схема генератора псевдовипадкових послідовностей (плакат А4). Схема генератора гами (плакат А4). Порівняльні характеристики алгоритмів потокового шифрування (плакат А4).

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Шелепало Г. В., к. ф.-м. н., доцент	11.03.24	31.03.24
2	Шелепало Г. В., к. ф.-м. н., доцент	11.03.24	16.04.24
3	Шелепало Г. В., к. ф.-м. н., доцент	11.03.24	16.04.24

7. Дата видачі завдання: 12 березня 2024 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз завдання. Вступ	12.03.24 – 14.03.24	
2	Аналіз інформаційних джерел за напрямком комплексної бакалаврської дипломної роботи	15.03.24 – 31.04.24	
3	Розробка моделей та алгоритмів	01.04.24 – 15.04.24	
4	Практична реалізація, моделювання, результати	16.04.24 – 01.05.24	
5	Оформлення пояснювальної записки	02.05.24 – 10.05.24	
6	Попередній захист БДР	30.05.24 – 07.06.24	
7	Виправлення зауважень, підготовка ілюстративного матеріалу	10.06.24 – 12.06.24	
8	Перевірка на наявність текстових запозичень	12.06.24 – 13.06.24	
9	Представлення БДР до захисту, рецензування	13.06.24 – 17.06.24	
10	Захист БДР	18.06.24 – 21.06.24	

Студент _____ Богдан ЗАГИР

Керівник роботи _____ Галина ШЕЛЕП

АНОТАЦІЯ

Комплексна бакалаврська дипломна робота складається з двох частин. Дана робота, присвячена другій частині, нараховує 55 сторінок формату А4, на яких є 18 рисунків, 4 таблиці, список використаних джерел, що налічує 26 найменувань.

Комплексною бакалаврською дипломною роботою пропонується підхід до потокового шифрування на основі латинських квадратів. Такий інструмент сприяє покращенню ефективності та безпеки шифрування, а також дозволяє збільшити стійкість криптографічних систем, що базуються на квазігрупах.

Ключові слова: захист інформації, квазігрупа, LW-криптографія, кібербезпека, криптопримітив, шифр, метод, алгоритм, латинський квадрат, оцінка складності.

ABSTRACT

The complex bachelor thesis consists of two parts. This work, devoted to the first part, consists of 55 A4 pages format, which have 18 figures, 4 tables, the list of sources used contains 26 items.

This comprehensive bachelor's thesis proposes an approach to stream encryption based on Latin squares. Such a tool helps to improve the efficiency and security of encryption, as well as to increase the stability of cryptographic systems based on quasigroups.

Keywords: information protection, quasigroup, LW-cryptography, security, cryptoprimitive, chipher, method, algorithm, latin square, difficulty assessment.

ЗМІСТ

ВСТУП

Сьогодні технологічний процес розвивається надзвичайно швидко, змінюючи життя суспільства, надаючи методи для виконання банківських операцій та процесів керування віддалено, в режимі реального часу. Це обумовлює стрімкий розвиток інтернет технологій та пристроїв для роботи з інформацією. Такі процеси покращили якість життя людей та змінили методи ведення сучасного бізнесу.

Такий стан речей в IT світі робить інформацію найціннішим ресурсом, який впливає на всі сфери життя людства. Питання існування та успішності сучасного бізнесу повністю залежить від інформації якою він володіє та як з нею працює. Тому очевидним є той факт, що будуть особи які прагнуть отримати доступ до інформації яка їм не належить. Це створює виклики для забезпечення захисту інформації.

Станом на сьогодні, інформація активно циркулює не тільки у традиційних інформаційних системах, які забезпечені великою кількістю ресурсів для роботи, а також у системах з обмеженими ресурсами, через мініатюрну апаратну реалізацію. Тому існує потреба розробки методів шифрування для області LW-криптографії (від англ. Lightweight - легковаговий). Основною метою LW-криптографії є створення криптографічних алгоритмів, методів, засобів та протоколів, що мають низьку апаратну складність для їх використання в мініатюрних пристроях, де важливим є розмір кожного елемента.

Для виконання поставлених задач чудово підходить концепція потокового шифрування. Ідея блокового шифрування значно поступається потоковому шифруванні, через особливості роботи алгоритмів, що значною мірою впливає на ефективність обробки інформації.

Одним із ефективних інструментів для реалізації потокового шифрування, серед різних алгебричних структур, вважаються квазігрупи та їх аналоги у вигляді латинських квадратів. Такі інструменти успішно використовуються у сучасних шифрах із високою ефективністю роботи алгоритмів.

Використання таких інструментів не зменшує ефективність та актуальність досліджень даної тематики, оскільки такий підхід не є стандартним та є не повністю дослідженим. Разом із тенденцією до зменшення розміру апаратури та використання систем з обмеженими ресурсами, обґрунтовує актуальність такого дослідження.

Об’єктом роботи є процес потокового шифрування. Предметом роботи є метод, алгоритм та засіб для потокового шифрування на основі латинських квадратів.

Метою комплексної бакалаврської роботи є зменшення складності засобу потокового шифрування шляхом розробки методу шифрування на основі латинських квадратів 4-го порядку.

Для досягнення поставленої мети сформовано такі задачі:

– Проаналізувати тематику потокового шифрування.
– Виконати порівняльний аналіз методів потокового шифрування в області малоресурсної криптографії.
– Розробити метод та алгоритм потокового шифрування на основі латинських квадратів 4-го порядку.
– Розробити структуру засобу потокового шифрування та оцінити його складність.

Результати дослідження за темою були представлені у публікаціях:

Микитченко Б. В. Загирняк Б. Д. Крайнічук (Шелепало) Г. В. Побудова псевдовипадкових послідовностей на основі двох латинських квадратів. *Матеріали LIII наук.-техн. конф. ФІТКІ*, ВНТУ, 20-22 берез. Вінниця. 2024. С. 378-379.

Загирняк Б. Д. Микитченко Б. В. Крайнічук (Шелепало) Г. В. Концепція потокового шифрування на основі двох квазігруп. *І Міжнародна науково-практична конференція «Проблеми комп’ютерних наук, програмного моделювання та безпеки цифрових систем»*, ВНУ, 13-16 чер. Луцьк-Світязь. 2024. 19 С.

1 АНАЛІЗ ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

Даний розділ присвячений аналізу літературних джерел, розвитку досліджень квазігруп, їх властивостям та використанню у криптографії.

1.1 Визначення актуальності дослідження

Потреба в забезпеченні конфіденційності та безпеки інформації є невід'ємною частиною сучасного цифрового світу. З розвитком технологій і зростанням кількості даних, що передаються через мережі, постає необхідність у надійних методах шифрування для захисту конфіденційної інформації від несанкціонованого доступу. Саме тому розробка ефективних і стійких алгоритмів потокового шифрування на основі латинських квадратів є актуальною темою дослідження.

Використання застарілих методів шифрування наражає на небезпеку сучасні інформаційні системи. Така проблема постає із покращенням інформаційної потужності обчислювальної техніки. Таким чином старі алгоритми не можуть забезпечити потрібний рівень захисту, оскільки можуть бути розкриті. Тому необхідно розробляти та впроваджувати нові методи та підходи до захисту даних, які зможуть витримати навантаження сучасних обчислювальних систем.

Також важливо розуміти що у сучасному інформаційному світі об’єм інформацій для передачі зростає досить швидко. Тому алгоритми шифрування повинні бути не лише стійкими а і відповідати необхідній швидкодії при їх використанні. Оскільки є необхідність використання таких алгоритмів у системах із обмеженими ресурсами та різного роду особливостями. Також можливе використання алгоритмів у різних сценаріях систем, що має бути передбачено у таких алгоритмах шифрування.

Потокове шифрування відіграє важливу роль у забезпеченні безпеки передачі даних через мережі, оскільки воно дозволяє шифрувати дані в реальному часі, не чекаючи на повне завантаження повідомлення [1]. Це робить

його незамінним для таких додатків, як захищені потокові передачі даних, телефонні розмови та відеоконференції. Однак класичні алгоритми потокового шифрування, наприклад RC4, виявилися вразливими до різних атак, що підкреслює необхідність розробки нових, більш стійких методів [2].

Латинські квадрати є потужним математичним інструментом, який може бути використаний для створення стійких алгоритмів шифрування. Завдяки своїм унікальним властивостям, вони забезпечують високу ентропію і складність, що ускладнює криптоаналіз [3]. Проте, незважаючи на їхній потенціал, дослідження щодо використання латинських квадратів у потоковому шифруванні досі обмежені, і потребують подальшого вивчення та вдосконалення.

Деякі широко використовувані алгоритми потокового шифрування, виявилися вразливими до атак, таких як атаки на основі слабких ініціалізаційних векторів та атаки на основі відомих відкритих текстів [4]. Це підкреслює необхідність розробки нових, більш стійких алгоритмів, які можуть забезпечити вищий рівень безпеки для конфіденційних даних.

Латинські квадрати мають унікальні математичні властивості, які роблять їх цікавим інструментом для побудови криптографічних алгоритмів. Їх висока ентропія та складність можуть забезпечити стійкість до криптоаналітичних атак [5]. Однак, незважаючи на цей потенціал, дослідження щодо використання латинських квадратів у потоковому шифруванні досі обмежені.

Обсяг даних, що передаються через мережу інтернет зростає, тому існує постійна потреба в ефективних і масштабованих алгоритмах шифрування, які можуть забезпечити належний рівень безпеки без значного впливу на продуктивність систем [6]. Дослідження нових підходів до потокового шифрування на основі латинських квадратів може запропонувати рішення, які відповідають цим вимогам.

У сучасному цифровому світі безпечний обмін даними є критично важливим для багатьох галузей, таких як фінанси, охорона здоров'я, уряд та військова сфера. Розробка надійних алгоритмів шифрування, наприклад

потокове шифрування на основі латинських квадратів, може забезпечити необхідний рівень безпеки для захисту конфіденційних даних під час передачі.

Будь-який новий алгоритм шифрування потребує ретельного криптографічного аналізу та тестування для оцінки його стійкості до різних видів атак. Дослідження потокового шифрування на основі латинських квадратів дозволить провести глибокий аналіз властивостей та характеристик даного підходу, а також визначити потенційні слабкі місця та можливості для вдосконалення.

У сфері криптографії відкритість та прозорість є важливими принципами для забезпечення довіри та широкого впровадження нових алгоритмів. Дослідження в галузі потокового шифрування на основі латинських квадратів сприятиме відкритому обговоренню та аналізу даного підходу в науковому співтоваристві, що може призвести до покращення та прийняття нових стандартів безпеки.

Дане дослідження поєднує елементи математики, криптографії та інформаційної безпеки. Такий міждисциплінарний підхід може сприяти розвитку нових ідей та інноваційних рішень, які виходять за рамки традиційних методів шифрування.

Враховуючи зазначені фактори, дослідження в галузі потокового шифрування на основі латинських квадратів є актуальним та важливим напрямком, який може зробити значний внесок у забезпечення безпеки та конфіденційності даних у цифровому світі.

1.2 Огляд потокового шифрування

Потокове шифрування – це тип симетричного криптографічного шифрування, де біти або байти відкритого тексту шифруються один за одним, на відміну від блокового шифрування, де вхідні дані обробляються блоками фіксованого розміру. У потоковому шифруванні шифротекст генерується шляхом комбінування відкритого тексту з псевдовипадковою цифровою послідовністю ключових елементів, що називається ключовим потоком.

Потокові шифри працюють шляхом породження ключового потоку псевдовипадкової послідовності даних, що використовується для шифрування відкритого тексту. Ця послідовність генерується за допомогою генератора псевдовипадкових чисел (ГПВЧ) на основі секретного ключа.

Процес шифрування відбувається шляхом виконання операції XOR (виключне АБО) між відкритим текстом і ключовим потоком. Результатом є шифротекст, який є засекреченими даними. На приймальному боці виконується та сама операція XOR між шифротекстом і тим самим ключовим потоком, що і відновлює вихідний відкритий текст.

Безпека потокового шифру цілком залежить від випадковості ключового потоку. Ідеальний ключовий потік повинен бути цілком випадковим і ніколи не повторюватися. Насправді ключовий потік генерується детермінованим ГПВЧ, який ініціалізується секретним ключем, тому він є псевдовипадковим. Синхронізація між передавачем і приймачем є критично важливою для успішного розшифрування. Будь-яка втрата синхронізації призведе до помилкового розшифрування шифротексту після точки розсинхронізації.

У синхронних потокових шифрах ключовий потік генерується незалежно від відкритого тексту чи шифротексту. ГПВЧ періодично ініціалізується секретним ключем, і потім продовжує генерувати псевдовипадкову послідовність на основі свого внутрішнього стану.

Найпоширенішим типом синхронних потокових шифрів є ті, що використовують регістрах зсуву з лінійним зворотним зв'язком (LFSR). LFSR - це регістр зсуву, біти якого зсуваються з кожним тактом, а нові біти обчислюються як деяка функція від попередніх бітів регістра відповідно до поліному зворотного зв'язку.

Одиночний LFSR має обмежений період повторення послідовності та легко передбачуваний. Тому на практиці часто комбінують кілька LFSR разом для збільшення періоду та ускладнення передбачуваності вихідної послідовності. Прикладами синхронних потокових шифрів на основі LFSR є A5/1 та A5/2, які використовувалися в GSM.

На відміну від синхронних, асинхронні потокові шифри генерують ключовий потік, використовуючи шифротекст від попереднього кроку шифрування. Вони засновані на режимах роботи блокових шифрів і фактично перетворюють блоковий шифр на псевдовипадковий генератор.

Найпоширенішими режимами асинхронних потокових шифрів є:

– Режим лічильника (CTR) - блоковий шифр шифрує послідовність лічильників, а результат виконує операцію XOR з відкритим текстом.
– Режим зчеплення шифротекстів (CFB) - вхідними даними для блокового шифру є попередній шифротекст, а результат виконує операцію XOR з поточним відкритим текстом.
– Режим зворотного зв'язку за виходом (OFB) - схожий на CFB, але в якості вхідних даних використовується попередній вихід шифру, а не шифротекст.

Прикладами асинхронних потокових шифрів є AES-CTR, який базується на AES у режимі лічильника.

Потоковий шифр повинен відповідати певним критеріям для забезпечення високого рівня безпеки, продуктивності та сумісності з різними платформами[7]. Р. Рюппель визначив чотири основні підходи до проектування потокових шифрів[1]:

– Системно-теоретичний підхід, що передбачає створення складної або невідомої задачі для спеціалістів криптоаналітиків.
– Комплексно-теоретичний підхід, заснований на відомій, але складній проблемі.
– Підхід інформаційних технологій, спрямований на приховування відкритого тексту від криптоаналітиків, незалежно від часу, витраченого на дешифрування.
– Випадковий підхід, метою якого є створення задачі, вирішення якої фізично неможливе для криптоаналітиків.

Під час проектування потокових шифрів необхідно дотримуватися низки вимог, включаючи теоретичні критерії Р. Рюппеля. Зокрема, схеми генераторів

псевдовипадкових послідовностей повинні мати великі періоди вихідної послідовності, гарні статистичні властивості, нелінійність або високу лінійну складність вихідної послідовності, а також забезпечувати складну залежність кожного біту вихідного потоку від більшості бітів ключа.

Ці вимоги досягаються шляхом ефективного поєднання різних компонентів, таких як базова структура алгоритму, лінійні та нелінійні перетворення тощо. Більшість існуючих схем потокового шифрування складаються з окремих блоків, основними з яких є регістри зсуву зі зворотним зв'язком, функції дискретної складності, запам'ятовуючі пристрої та вузли, що реалізують нерівномірний рух.

Безпека потокового шифру значною мірою залежить від властивостей ключового потоку, який повинен відповідати критеріям довгого періоду та псевдовипадковості ключового потоку, стійкості до атак та криптоаналізу.

Період - це довжина послідовності, після якої вона починає циклічно повторюватися. Для безпечного потокового шифру період ключового потоку має бути максимально довгим, в ідеалі не повторюватися взагалі протягом усього процесу обробки даних.

Послідовність ключового потоку повинна бути випадковою, для унеможливлення передбачити наступний біт на основі попередніх бітів. Вона має задовольняти всі статистичні тести на випадковість.

Потоковий шифр має бути стійким до різних типів атак, таких як атаки з відомим відкритим текстом, відомим шифротекстом, виведенням станів ГПВЧ тощо. Не повинно бути можливості відновити секретний ключ або частини ключового потоку.

Зусилля, необхідні для успішного криптоаналізу шифру і визначення ключа, повинні бути обчислювально нездійсненними для противника.

Для подальшої роботи та розробки необхідно проаналізувати конкретні приклади потокових ширів та їх застосувань. Одним з найвідоміших і широко використовуваних потокових шифрів був RC4, розроблений Роном Рівестом у 1987 році. Він став популярним завдяки простоті реалізації та швидкодії. Проте

з 2000-х років виявлено кілька серйозних вразливостей, і на сьогодні RC4 більше не вважається безпечним і не рекомендується до використання.

Сучасний високошвидкісний потоковий шифр ChaCha20 був запропонований Денієлом Дж. Бернстайном у 2008 році. Він базується на операціях над 32-бітними словами і демонструє високу продуктивність на сучасних процесорах. ChaCha20 пройшов ретельний криптоаналіз і вважається безпечним на поточний момент.

Поширений приклад асинхронного потокового шифру - це режим лічильника AES (AES-CTR). Він використовує блоковий шифр AES для шифрування послідовності лічильників,

Потоковий шифр ZUC був розроблений Datastorm та Samsung у 2011 році як частина стандарту 4G LTE. Він є основним потоковим шифром для мереж 5G і має високий ступінь паралелізму, що робить його ефективним для апаратних реалізацій.

Потокові шифри знаходять широке застосування у різноманітних сферах завдяки своїй особливості обробки даних у потоковому режимі, без необхідності збирати їх у блоки фіксованої довжини. Потокові шифри ідеально підходять для шифрування мультимедійних даних, таких як голосові або відео трансляції, де затримки неприпустимі. Вони можуть оброблювати дані по мірі їх надходження без накопичення.

Багато протоколів безпечного з'єднання, таких як SSL/TLS, SSH, IPsec, використовують потокові шифри, часто в режимах CTR або GCM для забезпечення конфіденційності трафіку. Потокові шифри відіграють важливу роль у захисті бездротових мереж, таких як GSM, GPRS, UMTS та LTE, де вони використовуються для шифрування трафіку між мобільними пристроями та базовими станціями.

Деякі системи шифрування файлів або дисків, такі як BitLocker, можуть використовувати потокові шифри в режимах XTS або XEX для ефективного шифрування великих обсягів даних. Через обмежені ресурси вбудованих

пристроїв часто використовуються легкі та ефективні потокові шифри, наприклад ChaCha, для захисту конфіденційності даних.

1.3 Аналіз методів потокового шифрування в області малоресурсної криптографії

Розробка та представлення алгоритмів потового шифрування відбувається постійно, що створює великий об’єм інформації для порівняння для аналізу. Але аналізуючи такі алгоритми, доцільно звернути увагу на результати конкурсу eSTREAM. Його мета полягала у виявленні найкращого шифру на заміну застарілим алгоритмам [8].

Серед учасників та переможців було багато цікавих рішень які показали хороші результати для аналізу.

Алгоритм Grain показав хороші результати, що дозволяють його реалізувати у системах з обмеженими ресурсами. Використовує два регістри зсуву (LFSR та NFSR) для генерації ключового потоку. Grain використовує 80-бітний секретний ключ та 64-бітний вектор ініціалізації [9].

Хоча такий алгоритм належить до області LW-криптографії, але є підходи та методи для його вдосконалення для більш розширених систем.

MICKEY 2.0 розроблений для високої стійкості при низькому використанні ресурсів. Алгоритм використовує 80-бітний ключ та вектор ініціалізації розміром до 80 бітів. Має два 100-бітних асинхронних регістри зсуву (лінійний та нелінійний), що робить алгоритм стійким до методів криптоаналізу, які використовуються проти синхронних шифрів. Але має і свої недоліки, а саме досить висока складність реалізації. Оскільки використовується два регістри які не є досить ефективниим для апартної реалізації. Не зважаючи на це алгоритм залишається перспективним для подальшого аналізу.

Trivium використовує 80-бітний ключ та 80-бітний вектор ініціалізації, його внутрішній стан описується трьома взаємопов'язаними регістрами зсуву з нелінійним зворотним зв'язком розмірами 93, 84 та 111 бітів [10]. Його особливістю стала здатність до паралелізації, що дозволяє обробляти до 64 бітів

повідомлення одночасно з незначним ускладненням апаратної реалізації. Але всі переваги розбиваються об призму складності його апаратної реалізації. Були спроби створити аналоги із меншою апаратною складністю, але вони були успішно зламані [11].

Enocoro-128 призначений для високої швидкості шифрування з використанням нерегулярного регістру зсуву. Шифр використовує 128-бітний секретний ключ та 64-бітний вектор ініціалізації. Алгоритм також демонструє хороші показники стійкості через особливості своєї реалізації. Але при цьому є складним для реалізації в області LW-криптографії.

F-FCSR-H використовує функціональні фідбек-здвигові регістри (FCSR) для генерації ключового потоку. Шифр застосовує 80-бітний секретний ключ та 80-бітний вектор ініціалізації. Його апаратна складність також висока і не підходить до вимог систем з обмеженими ресурсами. Також у ньому було виявнено вразливість яка розкривається у можливості перетворення нелінійної функції FCSR в лінійну [12].

Salsa20 відомий своєю швидкістю і безпекою, який використовує 32-бітні слова для шифрування блоків. Може використовувати 128 або 256-бітний секретний ключ та 64-бітний вектор ініціалізації. Алгоритм в першу чергу було розроблено для програмної реалізації, але згодом було проведено процес його апаратної реалізації. Такий алгоритм у апаратній реалізація показав непогані показники ефективності але через складність реалізації його не можна назвати малоресурсним [13].

LIZARD представлений на FSE 2017 та розроблений для області малоресурсної криптографії. Він використовує регістри з нелінійним зворотнім зв’язком довжиною 31 і 90 біт відповідно, і функцію виходу. LIZARD використовує 120-бітний секретний ключ та 64-бітний відкритий вектор ініціалізації. У порівнянні з Grain, має інший компонентний склад, що дозволяє зменшити апаратну складність [14].

Fruit-80 використовує секретний ключ довжиною 80 біт, розмір регістрів зсуву становить 43 та 37 бітів відповідно, що позитивно впливає на апаратну

складність алгоритму. Додатково використовує функцію g(.), яка проводить операції додавання за модулем 2 з ключем та значеннями заздалегідь визначених регістрів з регістру зсуву з нелінійним зв'язком [15]. Але алгоритм має недолік: вектор ініціалізації не може бути використаний повторно.

Plantlet використовує 80-бітний ключ та 80-бітний вектор ініціалізації, внутрішній стан якого описується двома взаємопов'язаними регістрами зсуву з нелінійним зворотним зв'язком розмірами 37 та 59 бітів. Алгоритм схожий на Fruit-80, але поліном, на основі якого побудовано регістр, та вихідна функція що впливає на ініціалізацію регістра відрізняються.

LILLE використовує 80-бітний ключ та його внутрішній стан є коротким і описується трьома взаємопов'язаними регістрами зсуву з нелінійним зворотним зв'язком розмірами 31, 32 та 37 бітів. Алгоритм не потребує багато ресурсів і має просту апаратну реалізацію. Під час роботи, повідомлення надходить у функцію, де частина ключа та повідомлення відкидаються, а інша частина накладається на результат як гама, що потім повертається в регістр стану.

У таблиці 1.1 наведено порівняльні характеристики досліджених потокових шифрів.

Таблиця 1.1 - Порівняльні характеристики потокових шифрів

Назва властивості потокового шифру	Назва шифру									
	Grain	MICKEY 2.0	Trivium	Enocoro-128	F-FCSR-H	Salsa20	LIZARD	Fruit-80	Plantlet	LILLE
Довжина ключа, біт	80	80	80	128	80	256	120	80	80	80
Програмно орієнтований	-	-	-	-	-	+	-	-	-	-
Апаратно орієнтований	+	+	+	+	+	-	+	+	+	+
Генератор ПВП з нелінійним перетворенням	+	+	+	+	+	+	+	+	+	+
Генератор ПВП з декількома регістрами зсуву	+	+	+	+	+	-	+	+	+	+
Генератор ПВП з різним тактуванням регістрів зсуву	-	-	±	-	-	-	-	-	-	-

Проаналізувавши потокові шифри, можна зробити висновок, що наведені алгоритми є ефективними для захисту даних. Вони відрізняються відносною

простотою реалізації та надійним рівнем захисту, що дозволяє використовувати їх у різних сферах. Ідея використання генераторів ПВП є досить перспективними, навіть у найпростішій реалізації. Сучасні алгоритми шифрування потребують вдосконалення з одного боку, а концепція псевдо невизначеної моделі перетворення дає можливість для цього з іншого боку.

1.4 Використання квазігруп у криптографії

Застосування квазігруп в криптографії є широким і різноманітним, вони займають важливе місце в розробці безпечних криптографічних протоколів, алгоритмів та систем.

Квазігрупи є алгебричними структурами, які знайшли широке застосування в криптографії завдяки своїм унікальним властивостям. У цій частині роботи ми розглянемо основні принципи використання квазігруп у криптографії, переваги таких методів та конкретні приклади їх застосування.

Квазігрупа $(Q,*)$ — це множина Q з бінарною операцією (*), яка задовольняє такій умові, що для будь-яких $a,b \in Q$ існують єдині $x,y \in Q$ такі, що

$$a*x = b \qquad y*a = b.$$

Іншими словами, у кожній квазігрупі для кожного елемента множини можна знайти єдиний інший елемент, що задовольняє вказаним рівнянням з операцією (*) [16].

Основні властивості квазігруп:

- Перейменування: кожний елемент можна перейменувати іншим з тієї самої множини, зберігаючи властивості основної операції.
- Оборотність: для кожного елемента існує обернений елемент.
- Зображуваність: квазігрупу можна подати у вигляді таблиці Келі з вказаною операцією, елементи якої зображені у вигляді латинського квадрату, як результату записаного у внутрішній таблиці Келі.
- Унікальність: латинський квадрат забезпечує єдиність кожного елемента у кожному рядку та стовпці, що й підтверджує унікальність квазігрупи.

Криптографія використовує квазігрупи для створення криптосистем через їх здатність забезпечувати високий рівень безпеки за допомогою складних математичних перетворень. Основні напрямки застосування включають генерацію псевдовипадкових послідовностей, розробку потокових шифрів, хеш-функцій і систем автентифікації.

Квазігрупи використовуються для генерації псевдовипадкових чисел, які є основою багатьох криптографічних алгоритмів. Використання квазігруп у генераторах псевдовипадкових послідовностей дозволяє отримувати високий рівень непередбачуваності та однорідності.

В потокових шифрах використовуються квазігрупи для перетворення відкритого тексту в шифротекст. Алгоритми на основі квазігруп забезпечують високий рівень криптостійкості, завдяки унікальним властивостям квазігруп, наприклад, відсутність комутативності та асоціативності, що ускладнює криптоаналіз.

Використання квазігруп у конструкції хеш-функцій дозволяє отримати хеші з високою стійкістю до колізій. Квазігрупи допомагають створювати складні перестановки вхідних даних, що ускладнює обчислення оборотного значення хешу, завдяки властивостям оборотності в бінарному випадку – односторонньої оборотності, лівої оборотності, правої оборотності, комутативної оборотності, напівсиметричної оборотності та асиметричної оборотності.

Квазігрупи застосовуються у протоколах автентифікації для забезпечення перевірки автентичності повідомлень. Завдяки своїм алгебричним властивостям, квазігрупи дозволяють створювати ефективні і безпечні алгоритми перевірки автентичності.

Прикладом застосування квазігруп є Алгоритм Лобо. Це один з класичних прикладів використання квазігруп у криптографії, який використовує квазігрупи для генерації псевдовипадкових чисел. Алгоритм базується на ітеративному застосуванні квазігрупових операцій для отримання послідовності чисел з високою ентропією [17].

Основні кроки алгоритму Лобо можна описати так:

– Ініціалізація: Вибрати початковий елемент квазігрупи та створити початкову таблицю.

– Рекурсія: Заповнити таблицю квазігрупи, перевіряючи при цьому, чи задовольняються умови Латинського квадрата (у кожному рядку та кожному стовпці повинні бути унікальні елементи).

– Перевірка та повернення: Якщо всі умови задоволені, повернути отриману квазігрупу. Якщо ні, повернутись до попереднього кроку та спробувати іншу комбінацію.

Іншим прикладом є метод Шута, який використовує квазігрупи для побудови потокових шифрів. Цей метод забезпечує високу стійкість до атак на основі аналізу частотності завдяки складним алгебричним перетворенням [18].

Основні етапи методу Шута:

– Вибирається квазігрупа Q з певною операцією (*). Квазігрупа визначається таблицею Келі, де для будь-яких елементів a і b існує єдиний елемент с такий, що $a * b = c$.

– Задається початкова послідовність S та початкове значення seed, яке буде використовуватися для генерації псевдовипадкових чисел.

– Виконується ітеративне перетасування послідовності S з використанням операції квазігрупи. Для кожного елемента послідовності S_i виконується операція квазігрупи з іншим елементом послідовності або з певним фіксованим елементом:

$$S_{i+1} = S_i * K$$

де K — деякий фіксований елемент або залежний від попередніх значень.

– Процедура повторюється певну кількість разів для досягнення необхідного рівня перетасування і забезпечення високої ентропії послідовності.

Потокові шифри на основі квазігруп та їх парастрофів описали С.Марковський, Д.Глігороський та В.Бакева у своїй праці "Quasigroup and hash functions" для застосування квазігруп та їх парастрофів у потокових шифрах. Вони показали, що використання парастрофів квазігруп дозволяє створювати

більш складні і стійкі до атак шифри. Парастрофи є варіантами квазігрупи, які отримуються шляхом зміни циклу операцій, що додає додатковий рівень складності і безпеки [19].

Перевагами використання квазігруп у криптографії вважаються: безпека, ефективність та варіативність. За безпеку відповідають деякі властивості квазігруп, зокрема некомутативність та неасоціативність, що значно ускладнюють криптоаналіз. Алгоритми на основі квазігруп можуть бути ефективно реалізовані на апаратному і програмному рівнях. Квазігрупи дозволяють створювати різноманітні криптографічні схеми, що можуть бути адаптовані до специфічних вимог безпеки.

Квазігрупи є потужним інструментом у криптографії завдяки своїм унікальним алгебричним властивостям. Вони забезпечують високий рівень безпеки та ефективності криптографічних алгоритмів, що робить їх невід'ємною частиною сучасних криптографічних систем. Застосування квазігруп для генерації псевдовипадкових послідовностей, потокових шифрів, хеш-функцій та систем автентифікації підкреслює їх важливість і перспективність у розробці криптографічних технологій.

1.5 Висновки до розділу

Перший розділ дипломної роботи, присвячений аналізу інформаційних джерел, що дає змогу зрозуміти сучасні виклики та потреби у галузі потокового шифрування. Зокрема, розділ висвітлює актуальність теми дослідження, враховуючи зростаючі вимоги до безпеки та конфіденційності інформації у цифровому світі.

Зі зростанням обсягів даних та розвитку технологій виникає необхідність у надійних методах шифрування для захисту конфіденційної інформації від несанкціонованого доступу. Це підкреслює важливість розробки ефективних і стійких алгоритмів потокового шифрування, зокрема на основі латинських квадратів.

Дослідження поєднує елементи математики, криптографії та інформаційної безпеки, що сприяє розвитку нових ідей та інноваційних рішень, виходячи за рамки традиційних методів шифрування. Такий підхід може значно підвищити рівень безпеки інформаційних систем.

Потокове шифрування, де біти відкритого тексту шифруються один за одним, є важливим для забезпечення конфіденційності даних. Основним елементом цього процесу є генерація псевдовипадкових послідовностей, що забезпечують високий рівень стійкості до атак.

Зі зростанням обсягів даних існує потреба в ефективних та масштабованих алгоритмах шифрування, які не впливають на продуктивність систем. Латинські квадрати можуть запропонувати перспективні рішення для таких вимог, забезпечуючи високу ентропію та стійкість до криптоаналітичних атак.

Латинські квадрати мають унікальні математичні властивості, що робить їх цікавим інструментом для побудови криптографічних алгоритмів. Вони можуть забезпечити стійкість до атак завдяки своїм математичним особливостям.

2 МЕТОД ШИФРУВАННЯ НА ОСНОВІ ЛАТИНСЬКИХ КВАДРАТІВ

Другий розділ присвячений розробці методу та алгоритму потокового шифрування на основі латинських квадратів.

2.1 Аналіз особливостей латинських квадратів

Латинський квадрат в криптографії визначається як n×n масив, заповнений n різними символами таким чином, що кожен символ з'являється точно один раз у кожному рядку та кожному стовпці [21].

Для бінарної квазігрупи, яку описує латинський квадрат, окрім головної операції, існує ще п’ять перетворень, які називаються парастрофами. Такими перетвореннями є:

- парастроф лівого ділення, тобто цикл (13);
- парастроф правого ділення, тобто цикл (23);
- парастроф комутування, тобто цикл (12);
- парастроф комутування правого ділення, тобто цикл (312)
- парастроф комутування лівого ділення, тобто цикл (231).

Якщо позначити операцію як *, то головна операція буде мати вигляд:

$$a * b = c \tag{2.1}$$

де a відповідає за рядок лятинського квадрату, а b за стовпець, тоді c вважається результатом перетину таких елементів a та b .

Ліве ділення визначається як операція, що задовольняє рівність:

$$c * b = a \tag{2.2}$$

де елементи a та c головної оперції було змінено місцями.

Парастрофом правого ділення вважається операція яка відповідає рівності:

$$a * c = b \tag{2.3}$$

такий парастроф як і ліве ділення полягає у зміненні розташування елементів b та c.

Операція комутування визначається рівністю:

$$a * b = b * a \tag{2.4}$$

що у свою чергу виконується до латинського квадрату і виконує процес транспонування матриці.

Перетворення комутування лівого або правого ділення полягають у виконанні операції комутування до рівностей лівого та правого ділення. Згідно цього парастроф комутування лівого ділення має вигляд:

$$c * a = b \tag{2.5}$$

Відповідно парастроф комутування правого ділення має вигляд:

$$b * c = a \tag{2.6}$$

Таким чином із одного латинського квадрату який описує головну операцію, є можливість, за допомогою перетворень, отримати ще п’ять латинських квадратів які будуть мати свої унікальні властивості та можуть бути використані у шифруванні.

Окрім базових парастрофних перетворень, розглянуто закони: асоціативності, напівсиметричності та тотальносиметричності.

Квазігрупа вважається асоціативною, якщо її головна операція задовольняє асоціативний закон:

$$(a * b) * c = a * (b * c) \tag{2.7}$$

Проте, варто зазначити, що в загальному випадку квазігрупа не обов'язково є асоціативною.

Квазігрупа називається напівсиметричною, якщо для її операції виконується симетрична властивість:

$$(a * b) * a = b \tag{2.8}$$

В подільшому для шифрування будуть обрані латинські квадрати, які будуть задовольняти різні описані закони.

2.2 Теоретична основа алгоритму шифрування

Вхідними даними для роботи алгоритму є повідомлення *M* та секретний ключ *K*.

Для роботи такого алгоритму є необхідний генератор псевдовипадкових послідовностей. На основі двох останніх біт ключа, в алгоритмі під час роботи

буде використано одну із чотирьох псевдовипадкових послідовностей отриманих від генератора.

Оскільки алгоритм є потоковим шифруванням, то робота з вхідним текстом буде відбуватись по два біти. При цьому для такого функціоналу буде використано регістр зсуву.

Регістр зсуву є послідовним логічним колом, що служить для зберігання та передачі бінарних даних. Він складається з каскаду тригерів (звичайно D-тригерів), де вихід одного тригера з'єднаний з входом наступного. Це дозволяє даним переміщуватися через регістр по одному біту на кожен тактовий імпульс. Регістр зсуву може працювати у різних конфігураціях: послідовний ввід/послідовний вивід (SISO), послідовний ввід/паралельний вивід (SIPO), паралельний ввід/послідовний вивід (PISO) та паралельний ввід/паралельний вивід (PIPO) [22].

Таким чином вхідне повідомлення буде у вигляді набору із пар бітів, які будуть використовуватись у шифруванні. І може бути описано послідовністю у вигляді:

$$M = (m1, m2, m3 \dots)$$

де m відповідає за набір із двох біт.

Ключ аналогічно з повідомленням є вхідними даними для роботи алгоритму і складаються з набору блоків по два біти. Тому такий ключ буде поданий у вигляді послідовності:

$$K = (k1, k2, k3 \dots)$$

де k також відповідає за набір із двох біт. При тому останню пару біт у ключі буде використано як вхідні дані для генератора із метою визначення псевдовипадкової послідовності яка буде використовуватись в алгоритмі.

Для додаткової крипостійкості ключа, він буде додатково проходити модифікацію за допомогою таблиці Келі Q_0, що описує CIP-квазігрупу [23].

Наступне значення ключа буде визначатись на основі вхідного повідомлення та поточного значення ключа, і може бути подано у вигляді виразу:

$$K_i = Q_0[m_i][k_i] \quad (2.9)$$

Такий процес описує створений генератор гами, схема якого наведена на рисунку 2.1.

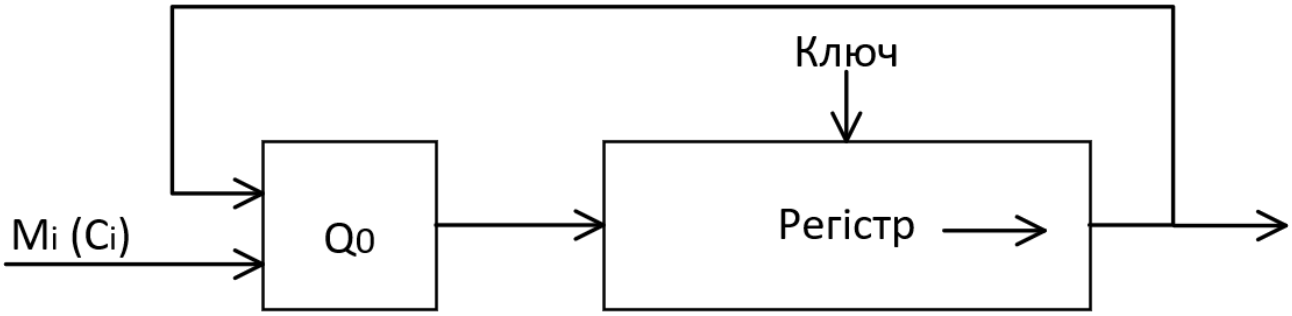


Рисунок 2.1 – Схема генератора гами

Таким чином використані біти ключа під час шифрування будуть відновлюватись на основі ключа та повідомлення.

Таблиця Келі Q_0 для роботи генератора гами наведено на рисунку 2.2.

*	0	1	2	3
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

Рисунок 2.2 – Таблиця Келі генератора гами

Також така таблиця Келі описується двома логічними виразами, за допомогою яких і буде обрано значення із такого квадрату. Логічні вирази використовуються для обрахунку апаратної складності засобу.

Після виконання процесу вибору псевдовипадкової послідовності, в алгоритмі вона буде подана аналогічно повідомленню, у вигляді набору блоків із двох біт. Тому можна подати у вигляді послідовності:

$$S = (s1, s2, s3 \dots)$$

де s це набір із двох біт.

Процес зашифрування повідомлення полягає у виборі значення з таблиці Келі, що описує CIP-квазігрупу, на основі вхідних даних. При цьому ключ k вказує на квазігрупу яку необхідно використати, а m та s вказують на рядок і стовпець звідки обрати елемент.

Набір таблиць Келі для роботи алгоритму наведено на рисунку 2.3.

Q1:

*	0	1	2	3
0	0	3	1	2
1	2	1	3	0
2	3	0	2	1
3	1	2	0	3

Q2:

*	0	1	2	3
0	1	2	0	3
1	3	0	2	1
2	2	1	3	0
3	0	3	1	2

Q3:

*	0	1	2	3
0	2	1	3	0
1	0	3	1	2
2	1	2	0	3
3	3	0	2	1

Q4:

*	0	1	2	3
0	3	0	2	1
1	1	2	0	3
2	0	3	1	2
3	2	1	3	0

Рисунок 2.3 – Набір латинських квадратів

Такий процес можна описати виразом:

$$C_i = Q_i[s_i][m_i] \tag{2.10}$$

де C_i відповідає за пару зишифрованих біт, s_i пара біт з генератора псевдовипадкової послідованості, m_i біти вхідного повідомлення а $Q_i[s_i][m_i]$ значення яке було отримано з перехрестя рядка і стовпця.

Алгоритм вважається закінченим коли вхідне повідомлення було вичерпано, тоді на виході буде отримано послідовність:

$$C = (c_1, c_2, c_3 \ldots)$$

Процес розшифрування повідомлення може відбуватись на основі набору таблиць Келі які було використано для зашифрування або на основі набору оберенених квазігруп. Метод розшифрування з використанням квазігруп для зашифрування полягає у визначенні стовпця у якому знаходиться зашифроване значення. Якщо f буде функцією пошуку значення стовпця у таблиці Келі, то такий процес можна описати за формулою:

$$m_i = f(c_i, s_i, Q_i) \tag{2.11}$$

де m_i розшифрована пара біт повідомлення, c_i зашифрована пара біт повідомлення, Q_i таблиця Келі обрана на основі ключа, s_i пара біт з генератора псевдовипадкової послідованості.

У результаті таких дій буде отримано послідовність біт яка є вхідним повідомленням.

2.3 Метод потокового шифрування

Перед розробкою алгоритму потокового шифрування, необхідно описати словесно кроки такого процесу для обох режимів роботи.

Крок 1. Введення даних для роботи алгоритму. Необхідно надати секретний ключ *K* та вхідне повідомлення *M (C)* для роботи алгоритму. Алгоритм не буде виконуватись без введення вхідних даних.

Крок 2. Використовуючи дані введеного секретного ключа, а саме двох останніх біт, виконується процес генерації псевдовипадкової послідовності *S*. Після завершення генерації такої послідовності, вона буде зафіксована і використана у поточному сеансі роботи алгоритму.

Крок 3. Для подальшої роботи алгоритму обирається його режим роботи зашифрування або розшифрування повідомлення.

Крок 4. Далі відбуваєтсья передача всіх необхідних даних *M (C)*, *K*, *S* на операційний блок для подальшої обробки інформації. Операційний блок працює із блоками біт вхідного повідомлення на основі вхідних даних та режиму роботи.

Крок 5. Після обробки кожного із блоків біт вхідного повідомлення, починаючи з першого, буде виконуватись операція відновлення ключа на основі латинського квадрату. Оскільки ключ працює на основі регістру зсуву, тому використане значення буде відкинуто, а наступне значення ключа буде визначено із латинського квадрату на основі повідомлення і поточного значення ключа. Така операція виконується в алгоритмі не залежно від вибору режиму його роботи. Однак для режиму зашифрування у процес вибору нового значення ключа на основі латинського квадрату подається вхідне значення повідомлення m_i. Тоді як для режиму роботи розшифрування, у процес вибору нового значення ключа буде передаватись поточне розшифроване значення повідомлення. Такі процеси у алгоритмі виконує генератор гами.

Крок 6. Якщо повідомлення було вичерпано, то робота алгоритму вважається завершеною. У ситуація коли значення повідомлення не є вичерпаним, алгоритм переходить до кроку 4 і виконує свої дії.

2.4 Алгоритм потокового шифрування

Для реалізації програмного засобу необхідно описати алгоритм потокового шифрування на основі латинських квадратів, треба описати кроки виконання такого алгоритму для режиму зашифрування та розшифрування повідомлення.

Такий алгоритм працює із набором даних: відкритий текст *M* або шифротекст *C*, секретний ключ *K*, псевдовипадкова послідовність *S*.

Алгоритм процесу зашифрування вхідного повідомлення описується кроками:

– Введення вхідних даних *M* та ключа *K*.

– Генерування псевдовипадкової послідовності *S* на основі ключа *K*.

– Передавання повідомлення *M*, ключа *K*, та *S* на вхід операційного блоку.

– Виконання головної операції операційного блоку із квазігрупою Q_1.

– Запис результату операційного блоку в шифротекст *C*.

– Зсув послідовностей *M*, *K* та *S* на основі регістра зсуву.

– Генерування наступного значення ключа K на основі квазігрупи Q_0 та *M* і так до моменту вичерпання вхідного повідомлення.

Алгоритм процесу розшифрування вхідного повідомлення концептуально схожий та описується кроками:

– Введення вхідних даних *C* та ключа *K*.

– Генерування псевдовипадкової послідовності *S* на основі ключа *K*.

– Передавання повідомлення *C*, ключа *K*, та *S* на вхід операційного блоку.

– Виконання головної операції операційного блоку.

– Запис результату операційного блоку у відкритий текст *M*.

– Зсув послідовностей *C*, *K* та *S* на основі регістра зсуву.

– Генерування наступного значення ключа *K* на основі квазігрупи Q_0 та *M* і так до моменту вичерпання вхідного повідомлення.

Коли алгоритм роботи методу шифрування на основі латинських квадратів описано покроково, доцільно буде скласти схему роботи алгоритму ширфування.

Схема роботи алгоритму шифрування описана на рисунку 2.4.

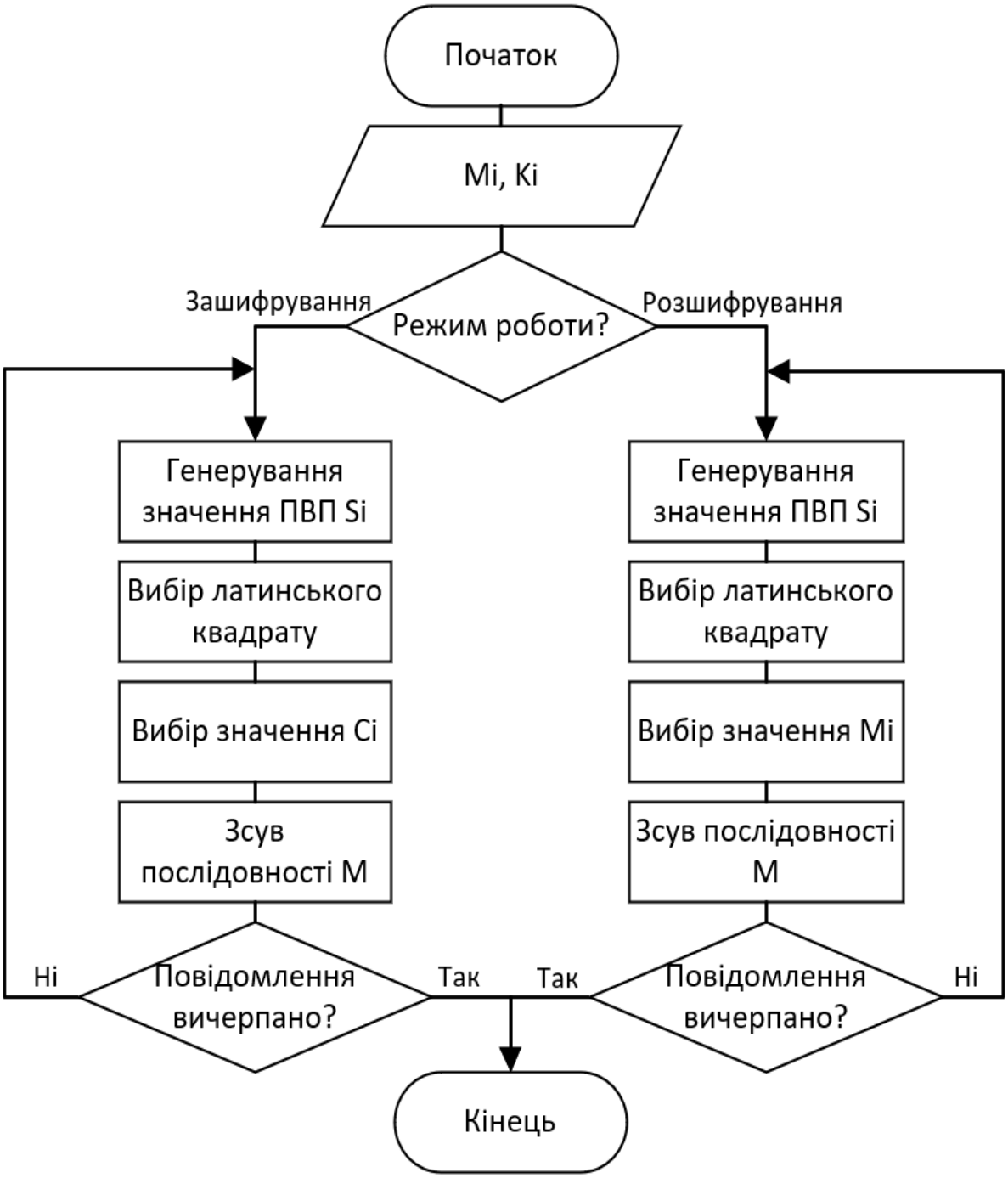


Рисунок 2.4 – Схема алгоритму шифрування

На основі введених даних, та вибору режиму роботи алгоритм виконує операції над вхідним текстом.

Використовуючи два останніх біти ключа, генератор псевдовипадкових послідовностей видає таку послідовність, яка складається із набору біт для подальшої роботи алгоритму.

Псевдовипадкова послідовність визначається як послідовність чисел, яка генерується за допомогою детермінованого алгоритму, але яка має властивості

випадковості. Тобто, хоча такі послідовності створюються на основі певного початкового значення, вони можуть виглядати випадковими, якщо алгоритм правильно спроектований. Ці послідовності широко використовуються в криптографії, моделюванні, чисельних методах та інших областях, де необхідна імітація випадковості.

Таким чином, псевдовипадкові послідовності поєднують детермінованість з властивостями випадковості, що робить їх корисними в криптографії [24].

Використовуючи такі дані, операційний блок виконує процес зашифрування або розшифрування інформації. Шляхом знаходження елементів на перехрестях рядка і стовпця у визначеному ключем латинському квадраті.

Після виконання поточної ітерації зашифрування або розшифрування, виконується зсув послідовностей *M*, *S* та *K* для подальшої роботи за допомогою регістру зсуву.

Після цього згідно рисунку 2.1 відбувається процес генерування значення гами із використанням вхідного значення ключа *K* і таблиці Келі Q_0 рис. 2.2. Такі дії підвищують стійкість ключа до розкриття, особливо атаки із відкритим текстом.

Алгоритм працює циклічно до моменту вичерпання повідомлення для роботи. Тоді алгоритм вважається виконаним.

2.5 Демонстрація роботи операційного блоку

Для кращого розуміння роботи та логіки алгоритму, доцільно розглянути фрагмент його роботи, а саме роботу операційного блоку.

Робота операційного блоку полягає у виборі елемента з одного із чотирьох латинських квадратів на основі вхідних даних. Для роботи операційного блоку необхідно: вхідне повідомлення, ключ, псевдовипадкова послідовність із генератора ПВП. Роботу операційного блоку буде розглянуто у режимах зашифрування та розшифрування інформації.

Приклади роботи буде розглянуто з точністю до перейменування, тобто значенням у четвірковій системі числення будуть відповідати значення двійкової системи числення: 0 – 00, 1 – 01, 2 – 10, 3 – 11.

Нехай для прикладу, на вхід операційного блоку буде подано послідовності:

– Вхідне повідомлення M (11, 10, 00, 01, 11 , 00).

– Ключ K (10, 11, 01, 00, 11 , 10).

– Послідовність генератора ПВП S (01, 00, 11, 10, 00, 11).

Для першої ітерації операційного блоку буде надіслано пари даних:

$$M_1 = (00_2) = 0_4,\ K_1 = (10_2) = 2_4,\ S_1 = (11_2) = 3_4$$

тоді K_1 вказує на те що буде використано таблицю Келі Q_3 яка наведена на рисунку 2.5.

*	0	1	2	3
0	2	1	3	0
1	0	3	1	2
2	1	2	0	3
3	3	0	2	1

Рисунок 2.5 – таблиця Келі Q_3

Результат зашифрування бітів буде на перетині рядка S_1 та стовпця M_1. Згідно введених значень, зашифровані біти c_1 зображено на рисунку 2.6.

*	0	1	2	3
0	2	1	3	0
1	0	3	1	2
2	1	2	0	3
3	③	0	2	1

Рисунок 2.6 – зашифровані біти c_1

Згідно виразу 2.10 зашифровані біти будуть $c_1 = Q_3[3][0] = 3_4 = 11_2$.

Для наступної ітерації операційного блоку буде надіслано пари даних:

$$M_2 = (11_2) = 3_4,\ K_2 = (11_2) = 3_4,\ S_2 = (00_2) = 0_4$$

тоді K_2 вказує на те що буде використано таблицю Келі Q_4 яка наведена на рисунку 2.7.

*	0	1	2	3
0	3	0	2	1
1	1	2	0	3
2	0	3	1	2
3	2	1	3	0

Рисунок 2.7 – таблиця Келі Q_4

Результат зашифрування бітів буде на перетині рядка S_2 та стовпця M_2. Згідно введених значень, зашифровані біти c_2 зображено на рисунку 2.8.

*	0	1	2	3
0	3	0	2	(1)
1	1	2	0	3
2	0	3	1	2
3	2	1	3	0

Рисунок 2.8 – зашифровані біти c_2

Згідно формули 2.10 зашифровані біти будуть $c_2 = Q_4[0][3] = 1_4 = 01_2$.

Для наступної ітерації операційного блоку буде надіслано пари даних:

$$M_3 = (01_2) = 1_4,\ K_3 = (00_2) = 0_4,\ S_3 = (10_2) = 2_4$$

тоді K_3 вказує на те що буде використано таблицю Келі Q_1 яка наведена на рисунку 2.9.

*	0	1	2	3
0	0	3	1	2
1	2	1	3	0
2	3	0	2	1
3	1	2	0	3

Рисунок 2.9 – таблиця Келі Q_1

Результат зашифрування бітів буде на перетині рядка S_3 та стовпця M_3. Згідно введених значень, зашифровані біти c_3 зображено на рисунку 2.10.

*	0	1	2	3
0	0	3	1	2
1	2	1	3	0
2	3	(0)	2	1
3	1	2	0	3

Рисунок 2.10 – зашифровані біти c_3

Згідно формули 2.10 зашифровані біти будуть $c_3 = Q_1[2][1] = 0_4 = 00_2$.

Для наступної ітерації операційного блоку буде надіслано пари даних:

$$M_4 = (00_2) = 0_4,\ K_4 = (01_2) = 1_4,\ S_4 = (11_2) = 3_4$$

тоді K_4 вказує на те що буде використано таблицю Келі Q_1 яка наведена на рисунку 2.11.

*	0	1	2	3
0	1	2	0	3
1	3	0	2	1
2	2	1	3	0
3	0	3	1	2

Рисунок 2.11 – таблиця Келі Q_2

Результат зашифрування бітів буде на перетині рядка S_4 та стовпця M_4. Згідно введених значень, зашифровані біти c_4 зображено на рисунку 2.12.

*	0	1	2	3
0	1	2	0	3
1	3	0	2	1
2	2	1	3	0
3	(0)	3	1	2

Рисунок 2.12 – зашифровані біти c_4

Згідно формули 2.10 зашифровані біти будуть $c_4 = Q_2[3][0] = 0_4 = 00_2$.

Для наступної ітерації операційного блоку буде надіслано пари даних:

$$M_5 = (10_2) = 2_4,\ K_5 = (11_2) = 3_4,\ S_5 = (00_2) = 0_4$$

тоді K_5 вказує на те що буде використано таблицю Келі Q_4 яка наведена на рисунку 2.7.

Результат зашифрування бітів буде на перетині рядка S_5 та стовпця M_5. Згідно виразу 2.10 зашифровані біти будуть $c_5 = Q_4[0][2] = 2_4 = 10_2$.

Для останньої ітерації операційного блоку буде надіслано пари даних:

$$M_6 = (11_2) = 3_4,\ K_6 = (10_2) = 2_4,\ S_6 = (01_2) = 1_4$$

тоді K_6 вказує на те що буде використано використано таблицю Келі Q_3 яка наведена на рисунку 2.5.

Результат зашифрування бітів буде на перетині рядка S_6 та стовпця M_6. Згідно виразу 2.10 зашифровані біти будуть $c_6 = Q_3[1][3] = 2_4 = 10_2$.

У результаті роботи операційного блоку було отримано зашифрований текст у вигляді послідовності С (10, 10, 00, 00, 01, 11).

Далі буде розглянуто процес розшифрування отриманої зашифрованої послідовності. Вхідні дані такі як ключ та послідовність генератора ПВП залишаються незмінними, а в якості вхідного повідомлення буде зашифрована послідовність.

Для першої ітерації операційного блоку буде надіслано пари даних:

$$M_1 = (10_2) = 2_4,\ K_1 = (10_2) = 2_4,\ S_1 = (11_2) = 3_4$$

тоді K_1 вказує на те що буде використано використано використано таблицю Келі Q_3 яка наведена на рисунку 2.5.

Результат розшифрування бітів буде номер стовпця у якому знаходиться зашифроване значення c_1. Згідно введених значень, розшифровані біти m_1 зображено на рисунку 2.13.

*	0	1	2	3
0	2	1	3	0
1	0	3	1	2
2	1	2	0	3
3	3	0	2	1

Рисунок 2.13 – розшифровані біти m_1

Згідно виразу 2.11 розшифровані біти будуть $m_1 = f(c_1, s_1, Q_3) = 0_4 = 00_2$.

Для наступної ітерації операційного блоку буде надіслано пари даних:

$$M_2 = (01_2) = 2_4,\ K_2 = (11_2) = 3_4,\ S_2 = (00_2) = 0_4$$

тоді K_2 вказує на те що буде використано таблицю Келі Q_4 яка наведена на рисунку 2.7.

Результат розшифрування бітів буде номер стовпця у якому знаходиться зашифроване значення c_2.

Згідно виразу 2.11 розшифровані біти будуть $m_2 = f(c_2, s_2, Q_4) = 2_4 = 11_2$.

Для наступної ітерації операційного блоку буде надіслано пари даних:

$$M_3 = (00_2) = 3_4,\ K_3 = (00_2) = 0_4,\ S_3 = (10_2) = 2_4$$

тоді K_3 вказує на те що буде використано таблицю Келі Q_1 яка наведена на рисунку 2.9.

Результат розшифрування бітів буде номер стовпця у якому знаходиться зашифроване значення c_3. Згідно введених значень, розшифровані біти m_3 зображено на рисунку 2.22.

Згідно виразу 2.11 розшифровані біти будуть $m_3 = f(c_3, s_3, Q_1) = 1_4 = 01_2$.

Для наступної ітерації операційного блоку буде надіслано пари даних:

$$M_4 = (00_2) = 0_4,\ K_4 = (01_2) = 1_4,\ S_4 = (11_2) = 3_4$$

тоді K_4 вказує на те що буде використано таблицю Келі Q_2 яка наведена на рисунку 2.11.

Результат розшифрування бітів буде номер стовпця у якому знаходиться зашифроване значення c_4.

Згідно виразу 2.11 розшифровані біти будуть $m_4 = f(c_4, s_4, Q_2) = 0_4 = 00_2$.

Для наступної ітерації операційного блоку буде надіслано пари даних:

$$M_5 = (10_2) = 2_4,\ K_5 = (11_2) = 3_4,\ S_5 = (00_2) = 0_4$$

тоді K_5 вказує на те що буде використано таблицю Келі Q_4 яка наведена на рисунку 2.7.

Результат розшифрування бітів буде номер стовпця у якому знаходиться зашифроване значення c_5.

Згідно виразу 2.11 розшифровані біти будуть $m_5 = f(c_5, s_5, L_4) = 2_4 = 10_2$.

Для останньої ітерації операційного блоку буде надіслано пари даних:

$$M_6 = (11_2) = 3_4,\ K_6 = (10_2) = 2_4,\ S_6 = (01_2) = 1_4$$

тоді K_6 вказує на те що буде використано таблицю Келі Q_3 яка наведена на рисунку 2.5.

Результат розшифрування бітів буде номер стовпця у якому знаходиться зашифроване значення c_6.

Згідно формули 2.11 розшифровані біти будуть $m_6 = f(c_6, s_6, L_3) = 3_4 = 11_2$.

У результаті роботи операційного блоку у режимі розшифрування було отримано текст у вигляді послідовності M (11, 10, 00, 01, 11 , 00).

2.6 Висновки до розділу

Під час дослідження основних характеристик латинських квадратів, їх використання в криптографії та принципів побудови відповідних алгоритмів, було визначено доцільність використання таких структур для розробки алгоритму потокового шифрування.

Після визначення доцільності роботи з такими структурами, було описано теоретичну частину роботи алгоритму та його особливості. Що підкріплені використанням генератора псевдовипадкових чисел, розробленого у першій частині роботи, для збільшення стійкості отриманого значення.

Розроблено метод шифрування який описує кроки виконання процесу шифрування використовуючи латинські квадрати. Розробка методу такого шифрування допомогла зрозуміти деталі які необхідно враховувати при розробці такого алгоритму. Особливо розробка генератора гами. Оскільки це є однією із важливих частин стійкості алгоритму.

Наступним кроком було розроблено алгоритм потокового шифрування на основі латинських квадратів. У якому покроково описано дії, які повинні бути виконані для досягнення результату. Окремо розписано процес зашифрування та розшифрування повідомлення таким методом.

Коли алгоритм було розроблено, виконано моделювання його роботи шляхом демонстрації процесу зашифрування та розшифрування на тестовому наборі даних. У результаті такої демонстрації, вхідне повідомлення було зашифровано та успішно розшифровано із використанням такого алгоритму.

3 РОЗРОБКА ТА ОЦІНЮВАННЯ СКЛАДНОСТІ ЗАСОБУ ПОТОКОВОГО ШИФРУВАННЯ

Третій розділ присвячений розробці апаратного засобу який реалізує метод потокового шифрування на основі латинських квадратів 4-го порядку. Опису структури блоків такого засобу, оцінювання його складності та порівняння з аналогами.

3.1 Модель програмного засобу шифрування

Після розробки методу та алгоритму потокового шифрування на основі латинських квадратів 4-го порядку, необхідно розробити програмний засіб для демонстрації працездатності такого алгоритму шифрування.

Програмний засіб реалізовує основний функціонал такого алгоритму, а саме зашифрування та розшифрування вхідного повідомлення.

Для досягнення мети такого функціоналу, реалізовано функції перетворення вхідного повідомлення у потік біт для подальшої роботи. Операції зсуву бінарних послідовностей за аналогією роботи регістру зсуву. Функції вибору необхідного для роботи латинського квадрату та вибору необхідного значення із такого квадрату. Однією із головних функцій буде реалізація процесу генерування значення гами на основі вхідного повідомлення, ключа K та значень таблиці Келі Q_0, яка зображена на рисунку 2.2.

Для реалізації такої моделі, було розроблено структурну схему роботи програмного засобу потокового шифрування на основі латинських квадратів, яка зображена на рисунку 3.1.

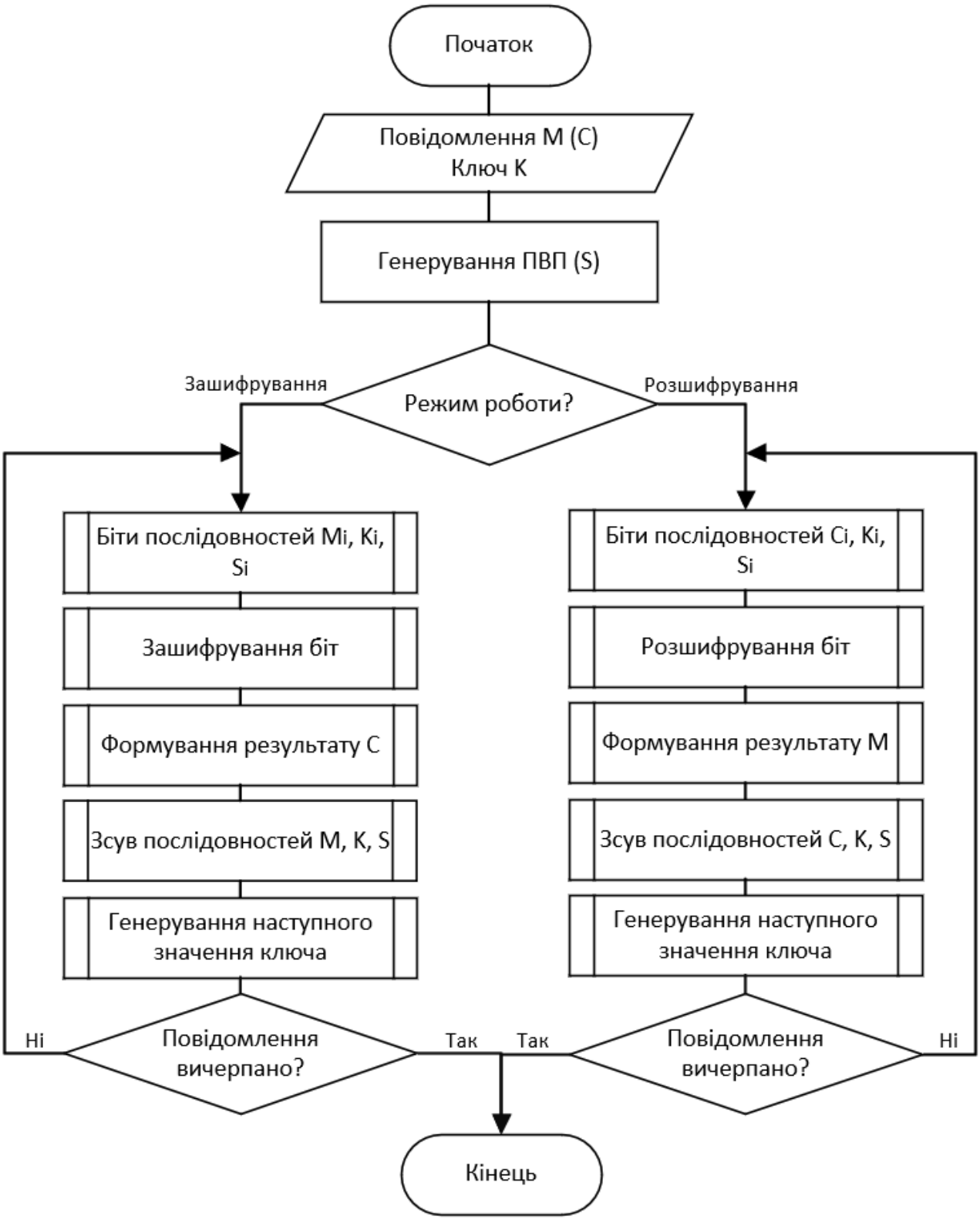


Рисунок 3.1 – Структурна схема роботи програмного засобу потокового шифрування на основі латинських квадратів

На основі такої структурної схеми було реалізовано програмний засіб, який є моделлю для демонстрації працездатності такого алгоритму шифрування.

Вхідними даними від користувача вважається повідомлення для зашифрування або розшифрування та секретний ключ.

Для демонстрації працездатності моделі яка відображає алгоритм роботи потокового шифрування на основі латинських квадратів 4-го порядку, було проведено тестування такого програмного засобу для різних повідомлень, що показано на рисунках 3.2 та 3.3, а код програмного засобу наведено у додатку Б.

Тестування роботи алгоритму для короткого повідомлення українського алфавіту наведено на рисунку 3.2.

```
Input string for encription:
Привіт світ!
Encrypted message:
▯▯Dr◆H◆◆54◆▯3◆◆◆◆▯▯◆▯◆◆
Decrypted message:
Привіт світ!
```

Рисунок 3.2 – Результат шифрування повідомлення українського алфавіту

Тестування роботи алгоритму для повідомлення англійського алфавіту наведено на рисунку 3.3

```
Input string for encription:
Petals cradled moonlight's hushed confessions.
Encrypted message:
▯◆▯◆◆▯&◆/Z}C◆N◆◆T▯◆◆G3◆◆◆r◆{H◆▯&◆◆^◆V◆◆F◆◆L◆◆◆
Decrypted message:
Petals cradled moonlight's hushed confessions.
```

Рисунок 3.3 – Результат шифрування повідомлення англійського алфавіту

Після проведених тестувань, програмний засіб розроблений на основі спроектованого алгоритму потокового шифрування на основі латинських квадратів 4-го порядку показав працездатність такого алгоритму. Функції зашифрування та розшифрування працюють правильно, що підтверджує працездатність такого алгоритму шифрування.

3.2 Структура апаратного засобу шифрування

Після розробки програмного засобу та перевірки його функціоналу, що підтвердило працездатність розробленого алгоритму, відкривається можливість розробки апаратного засобу потокового шифрування на основі латинських квадратів 4-го порядку.

Для реалізації апаратного засобу, необхідно розробити його структуру, за допомогою якої буде досягатись головна мета алгоритму.

Ключовими блоками апаратного засобу є: генератор гами, блок керування, генератор псевдовипадкової послідовності, операційний блок.

Генератор гами відображає схему на рис. 2.1 і виконує генерування псевдовипадкової послідовності ключа на основі таблиці Келі Q_0 що наведена на рис. 2.2 та вхідного повідомлення. Таким чином досягається стійкість ключа для роботи алгоритму. Оскільки такий підхід використовує процес перенесення повідомлення у якості нового ключа для шифрування, то стійкість такого ключа зростає. Але з’являється проблема атаки із відкритим вхідним текстом. Для обробки такої проблеми використано таблицю Келі Q_0, яка за допомогою своїх операцій розбавляє ключ значеннями, що зменшує ризик такої атаки на ключ.

Блок керування обирає режим роботи програмного засобу. Після сигналу такого блоку, операційний блок буде мати розуміння який процес, зашифрування чи розшифрування, виконувати з вхідним повідомленням.

Генератор псевдовипадкової послідовності виконує процес генерування послідовності яка вважається псевдовипадковою і має максимальну довжину для латинських квадратів 4-го порядку. Оскільки послідовність генерується за допомогою математичних формул, назвати її повністю випадковою не є можливим. Такий генератор на вихід подає біти послідовності на основі двох останніх біт введеного ключа. Довжина такої послідовності є максимальною для латинських квадратів 4-го порядку.

Операційний блок обробляє основну логіку засобу, а саме зашифрування або розшифрування повідомлення в залежності від сигналу блоку керування.

Процес шифрування у такому блоці не буде виконуватись, якщо не надано всі необхідні вхідні дані для роботи. Такий блок використовує набір із чотирьох латинських квадратів 4-го порядку для виконання операції зашифрування. Процес зашифрування біт описано за допомогою формули 2.10. Для виконання процесу розшифрування у апаратній реалізації засобу буде використано набір із чотирьох латинських квадратів які є оберненими відносно тих які були використані під час зашифрування. А процес розшифрування буде за логікою схожий на процес зашифрування повідомлення, тільки на іншому наборі латинських квадратів. Такий підхід дозволить покращити ефективність апаратної реалізації. Після зашифрування або розшифрування біт вони формують результуючу послідовність, яка подається як результат роботи операційного блоку і апаратного засобу в загальному.

Для правильності роботи засобу необхідна чітка та злагоджена робота кожного із блоків. Оскілки процес шифрування даних є досить чутливим спотворення даних, то колізії інформації можуть виникнути у будь-якій точці де буде виконуватись неправильна обробка біт. Тому важливо забезпечити правильність виконання кожного блоку та їх злагоджену роботу.

Таким чином було описано структуру апаратного засобу потокового шифрування на основі латинських квадратів 4-го порядку, яка допоможе побудувати блоки такого апаратного засобу для правильного шифрування вхідних повідомлень.

Після опису кожного із блоків доцільно буде відобразити структуру такого засобу. Структура засобу наведена на рисунку 3.4.

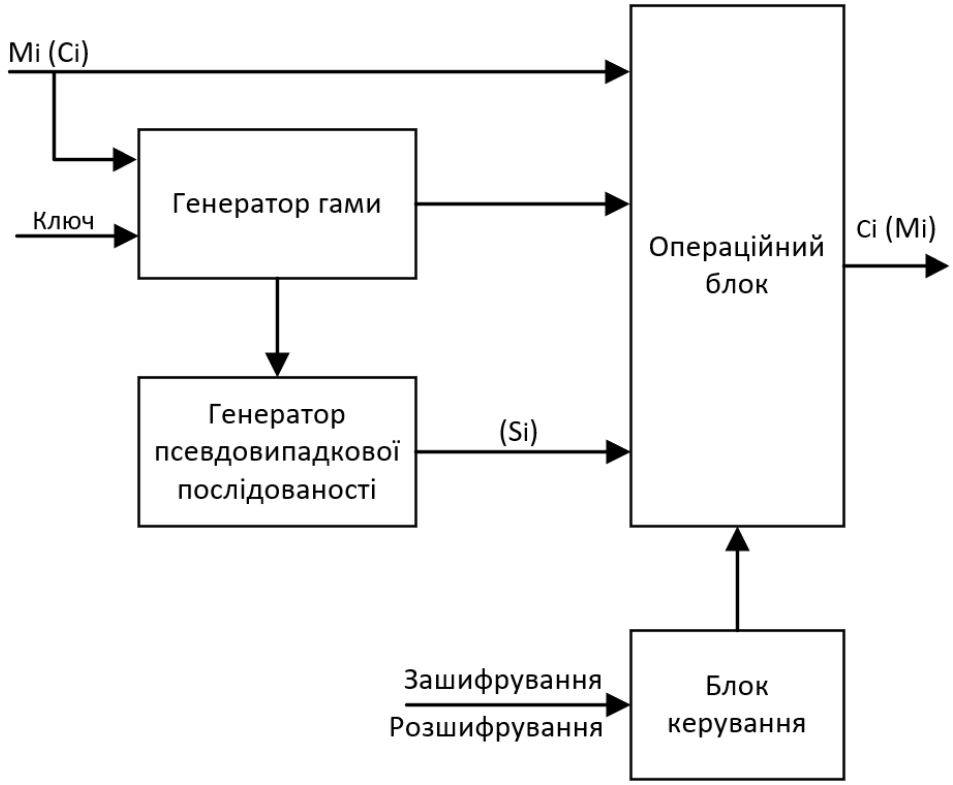


Рисунок 3.4 – Структура засобу шифрування

Така візуалізація зображає процес роботи засобу шифрування. На рисунку наведено всі необхідні для роботи засобу блоки та взаємодію між ними.

3.3 Оцінювання складності апаратного засобу

Визначення складності апаратного засобу є важливим критерієм для оцінки алгоритму в загальному. Оскільки алгоритм призначений для використання у сфері малоресурсної криптографії, тому показники складності не мають перевищувати необхідних норм, а краще бути якомога меншими.

Оцінка складності апаратного засобу відбувається на основі таблиці 3.1.

Таблиця 3.1 – показники складності основних апаратних елементів

Елемент	GE(Gate Equivalent)
НІ	0,67
І	1,33
АБО	1,33
Виключне АБО	2,67
І-НІ	1
АБО-НІ	1
Мультиплексор	2,33
Тригер	5,33

На основі таких показників оцінено складність апаратного засобу.

3.3.1 Генератор гами

Генератор гами визначає біти гами на основі біт ключа та біт вхідного повідомлення. Такий генератор використовує для роботи регістр зсуву та квазігрупу Q_0. Схему такого генератора гами наведено на рисунку 2.1.

Також для генерування значень використовується таблиця Келі Q_0, яка наведена на рисунку 2.2.

Для оцінки складності такого генератора, необхідно обрахувати складність реалізації регістру зсуву та складність латинського квадрату.

Регістр зсуву складається з 64 тригерів типу D. Тригер типу D має складність 5.33 умовних одиниць GE. Тоді загальна складність регістру зсуву буде рівна 341.12 GE.

Таблиця Келі Q_0 описує квазігрупу, що реалізується двома логічними виразами Q_0d_1 та Q_0d_2. Такі логічні вирази розраховані у першій частині роботи і мають вигляд:

$$Q_0\, d_1 = (\overline{x_1}\,\overline{x_2} + \overline{x_1}\,x_2)(y_1\overline{y_2} + y_1y_2)(x_1\overline{x_2} + x_1x_2)(\overline{y_1}\,\overline{y_2} + \overline{y_1}\,y_2)$$

$$Q_0\, d_2 = (\overline{x_1}\,x_2 + x_1\,x_2)(y_1\overline{y_2} + \overline{y_1}\,\overline{y_2})(x_1\overline{x_2} + \overline{x_1}\,\overline{x_2})(\overline{y_1}\,y_2 + y_1\,y_2)$$

Для апаратної реалізації їх необхідно скоротити, з метою зменшення та оптимізації ресурсів.

Логічні вирази після скорочення мають вигляд:

$$Q_0\, d_1 = \overline{x_1}\,y_1 + x_1\overline{y_1}$$

$$Q_0\, d_2 = x_2\overline{y_2} + \overline{x_2}\,y_2$$

Після скорочення логічні вирази стали набагато компактнішими, але ще є можливість їх скоротити, для зменшення апаратних витрат. Результуючі логічні вирази мають вигляд:

$$Q_0\, d_1 = x_1 \oplus y_1$$

$$Q_0\, d_2 = x_2 \oplus y_2$$

Такий латинський квадрат який описує таблиця келі Q_0 є тотально-симетричним, що робить його логічне рівняння досить компактним і не потребує багато ресурсів для реалізації.

Виходячи з цього, для реалізації латинського квадрату який подано таблицею Келі Q_0 потрібно два елементи виключного АБО.

Тоді складність такого квадрату згідно таблиці 3.1 буде рівна 5.34 GE. Отже складність генератору гами становить 346.46 GE.

3.3.2 Блок керування та генератор псевдовипадкової послідованості

Для визначення режиму роботи операційного блоку використовується блок керування. Такий блок подає на вхід операційного блоку біти керуючого сигналу, що відповідає за режим роботи зашифрування чи розшифрування біт. Такий блок буде складатись лише з елементу “НІ” для вибору режиму роботи і генерувати керуючий сигнал для операційного блоку. Тому складність такого блоку згідно таблиці 3.1 можна вважати 0.67 GE.

Генератор псевдовипадкової послідованості було розроблено у першій частині роботи. Такий генератор є обов’язковою частиною алгоритму, забезпечуючи набір псевдовипадкових значень, на основі яких відбувається процес шифрування повідомлення. Генератор використовує пару латинських квадратів для генерування послідовності значень використовуючи формулу рекурсії [20]. Принцип роботи генератора ПВП показано на рисунку 3.5.

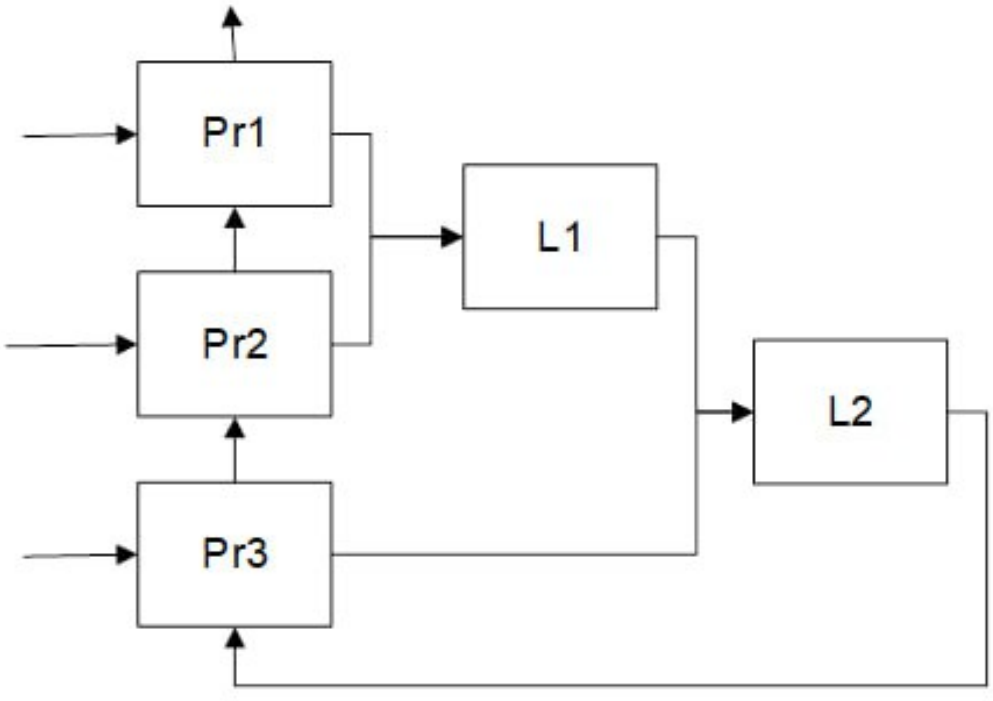


Рисунок 3.5 – Схема генератора ПВП

У такій схемі продемонстровано принцип отримання псевдовипадкової послідовності використовуючи пару латинських квадратів L_1, L_2 та регістри Pr_1, Pr_2, Pr_3.

Дослідження та детальний принцип роботи такого генератора ПВП описано у першій частині роботи. Оскільки такий генератор використовує латинські квадрати та деякі логічні елементи для реалізації свого функціоналу, його складність буде описуватись складністю використаних латинській квадратів та набору логічних елементів для реалізації. Апаратна складність такого блоку обраховано у першій частині роботи, вона становить 71.65 GE. Тоді апаратна складність блоку керування та ГПВП складає 72.32 GE.

3.3.3 Операційний блок

Операційний блок отримує на вхід біти повідомлення $M_i(C_i)$, біти генератора гами K_i, біти генератора псевдовипадкової послідовності S_i та сигнал керування для зашифрування або розшифрування повідомлення.

Для мінімізації апаратних витрат, для виконання розшифрування вхідних біт зашифрованого повідомлення буде використано обернені латинські квадрати, відносно тих які були використані під час шифрування. Таким чином досягається спрощення процесу розшифрування, шляхом простого вибору значення із оберненого латинського квадрату [23]. Такий підхід не впливає на логіку та працездатність алгоритму, але дозволяє значно зменшити його апаратну складність. Набір обернених латинських квадратів у вигляді таблиць Келі наведено на рисунку 3.6.

Q5:

*	0	1	2	3
0	0	2	3	1
1	3	1	0	2
2	1	3	2	0
3	2	0	1	3

Q6:

*	0	1	2	3
0	1	3	2	0
1	2	0	1	3
2	0	2	3	1
3	3	1	0	2

Q7:

*	0	1	2	3
0	2	0	1	3
1	1	3	2	0
2	3	1	0	2
3	0	2	3	1

Q8:

*	0	1	2	3
0	3	1	0	2
1	0	2	3	1
2	2	0	1	3
3	1	3	2	0

Рисунок 3.6 – Набір обернених таблиць Келі

Для кращого розуміння процесу роботи операційного блоку та його основної логіки було створено відповідну схему роботи. Схема операційного блоку наведена на рисунку 3.7.

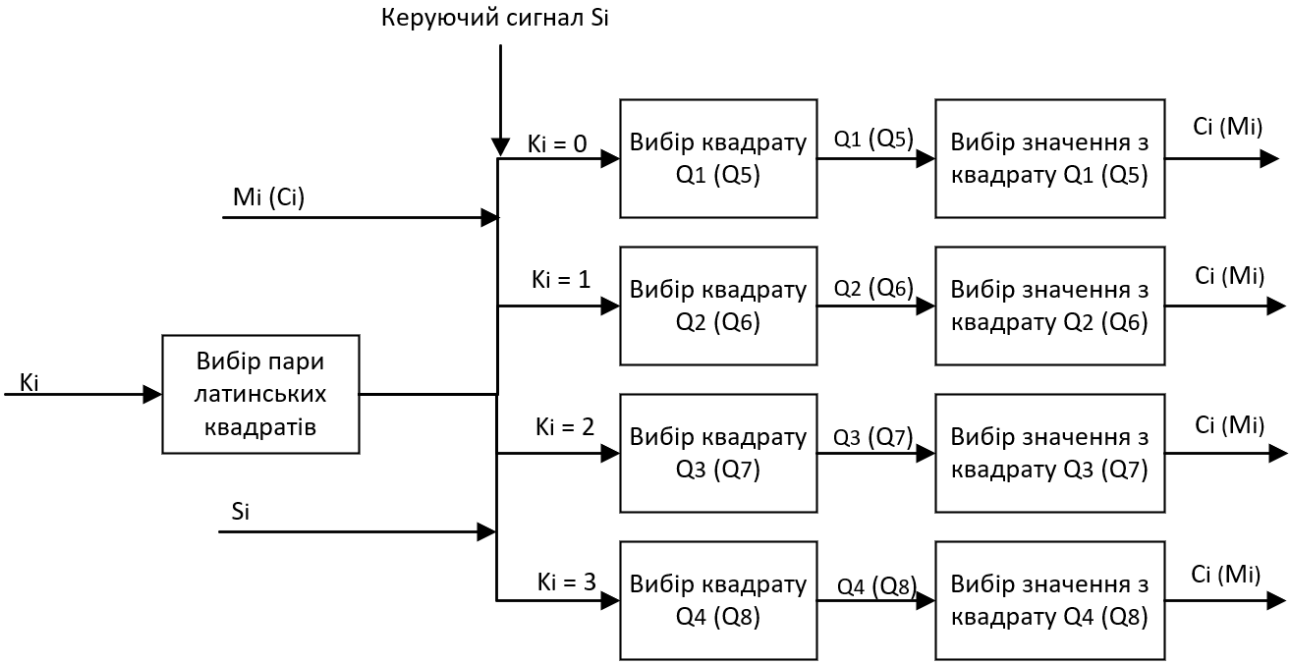


Рисунок 3.7 – Схема операційного блоку

Де M_i (C_i) – вхідне зашифроване або розшифроване повідомлення, S_i керуючий сигнал який вказує на режим зашифрування або розифрування, K_i біти генератора гами.

Така апаратна реалізація описується елементами: 1 елемент “НІ”, 16 елементів “І” та 8 елементів “АБО”. Згідно даних таблиці 3.1 складність такої частини операційного блоку рівна 32.59 GE.

Операційний блок використовує логічні вирази для отримання значення з латинських квадратів. Такі логічні вирази описують два набори по 4 латинських квадрати для зашифрування та розшифрування.

Набір латинських квадратів для зашифрування поданих у вигляді таблиць Келі, зображений на рисунку 2.3. Такий набір квадратів описується наступними логічними виразами:

$$Q_1\ d_1 = Q_2\ d_1 = y_2\overline{(x_1 \oplus x_2)} + \overline{y}_2(x_1 \oplus x_2)$$

$$Q_1\ d_2 = Q_4\ d_2 = y_2\overline{(x_1 \oplus y_2)} + \overline{y}_2(x_1 \oplus y_2)$$

$$Q_2\ d_2 = Q_4\ d_2 = y_2(x_1 \oplus y_1) + \overline{y}_2\overline{(x_1 \oplus y_1)}$$

$$Q_3\ d_1 = Q_4\ d_1 = x_2(x_1 \oplus y_2) + \overline{x}_2\overline{(x_1 \oplus y_2)}$$

Для реалізації функції розшифрування, операційний блок використовує набір з обернених латинських квадратів відносно тих які були для зашифрування. Такі латинські квадрати описуються таким набором логічних виразів:

$$Q_5\, d_1 = Q_8\, d_1 = x_2\overline{(y_1 \oplus y_2)} + \overline{x}_2(y_1 \oplus y_2)$$

$$Q_8\, d_2 = Q_8\, d_2 = x_1(x_2 \oplus y_1) + \overline{x}_1\overline{(x_2 \oplus y_1)}$$

$$Q_6\, d_1 = Q_7\, d_1 = x_2(y_1 \oplus y_2) + \overline{x}_2\overline{(y_1 \oplus y_2)}$$

$$Q_5\, d_2 = Q_6\, d_2 = x_2\overline{(x_1 \oplus y_1)} + \overline{x}_2(x_1 \oplus y_1)$$

Де d_1 та d_2 є вхідними сигналами. Таким чином на основі вхідного сигналу про режим роботи, буде обрано набір латинських квадратів для роботи, потім виходячи із вхідних сигналів буде вибрано значення із обраного квадрату, що буде зашифрованим або розшифрованим значенням.

Для реалізації одного логічного виразу використовуються: операція “АБО”, дві операції “І”, дві операції “Виключне АБО” та дві операції “НІ”. Складність такого виразу буде рівна 10.67 GE. Оскільки вирази були максимально оптимізовані і є однакові за складністю, то можна обрахувати складність для всіх восьми логічних виразів, їх складність буде рівною 85.36 GE.

Для формування результату роботи використано: шість операцій “АБО” та вісім операцій “НІ”. Тоді складність такої частини буде рівна 13.34 GE.

Отримавши результати оцінки всіх частин операційного блоку можна обчислити загальну складність такого блоку. Складність апаратної реалізації операційного блоку буде рівна 131.29 GE.

3.3.4 Загальна складність апаратної реалізації

Оцінка апаратного засобу проведена на основі даних таблиці 3.1, яка дає можливість оцінити апаратну складність засобу шифрування, шляхом підрахунку логічних елементів які було використано у апаратній реалізації та виведення їхньої апаратної складності в умовних одиницях обрахунку GE (Gate Equivalent).

Після оцінки кожного компонента апаратного засобу, є можливість зробити загальну оцінку складності такої реалізації. Складність блоків апаратного засобу:

– генератор гами має складність 362.36 GE;
– блок керування та генератор ПВП мають складність 72.32 GE;
– складність операційного блоку 131.29 GE.

Тоді складність такого апартного засобу буде рівна 565.97 GE, що не є ведилим значенням і задовольняє умови малоресурсної криптографії.

Складність кожного блоку апаратної реалізації засобу шифрування наведено у таблиці 3.2.

Таблиця 3.2 – Складність блоків засобу шифрування

Блок	Апаратна складність (GE)
Генератор гами	362.36
Блок керування	0.67
Генератор псевдовипадкової послідовності	72.32
Операційний блок	131.29
Всього	565.97

При цьому є можливісь зменшити апаратну складність такого засобу, для деяких систем, шляхом зменшення регістра зсуву генератора гами. Якщо регістр зсуву буде складатись із 32 тригерів, то складність генератора гами буде 191.8 GE.

3.4 Порівняння апаратного засобу

Після розробки алгоритму та розрахунку складності його апаратної реалізації необхідно зробити порівняльну оцінку такого алгоритму потокового шифрування з іншими існуючими алгоритмами.

Grain хоч і є досить ефективним представником потокового шифрування для систем з обмеженими ресурсами, але через використання 160 тригерів у регістрах зсуву та інші особливості його апаратної реалізації має досить велику

оцінку апаратної складності засобу. Загалом алгоритм має оцінку у 1294 GE, що прийнятно для використання у сфері LW-криптографії. Але запропонований алгоритм має складінсть 565.97 GE що менше за наведений шифр на 728.03 GE, що демонструє ефективність такої розробки.

Наступний представник шифрів для систем з обмеженими ресурсами це MICKEY 2.0. Його апаратна реалізація вимагає використання двох регістрів зсуву по 100 тригерів кожен. Такий підхід примо впливає на його апаратну складність яка обраховується у 1846 GE, що є досить великим значенням для малоресурсних пристроїв. Станом на сьогодні не було знайдено ефективної методики для зменшення його апаратної реалізації, тому значення його складності не є змінним. Тому можна вважати що розроблений алгоритм є ефективнішим у плані апаратної реалізації, і різниця оцінок їх складності становить 1280.03 GE.

Алгоритм Trivium є надійним методом потокового шифрування, який за рахунок своєї можливості паралельних обчислень, може протистояти багатьом криптоаналітичним атакам. Але при цьому через його особливості роботи та апаратної реалізації, його складність у найменш затратному режимі оцінюється у 2599 GE. Виходячи з цього він не можу бути використаний у пристроях які мають обмежені ресурси. Порівнюючи його із розробленим під час роботи методом, то складність розробленого методу приблизно у п’ять разів менше ніж представлений екземпляр.

Enocoro-128 працює з ключем довжиною 128 біт, та містить два буфери по 32 біти з регістрами зсуву. Такий підхід робить його дещо схожим за швидкодією за алгоритм Trivium. Тому його апаратна складність становить 2700 GE. Такий показник є мінімальним якщо алгоритм буде працювати з ключем 80 біт. У іншому випадку апаратна складність алгоритму стрімко зростає. Порівнюючи такий алгоритм із розробленим, різниця апаратної складності складає 2134.03 GE одиниць.

Ще один представник потокових шифрів є F-FCSR-H. Він використовує регістр зсуву з лінійним зворотнім зв’язком і має довжину регістра 160 біт.

Складність його реалізації обраховується у 4800 GE, більш чим на 4200 GE більше ніж запропонований метод потокового шифрування.

Один із найкращих потокових шифрів це Salsa20. Оскільки алгоритм був розроблений виключно для програмної реалізації, то складно оцінити його апаратну складність. Але за приблизними оцінками його апаратна складність становить 3842 GE. Це на 3276.03 GE більше ніж запропонований алгоритм.

LIZARD реалізовано на основі двох регістрів з нелінійним зворотнім зв'язком. Такий підхід перегукується із реалізацією Grain але в наслідок зменшення довжини регістрів складність апаратної реалізації LIZARD становить 1161 GE. Такий показник є досить хорошим із точки зору систем з обмеженими ресурсами. У порівнянні з розробленим у роботі алгоритмом, складність апаратної реалізації LIZARD більша за розроблений алгоритм на 595.03 GE.

Алгоритм Fruit-80 також використовує два регістри з нелінійним зворотнім. Але їх довжина є меншою у порівнянні з Grain та LIZARD. Виходячи з цього апаратна реалізація Fruit-80 складає 960 GE. Хоч такий алгоритм і має свої особливості, складність його реалізації дозволяє використовувати його у малоресурсних системах. Порівнюючи його із розроблений протягом роботи алгоритмом, складність Fruit-80 на 394.03 GE більше ніж розроблений алгоритм.

Planet є дещо схожим на попередній алгоритм. Він використовує регістр зсуву з лінійним зворотнім зв'язком і його складність апаратної реалізації становить 928 GE. Такі показники задовольняють вимоги LW-криптографії, але така складність є на 362.03 GE більша ніж у описаного у роботі алгоритму.

Ще одним представником алгоритмів для систем з обмеженими ресурсами є LILLE. Його головною метою була низька апаратна складність, що досягалось ефективністю виконання процесів під час роботи шифру. Його апаратна складність 911 GE, що на 345.03 GE більше ніж представлений у роботі алгоритм.

Одними із головних аналогів до представленого у роботі алгоритму є шифри Edon-80 та Edon-R. У їх реалізації є деякі відмінності, що відображено у маркуванні назви алгоритму. Edon-80 є компактнішою версією, яка використовує ключ довжиною 80 біт, а Edon-R використовує ключ 128 біт. Ці два алгоритми

при роботі використовують квазігрупи як одна із основ своєї логіки [25]. Складність алгоритму Edon-80 становить близько 831 GE. Edon-R має дещо інший функціонал і тому складність його апаратної реалізації становить близько 1198 GE [26]. У порівнянні з розробленим під час роботи алгоритмом, Edon-80 виявився найбільш схожим за оцінкою. Апаратна реалізація алгоритму Edon-80 є на 265.03 GE більшою за розроблений алгоритм. Edon-R вимагає ще більше ресурсів, тому його реалізація на 632.03 GE більша ніж пропонований у роботі алгоритм.

Студентами ВНТУ був розроблений алгоритм “LKRP” заснований на CIP-квазігрупах, який має ключ 64 біти та 32 такти ініціалізації [23]. Алгоритм через особливості своєї реалізації має складність 482.39 GE, що менше за розроблений алгоритм на 83.53 GE.

Після детальної оцінки всіх алгоритмів доцільно буде скласти таблицю 3.3 порівняльних характеристик, для демонстрації результатів роботи.

Назва шифру	Довжина ключа (біт)	Складність (GE)
Розроблений (LSMZ)	64	565.97
LKRP	64	482.39
Edon-80	80	831
LILLE	80	911
Planet	80	928
Fruit-80	80	960
LIZARD	120	1161
Edon-R	256	1198
Grain	80	1294
MICKEY 2.0	80	1846
Trivium	80	2599
Enocoro-128	128	2700
Salsa20	256	3842
F-FCSR-H	80	4800

Після проведеного аналізу, визначено, що запропонований алгоритм потокового шифрування заснований на латинських квадратах 4-го порядку є ефективним і при тому має низьку апаратну складність реалізації. Таким чином він може бути використаний у системах з обмеженими ресурсами.

3.5 Висновки до розділу

На основі розробленого алгоритму та методу потокового шифрування на основі латинських квадратів 4-го порядку, було реалізовано модель програмного засобу для перевірки працездатності алгоритму.

Після реалізації програмного засобу такого шифрування та перевірки правильності його роботи, було розроблено структуру апаратного засобу шифрування. Визначено всі необхідні блоки для правильної роботи алгоритму, та взаємодію таких блоків між собою.

Таким чином апаратний засіб складається з чотирьох блоків, робота яких є злагодженою, запобігаючи спотворенню інформації.

Перелік блоків апаратного засобу шифрування:

- генератор гами;
- блок керування;
- генератор псевдовипадкової послідовності;
- операційний блок.

Після реалізації такого засобу, було проведено підрахунок його апаратної складності. Для отримання результату складності засобу було проведено аналіз складності логічних елементів які потрібні для реалізації такого засобу.

Після проведення всіх обрахунків, апаратна складність апаратного засобу була визначена як 565.78 GE. Такий результат є хорошим із точки зору області малоресурсної криптографії, та відповідає її вимогам.

Проведено порівняння отриманого апаратного засобу з відомими аналогами. Розроблений алгоритм виявився набагато простішим із точки зору апаратної складності. Оцінка складності його апаратної реалізації виявилась набагато меншою ніж у розглянутих аналогів.

ВИСНОВКИ

Аналіз рішень та стану області потокового шифрування, показав необхідність створення нових алгоритмів шифрування із покращенням показників ефективності та крипостійкості.

Під час аналізу інформаційних джерел було проведено комплексний аналіз предметної області дослідження та сучасних викликів і потреб у галузі потокового шифрування, що дозволило визначити перспективні напрями для подальших досліджень і розробок.

Результати аналізу показали необхідність такого дослідження та розробки ефективних методів шифрування для захисту конфіденційної інформації у зв'язку зі зростаючими обсягами даних та розвитком технологій. Надійні алгоритми потокового шифрування є необхідними для забезпечення безпеки у цифровому світі.

Проаналізовані інформаційні ресурси продемонстрували, що поєднання математики, криптографії та інформаційної безпеки є продуктивним підходом для розробки нових ідей та інноваційних рішень в області захисту інформації.

Латинські квадрати, як математичний об’єкт, завдяки своїм властивостям, пропонують перспективні рішення для задоволення таких вимог, забезпечуючи високу ентропію та стійкість до криптоаналітичних атак. Такі властивості дозволяють використовувати такі математичні об’єкти для реалізації нових алгоритмів потокового шифрування.

Використовуючи отриману інформацію, було розроблено алгоритм та метод потокового шифрування на основі латинських квадратів 4-го порядку. Описано основні кроки виконання такого алгоритму, для виконання шифрування вхідних повідомлень таким методом.

На основі розробленого методу, реалізовано модель програмного засобу потокового шифрування. Така модель продемонструвала працездатність розробленого методу, шляхом зашифрування та розшифрування вхідних

повідомлень. Таким чином доведено коректність та можливість апаратної реалізації алгоритму.

На основі реалізованого та протестованого алгоритму було розроблено структуру апаратного засобу потокового шифрування. Структура апартного засобу складається з компонентів: генератор гами, блок керування, генератор псевдовипадкової послідовності, операційний блок.

Розроблений генератор гами використовує обрану тотальносиметричну квазігрупу Q_0, вхідне повідомлення та секретний ключ для генерування значень які необхідні для роботи операційного блоку. Такий підхід дозволив збільшити стійкість ключа до різного роду атак. Враховуючи всі компоненти які необхідно для апаратної реалізації, апаратна складність такого генератора гами становить 362.36 GE.

Блок керування виконує вибір операції яка буде використовуватись в операційному блоці засобу, зашифрування або розшифрування біт повідомлення. Апаратна складність такого блоку становить 0.67 GE.

Генератор псевдовипадкової послідовності було розроблено у першій частині роботи. Його завдання забезпечити генерування псевдовипадкових значень на основі латинських квадратів, використовуючи значення біт секретного ключа. Апаратна складність такого генератора становить 71.65 GE.

Операційний блок виконує головну операцію алгоритму, зашфрування або розшифрування на основі вхідних даних із використанням наборів квазігруп. З урахуванням всіх компонентів такого блоку, апаратна складість такого блоку становить 131.29 GE.

Складність апаратного засобу цілком становить 565.78 GE, що у більшій чи меншій мірі менше за розглянуті аналоги такої сфери шифрування.

Таким чином було виконано всі поставлені завдання і досягнуто головну мету дослідження, яка формується як зменшення апаратної складності засобу потокового шифрування на основі латинських квадратів 4-го порядку.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Rueppel R. A. Stream Ciphers. Analysis and Design of Stream Ciphers. Berlin, Heidelberg, 1986. pp. 5–16. DOI: 10.1007/978-3-642-82865-2_2 (date of access: 21.05.2024).

2. Canteaut A. Stream Ciphers. Encyclopedia of Cryptography, *Security and Privacy*. Berlin, Heidelberg, 2023. pp. 1–3. DOI: 10.1007/978-3-642-27739-9_374-2 (date of access: 18.05.2024).

3. Ashwini P., Venkaiah V. C., Bhukya W. N. Secret sharing scheme based on Latin squares. *Journal of Discrete Mathematical Sciences and Cryptography*. 2021. pp. 1–15. DOI: 10.1080/09720529.2021.1925447 (date of access: 18.05.2024).

4. Cryptography and Security: From Theory to Applications / ed. by D. Naccache. Berlin, Heidelberg : Springer Berlin Heidelberg, 2012. pp. 35–54. DOI: 10.1007/978-3-642-28368-0 (date of access: 18.05.2024).

5. Karpman R., Roldán É. Isotopy graphs of Latin tableaux. *Advances in Applied Mathematics*. 2021. Vol. 130. pp. 102-204. DOI: 10.1016/j.aam.2021.102204 (date of access: 21.05.2024).

6. An Enhanced Security Framework for Secured Data Storage and Communications in Cloud Using ECC, Access Control and LDSA / B. Prabhu Kavin et al. *Wireless Personal Communications*. 2020. Vol. 115, no. 2. pp. 1107–1135. DOI: 10.1007/s11277-020-07613-7 (date of access: 21.05.2024).

7. Bauer C. P. Stream Ciphers. Secret History. 2nd ed. 2021. pp. 533–544. DOI: 10.1201/9781315162539-19 (date of access: 24.05.2024).

8. Asim M., Arif M. Internet of things adoption and use in academic libraries: A review and directions for future research. *Journal of Information Science*. 2023. pp. 533–544. DOI: 10.1177/01655515231188338 (date of access: 24.05.2024).

9. Hell M., Johansson T., Meier W. Grain: a stream cipher for constrained environments. *International journal of wireless and mobile computing*. 2007. Vol. 2, no. 1. pp. 8–6. DOI: 10.1504/ijwmc.2007.013798 (date of access: 21.05.2024).

10. De Cannière C., Preneel B. Trivium. *Lecture Notes in Computer Science*. Berlin, Heidelberg, 2008. pp. 244–266. DOI: 10.1007/978-3-540-68351-3_18 (date of access: 21.05.2024).

11. Analysis of Area-Efficiency vs. Unrolling for eSTREAM Hardware Portfolio Stream Ciphers / F. Alharbi et al. *Electronics*. 2020. Vol. 9, no. 11. pp. 19–35. DOI: 10.3390/electronics9111935 (date of access: 21.05.2024).

12. Hell M., Johansson T. Breaking the F-FCSR-H Stream Cipher in Real Time. *Advances in Cryptology - ASIACRYPT 2008*. Berlin, Heidelberg, 2008. pp. 557–569. DOI: 10.1007/978-3-540-89255-7_34 (date of access: 21.05.2024).

13. A Lightweight Cipher Based on Salsa20 for Resource-Constrained IoT Devices / E. Lara et al. *Sensors*. 2018. Vol. 18, no. 10. pp. 33–26. DOI: 10.3390/s18103326 (date of access: 21.05.2024).

14. Zhen M., Tian T., Wenfeng Q. Differential Fault Attack on the Stream Cipher LIZARD. *Chinese Journal of Electronics*. 2021. Vol. 30, no. 3. pp. 534–541. DOI: 10.1049/cje.2021.04.007 (date of access: 21.05.2024).

15. Ghafari V. A., Lin F., Zhou Z. A new idea in response to fast correlation attacks on small-state stream ciphers. *Microprocessors and Microsystems*. 2022. pp. 104–720. DOI: 10.1016/j.micpro.2022.104720 (date of access: 21.05.2024).

16. Shcherbacov V. Quasigroups in cryptology. *Elements of Quasigroup Theory and Applications*. 2017. pp. 471–508. DOI: 10.1201/9781315120058-14 (date of access: 21.05.2024).

17. Zhang Y., Zhang H. An algorithm for judging and generating multivariate quadratic quasigroups over Galois fields. *SpringerPlus*. 2016. Vol. 5, no. 1. DOI: 10.1186/s40064-016-3525-2 (date of access: 27.05.2024).

18. Applications of Quasigroups in Cryptography and Coding Theory / S. Markovski et al. *Algebra without Borders – Classical and Constructive Nonassociative Algebraic Structures*. Cham, 2023. pp. 419–489. DOI: 10.1007/978-3-031-39334-1_10 (date of access: 27.05.2024).

19. Markovski S, Gligoroski D, and Bakeva V. Quasigroup and hash functions Proc. of the 6th ICDMA. Bansko. 2001. pp. 43–50.

20. Микитченко Б. В. Загирняк Б. Д. Крайнічук (Шелепало) Г. В. Побудова псевдовипадкових послідовностей на основі двох латинських квадратів. *Матеріали LIII наук.-техн. конф. ФІТКІ*, ВНТУ, 20-22 берез. Вінниця. 2024. С. 378-379.

21. Nathan O. Latin Squares and Their Applications to Cryptography. *Boise state university theses and dissertations*. 2016. 230 P. URL: https://scholarworks.boisestate.edu/td/1223/ (date of access: 27.05.2024).

22. https://electronicscoach.com/shift-register.html

23. Лужецький В.А., Крайнічук Г.В., Радченко Є.В., Пилявець І.Ю. алгоритм шифрування на основі CIP-квазігруп // Матеріали тезів IX Міжн. наук.-техн. конф. «ЗАХИСТ ІНФОРМАЦІЇ І БЕЗПЕКА ІНФОРМАЦІЙНИХ СИСТЕМ», НУ «Львівська політехніка». Львів. 2023. 258 С.

24. Movahedi Z, Defude B, Hosseininia A. An efficient population-based multi-objective task scheduling approach in fog computing systems. *Journal of Cloud Computing*. 2021. URL: https://journalofcloudcomputing.springeropen.com/articles/10.1186/s13677-021-00264-4 (date of access: 27.05.2024).

25. Gligoroski D., Markovski S., Knapskog S. J. The Stream Cipher Edon80. *Lecture Notes in Computer Science*. Berlin, Heidelberg. P. 152–169. DOI: 10.1007/978-3-540-68351-3_12 (date of access: 27.05.2024).

26. Gligoroski D., Markovski S., Kocarev L. Edon–R, An Infinite Family of Cryptographic Hash Functions. *International Journal of Network Security*. 2009. Vol.8, no.3. pp. 293–300. URL: http://ijns.jalaxy.com.tw/contents/ijns-v8-n3/ijns-2009-v8-n3-p293-300.pdf (date of access: 27.05.2024).

ДОДАТКИ

Додаток А
ПРОТОКОЛ ПЕРЕВІРКИ
БАКАЛАВРСЬКОЇ ДИПЛОМНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Засіб потокового шифрування на основі латинських квадратів. Частина 2. Операційний блок

Автор роботи: Загирняк Богдан Дмитрович

Тип роботи: бакалаврська дипломна робота .

Підрозділ кафедра захисту інформації ФІТКІ .
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність – 93,4%. Схожість – 6,6%.

Аналіз звіту подібності (відмітити потрібне):

√ 1. Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

☐ 2. Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

☐ 3. Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Валентина КАПЛУН
(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи _____ Богдан ЗАГИРНЯК
(підпис) (прізвище, ініціали)

Керівник роботи _____ Галина ШЕЛЕПАЛО
(підпис) (прізвище, ініціали)

Додаток Б
КОД ПРОГРАМНОГО ЗАСОБУ

Main.java

```
import java.math.BigInteger;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.Scanner;

public class Main {
    private static final int[][] SQUARE_FR = {
            {0, 2, 3, 1},
            {3, 1, 0, 2},
            {1, 3, 2, 0},
            {2, 0, 1, 3}
    };
    private static final int[][] SQUARE_1 = {
            {0, 3, 1, 2},
            {2, 1, 3, 0},
            {3, 0, 2, 1},
            {1, 2, 0, 3}
    };

    private static final int[][] SQUARE_2 = {
            {1, 2, 0, 3},
            {3, 0, 2, 1},
            {2, 1, 3, 0},
            {0, 3, 1, 2}
    };

    private static final int[][] SQUARE_3 = {
            {2, 1, 3, 0},
            {0, 3, 1, 2},
            {1, 2, 0, 3},
            {3, 0, 2, 1}
    };

    private static final int[][] SQUARE_4 = {
            {3, 0, 2, 1},
            {1, 2, 0, 3},
            {0, 3, 1, 2},
            {2, 1, 3, 0}
    };

    public static void main(String[] args) {
        BigInteger key = new
BigInteger("53882770620174092286102347100892329 40");
        //BigInteger generatorResult = new
BigInteger("819781413133707871275461762532462 6408");
        BigInteger generatorResult = Dec.Gen(key.and(new
BigInteger("3")).intValue());
        /*BigInteger message = new
BigInteger("474424181238947189234712412238904523908492342342352312341459178236412
7845235129038129048124");

        System.out.println(Dec.Gen(0));

        System.out.println("Input message:     " + message);

        BigInteger messageEnc = Encrypting(key, generatorResult, message);
```

```
        System.out.println("Encrypted message: " + messageEnc);

        BigInteger messageDec = Decrypting(key, generatorResult, messageEnc);

        System.out.println("Decrypted message: " + messageDec);

        System.out.println("---------------------------------");*/

        Scanner sc = new Scanner(System.in);
        System.out.println("Input string for encription: ");
        String massage_v2 = sc.nextLine();

        //System.out.println("Encrypted message: " + "\n" + "Dec format: " +
"\n" + Encrypting(key, generatorResult, massage_v2) + "\nChar format: " + "\n" +
convertBinaryToString(Encrypting(key, generatorResult, massage_v2)));
        //System.out.println("Decrypted message: " + "\n" + "Dec format: " +
"\n" + Decrypting(key, generatorResult, convertBinaryToString(Encrypting(key,
generatorResult, massage_v2))) + "\nChar format: " + "\n" +
convertBinaryToString(Decrypting(key, generatorResult,
convertBinaryToString(Encrypting(key, generatorResult, massage_v2)))));
        System.out.println("Encrypted message: ");
        //System.out.println("Binary: " + Encrypting(key, generatorResult,
massage_v2));
        System.out.println(convertBinaryToString(Encrypting(key,
generatorResult, massage_v2)));
        System.out.println("Decrypted message: " );
        System.out.println(Decrypting(key, generatorResult, Encrypting(key,
generatorResult, massage_v2)));

    }


    public static BigInteger Encrypting(BigInteger key, BigInteger
generatorResult, String message1) {
        BigInteger message = convertStringToBinary(message1);
        BigInteger messageEnc = new BigInteger("1");
        BigInteger originalGeneratorResult = generatorResult;
        BigInteger temp_key = new BigInteger("1");

        while (!message.equals(BigInteger.ZERO)) {
            BigInteger[] keyAndResultBits = getLastTwoBits(key, generatorResult,
message);
            BigInteger keyBits = keyAndResultBits[0];
            BigInteger resultBits = keyAndResultBits[1];
            BigInteger messageBits = keyAndResultBits[2];

            messageEnc = attachTwoBits(messageEnc,
BigInteger.valueOf(getSquareValue(keyBits, resultBits, messageBits)));
            temp_key = attachTwoBits(temp_key,
BigInteger.valueOf(SQUARE_FR[messageBits.intValue()][keyBits.intValue()]));///!!
!!!

            key = key.shiftRight(2);
            generatorResult = generatorResult.shiftRight(2);
            message = message.shiftRight(2);

            // Перевірка та відновлення змінних
            if (generatorResult.equals(BigInteger.ZERO)) {
                generatorResult = originalGeneratorResult;
            }
            if (key.equals(BigInteger.ONE)) { //ZERO
                key = temp_key;
                temp_key = new BigInteger("1");
            }
```

```
        }
        return messageEnc; //convertBinaryToString(messageEnc);
    }

    public static String Decrypting(BigInteger keyValue, BigInteger outGen,
BigInteger encryptedMessage1) {
        BigInteger encryptedMessage =
encryptedMessage1;//convertStringToBinary(encryptedMessage1);
        BigInteger messageDec = BigInteger.ONE;
        BigInteger originalGeneratorResult = outGen;
        int numBits = encryptedMessage.bitLength(); // Отримуємо кількість бітів
у вхідному повідомленні
        BigInteger temp_key = new BigInteger("1");
        boolean start = false;

        for (int i = numBits - 1; i >= 0; i -= 2) {
            BigInteger bits =
encryptedMessage.shiftRight(i).and(BigInteger.valueOf(3)); // Зчитуємо два біти
з використанням побітових операцій
            if (!bits.equals(BigInteger.ZERO) || start) {
                if (start) {
                    BigInteger[] keyAndResultBits = getLastTwoBits(keyValue,
outGen);
                    BigInteger keyBits = keyAndResultBits[0];
                    BigInteger resultBits = keyAndResultBits[1];
                    messageDec = attachTwoBits(messageDec,
BigInteger.valueOf(decryptBit(keyBits, resultBits, bits)));
                    temp_key = attachTwoBits(temp_key,
BigInteger.valueOf(SQUARE_FR[messageDec.and(new
BigInteger("3")).intValue()][keyBits.intValue()]));///!!!!!
                    keyValue = keyValue.shiftRight(2);
                    outGen = outGen.shiftRight(2);

                    // Перевірка та відновлення змінних
                    if (outGen.equals(BigInteger.ZERO)) {
                        outGen = originalGeneratorResult;
                    }
                    if (keyValue.equals(BigInteger.ONE)) {//ZERO
                        keyValue = temp_key;
                        temp_key = new BigInteger("1");
                    }

                }
                start = true;
            }

        }
        BigInteger temp = BigInteger.ZERO;
        while (!messageDec.equals(BigInteger.ONE)) {
            BigInteger mask = new BigInteger("3"); // Маска для отримання перших
двох бітів
            BigInteger messageDecbits = messageDec.and(mask);
            temp = attachTwoBits(temp, messageDecbits);
            messageDec = messageDec.shiftRight(2);

        }
        return convertBinaryToString(temp);
    }

    public static BigInteger[] getLastTwoBits(BigInteger key, BigInteger
generatorResult, BigInteger message) {
        BigInteger mask = new BigInteger("3"); // Маска для отримання перших
двох бітів
```

```
        BigInteger keyBits = key.and(mask);
        BigInteger resultBits = generatorResult.and(mask);
        BigInteger messageBits = message.and(mask);

        return new BigInteger[]{keyBits, resultBits, messageBits};
    }

    public static BigInteger[] getLastTwoBits(BigInteger key, BigInteger
generatorResult) {
        BigInteger mask = new BigInteger("3"); // Маска для отримання перших
двох бітів

        BigInteger keyBits = key.and(mask);
        BigInteger resultBits = generatorResult.and(mask);

        return new BigInteger[]{keyBits, resultBits};
    }

    public static BigInteger attachTwoBits(BigInteger key, BigInteger
additionalBits) {
        // Зсуваємо ключ на 2 біти уліво, щоб звільнити місце для додавання двох
бітів
        key = key.shiftLeft(2);

        // Додаємо два біти
        BigInteger modifiedKey = key.or(additionalBits);

        return modifiedKey;
    }

    private static int getSquareValue(BigInteger keyValue, BigInteger row,
BigInteger column) {
        return switch (keyValue.intValue()) {
            case 0 -> SQUARE_1[row.intValue()][column.intValue()];
            case 1 -> SQUARE_2[row.intValue()][column.intValue()];
            case 2 -> SQUARE_3[row.intValue()][column.intValue()];
            case 3 -> SQUARE_4[row.intValue()][column.intValue()];
            default -> throw new IllegalArgumentException("Invalid key value: "
+ keyValue);
        };
    }

    public static int decryptBit(BigInteger keyValue, BigInteger outGen,
BigInteger encryptedMessage) {
        int[][] square;
        switch (keyValue.intValue()) {
            case 0 -> square = SQUARE_1;
            case 1 -> square = SQUARE_2;
            case 2 -> square = SQUARE_3;
            case 3 -> square = SQUARE_4;
            default -> throw new IllegalArgumentException("Invalid key value: "
+ keyValue);
        }

        int[] row = square[outGen.intValue()];
        for (int i = 0; i < row.length; i++) {
            if (row[i] == encryptedMessage.intValue()) {
                return i;
            }
        }
        throw new IllegalArgumentException("Value not found in the row: " +
encryptedMessage);

    }
```

```
    public static BigInteger convertStringToBinary(String input) {
        byte[] bytes = input.getBytes(StandardCharsets.UTF_8);
        StringBuilder binaryBuilder = new StringBuilder();

        for (byte b : bytes) {
            int val = b;
            for (int i = 7; i >= 0; i--) {
                binaryBuilder.append((val & (1 << i)) != 0 ? '1' : '0');
            }
        }

        return new BigInteger(binaryBuilder.toString(), 2);
    }

    public static String convertBinaryToString(BigInteger binary) {
        String binaryString = binary.toString(2);

        // Ensure the binary string length is a multiple of 8
        int paddingLength = 8 - (binaryString.length() % 8);
        if (paddingLength < 8) {
            for (int i = 0; i < paddingLength; i++) {
                binaryString = "0" + binaryString;
            }
        }

        int byteLength = binaryString.length() / 8;
        byte[] bytes = new byte[byteLength];

        for (int i = 0; i < byteLength; i++) {
            String byteString = binaryString.substring(i * 8, (i + 1) * 8);
            bytes[i] = (byte) Integer.parseInt(byteString, 2);
        }

        return new String(bytes, StandardCharsets.UTF_8);
    }
}
```

Dec.java

```
import java.math.BigInteger;
import java.util.ArrayList;

public class Dec {
    static int[][] LS1 = {//91
            {0, 2, 3, 1},
            {3, 1, 0, 2},
            {1, 3, 2, 0},
            {2, 0, 1, 3}
    };

    static int[][] LS2 = {//0
            {0, 1, 2, 3},
            {1, 0, 3, 2},
            {2, 3, 0, 1},
            {3, 2, 1, 0}
    };
    static int[][] LS3 = {//28
            {0, 1, 3, 2},
            {1, 2, 0, 3},
            {2, 3, 1, 0},
            {3, 0, 2, 1}
    };
```

```
    static int[][] LS4 = {//184
            {1, 0, 3, 2},
            {3, 1, 2, 0},
            {0, 2, 1, 3},
            {2, 3, 0, 1}
    };

    public static BigInteger Gen(int getb) {
        //int getb = 1;
        ArrayList<Integer> sequence;
        switch (getb) {
            case 0 -> sequence = generateSequence(1, 2, 3, LS1, LS2);
            case 1 -> sequence = generateSequence(1, 2, 3, LS2, LS1);
            case 2 -> sequence = generateSequence(1, 2, 3, LS3, LS4);
            case 3 -> sequence = generateSequence(1, 2, 3, LS4, LS3);
            default -> sequence = generateSequence(1, 2, 3, LS1, LS4);
        }
        sequence.removeLast();
        sequence.removeLast();
//        System.out.println(sequence);
//        System.out.print(sequence.size());
        BigInteger decimalValue = convertToDecimal(sequence);
        //System.out.println("У десятковому вигляді: " + decimalValue);
        return decimalValue;
    }
    static ArrayList<Integer> generateSequence(int u1, int u2,int u3, int
[][]latinSquare0,int [][]latinSquare) {
        ArrayList<Integer> sequence = new ArrayList<>();
        sequence.add(u1);
        sequence.add(u2);
        sequence.add(u3);
        generateSequenceRecursively(u1, u2,u3,
sequence,latinSquare0,latinSquare);
        return sequence;
    }
    static void generateSequenceRecursively(int u1, int u2,int
u3,ArrayList<Integer> sequence,int [][]latinSquare0,int [][]latinSquare) {
        int next = latinSquare[latinSquare0[u1][u2]][u3]; // Отримуємо наступний
елемент за вказаним квадратом
        sequence.add(next);
        if (u2 != sequence.get(0) || u3 != sequence.get(1) || next !=
sequence.get(2)) {
            generateSequenceRecursively(u2, u3, next,
sequence,latinSquare0,latinSquare);
        }
    }
    public static BigInteger convertToDecimal(ArrayList<Integer> sequence) {
        BigInteger base = BigInteger.valueOf(4); // Основа системи числення (4
для чотирикової)
        BigInteger decimalValue = BigInteger.ZERO;

        for (int i = 0; i < sequence.size(); i++) {
            BigInteger digit = BigInteger.valueOf(sequence.get(i));
            decimalValue = decimalValue.multiply(base).add(digit);
        }

        return decimalValue;
    }
}
```

ІЛЮСТРАТИВНА ЧАСТИНА

ЗАСІБ ПОТОКОВОГО ШИФРУВАННЯ НА ОСНОВІ ЛАТИНСЬКИХ КВАДРАТІВ. ЧАСТИНА 2. ОПЕРАЦІЙНИЙ БЛОК.

Виконав: студент 4 курсу групи 1БКС-20б спеціальності 125 Кібербезпека
_____ Богдан ЗАГИРНЯК
17 червня 2024 р.
Керівник: к.ф.-м.н., доцент каф. ЗІ
_____ Галина ШЕЛЕПАЛО
17 червня 2024 р.

Таблиці Келі 4-го порядку для виконання операції шифрування з властивостями схрещеної оборотності

Q1:

*	0	1	2	3
0	0	3	1	2
1	2	1	3	0
2	3	0	2	1
3	1	2	0	3

Q2:

*	0	1	2	3
0	1	2	0	3
1	3	0	2	1
2	2	1	3	0
3	0	3	1	2

Q3:

*	0	1	2	3
0	2	1	3	0
1	0	3	1	2
2	1	2	0	3
3	3	0	2	1

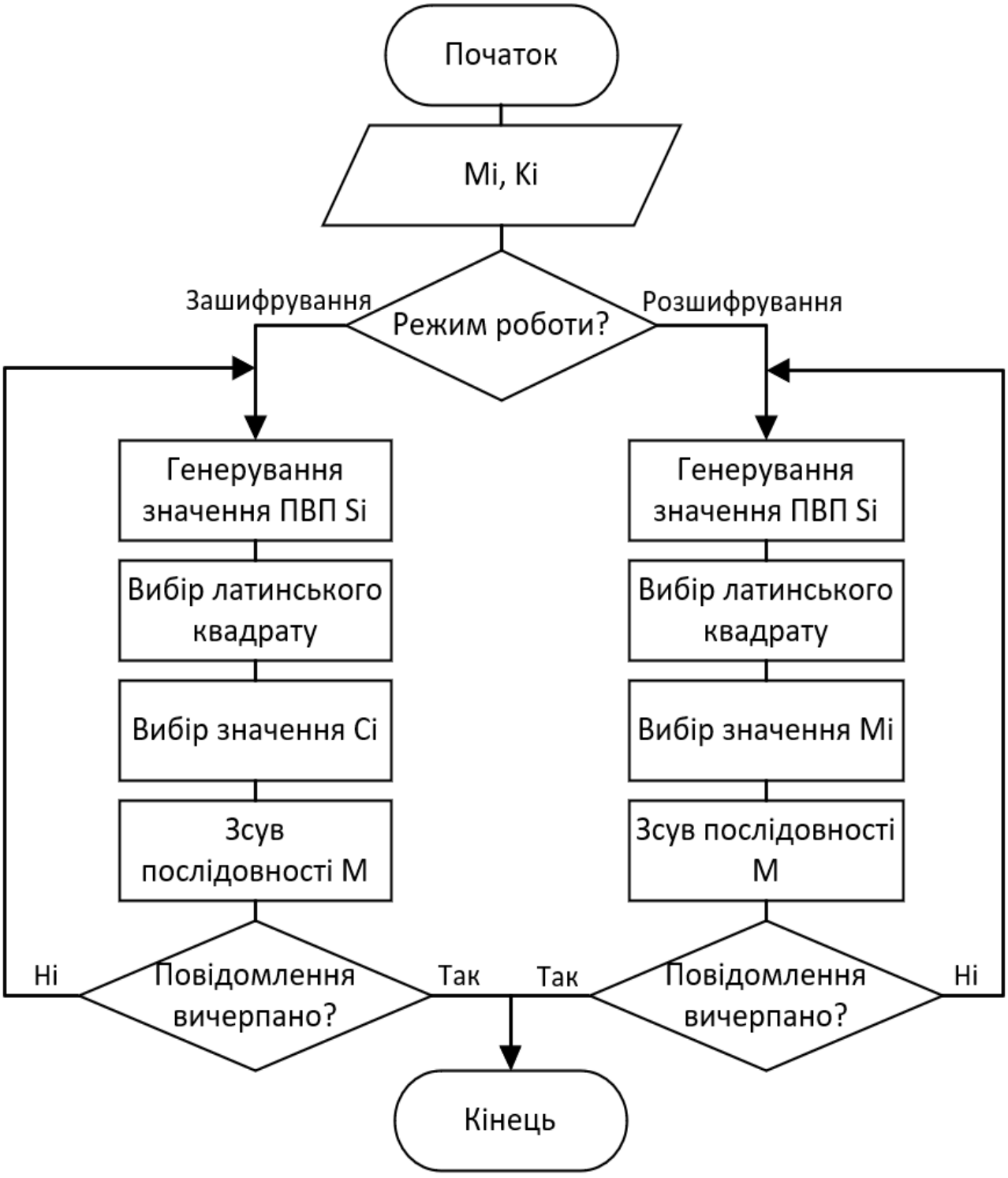
Q4:

*	0	1	2	3
0	3	0	2	1
1	1	2	0	3
2	0	3	1	2
3	2	1	3	0

Тотальносиметрична таблиця Келі 4-го порядку для генератора гами

*	0	1	2	3
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

Схема алгоритму шифрування



Таблиці Келі 4-го порядку для виконання операції розшифрування

Q5:

*	0	1	2	3
0	0	2	3	1
1	3	1	0	2
2	1	3	2	0
3	2	0	1	3

Q6:

*	0	1	2	3
0	1	3	2	0
1	2	0	1	3
2	0	2	3	1
3	3	1	0	2

Q7:

*	0	1	2	3
0	2	0	1	3
1	1	3	2	0
2	3	1	0	2
3	0	2	3	1

Q8:

*	0	1	2	3
0	3	1	0	2
1	0	2	3	1
2	2	0	1	3
3	1	3	2	0

Структура засобу шифрування

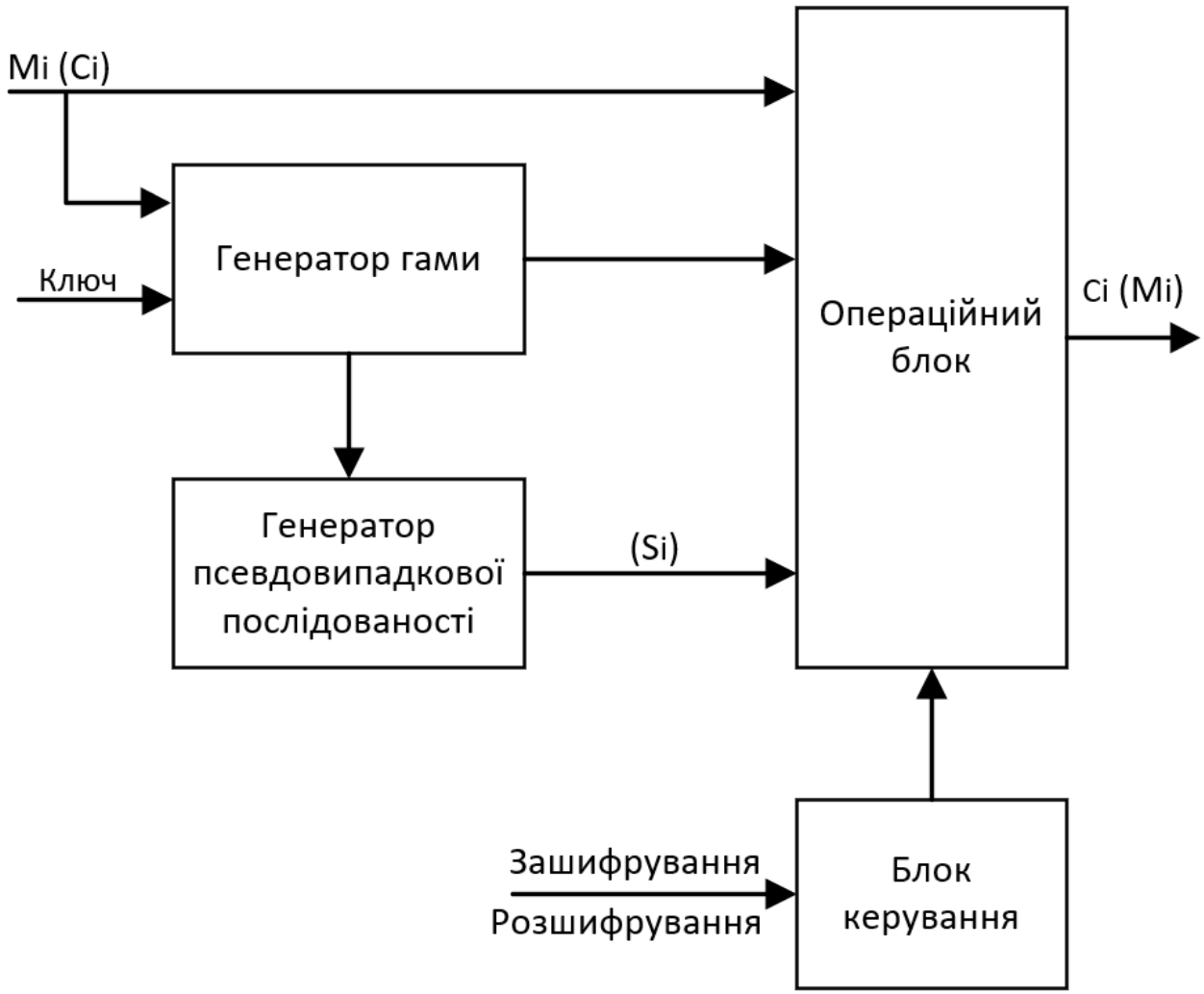


Схема генератора псевдовипадкових послідовностей

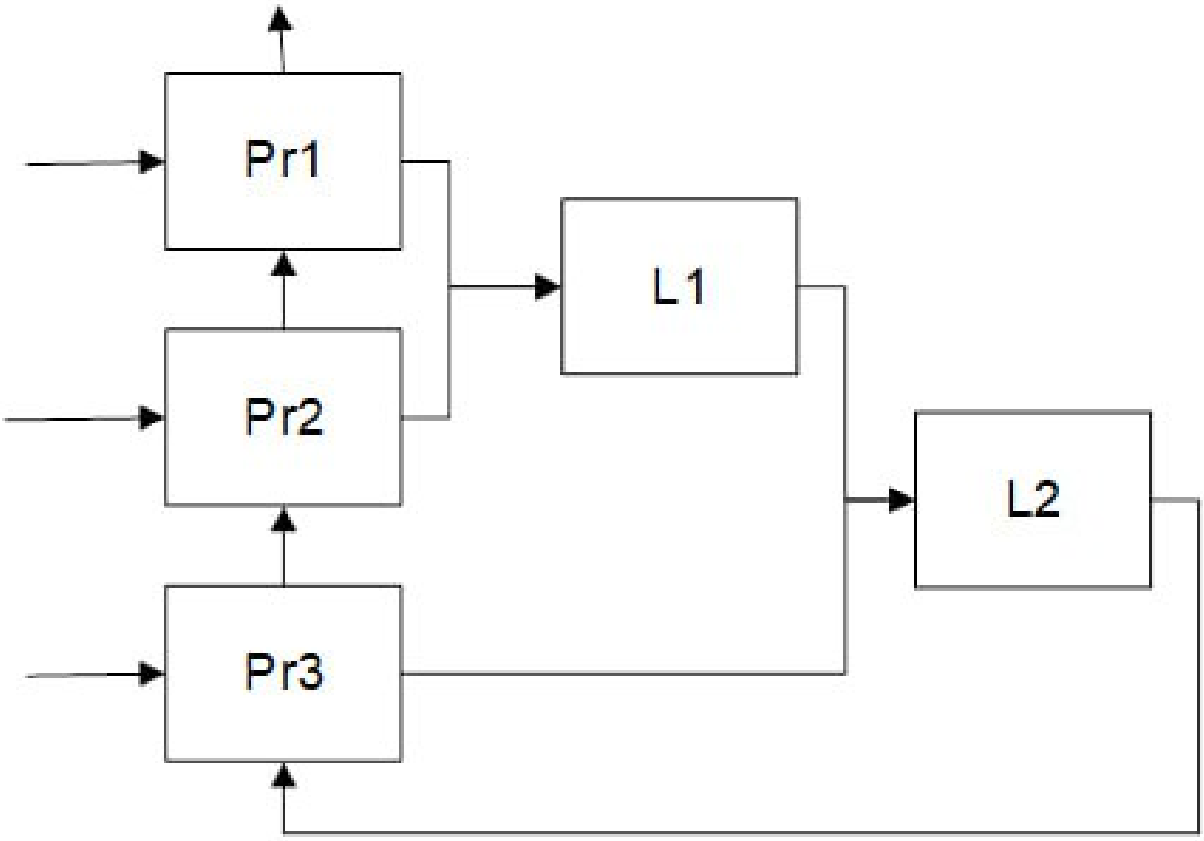
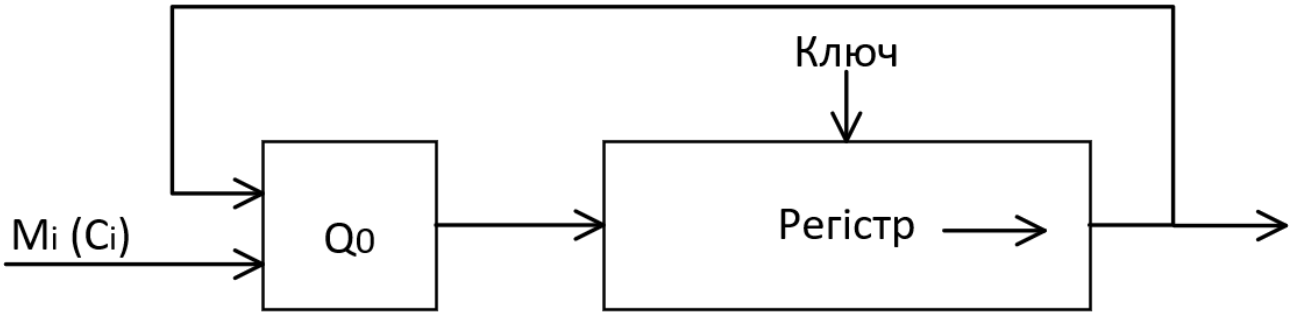


Схема генератора гами



Порівняльні характеристики алгоритмів потокового шифрування

Назва шифру	Довжина ключа (біт)	Складність (GE)
Розроблений (LSMZ)	64	565.97
LKRP	64	482.39
Edon-80	80	831
LILLE	80	911
Planet	80	928
Fruit-80	80	960
LIZARD	120	1161
Edon-R	256	1198
Grain	80	1294
MICKEY 2.0	80	1846
Trivium	80	2599
Enocoro-128	128	2700
Salsa20	256	3842
F-FCSR-H	80	4800