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

Комплексна магістерська кваліфікаційна робота на тему:

«Метод та засіб потокового шифрування на основі квазігруп.
Частина 2. Операційний блок.»

Виконав: студент 2 курсу групи 1БС-24м
спеціальності 125 Кібербезпека та захист інформації
_____ Богдан ЗАГИРНЯК
Керівник: к. ф.-м. н., доцент каф. ЗІ
_____ Галина ШЕЛЕПАЛО
«16» грудня 2025 р.
Рецензент: к. т. н., доц., доц. каф. ПЗ
_____ Олександр ХОШАБА
«16» грудня 2025 р.

Допущено до захисту
В. о. зав. каф. ЗІ
д. т. н., проф.
_____ Володимир ЛУЖЕЦЬКИЙ
«16» грудня 2025 р.

Вінниця ВНТУ – 2025 року

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації
Рівень вищої освіти II (магістерський)
Галузь знань – 12 Інформаційні технології
Спеціальність – 125 Кібербезпека та захист інформації
Освітньо-професійна програма – Безпека інформаційних і комунікаційних систем

ЗАТВЕРДЖУЮ
В. о. зав. кафедри ЗІ, д. т. н., проф.
Володимир ЛУЖЕЦЬКИЙ
24.09 **2025 року**

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

Загирняку Богдану Дмитровичу

1. Тема роботи: "Метод та засіб потокового шифрування на основі квазігруп. Частина 2. Операційний блок.",
керівник роботи: Шелепало Галина Василівна к.ф.-м.н., доцент, затверджені наказом ректора ВНТУ від 24 вересня 2025 року №313.
2. Строк подання студентом роботи 16 грудня 2025 р.
3. Вихідні дані до роботи:
   - приклади використання криптографічно стійких квазігруп у криптографії;
   - сучасні алгоритми шифрування в області LW-криптографії;
   - вимоги до засобів LW-криптографії.
4. Зміст текстової частини: Вступ. 1 Аналіз інформаційних джерел. 2 Розробка та оцінювання операційного блоку шифру. 3 Інтеграція та випробування моделі апаратно-програмного комплексу. 4. Економічна частина. Висновки. Список використаних джерел. Додатки.
5. Перелік ілюстративного матеріалу: Порівняльні характеристики досліджених потокових шифрів. Таблиця Келі 4-го порядку для генератора гами. Схема генератора гами. Таблиці Келі 4-го порядку для операції зашифрування з властивостями схрещеної оборотності. Таблиці Келі 4-го порядку для операції розшифрування. Схема алгоритму апаратного засобу. Загальна структурна схема операційного блоку. Діаграма станів керуючого автомату (FSM). Загальна структурна схема апаратного інтерфейсу. Схема роботи автомату FSM розробленого протоколу. Схема ієрархічної структури рівнів абстракції даних. Схема взаємодії компонентів стенду.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Галина ШЕЛЕПАЛО, к. ф.-м. н., доц. каф. ЗІ	25.09.25	26.09.25
2	Галина ШЕЛЕПАЛО, к. ф.-м. н., доц. каф. ЗІ	25.09.25	03.11.25
3	Галина ШЕЛЕПАЛО, к. ф.-м. н., доц. каф. ЗІ	25.09.25	17.11.25
4	Олександр ЛЕСЬКО, к. е. н., проф. зав. каф. ЕПВМ	25.09.25	17.11.25

7. Дата видачі завдання: 25 вересня 2025 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз завдання. Вступ	24.09.25 – 26.09.25	
2	Аналіз інформаційних джерел за напрямком комплексної магістерської дипломної роботи	27.09.25 – 07.10.25	
3	Розробка моделей та алгоритмів	23.10.25 – 02.11.25	
4	Практична реалізація, моделювання, результати	03.11.25 – 17.11.25	
5	Розробка розділу економічного обгрунтування доцільності розробки	18.11.25 – 22.11.25	
6	Оформлення пояснювальної записки	23.11.25 – 29.11.25	
7	Попередній захист та доопрацювання МКР	29.11.25 – 11.12.25	
8	Перевірка на наявність текстових запозичень	12.12.25 – 15.12.25	
9	Представлення МКР до захисту, рецензування	16.12.25 – 19.12.25	
10	Захист МКР	19.12.25 – 23.12.25	

Студент _____ Богдан ЗАГИРНЯ

Керівник роботи _____ Галина ШЕЛЕПАЛ

АНОТАЦІЯ

Загирняк Б. Метод та засіб потокового шифрування на основі квазігруп. Частина 2. Операційний блок. Магістерська кваліфікаційна робота зі спеціальності 125 – Кібербезпека та захист інформації, освітня програма – Безпека інформаційних і комунікаційних систем. Вінниця: ВНТУ, 2025. 85 с.

Укр. мовою. Бібліогр.: 26 назв; рис. 22, табл. 15.

Комплексна магістерська робота зосереджена на апаратній реалізації потокового алгоритму шифрування на основі квазігруп. У роботі було проведено аналіз існуючих рішень у сфері криптографії. Розроблено апаратну реалізацію операційного блоку шифру. Інтегровано розроблене рішення у модель апаратно-програмного комплексу шифрування. Такий інструмент сприяє покращенню ефективності та безпеки шифрування, а також дозволяє збільшити стійкість криптографічних систем, що базуються на квазігрупах.

Ілюстративна частина складається з 12 плакатів з демонстрацією результатів моделювання, розробки та проведених досліджень.

Економічна частина присвячена оцінці витрат на розробку. В ній доведено, що науково-дослідна робота має високий потенціал та є економічно доцільною.

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

ABSTRACT

Zahyrniak B. Method and means of stream cipher based on quasi-groups. Part 2. Operational block. Bachelor's thesis in specialty 125 – cybersecurity. Vinnitsa: VNTU, 2024. – 85 p.

In Ukrainian language. Bibliographer: 26 titles; fig. 22, tabl. 15.

This comprehensive master's thesis focuses on the hardware implementation of a stream cipher algorithm based on quasi-groups. The thesis analyzes existing solutions in the field of cryptography. A hardware implementation of the cipher's operating block has been developed. The developed solution is integrated into the hardware and software encryption complex. Such a tool contributes to improving the efficiency and security of encryption, as well as increasing the stability of cryptographic systems based on quasi-groups.

The illustrative part consists of 12 posters demonstrating the results of modeling, development, and research conducted.

The economic section is devoted to assessing development costs. It proves that research and development work has high potential and is economically viable.

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

ЗМІСТ

ВСТУП

Сучасна епоха характеризується глибокою цифровою трансформацією, що пронизує усі сфери людської діяльності. Від промислової автоматизації та критичної інфраструктури до персональних пристроїв Інтернету речей та кібер-фізичних систем, суспільство стає дедалі залежним від коректної та безпечної роботи розподілених обчислювальних мереж.

В таких умовах інформація перетворюється на ключовий актив та, водночас, на головну ціль для атак. Це відкриває нові вектори загроз для зловмисників, які прагнуть отримати несанкціонований доступ, порушити конфіденційність чи цілісність даних. Відтак, проблема захисту інформації вимагає постійного вдосконалення криптографічних методів.

Складні задачі захисту інформації формуються у сегменті вбудованих систем. На відміну від потужних серверів чи персональних комп’ютерів, пристрої з обмеженими ресурсами як от сенсори, контролери чи RFID-мітки функціонують в умовах жорстких обчислювальних, енергетичних та апаратних обмежень. Це унеможливлює застосування класичних криптографічних стандартів та формує сталий попит на розробку у галузі LW-криптографії (від англ. Lightweight - легковаговий).

У контексті LW-криптографії, де апаратна реалізація повинна бути максимально компактною та ефективною, потокові шифри часто демонструють значні переваги. На відміну від блокових алгоритмів, що оперують великими порціями даних та часто вимагають значних ресурсів, потокові шифри обробляють інформацію у вигляді потоку. Такий підхід дозволяє досягти нижчої затримки та, меншої площі на кристалі при апаратній реалізації.

Для генерації псевдовипадкових послідовностей у потокових шифрах використовуються різноманітні математичні апарати. Серед них, значний науковий інтерес представляють алгебраїчні структури квазігруп та їх аналоги у вигляді латинських квадратів. Їхні специфічні властивості дозволяють будувати ефективні криптографічні перетворення.

Використання таких інструментів не зменшує ефективність та актуальність досліджень даної тематики, оскільки такий підхід не є стандартним та є не повністю дослідженим. Поєднання стрімкого росту ринку пристроїв з обмеженими ресурсами з одного боку, та унікальних властивостей квазігруп з іншого, визначає актуальність дослідження.

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

Метою комплексної магістерської роботи є зменшення складності засобу потокового шифрування шляхом розробки методу шифрування на основі квазігруп.

Для досягнення поставленої мети сформовано такі задачі:

– Проаналізувати тематику потокового шифрування.
– Виконати порівняльний аналіз методів потокового шифрування в області малоресурсної криптографії.
– Розробити метод потокового шифрування на основі квазігруп.
– Розробити структуру операційного блоку потокового шифрування та оцінити його складність.
– Розробити модель апаратно-програмного комплексу потокового шифрування на основі квазігруп.

Науковою новизною роботи є зменшення апаратної складності засобу потокового шифрування шляхом розробки методу та засобу шифрування на основі квазігруп.

Результати дослідження за темою були представлені у публікації:

Загирняк Б. Д. Крайнічук (Шелепало) Г. В. Алгоритм потокового шифрування на основі латинських квадратів. *SMICS: Безпека сучасних інформаційно-комунікаційних систем: матеріали міжнар. наук.-техн. конф., м. Львів, 16-18 жовтня 2025 р.* ЛНУ ім. І. Франка, 2025, С. 246 – 250.

1 АНАЛІЗ ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

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

1.1 Проблематика та актуальність дослідження

Забезпечення конфіденційності передачі інформації у сучасному цифровому світі є необхідною умовою для безпечного функціонування інформаційних систем. Виконання такої умови залежить безпосередньо від методів шифрування. Таким чином розробка нових та покращення існуючих криптографічних рішень є актуальним завданням для сучасних науковців та розробників.

Проблематика сучасного цифрового світу полягає не тільки у методах обробки чутливої інформації а й у засобах на яких така інформація обробляється. Сучасні системи все частіше використовують малоресурсні рішення Інтернету речей (IoT), що у свою чергу створює серйозні виклики для криптографії. Оскільки рішення IoT вимагають від криптографічних алгоритмів зниження використання ресурсів, водночас забезпечення якомога кращого захисту інформації. Відповіддю на такі виклики прийнято вважати область легковагової криптографії (від англ. Lightweight - легковаговий). Ця область криптографії створена саме для таких цілей, а саме зменшення необхідних ресурсів для забезпечення захисту інформації у системах з обмеженими ресурсами.

Ключова ідея легковагової криптографії полягає у досягненні оптимального балансу між трьома основними параметрами: безпекою, вартістю (кількістю ресурсів) та продуктивністю. На відміну від традиційних алгоритмів, що розроблялися для потужних комп'ютерів, LWC-рішення проєктуються з урахуванням жорстких обмежень апаратної частини, таких як: малий обсяг пам’яті (RAM/ROM), низьке енергоспоживання (що критично для пристроїв з автономним живленням) та обмежена обчислювальна потужність процесора. Важливість цього напряму підкреслюється діяльністю світових організацій,

зокрема проектом Національного інституту стандартів і технологій США (NIST), який у 2023 році завершив процес стандартизації легковагових алгоритмів, обравши переможцем сімейство ASCON [1]. Метою таких ініціатив є надання надійних та ефективних інструментів для захисту даних у специфічних умовах.

Особливості рішень LW-криптографії полягають не лише в обмежених ресурсах, але й в унікальних загрозах, орієнтованими на перехоплення даних та компрометацію самого пристрою. Багато IoT-пристроїв використовує бездротові канали зв'язку, що робить їх вразливими до атак прослуховування. Без надійного шифрування конфіденційні дані, наприклад показники медичних сенсорів чи команди управління критичною інфраструктурою, можуть бути легко перехоплені [2]. Також критичним для апаратних реалізацій є загроза фізичного доступу. На відміну від серверів у дата-центрах, IoT-пристрої часто є фізично доступними для зловмисників. Це відкриває шлях до потужних атак сторонніми каналами (Side-Channel Attacks, SCA), де аналіз коливань електроспоживання чи електромагнітного випромінювання дозволяє викрасти секретний ключ [3]. Також можливі атаки внесенням збоїв (Fault Injection Attacks, FIA), де цілеспрямоване втручання в роботу чипа (наприклад, зміною напруги) може призвести до пропуску інструкцій безпеки або витоку даних. Таким чином, легковаговий алгоритм має бути не лише ефективним, але й стійким до реалізації в апаратурі, що захищає від фізичного втручання [4].

Для відповіді на ці загрози в умовах обмежених ресурсів було розроблено декілька основних класів алгоритмів LW-криптографії. Першу групу складають легковагові блокові шифри, які оперують над блоками даних фіксованого розміру. Вони є фундаментальними будівельними блоками для багатьох протоколів. Яскравим представником цього класу є шифр PRESENT, який став одним із перших, який був стандартизований ISO/IEC 29192-2, і довів можливість створення надзвичайно компактних апаратних реалізацій [5]. Другу, надзвичайно важливу групу, складають легковагові хеш-функції та схеми автентифікованого шифрування (AEAD). Вони вирішують не лише задачу конфіденційності, але й критичну проблему забезпечення цілісності та

автентичності даних. Саме до цього класу належить родина алгоритмів ASCON, обрана NIST у 2023 році переможцем процесу стандартизації LWC [6]. Третю, окрему гілку, формують легковагові потокові шифри. На відміну від блокових, вони генерують псевдовипадковий ключовий потік (гамму), який побітово поєднується з відкритим текстом. Історично, значний поштовх до їхнього розвитку дав європейський проєкт eSTREAM [7].

Потокові шифри, демонструють низку унікальних переваг, які роблять їх ідеальними кандидатами для апаратних рішень LW-криптографії. Фундаментальна відмінність від блокових шифрів полягає у відсутності потреби в буферизації даних, оскільки шифрування відбувається потоком, побітово або побайтово. Це забезпечує мінімальну затримку, що є критичною вимогою для роботи систем у реальному часі, таких як обробка даних з сенсорів чи швидкі протоколи зв'язку [8]. Також багато потокових шифрів, особливо ті, що базуються на зсувних регістрах, можуть бути реалізовані з низькою площею на кристалі, часто значно перевершуючи за компактністю навіть легковагові блокові шифри [9]. Додатковою перевагою є відсутність потреби у механізмах доповнення, оскільки шифр оперує над потоком будь-якої довжини, що усуває зайві обчислення та накладні витрати на передачу даних [7].

Основою будь-якого криптографічного алгоритму є математичний апарат, що визначає його ключові властивості. Дане дослідження розглядає квазігрупи як математичний апарат, для реалізації алгоритму потокового шифрування. Завдяки своїм математичним властивостям, квазігрупи є перспективним інструментом для криптографії, здатним забезпечити високу ентропію та обчислювальну складність алгоритму. Таким чином, дослідження в галузі потокового шифрування на основі квазігруп є актуальним напрямком, що може зробити значний внесок у сферу сучасної криптографії.

1.2 Огляд архітектурних особливостей побудови апаратних потокових шифрів

В основі архітектури переважної більшості апаратних, зокрема легковагових, потокових шифрів лежить принцип синхронного шифрування. Його ядром є Генератор Ключового Потоку (Keystream Generator, KSG) скінченний автомат, який, отримавши на вхід секретний ключ та (зазвичай) публічний вектор ініціалізації, генерує псевдовипадкову послідовність бітів. Ця послідовність, відома як ключовий потік або гамма, побітово поєднується з відкритим текстом за допомогою операції виключного АБО (XOR) для отримання шифротексту. Процес дешифрування є ідентичним, де KSG генерує ту саму гаму використовуючи той самий ключ та вектор ініціалізації, і поєднує її з шифротекстом [8]. Таким чином, вся криптографічна безпека системи повністю покладається на одну вимогу, а саме неможливість для зловмисника, який не знає ключа, передбачити біти ключового потоку [9].

Головним будівельним блоком для апаратних генераторів ключового потоку (KSG) є Регістр Зсуву з Лінійним Зворотним Зв'язком (Linear Feedback Shift Register, LFSR). Його надзвичайна популярність у LW-криптографії пояснюється декількома причинами. Перш за все, він є апаратно надлегким, для реалізації n-бітного LFSR потрібні лише n комірок пам'яті та декілька модулів XOR для організації зворотного зв'язку, що дорівнює мінімальній площі на кристалі (GE) [7]. LFSR здатні працювати на дуже високих тактових частотах. Також, при правильному виборі поліному зворотного зв'язку, LFSR довжиною n біт здатен згенерувати псевдовипадкову послідовність максимальної довжини $2^n - 1$, відому як m-послідовність. Такі послідовності мають хороші статистичні властивості, зокрема великий період [11]. Ці якості роблять LFSR ідеальною відправною точкою для LW-криптографії, однак він має фундаментальну ваду лінійнійності.

Ключова вразливість LFSR, про яку було згадано, є його лінійність. Послідовність, згенерована LFSR, має низьку лінійну складність, яка дорівнює

довжині самого регістра. На практиці це означає, що зловмисник може повністю відновити всю структуру LFSR, тобто знайти його поліном зворотного зв'язку і початковий стан, знаючи лише 2n послідовних бітів ключового потоку, де n - довжина LFSR. Для цього використовується високоефективний алгоритм Берлекемпа-Мессі [11]. Як тільки структура LFSR стає відомою, зловмисник може згенерувати весь майбутній і минулий ключовий потік, що призводить до повної компрометації шифру [10]. Через цю фатальну вразливість, LFSR ніколи не використовуються у такому вигляді для шифрування. Це формує фундаментальний принцип побудови апаратних KSG з необхідністю внесення нелінійності для протидії атакам.

Першим архітектурним принципом для внесення нелінійності є використання нелінійних комбінаторів. Ідея полягає у використанні декількох паралельно працюючих LFSR, кожен з яких генерує власну лінійну послідовність. Після чого вихідні біти з кожного регістра одночасно подаються на вхід нелінійної булевої функції f, яка перемішує їх. Вихідний біт цієї функції і стає бітом фінального ключового потоку. Класичним прикладом такого підходу є генератор Геффе, який використовує три LFSR та функцію $f(x_1, x_2, x_3) = (x_1 \wedge x_2) \oplus (\neg x_1 \wedge x_3)$. Однак, незважаючи на введення нелінійності, ця проста архітектура є вразливою до кореляційних атак. Оскільки вихід z_i збігається з виходом x_2 у 75% випадків, зловмисник може проаналізувати кожен LFSR окремо, що зводить складність атаки до аналізу найслабшого компонента, а не всієї системи [12].

Наступним архітектурним принципом є генератор з нелінійним фільтром (Non-linear Filter Generator). На відміну від комбінатора, ця архітектура зазвичай використовує лише один LFSR великої довжини. Нелінійність вноситься шляхом того, що біти для ключового потоку беруться не з одного виходу, а з кількох визначених комірок внутрішнього стану регістра. Ці біти потім подаються на вхід нелінійної фільтруючої булевої функції, вихід якої і є бітом гамми. Цей підхід став успішним і лежить в основі багатьох шифрів, зокрема шифрів родини

Grain , які були обрані для портфоліо апаратного профілю проекту eSTREAM саме завдяки своїй винятковій компактності [12].

Іншим архітектурний рішенням, є використання Регістру Зсуву з Нелінійним Зворотним Зв'язком (Non-linear Feedback Shift Register, NFSR), яке вважається більш стійким до певних видів криптоаналізу. На відміну від LFSR, де наступний стан лінійно залежить від попереднього, в NFSR ця залежність описується нелінійною функцією. Тобто, нелінійність закладена безпосередньо в механізм оновлення стану, а не додається на виході. Вважається, що це значно ускладнює алгебраїчні атаки, оскільки стан системи описується значно складнішою системою рівнянь [13]. Через це NFSR часто використовуються в сучасних легковагих шифрах. Наприклад, шифр Grain-v1 насправді є гібридною конструкцією, що використовує і LFSR, і NFSR, комбінуючи їхні стани для підвищення загальної складності [14].

Аналіз показує, що незалежно від обраної схеми, криптографічним ядром будь-якого генератора є його нелінійний компонент. Саме він єдиний забезпечує плутанину значень і є головним бар'єром проти лінійного та алгебраїчного криптоаналізу. В апаратних реалізаціях цей компонент найчастіше має форму булевої функції або таблиці заміни, відомої як S-блок. До цього компонента висувається подвійний, часто суперечливий набір вимог. З одного боку, він повинен мати сильні криптографічні властивості, а саме високу нелінійність , для протидії лінійним атакам, високий алгебраїчний ступінь, для протидії алгебраїчним атакам, низьку диференціальну рівномірність, для протидії диференціальним атакам. З іншого боку, в контексті LW-криптографії, він має бути апаратно легким, тобто мати мінімальну площу на кристалі [15].

Таким чином, уся безпека сучасного потокового шифру зводиться до фундаментальної проблеми проєктування, а саме синтез нелінійного компоненту, що одночасно задовольняє всім суворим криптографічним критеріям і має мінімальну апаратну вартість. Традиційно, математичною основою для побудови таких S-блоків слугують алгебраїчні операції, зокрема інверсія, у скінченних полях Галуа, як в AES. Однак, досягнення оптимального

балансу між високою нелінійністю та низькою кількістю вентилів у цій парадигмі є надзвичайно складною задачею. Саме ця складність та постійний пошук інноваційних рішень спонукають дослідників вивчати альтернативні алгебраїчні структури, що здатні стати новою основою для синтезу ефективних та безпечних криптографічних перетворень.

1.3 Квазігруповий підхід у криптографії

Особливості проектування легковагових шифрів полягає у створенні нелінійних компонентів, що є одночасно апаратно легкими та криптографічно стійкими [15]. Традиційний підхід, що базується на алгебраїчних операціях у скінченних полях Галуа, стикається зі значними труднощами у досягненні цього балансу. Ця проблема спонукає дослідників до пошуку альтернативних алгебраїчних структур. Однією з таких перспективних, але ще недостатньо досліджених у сфері LW-криптографії, структур є квазігрупи [16]. Їхні унікальні математичні властивості відкривають нові підходи до синтезу нелінійних перетворень, що потенційно краще відповідають жорстким апаратним вимогам.

Квазігрупи є алгебраїчними 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 умові, що для будь-яких $a,b \in Q$ існують єдині $x,y \in Q$ такі, що $a*x = b$ і $y*a = b$. Інакше кажучи, у кожній квазігрупі для кожного елемента множини можна знайти єдиний інший елемент, що задовольняє рівнянням з операцією * [17].

Основні властивості квазігруп:

– Заміщуваність: кожний елемент можна замістити іншим, зберігаючи властивості групи.
– Інверсія: для кожного елемента існує інверсний елемент.

– Латинський квадрат: таблиця Келі квазігрупи є латинським квадратом, що забезпечує унікальність кожного елемента у кожному рядку та стовпці.

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.

Ключова перевага квазігруп випливає безпосередньо з властивості латинського квадрата, що описує їхню таблицю Келі. Ця властивість забезпечує ідеальну дифузію, що є однією з фундаментальних вимог до стійких криптографічних перетворень, сформульованих ще Клодом Шенноном [18]. Унікальність кожного елемента в кожному рядку та стовпці таблиці Келі гарантує, що будь-яка зміна в одному з двох вхідних операндів a або b неминуче призводить до зміни вихідного елемента z = a * b. Це створює миттєвий лавинний ефект вже на рівні однієї бінарної операції. Така вбудована дифузія є суттєвою перевагою порівняно з лінійними операціями, як-от XOR, де зміна одного вхідного біта впливає лише на один вихідний біт.

Окрім дифузії, квазігрупи забезпечують високу нелінійність. Нелінійність є ключовим фактором для протидії лінійному та диференціальному криптоаналізу. У той час як в традиційних конструкціях на полях Галуа досягнення високої нелінійності часто пов'язане з обчислювально складними операціями, як інверсія, що використовується в AES, квазігрупи пропонують значно ширший клас потенційно нелінійних бінарних операцій [17]. Якщо квазігрупа не є ізотопною абелевій групі, тобто не може бути зведена до простої лінійної функції, її операція за своєю природою є нелінійною, що ускладнює апроксимацію лінійними або афінними функціями.

Однією з найважливіших переваг використання квазігруп, як основи для криптографічних перетворень, є величезний комбінаторний простір, який вони пропонують. Кількість різних неізотопних квазігруп, а отже й латинських

квадратів, зростає надзвичайно швидко зі збільшенням їхнього порядку n. Навіть для малих порядків, таких як n = 8, існують мільярди неізотопних квазігруп. Для порядку n = 16, який є типовим для сучасних блокових шифрів, ця кількість стає значно більшою. Таке розмаїття дає конструкторам шифрів значно більшу свободу порівняно з обмеженим набором операцій у полях Галуа. Це дозволяє цілеспрямовано шукати та конструювати квазігрупи або їхні представлення у вигляді латинських квадратів, які мають не лише високу нелінійність та дифузію, але й інші специфічні криптографічні властивості, оптимізовані для протидії конкретним атакам для прикладу низьку диференціальну рівномірність або для максимально ефективної апаратної реалізації [19].

Одне з прямих застосувань квазігруп в апаратній криптографії полягає у використанні їхньої таблиці Келі поданої як латинський квадрат, безпосередньо як блоку підстановки S-box. Якщо є квазігрупа $(Q,*)$ порядку $n=2^k$, її латинський квадрат n × n можна інтерпретувати як S-блок, що відображає 2k вхідних бітів у k вихідних біти [16]. Конкретніше, 2k вхідних бітів поділяються на два k-бітні блоки, скажімо a та b. Ці блоки представляють індекси рядка та стовпця, або операнди в латинському квадраті. Тоді результуючий елемент z знаходиться на їхньому перетині z = a * b. Це відрізняється від традиційних S-блоків, як в AES, які зазвичай відображають k бітів у k бітів (k × k). Однак така структура, отримана з квазігрупи, не є особливо ефективною для апаратної реалізації, оскільки вимагає таблицю пошуку необхідного значення. Тому для реалізації зі зменшенням апаратної складності необхідно переводити її у просту комбінаційну логіку.

Окрім використання як статичних S-блоків, квазігрупи також є перспективним рішенням для проектування потокових шифрів, особливо в контексті генераторів ключового потоку, що базуються на регістрах зсуву. Замість того, щоб використовувати лише лінійну операцію XOR для зворотного зв'язку, як в LFSR або додавати нелінійність лише на виході, як у фільтр-генераторах, операцію квазігрупи * можна інтегрувати безпосередньо в процес оновлення стану регістра. Наприклад, стан регістра може оновлюватися за

правилом, де новий біт залежить від нелінійної комбінації попередніх бітів за допомогою операції квазігрупи. Це дозволяє створювати структури, подібні до регістрів з нелінійним зворотним зв'язком, але засновані на іншій, потенційно більш різноманітній, алгебраїчній базі. Такий підхід був використаний у низці експериментальних потокових шифрів, демонструючи потенціал квазігруп для генерації складних, нелінійних послідовностей [15].

Таким чином використання квазігруп як алгебраїчної основи відкриває нові можливості для проектування ефективних криптографічних перетворень, особливо для потокових шифрів у сфері LW-криптографії. Їхні властивості поєднані з потенціалом компактної апаратної реалізації, роблять їх хорошою альтернативою класичним підходам. Саме тому доречно проаналізувати існуючі рішення для обґрунтування напрямку подальшого дослідження.

1.4 Аналіз існуючих рішень у малоресурсній криптографії

Для аналізу існуючих рішень у області LW-криптографії доцільно проаналізувати конкурс Lightweight Cryptography (LWC) ініційованим Національним інститутом стандартів і технологій США (NIST) з метою стандартизації криптографічних алгоритмів, оптимізованих для пристроїв з обмеженими ресурсами.

Традиційні криптографічні стандарти, як AES (Advanced Encryption Standard), хоч і є надзвичайно надійними, проте створювалися для настільних комп'ютерів та серверів. Вони можуть бути надто важкими для мікроконтролерів, RFID-міток, сенсорів Інтернету речей та інших вбудованих систем, які мають суттєві обмеження у обсягу пам’яті, низькій обчислювальній потужності центрального процесора чи обмеженому споживанню енергії. Метою конкурсу було обрати один або декілька алгоритмів, що забезпечують автентифіковане шифрування з приєднаними даними (AEAD) та, опціонально, гешування, які б найкраще відповідали цим вимогам.

Конкурс був відкритим і тривав з 2018 по 2023 рік, зібрав 56 початкових кандидатів від провідних криптографічних команд з усього світу. Ці алгоритми

пройшли декілька раундів інтенсивного публічного криптоаналізу, ретельної оцінки продуктивності та аналізу безпеки. Таким чином, 10 фіналістів цього конкурсу формують золотий стандарт та еталонний зріз сучасних легковагових рішень, з яким доцільно порівнювати будь-яку нову розробку в цій галузі.

За результатами багаторічного та всебічного аналізу, в лютому 2023 року Національний інститут стандартів і технологій США (NIST) офіційно оголосив переможцем конкурсу Lightweight Cryptography (LWC) сімейство алгоритмів ASCON. Це рішення стало кульмінацією п'ятирічного процесу, що залучив провідних світових експертів з криптографії до публічного аналізу та тестування десятків кандидатів. Перемога ASCON є знаковою подією, оскільки він став першим в історії стандартизованим портфоліо легковагових алгоритмів AEAD, що було закріплено у федеральному стандарті NIST SP 800-232. Цей документ тепер слугує офіційною рекомендацією для захисту даних у пристроях Інтернету речей (IoT) та інших середовищах з обмеженими ресурсами, де раніше домінували нестандартизовані або надто важкі рішення.

ASCON не новий, а вже добре перевірений часом набір криптографічних примітивів, розроблений у 2014 році командою дослідників з Грацького технічного університету (Австрія), Infineon Technologies, Intel Labs та Університету Радбауд. Його вибір не був несподіваним для криптографічної спільноти, адже ASCON раніше вже довів свою ефективність, ставши одним із переможців у категорії легковагових застосунків попереднього масштабного конкурсу CAESAR (Competition for Authenticated Encryption: Security, Applicability, and Robustness). Ця тривала історія публічного аналізу, що налічує майже десятиліття, надала NIST високу впевненість у надійності та безпеці даного сімейства алгоритмів.

Архітектура сімейства ASCON побудована навколо єдиного елегантного ядра — 320-бітної криптографічної перестановки, як прийнято називати інакше, пермутації, Ascon-p. Ця пермутація є основною функцією, що забезпечує нелінійність та дифузію. Вона оперує над внутрішнім 320-бітним станом і використовується в рамках губкової конструкції, що концептуально схожа на

підхід, стандартизований у SHA-3. Ця конструкція реалізує фазу втягування, коли вхідні дані блоками інтегруються в частину внутрішнього стану та перемішуються пермутацією, і фазу вижимки, коли вихідні дані формуються з іншої частини цього ж стану. Внутрішній стан ділиться на зовнішню частину (rate, r), що має розмір 64 або 128 біт і слугує для поглинання XOR блоків повідомлення та приєднаних даних, та внутрішню (capacity, c), що залишається прихованою і забезпечує 128-бітний рівень безпеки. Процес роботи складається з фази отримання даних послідовно поєднучись зі станом, і фази відправки, де генерується шифротекст або геш. Саме ця гнучка губкова архітектура дозволяє на основі єдиної пермутації Ascon-p реалізувати функціонал автентифікованого шифрування (Ascon-128, Ascon-128a), гешування (Ascon-Hash, Ascon-HashA) та функції розширеного виводу (Ascon-Xof, Ascon-XofA).

Ключовим фактором перемоги ASCON став його винятковий загальний баланс характеристик, а не перевага в якійсь одній вузькій категорії. Він продемонстрував чудову всебічну продуктивність. Алгоритм однаково ефективний як у програмних реалізаціях, демонструючи високу швидкість та низькі вимоги до RAM на 8-бітних та 32-бітних мікроконтролерах, так і в апаратних імплементаціях, досягаючи високої пропускної здатності при малій площі на кристалі FPGA та ASIC. Також його легкий та відносно простий дизайн значно спрощує реалізацію контрзаходів проти атак побічними каналами (SCA). Це означає, що захист від аналізу енергоспоживання (DPA) чи електромагнітного випромінювання (EMA) можна вбудувати з низькими накладними витратами, що є критичним для багатьох апаратних застосувань.

Таким чином, офіційна стандартизація ASCON встановлює новий глобальний еталон (baseline) для всієї галузі легковагової криптографії. Відтепер будь-який новий алгоритм, що пропонується для застосувань у пристроях з обмеженими ресурсами, неминуче буде порівнюватися саме з показниками безпеки, продуктивності та вартості реалізації ASCON.

Хоча ASCON і був обраний переможцем та новим стандартом, решта дев'ять фіналістів конкурсу LWC представляють надзвичайну наукову та

інженерну цінність. Кожен з них пропонує унікальний набір компромісів між рівнем безпеки, апаратною вартістю, програмною швидкістю та потенційною стійкістю до атак побічними каналами. Тому для систематичного аналізу сучасних підходів до проектування легковагових шифрів доцільно проаналізувати кожного з них.

Перша і найбільша група фіналістів, що включає також переможця ASCON це алгоритми, що базуються на губкових або тісно пов'язаних з ними пермутаційних конструкціях. Їх місце у фіналі підкреслює фундаментальний успіх губкової парадигми, популяризованої стандартом SHA-3. Цей підхід забезпечує елегантну гнучкість, дозволяючи на основі єдиної базової пермутації реалізувати як автентифіковане шифрування, так і гешування, що часто є вигідним з точки зору апаратної площі. До цієї групи увійшли Xoodyak, SPARKLE, PHOTON-Beetle та ISAP.

Xoodyak є одним з конкурентів ASCON. Цей алгоритм розроблений видатною командою криптографів, до якої входять Йоан Даймен (Joan Daemen) та Жиль Ван Ассхе (Gilles Van Assche), які є співавторами стандартів AES (Rijndael) та SHA-3 (Keccak). В основі Xoodyak лежить 384-бітна пермутація Xoodoo, яка була спеціально розроблена для високої продуктивності на 32-бітних програмних платформах ARM чи Cortex-M, зберігаючи при цьому хорошу ефективність в апаратурі.

Архітектурно Xoodyak використовує унікальний режим роботи, що називається "Cyclist mode". Він є варіацією губкової конструкції, що дозволяє більш гнучко та ефективно приймати вхідні дані та відправляти вихідні. Головною перевагою Xoodyak, що виділила його у фіналі, є виняткова програмна продуктивність, де він часто перевершував ASCON та інших фіналістів за швидкістю. При цьому він залишається конкурентоспроможним і в апаратних реалізаціях, демонструючи хороший баланс між складністю та пропускною здатністю.

Наступним помітним фіналістом у цій групі є SPARKLE, який слугує базовою пермутацією для набору криптографічних схем SCHWAEMM для

AEAD та ESCH для гешування. Ключовою особливістю цього кандидата є його інноваційний підхід до побудови самої пермутації. На відміну від багатьох інших шифрів, що покладаються на S-box як основне джерело нелінійності, SPARKLE використовує Add-Rotate-XOR (ARX) дизайн. Тобто, його нелінійність досягається за рахунок комбінації операцій модульного додавання (Add), циклічного зсуву (Rotate) та виключного АБО (XOR).

Такий ARX-підхід є надзвичайно корисним для деяких платформ, оскільки ці операції є зрозумілими для більшості сучасних мікропроцесорів, що забезпечує високу програмну швидкість. Також ARX-структури можуть пропонувати певну вбудовану стійкість до атак побічними каналами (SCA), зокрема до DPA (Differential Power Analysis), що більш важливо для апаратних реалізацій. Оскільки операції додавання та зсуву складніше аналізувати порівняно з простими табличними підстановками S-box, вартість додавання контрзаходів, наприклад маскуваня може бути нижчою. Таким чином, SPARKLE був представлений як рішення, що пропонує сильний захист від SCA, зберігаючи при цьому високу продуктивність.

PHOTON-Beetle ще один фіналіст, що використовує губкову конструкцію. Унікальність цього алгоритму полягає в тому, що він побудований на основі вже існуючого та добре вивченого примітиву 256-бітної пермутації PHOTON. Ця пермутація, спочатку розроблена як частина однойменної легковагової геш-функції яка, є міжнародним стандартом ISO/IEC 29192-5 та має архітектуру, що сильно нагадує AES. Вона використовує 4×4 матрицю стану, де кожен елемент є ніблом, і раундова функція складається з чітких кроків: AddConstant, SubCells, ShiftRows та MixColumnsSerial.

У такому алгоритму PHOTON це пермутація, а Beetle це назва самого режиму роботи AEAD, який використовує цю пермутацію. Головною перевагою та метою проектування PHOTON-Beetle є чудова оптимізація під апаратні реалізації. Використання 4-бітного S-box та послідовної операції MixColumns дозволяє досягти низького значення апаратної складності. Цей підхід робить його ідеальним кандидатом для сценаріїв, де кожен логічний елемент має

значення, наприклад RFID-мітках чи глибоко вбудованих сенсорах. Водночас, така архітектура, орієнтована на нібли, є менш ефективною у програмних реалізаціях на 32-бітних чи 64-бітних процесорах, де він суттєво програє у швидкодії ARX-шифрам як SPARKLE чи програмно-орієнтованим губкам як Xoodyak.

ISAP ще один алгоритм, що базується на губковій конструкції, але він має одну надзвичайно важливу та унікальну мету, а саме вбудована стійкість до певних класів атак побічними каналами, зокрема до DFA (Differential Fault Attacks), тобто атак, що базуються на внесенні збоїв. У той час як більшість алгоритмів потребують додаткових контрзаходів, маскування або дублювання обчислень для захисту від SCA, що збільшує їхню площу та знижує продуктивність, ISAP був розроблений таким чином, щоб бути безпечним за замовчуванням на рівні самого режиму роботи.

Для досягнення цієї мети ISAP комбінує два елементи. Першим є апаратно-орієнтована 256-бітна пермутація PHOTON, що забезпечує малу апаратну площу. Також він використовує специфічний дуплексний режим губкової конструкції. Головна ідея полягає в тому, що після фази ініціалізації алгоритм поглинає один блок відкритих даних, а перед витисканням блоку шифротексту він виконує додаткову пермутацію. Це означає, що внутрішній стан перемішується навіть тоді, коли нові дані не вводяться. Цей підхід значно ускладнює для атакуючого можливість внесення контрольованого збою та аналізу його впливу на вихідні дані, забезпечуючи високий рівень стійкості до DFA. Ціною за таку вбудовану безпеку є дещо нижча пропускна здатність порівняно з більш агресивно оптимізованими шифрами.

Другою архітектурною парадигмою серед фіналістів LWC є класичні конструкції на основі блочних шифрів. На відміну від губкових конструкцій, які використовують одну пермутацію для обробки даних, цей підхід розділяє логіку на два компоненти, а саме блочний шифр та режим автентифікованого шифрування, в якому цей шифр використовується. Хоча стандартний AES-GCM вважається занадто важким для мікроконтролерів, фіналісти LWC довели, що

при використанні легковагового блочного шифра ця архітектура залишається абсолютно конкурентоспроможною. До цієї групи належать GIFT-COFB, Romulus та Elephant.

Одним з представників такої групи є GIFT-COFB. Він поєднує два компоненти, як GIFT, що є надзвичайно легковаговим блочним шифром, та COFB (Combined Feedback), що є ефективним режимом AEAD. Сам блочний шифр GIFT, поданий у двох варіантах, з 64-бітним та 128-бітним блоком, був розроблений спеціально для апаратних реалізацій. Його архітектура базується на мережі підстановок-перестановок (SPN), подібно до AES, але використовує 4-бітні S-box, подібно до PRESENT та просту бітову перестановку, що робить його вкрай компактним в апаратурі.

Режим COFB, в свою чергу, є ефективним режимом автентифікованого шифрування, що потребує лише прямого напрямку роботи блочного шифру, що дозволяє додатково зекономити площу на кристалі. Завдяки такій комбінації, GIFT-COFB продемонстрував одну з найменших апаратних реалізацій серед усіх фіналістів, Це зробило його головним кандидатом для сценаріїв, де апаратна складність є вирішальною компонентою.

Наступним кандидатом у цій групі є Romulus. Це сімейство алгоритмів AEAD, яке, подібно до GIFT-COFB, також базується на вже існуючому та добре перевіреному легковаговому блочному шифрі. В його основі лежить SKINNY так званий налаштовуваний (tweakable, TBC) блочний шифр. Концепція TBC означає, що шифр приймає не лише ключ і відкритий текст, але й третій вхідний параметр, твік, який дозволяє ефективно змінювати функцію шифрування для кожного блоку без дорогої операції зміни ключа. Це робить TBC надзвичайно ефективними для побудови режимів автентифікованого шифрування.

Головною перевагою та інженерною метою Romulus є забезпечення стійкості до неправильного використання. Багато стандартних шифрів можуть втратити безпеку, якщо користувач випадково повторно використає ту саму пару (ключ, нонс) для шифрування двох різних повідомлень. Romulus розроблений так, щоб протистояти цьому сценарію. Навіть при повторному використанні

нонса, він гарантує збереження конфіденційності даних, хоча й не може зберегти автентичність, що є неминучим у такій ситуації. Ця властивість робить його дуже цінним для систем, де генерація унікальних нонсів не може бути надійно гарантована. Його апаратна реалізація є дещо більшою за GIFT-COFB, але він пропонує цей важливий додатковий рівень безпеки.

Найбільш гібридним та архітектурно складним дизайном серед усіх фіналістів є алгоритм Elephant. Він намагається поєднати переваги губкових конструкцій та паралелізму блокових шифрів. В його основі лежить широка пермутація, схожа на ті, що використовуються у губкових конструкціях, але режим її роботи імітує паралельну обробку, притаманну блоковим шифрам. Дизайн складається з двох версій: Elephant-v1 та Elephant-v2, що використовують різні базові пермутації Keccak-p[200] та SPONGENT-p[176] відповідно.

Ключова ідея Elephant полягає у розділенні обробки даних. Він використовує губкову конструкцію лише для обробки приєднаних даних та одноразового числа (нонсу), тоді як саме повідомлення шифрується за допомогою окремого потокового шифру, що генерується на основі ключа та стану губки. Цей підхід дозволяє досягти високого паралелізму при шифруванні самого повідомлення, що може бути перевагою у високошвидкісних апаратних реалізаціях. Однак, така подвійна архітектура є значно складнішою для реалізації та аналізу порівняно з чистими губковими чи блоковими підходами, що стало однією з причин, чому NIST віддав перевагу простішому та елегантнішому дизайну ASCON.

Третя група фіналістів LWC складається з алгоритмів, що представляють вузькоспеціалізовані, але критично важливі ніші у легковаговій криптографії. Коли губкові та блочні шифри змагалися за найкращий загальний баланс характеристик, то алгоритми цієї групи були розроблені з однією ключовою метою. Ця група складається з TinyJambu, що націлений на абсолютний мінімум апаратної складності, та Grain-128AEAD, єдиного представника потокових шифрів у фіналі.

TinyJambu є яскравим прикладом цільового дизайну, де головною і єдиною метою було досягнення найменшої апаратної складності. Його архітектура надзвичайно мінімалістична і базується на єдиному 128-бітному регістрі зсуву з нелінійним зворотним зв'язком (NFSR), що віддалено нагадує потокові шифри, але реалізований у режимі AEAD. Цей дизайн вимагає дуже малої кількості логічних елементів, що робить його абсолютним лідером за показником Gate Equivalents (GE) серед усіх фіналістів.

Така оптимізація задля зменшення апартної складності досягається ціною пропускної здатності. Алгоритм є значно повільнішим за ASCON, Xoodyak чи GIFT-COFB, оскільки він обробляє дані дуже малими порціями, по 32 біти за багато тактів. Проте його існування у фіналі доводить, що існує цілий клас пристроїв, наприклад, пасивні RFID-мітки, імплантовані медичні сенсори, де вартість, виміряна в площі кристалу, є набагато важливішою за швидкість шифрування. Він довів, що шифри з мінімальним станом та серіалізованою обробкою є життєздатним напрямком.

Grain-128AEAD єдиний класичний потоковий шифр, що дійшов до фіналу конкурсу NIST LWC. Всі інші фіналісти, включно з губками та блоковими шифрами, працюють за принципом обробки даних дискретними блоками. Grain, натомість, є представником сімейства шифрів, що генерують нескінченний псевдовипадковий ключовий потік, який потім за допомогою операції XOR поєднується з відкритим текстом для отримання шифротексту.

Архітектурно, Grain-128AEAD є розвитком відомого сімейства потокових шифрів Grain. Його ядро складається з двох регістрів зсуву, одного лінійного (LFSR) та одного нелінійного (NFSR), загальним розміром 256 біт. Ключовий потік та тег автентифікації генеруються за допомогою складної нелінійної функції, що комбінує біти зі станів обох регістрів. Його перевагами є мала апаратна складність та дуже низьке енергоспоживання, оскільки його логіка є простою та послідовною.

У таблиці 1.1 наведено порівняльні характеристики досліджених потокових шифрів.

Таблиця 1.1 – порівняльні характеристики досліджених потокових шифрів

Алгоритм	Ключ (біт)	Апаратна складність (GE)
ASCON-128	128	2650 - 3700
Xoodyak	128	8100
SPARKLE	128	7400
PHOTON-Beetle	128	2380
ISAP	128	2750
GIFT-COFB	128	2400
Romulus-M	128	4200
Elephant	128	5300
TinyJambu-128	128	1960 - 2700
Grain-128AEAD	128	2400

Проаналізовані криптографічні рішення демонструють широкий спектр різних реалізацій та методів підвищення стійкості до криптографічних атак. Всі алгоритми так чи інакше показували свої досягнення у криптографії та особливості їх використання. Таким чином актуальність досліджень у сфері криптографії з метою розробки різних за способом використання та архітектурними особливостями алгоритмів шифрування є високою.

1.5 Висновки до розділу

У першому розділі було проведено огляд інформаційних джерел, який дозволяє зрозуміти сучасні виклики у галузі потокового шифрування. Зокрема розділ підтверджує актуальність теми дослідження опираючись на сучасні потреби цифрового світу.

Стрімкий розвиток цифрових технологій створює нові канали витоку інформації у зв’язку з збільшенням вузлів обробки інформації. Це підкреслює важливість захисту даних у різних точках інформаційних систем шляхом розробки стійких та ефективних алгоритмів шифрування.

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.

Потокові шифри є одними з найкращих кандидатів для легковагових апаратних рішень. Їхня архітектура забезпечує мінімальну затримку, відсутність потреби в буферизації даних та високий потенціал для компактної реалізації.

Зі зростанням обсягів даних існує актуальна науково-технічна задача розробки нових архітектур потокових шифрів, які б досліджували альтернативні алгебраїчні основи для досягнення конкурентних показників апаратної складності та безпеки. Використання квазігруп є перспективним та недостатньо дослідженим напрямком.

Квазігрупи забезпечують вбудовану дифузію та високу нелінійність. Великий комбінаторний простір дає значно ширші можливості для пошуку операцій, що оптимально поєднують криптографічні властивості та апаратну компактність.

2 РОЗРОБКА ТА ОЦІНЮВАННЯ ОПЕРАЦІЙНОГО БЛОКУ ШИФРУ

Другий розділ присвячений розробці ключових апаратних блоків алгоритму потокового шифрування на основі латинських квадратів.

2.1 Структурна модель засобу шифрування

Операційний блок реалізує роботу алгоритму потокового шифрування на основі квазігруп. Модель такого блоку містить опис його структурних блоків на основі теоретичної основу алгоритму.

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

Одним із структурних блоків алгоритму є генератор ПВП який детально проаналізовано та реалізовано у першій частині роботи. Ключовою задачею такого генератора є генерація псевдовипадкового нескінченного потоку біт для роботи ядра шифрування ініціалізацію якого виконує ключ *K*. Такий потік значень є однією з компонент забезпечення стійкості алгоритму.

Оскільки алгоритм є потоковим, то обробка значень буде відбуватись по одному за такт по два біти, таким чином алгоритм зможе у режимі потоку формувати з вхідного повідомлення результуючий шифротекст.

Вхідне повідомлення інтерпретується як скінченний потік біт, які будуть тактово подаватись до ядра шифрування з метою їх зашифрування. Тому такий набір даних, у контексті роботи алгоритму буде вважатись послідовністю біт повідомлення *M*.

Основою стійкості алгоритму є ключ, оскільки він відіграє ключову роль у формуванні зашифрованої послідовності біт. Ключ *K* хоч і є вхідними даними для роботи алгоритму та аналогічно повідомленню інтерпретується ним як потік біт, але він є скінченним, що неприпустимо у контексті потокового шифрування. Таким чином важливо забезпечити процес його відновлення та стійкість таких нових значень.

Для забезпечення стійкості такого потоку нових бітів ключа доцільно використати квазігрупу описану латинським квадратом, що задовольняє ключові параметри стійкості та забезпечить складну логіку формування наступних значень для запобігання компрометації ключового потоку.

Тоді під час такту роботи алгоритму на основі біт ключа та повідомлення поточного такту і логічного виразу що описує таку квазігрупу буде сформоване нове значення ключа.

Таблиця Келі Q_0 для оновлення значень ключа наведено на рисунку 2.1.

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

Рисунок 2.1 – Таблиця Келі генератора гами

Однією з властивостей такого квадрату є мінімальна асоціативність, оскільки головна операція віддалена від звичайної арифметики груп. Це критично важливо для потокових шифрів на квазігрупах, оскільки руйнує алгебраїчні структури, якими міг би скористатися зловмисник.

Таким чином процес вибору нового значення ключа описується наступними логічними виразами:

$$O[1] = K[1] \oplus M[0]$$

$$O[0] = K[0] \oplus M[1] \oplus (K[1] \wedge NOT(M[0]))$$

де O біти нового значення ключа, M біти повідомлення, K біти поточного ключа.

Хоч у арифметиці така операція є лінійною, але у булевій алгебрі вона подається як нелінійна за рахунок операції ∧ (AND), що уникає лінійності та роблячи зв'язок між старим і новим ключем складним і непередбачуваним. Таким чином забезпечується надійність потоку ключа і забезпечується захист від атак на ключ.

Таким чином структура генератора гами наведено на рисунку 2.2.

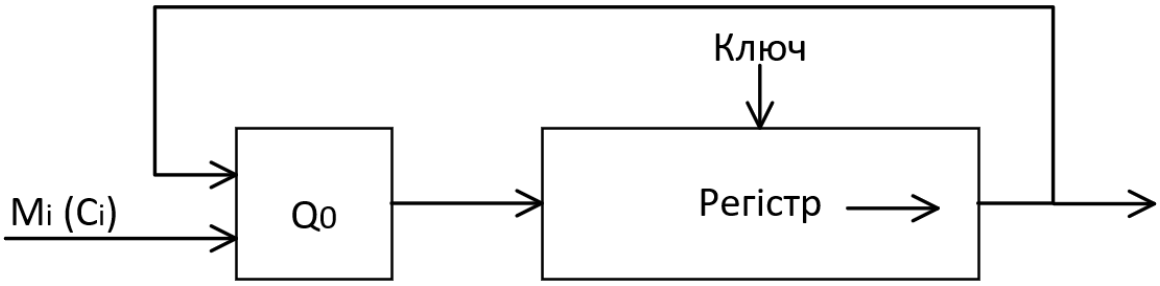


Рисунок 2.2 – Генератор гами

Для реалізації функціоналу алгоритму шифрування буде використано два набори квазігруп. Перший набір є прямими квазігрупами описаними латинськими квадратами, що забезпечують операцію зашифрування. Для симетричного відтворення зишифрованих значень логічно обрати набір з обернених квазігруп, які забезпечать точне відтворення початкових біт.

Набір таблиць Келі для процесу зашифрування 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

Тоді процес зашифрування на основі таких квадратів описується наступними логічними виразами:

$$LS1(M, X) = M[0] \oplus X[1] \oplus X[0], M[1] \oplus M[0] \oplus X[1]$$

$$C = LS1(M, X) \oplus K$$

де LS1 лінійне перетворення, X біти генератора ПВП, C шифротекст.

Властивості операції XOR ($\oplus$) дозволяють скоротити вирази, та переписати їх у компактнішому вигляді:

$$C[1] = M[0] \oplus X[1] \oplus X[0] \oplus K[1]$$

$$C[0] = M[1] \oplus M[0] \oplus X[1] \oplus K[0]$$

де C біти шифротексту.

Набір Келі для процесу розшифрування наведено на рисунку 2.4.

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 2.4 – Набір обернених таблиць Келі

Тоді процес розшифрування на основі таких квадратів описується наступними логічними виразами:

$$C' = C \oplus K$$

$$LS1^{-1}(C', X) = C'[1] \oplus C'[0] \oplus X[0], C'[1] \oplus X[1] \oplus X[0]$$

де $LS1^{-1}$ обернене лінійне перетворення, X біти генератора ПВП, C шифротекст, C' проміжне значення, що отримується після накладання ключа на шифротекст.

Такі вирази також можна скоротити та переписати у компактнішому вигляді:

$$M[1] = C[1] \oplus C[0] \oplus K[1] \oplus K[0] \oplus X[0]$$

$$M[0] = C[1] \oplus K[1] \oplus X[1] \oplus X[0]$$

На основі описаних особливостей алгоритму можна описати узагальнені кроки його виконання.

Алгоритм процесу зашифрування вхідного повідомлення описується кроками:

- Введення вхідних даних *M* та ключа *K*.
- Генерування псевдовипадкової послідовності *S* на основі ключа *K*.
- Передавання біт повідомлення *M*, ключа *K*, ПВП *S* та режиму роботи 0 для зашифрування до блоку шифрування.
- Виконання головної операції блоку шифрування та визначення наступного значення ключа.
- Запис результату операційного блоку в шифротекст *C* та ключ *K*.
- Зсув послідовностей *M*, *K*, *S* та *K*.

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 біт повідомлення *C,* ключа *K*, ПВП *S* та режиму роботи 1 для розшифрування до блоку шифрування.
– Виконання головної операції блоку шифрування та визначення наступного значення ключа.
– Запис результату операційного блоку у відкритий текст *M* та ключ *K.*
– Зсув послідовностей *C, K*, *S* та *K*.

і так до моменту вичерпання вхідного повідомлення.

Апаратна реалізація такого алгоритму буде складатись з трьох блоків, а саме ядро шифрування для виконання власне операції зашифрування чи розшифрування біт, генератор ПВП який описаний у першій частині роботи та відповідає за генерацію псевдовипадкової послідовності і контролера який відповідає за ключову логіку роботу всієї системи і керує всіма блоками для досягнення результату роботи.

Таким чином схема роботи алгоритму роботи апаратного засобу зображена на рисунку 2.5.

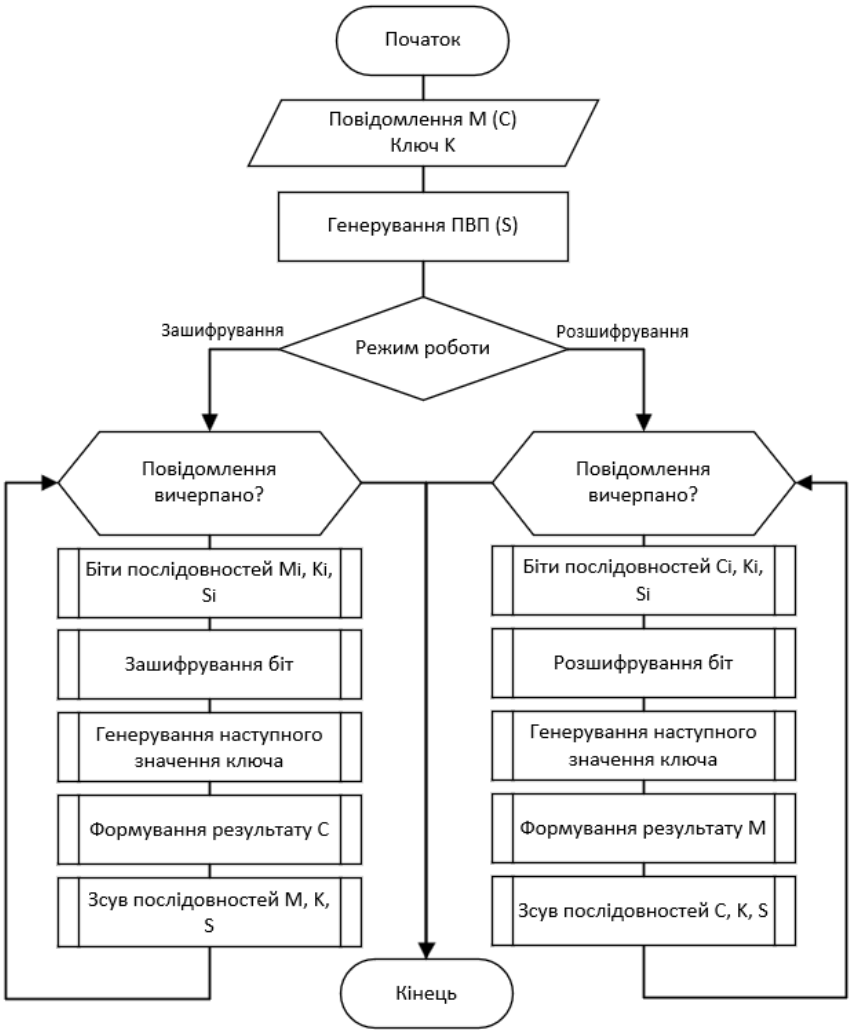


Рисунок 2.5 – Схема алгоритму апаратного засобу

Такі блоки забезпечують апаратну реалізацію описаного алгоритму потокового шифрування на основі квазігруп.

2.2 Архітектура апаратного блоку шифрування

Перехід від формальної структурної моделі до апаратної реалізації вимагає чіткого проектування архітектури та декомпозиції системи на керовані логічні блоки. Обрана архітектура базується на ієрархічному принципі, де вся система поділена на три незалежні модулі, що взаємодіють через чітко визначені інтерфейси.

Така декомпозиція дозволяє ізолювати логіку керування, логіку генерації псевдовипадкової послідовності та логіку криптографічного перетворення.

Загальна структурна схема операційного блоку, що ілюструє цю взаємодію, наведена на рисунку 2.6.

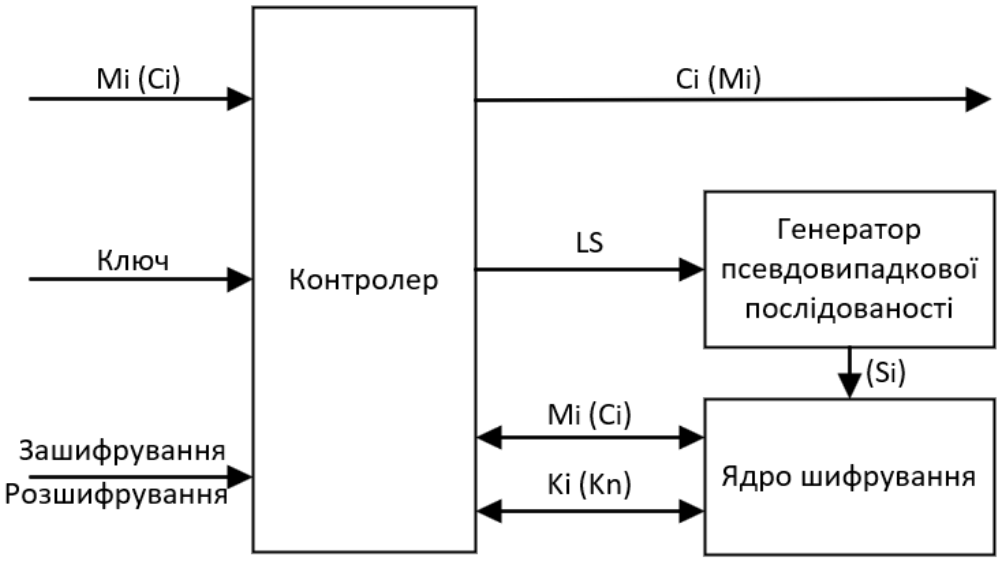


Рисунок 2.6 – Загальна структурна схема операційного блоку

З рисунку можна побачити, що апаратна реалізація алгоритму складається з трьох блоків, які взаємодіють між собою.

Ядро шифрування є центральним обчислювальним блоком, спроектованим як суто комбінаційний модуль. Це означає відсутність внутрішніх регістрів чи стану, оскільки модуль миттєво обчислює вихідні значення на основі поточних вхідних сигналів. Така реалізація мінімізує затримки та спрощує інтеграцію з керуючим автоматом.

Для виконання обчислень модуль приймає на вхід три два бітні потоки даних: біти ключа *K*, біти від генератора ПВП *S* та біти вхідного повідомлення *M(C)*. Додатковий одно бітний керуючий вхід визначає режим роботи, де ‘0’ для зашифрування та ‘1’ для розшифрування.

Внутрішня логіка модуля паралельно виконує операції зашифрування, розшифрування, оновлення значення ключа *Kn*.

В залежності від обраного режиму роботи модуль виконує обчислення зашифрованих чи розшифрованих біт. При тому логіка відновлення нового значення ключа також залежить від режиму роботи. Під час зашифрування використовується відкритий текст, а під час розшифрування розшифрований

відкритий текст, що забезпечує синхронність оновлення ключа на сторонах відправника та одержувача.

На основі цих обчислень модуль формує два виходи по два біти, а саме результат шифрування чи розшифрування та новий два бітний сегмент ключа.

Модуль генератора ПВП є синхронним пристроєм, відповідальним за генерацію псевдовипадкової послідовності. Його робота базується на використанні квазігрупних операцій.

Інтерфейс модуля призначений для взаємодії з керуючим FSM. Для початку його роботи контролер передає значення LS для вибору пари квазігруп. Модуль отримує тактовий сигнал, сигнал асинхронного скидання та сигнал запуску. Сигнал запуску є імпульсним і ініціює генерацію одного два бітного блоку ПВП. Вхідний керуючий сигнал дозволяє обрати одну з чотирьох пар квазігруп для генерації послідовності. Після завершення обчислень модуль видає два бітний результат та встановлює одно бітний прапорець готовності.

Внутрішній стан генератора зберігається у трьох два бітних регістрах. При скиданні вони ініціалізуються ненульовими константами для уникнення тривіальних станів.

Внутрішня комбінаційна логіка є двоступеневою та виконує два послідовні обчислення для генерації наступного стану генератора. Перший блок обчислюється проміжне два бітне значення. Після чого наступний блок обчислює фінальне два бітне значення. Цей блок використовує як входи результат першого етапу та третій регістр стану. У таких блоках квазігрупи описані у вигляді логчних виразів, що зменшує їх апаратну складність.

На відміну від простого вибору, два бітний сигнал входу керує вибором пари квазігруп. Він подається на логіку, що генерує два незалежні два бітні індекси. Таким чином, вхідний сигнал дозволяє обрати які квазігрупи використовувати, що значно посилює нелінійність перетворення.

Процес оновлення стану та генерації відбувається при отриманні сигналу початку. Нове значення для генератора обчислюється нелінійно. На вихід при

цьому видається значення, яке знаходилось у результаті роботи до оновлення його стану, забезпечуючи коректну послідовність.

Модуль контролера керує всією системою та виконує дві основні функції. Перша це інтеграція модулів ядра шифрування та генератора ПВП, керуючи їх взаємодією. Друга це обробка потоку даних та реалізація скінченного автомату (FSM), що керує усім конвеєром обробки.

Зовнішній інтерфейс модуля забезпечує зв'язок із зовнішнім середовищем. Він приймає початковий 64-бітний ключ, сигнал старту сесії, потік вхідних даних, відкритий чи зашифрований текст, та видає потік вихідних даних, як результат обробки, і сигнали стану.

Ключовим елементом потоку даних є логіка формування потоку ключа K. Для цього в модулі реалізовано 64-бітний регістр зсуву. На відміну від простої пам'яті, цей регістр функціонує як регістр зсуву з нелінійним зворотним зв'язком.

У стані очікування *WAIT_DATA*, керуючий автомат обирає два старші біти з регістра ключа і подає їх на вхід ядра шифрування. Після чого, у стані *PROCESS*, коли ядро шифрування обчислило нові два біти ключа, регістр ключа оновлюється. Регістр зсувається на два біти, відкидаючи старші біти, а на звільнені молодші позиції записуються нові два біти, отримані від ядра шифрування. Це забезпечує постійне оновлення 64-бітного внутрішнього стану ключа.

Управління всіма операціями та синхронізацією потоків виконує скінченний автомат (FSM), що реалізує 7 станів . Діаграма станів FSM зображена на рисунку 2.7

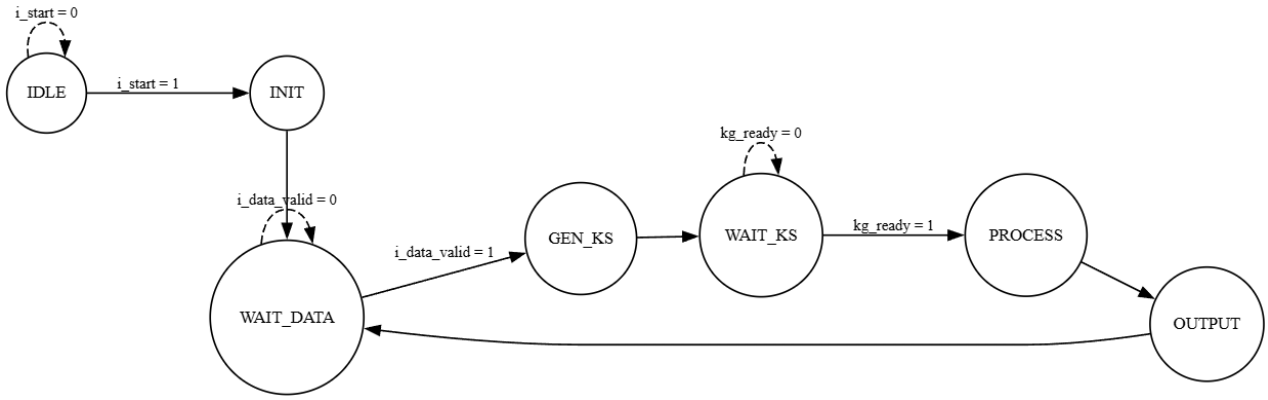


Рисунок 2.7 – Діаграма станів керуючого автомату (FSM)

Логіка автомату FSM описується такими станами:

– IDLE: Початковий стан очікування. При отриманні сигналу старт, FSM переходить до ініціалізації.
– INIT: Стан одноразової ініціалізації сесії, під час якого ключ завантажується у регстір, фіксується режим роботи, і система сигналізує про готовність приймати дані.
– WAIT_DATA: Основний стан очікування. FSM чекає на дані вхідні дані, коли дані надходять, автомат фіксує їх та поточні два біти ключа і переходить до генерації ПВП.
– GEN_KS: Стан запуску ПВП. FSM подає тактовий імпульс початку на генератор ПВП.
– WAIT_KS: Стан очікування ПВП. FSM чекає на сигнал готовності від генератора. Отримавши його, FSM фіксує два біти ПВП.
– PROCESS: Стан обробки. У цей момент усі три компоненти *M(C), K, S* готові. Ядро шифрування комбінаційно обчислює результат. FSM виконує єдину синхронну операцію зсуву ключа.
– OUTPUT: Стан видачі результату. FSM виставляє обчислені біти результату на вихід та сигналізує про їх готовність.

Після видачі результату, FSM негайно повертається у стан WAIT_DATA для обробки наступного два бітного блоку, забезпечуючи безперервний потоковий конвеєр.

2.3 Розробка тестового оточення та методологія верифікації

Тестування є важливим етапом проектування, що підтверджує функціональну коректність розроблених модулів апаратного засобу. Для забезпечення надійності операційного блоку виконано багатоступеневий процес верифікації отриманих результатів розробки.

Такий підхід дозволить перевірити працездатність кожного з модулів та в результаті перевірити працездатність всіх модулів разом як одну систему. Тому

така стратегія буде складатись з модульного тестування та інтеграційного тестування.

Модульне тестування спрямоване на перевірку коректності функціонування найменших логічних компонентів програми, модулів або юнітів. Основна мета якого полягає в ізоляції кожної частини системи та демонстрації того, що окремі її компоненти працюють правильно. Таке тестування, як правило, виконується на ранніх етапах життєвого циклу розробки і слугує фундаментом для побудови більш складних рівнів тестування, таких як інтеграційне та системне [20]. Це дозволяє виявляти та усувати дефекти на ранній стадії, що значно спрощує процес їх виправлення.

Таким чином буде виконано ізольовану перевірку боків ядра шифрування та генератора ПВП для підтвердження коректності їхньої внутрішньої логіки.

Інтеграційне тестування є важливим етапом життєвого циклу розробки, основним завданням якого є перевірка коректної взаємодії та спільної роботи індивідуальних модулів системи після їх об'єднання. Таке тестування спрямоване на виявлення дефектів, що виникають на межах компонентів, включаючи невідповідності в інтерфейсах, помилки передачі даних та проблеми з глобальними структурами даних. Дослідники підкреслюють, що особливістю цього рівня тестування є необхідність перевірки саме взаємодії між вже протестованими модулями, що вимагає спеціальних підходів до організації тестового процесу та пріоритизації тестових сценаріїв [21]

Виходячи з цього, буде перевірено модуль контролера, який об'єднує всі компоненти. Такий процес приділяє увагу коректній взаємодії модулів, роботі FSM та правильній організації загального конвеєра обробки даних.

Для модульного тестування ядра шифрування та генератора ПВП розроблено відповідні тестові стенди для перевірки правильності роботи.

Особливістю тестового стенду для ядра шифрування є автоматизований процес запуску тестових векторів. Він реалізує фундаментальний для криптографії тест зі зворотним зв'язком, та автоматично виконує три послідовні кроки:

– Шифрування: Встановлює режим шифрування та подає набір тестових даних *M*, *K*, *S*, після чого фіксує вихідний шифротекст *C*.
– Розшифрування: Одразу подає отриманий шифротекст *C* назад на вхід модуля, змінивши режим на розшифрування.
– Перевірка: Стенд автоматично порівнює кінцевий вихід що є розшифрованим початковим повідомленням з початковим оригінальним повідомленням. У разі повного збігу тест вважається пройденим в іншому випадку ні.

Таке тестування буде проведено з різними наборами вхідних векторів з метою комплексної перевірки модуля.

Результати проходження тестування модуля ядра шифрування наведено на рисунку 2.8

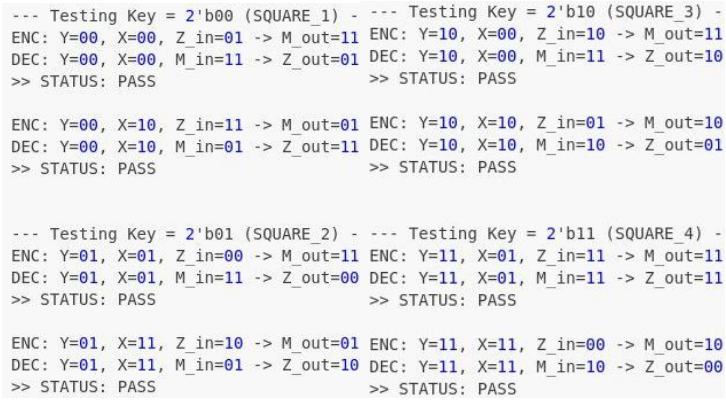


Рисунок 2.8 – Результати тестування модуля ядра шифрування

З рисунка видно, що модуль ядра шифрування успішно пройшов тести.

Тестовий стенд для модуля генератора ПВП має іншу мету, адже він фокусується на коректності оновлення його внутрішнього стану та синхронній поведінці. Таким чином він ініціалізує роботу генератора з метою перевірки генерування псевдовипадкової послідовності.

Оскільки генератор ПВП має видавати непередбачувану послідовність, автоматична перевірка тут є недоцільною. Тому основне завдання для тестового середовища буде у моніторингу змін виходу генератора ПВП. Визначивши будь-які зміни на виході генератора стенд виводить у консоль кожне нове згенероване 2-бітне значення разом із часом симуляції. Це дозволяє візуально проаналізувати

згенеровану послідовність на предмет повторень значення, нульових векторів чи інших аномалій, підтверджуючи працездатність логіки генератора.

Результати проходження тестування модуля генератора ПВП наведено на рисунку 2.9

```
Selected LS pair: 01
Time(ns) | Keystream Output
--------------------------
      20 | 00
      25 | 11
      45 | 10
      55 | 11
      65 | 10
      75 | 00
      85 | 11
      95 | 00
     105 | 10
     115 | 01
     125 | 11
     145 | 00
     165 | 10
     175 | 11
     185 | 00
     195 | 11

--- Simulation Finished ---
```

Рисунок 2.9 – Результати тестування модуля генератора ПВП

Тестування модуля генератора ПВП також показує правильність генерування потоку значень без колізій чи зациклення значень.

Після підтвердження коректності окремих модулів доцільно виконати інтеграційне тестування зібраного операційного блоку контролера. Для цього розроблено тестовий стенд, який емулює повний цикл роботи з пристроєм у апаратному середовищі.

Тестовий стенд для такого модуля є значно складнішим і реалізує високорівневу абстракцію взаємодії із засобом. Його основними процесами є очікування, передача та обробка даних з використанням під’єднаних модулів ядра шифрування та генератора ПВП. Такий стенд також виконує операцію самоперевірки, для валідації отриманого відкритого повідомлення з нещодавно зашифрованих значень.

Результати проходження тестування модуля контролера наведено на рисунку 2.10

```
=== TEST 1: Encrypt/Decrypt "hi" (2 bytes) ===
Encrypting...
Plaintext:  "hi" (hex: 68 69)
Ciphertext: (hex: 9b 4e )
Decrypting...
Decrypted:  "hi" (hex: 68 69 )
>>> TEST 1 PASSED


=== TEST 2: Encrypt/Decrypt "hello" (5 bytes) ===
Encrypting...
Plaintext:  "hello" (hex: 68 65 6c 6c 6f)
Ciphertext: (hex: 9b 46 07 d0 48 )
Decrypting...
Decrypted:  "hello" (hex: 68 65 6c 6c 6f )
>>> TEST 2 PASSED: Full "hello" encrypted/decrypted correctly!


=== TEST 3: Encrypt/Decrypt "helloworld" (10 bytes) ===
Encrypting...
Plaintext:  "helloworld"
Ciphertext: (hex: 6d b7 16 30 6d 79 e9 7c 24 dc )
Decrypting...
Decrypted:  "helloworld" (hex: 68 65 6c 6c 6f 77 6f 72 6c 64 )
>>> TEST 3 PASSED: 10 bytes OK
```

Рисунок 2.10 – Результати тестування модуля контролера

Тестування модуля контролера також пройшло успішно і всі вхідні дані були зашифровані та правильно розшифровані.

Такий багатоетапний підхід до перевірки всіх модулів забезпечує високу впевненість у функціональній коректності усього розробленого операційного блоку. Це підтверджує, що пристрій готовий до подальшої інтеграції у складніші апаратно-програмні системи.

2.4 Аналіз та оцінка операційного блоку

Після проектування та тестування операційного блоку необхідно оцінити апаратну складність його реалізації. Такий показник є важливим оскільки розробка передбачена для сфери LW-криптографії, тому отримане значення має бути у межах норми та бути якомога меншим.

Оцінка складності апаратного засобу відбувається на основі таблиці 2.1.

Таблиця 2.1 – 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

Генератор ПВП було розроблено у першій частині роботи. Він забезпечує потік псевдовипадкових значень для виконання готової операції операційного блоку. Такий модуль містить 9 одно бітних тригерів (FF) для зберігання внутрішнього стану та вихідних значень. Їх апаратна складність становить 48 GE. Комбінаційні блоки що реалізують математику чотирьох квазігруп та вибір між ними складаються з мультиплексорів та XOR-елементів та оцінюютсья у 51.94 GE. Загальна складність модуля генератора ПВП становить 108.97 GE.

Модуль ядра шифрування є суто комбінаційною схемою. Його складність формується виключно логічними елементами. Реалізація прямих та обернених перетворень включає дванадцять XOR-елементів, що оцінюються у 32.04 GE. Мультиплексори вибору режиму та логіка оновлення ключа додають ще 20 GE складності. Загальна складність модуля ядра шифрування є надзвичайно низькою і становить 52.04 GE.

Контролер реалізує керування всіма модулями та сигналами у засобі, тому він має найбільшу апаратну складність та кількість необхідних компонентів. Кількість тригерів для реалізації такого модуля становить 84, з яких 64 використовується для ключа. Тому апаратна складність регістрів становить 447.72 GE. Комбінаційна логіка, що реалізує скінченний автомат керування та формування вихідних сигналів, потребує додатково 93 GE, у тому числі 23 GE для FSM та 70 GE для вихідної логіки. Таким чином, сумарна апаратна складність модуля контролера становить 540.72 GE, що підтверджує його найбільшу складність серед усіх модулів засобу.

Складність кожного блоку апаратної реалізації засобу шифрування наведено у таблиці 2.2.

Таблиця 2.2 – Складність блоків засобу шифрування

Блок	Апаратна складність (GE)
Генератор ПВП	108.97
Ядро шифрування	52.04
Контролер	540.72
Всього	701.73

Таким чином апаратна складність всього засобу становить 701.73 GE. Отримане значення знаходиться у межах допустимих для LW-криптографії та демонструє те що розроблений алгоритм є перспективним для використання у малоресурсних пристроях. Апаратна складність такого засобу може змінюватись зі зміною довжини ключа чи особливостей реалізації FSM чи логіки виведення оброблених значень.

Якщо порівняти отриманий апаратний засіб з учасниками конкурсу NIST Lightweight Cryptography який був зосереджений на легковагових алгоритмах, можна побачити, що реалізований алгоритм значною мірою випереджає учасників за показником апаратної складності. Навіть якщо перерахувати результати для реалізації зі 128-бітним ключем, що є вимогою конкурсу, основні апаратні витрати зростуть лише за рахунок регістра ключа, на 341.12 GE. Підсумкова апаратна складність у 1042.85 GE все одно залишається значно нижчою, ніж у багатьох конкурентів, наприклад, у стандарта PRESENT-80, складність якого 1570 GE.

Отримані результати показують перспективу використання такого алгоритму у системах зі значними апаратними обмеженнями. При тому дане дослідження та результати демонструють перспективи дослідження квазігруп як математичної основи для побудови легковагових алгоритмів шифрування.

2.5 Висновки до розділу

У другому розділі було виконано реалізацію алгоритму потокового шифрування на основі казігруп. Для досягнення мети було описано теоретичну частину та структурну модель апаратного засобу.

Наступним кроком було спроектовано апаратну архітектуру засобу шифрування. Для цього було проведено декомпозицію системи на три ключові, логічно ізольовані модулі. Виконано процес проектування та опису кожного з модулів такого засобу.

Розроблено тестові оточення та проведено комплексну верифікацію за допомогою багатоетапної методології тестування. Для цього було провдено модульне та інтеграційне тестування для розробленого апаратного засобу з метою перевірки правильності роботи логіки кожного з ключових модулів окремо. Після перевірки кожного з модулів виконано тестування роботи всіх модулів як однієї системи. Всі тести пройдені успішно, система працює стабільно.

Коли алгоритм спроектовано та протестовано, виконано аналіз та оцінку апаратної складності. Кількісна оцінка апаратної вартості в еквівалентних логічних вентилях (Gate Equivalents, GE) показала низькі показники які лежать у межах прийнятних для сфери малоресурсної криптографії. Загальна складність спроектованого апаратного засобу становить 701.73 GE, що є перспективним для використання у малоресурсних пристроях і показує актуальність використання квазігруп як математичного апарату для створення алгоритмів шифрування.

3 ІНТЕГРАЦІЯ ТА ВИПРОБУВАННЯ МОДЕЛІ АПАРАТНО-ПРОГРАМНОГО КОМПЛЕКСУ

Третій розділ присвячений розробці апаратного-програмного комплексу який реалізує метод потокового шифрування на основі квазігруп. Опису структури такого засобу, тестування розробленого комплексу та оцінювання.

3.1 Розробка апаратного інтерфейсного модуля

Після розробки та верифікації операційного блоку шифрування на рівні RTL у Розділі 2, наступним кроком є його інтеграція у систему вищого рівня. Для демонстрації практичного застосування розробленого ядра, необхідно забезпечити йому можливість взаємодії із іншими пристроями, наприклад, з керуючим мікроконтролером або персональним комп'ютером. Для цього було розроблено апаратний інтерфейсний модуль.

Як основний інтерфейс зв’язку було обрано UART (Universal Asynchronous Receiver-Transmitter).

UART є апаратним модулем для асинхронної послідовної передачі даних, який перетворює паралельні байти у послідовні біти для передачі по лінії TX і відновлює їх на приймальній стороні. Розподілення на кадри здійснюється за допомогою старт і стоп бітів та, за потреби, біта парності [22].

Отже такий вибір обґрунтований кількома причинами:

– Універсальність: UART є одним з найпоширеніших послідовних інтерфейсів, що підтримується практично будь-яким мікроконтролером та ПК через USB-UART перетворювачі.
– Простота реалізації: апаратна реалізація UART на ПЛІС є відносно простою і не вимагає значних апаратних ресурсів.
– Достатність: для реалізації розробленого алгоритму, пропускної здатності UART буде достатньо для безперешкодної роботи.

Модуль апаратного інтерфейсу є ієрархічним блоком верхнього рівня, який об'єднує три ключові компоненти модуль приймача, модуль передавача та операційного блоку.

Модуль приймача (RX) приймає послідовний потік біт з лінії, десеріалізує його та видає 8-бітний паралельний байт разом із сигналом готовності.

Модуль передавача (TX) приймає 8-бітний байт по сигналу валідації, серіалізує його та відправляє у лінію, дотримуючись протоколу старт-біт, дані, стоп-біт.

Операційний блок виконує головну дію алгоритму та його структуру було описано у Розділі 2.

Кінцевий автомат (FSM) протоколу є керуючим центром модуля, який керує взаємодією між UART та ядром шифру.

Схема модуля апаратного інтерфейсу наведена на рисунку 3.1.

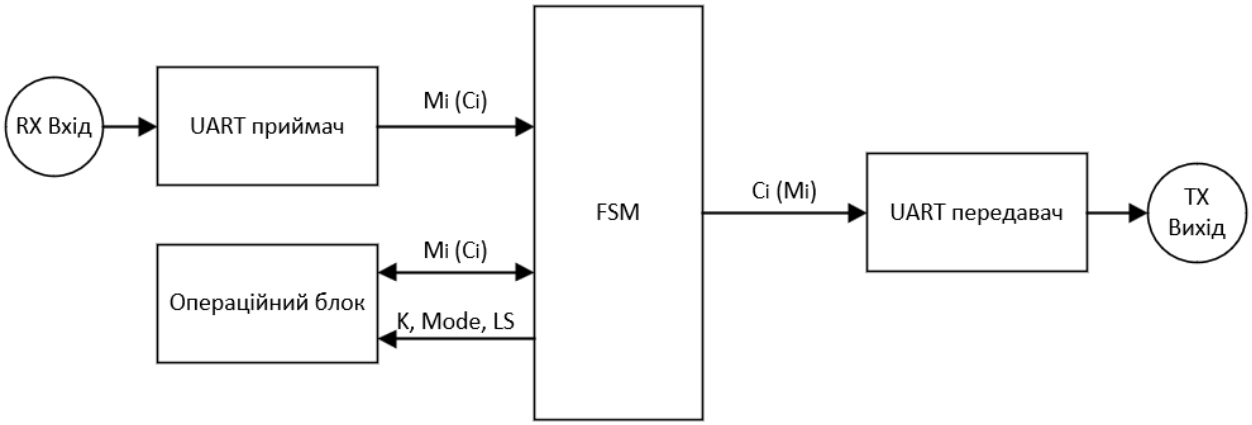


Рисунок 3.1 – Загальна структурна схема апаратного інтерфейсу

Оскільки блок ядра шифрування вимагає ініціалізації та має специфічний два бітний інтерфейс даних, було розроблено простий байт-орієнтований протокол керування. Керуючий пристрій виступає в ролі керуючого, а розроблений модуль виконавця.

Для такого протоколу визначені команди керування, а саме ініціалізації сесії, обробки одного байту даних та отримання статусу.

Сигнал ініціалізації сесії описується кодом “0x01” та має структуру з 11 байт. Така команда складається з коду, режиму роботи, визначення необхідної квазігрупи та 8 байт вхідного ключа. Відповідь описується командою “0x80

(ACK)” та означає підтвердження ініціалізації. Таким чином модуль зчитує вхідні дані, завантажує їх у внутрішні регістри та ініціалізує ядро шифру.

Сигнал обробки одного байту даних описується кодом “0x02” та має структуру з двох байт. Така команда складається з коду та байту вхідних даних. Як відповідь отримується оброблений байт даних. Це ключова частина логіки. FSM модуля приймає 8-бітний байт даних у буфер. Потім він послідовно, протягом чотирьох ітерацій, розбиває цей байт на два бітні пари і подає їх на вхід ядра шифрування. Результати збираються у буфер результату і по завершенні відправляються назад через UART.

Сигнал статусу описується кодом “0x03” та має структуру в один байт. Його відповідь це біти статусу готовності та обробки системи.

Схему роботи автомату FSM для розробленого протоколу наведено на рисунку 3.2

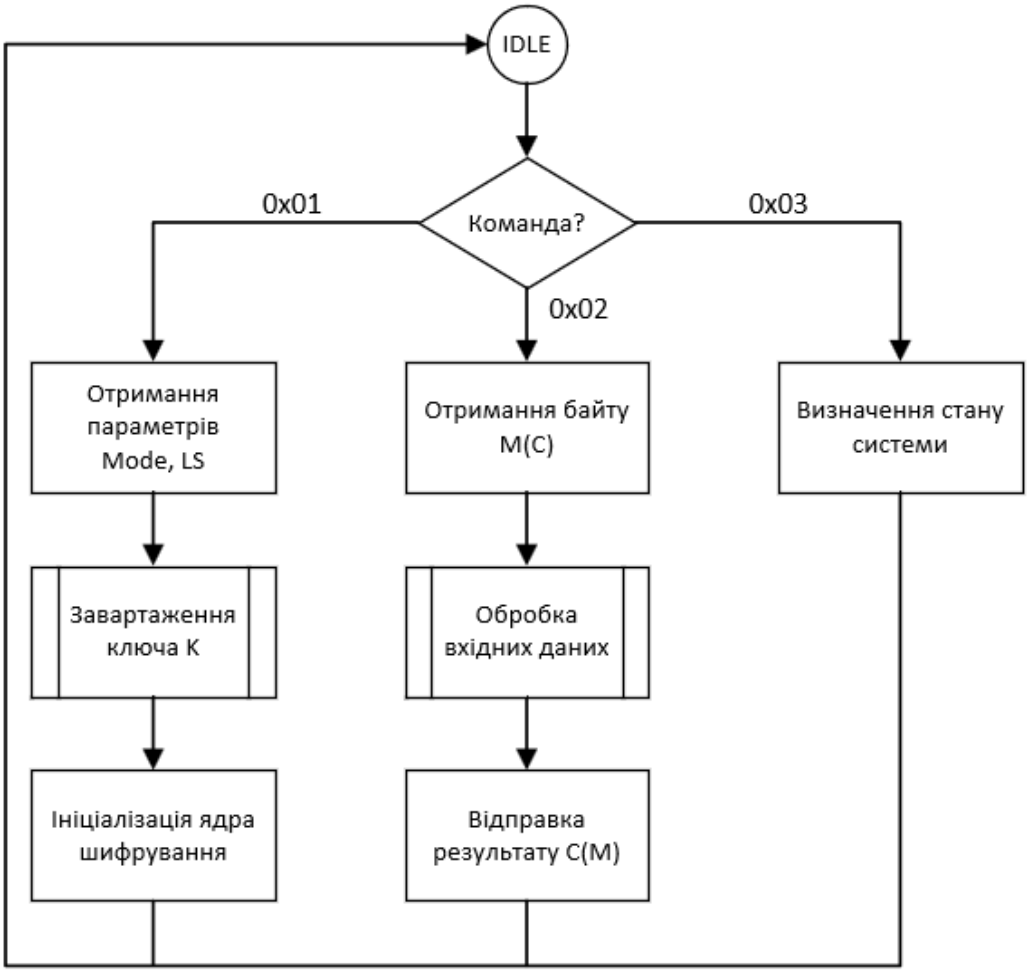


Рисунок 3.2 – Схема роботи автомату FSM розробленого протоколу

Таким чином забезпечується безперервна обробка вхідного повідомлення.

3.2 Розробка програмного драйвера

Після розробки інтерфейсного модуля необхідно розробити програмний драйвер для подальшої роботи системи.

Таке рішення забезпечує взаємодію прикладного програмного забезпечення з розробленим апаратним модулем створюючи рівень абстракції. Основна мета драйвера полягає у приховуванні низькорівневих деталей реалізації протоколу обміну та керування сигналами, надаючи користувачеві зручний та зрозумілий API (Application Programming Interface).

Такий драйвер реалізовано з використанням мови C++ у вигляді єдиного класу. Такий підхід дозволяє інкапсулювати стан сесії та методи обміну даними в єдину програмну сутність.

Архітектура драйвера побудована на принципах композиції де клас драйвера містить у собі екземпляр емулятора інтерфейсу UART та посилання на модель розробленого пристрою.

Ієрархічну структуру , яка демонструє рівні абстракції даних, зображено на рисунку 3.3

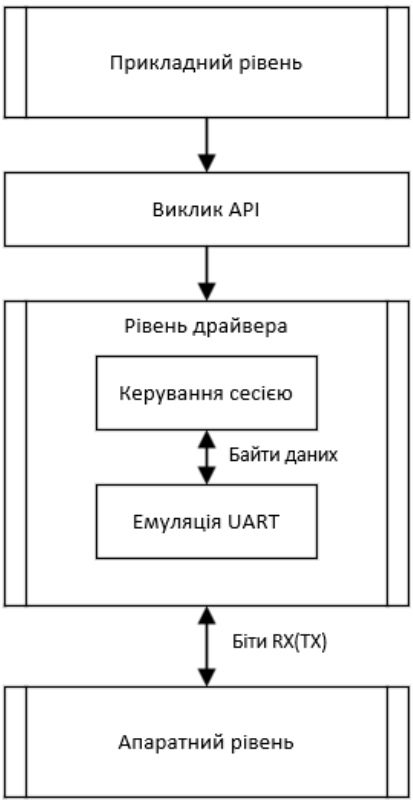


Рисунок 3.3 – Схема ієрархічної структури рівнів абстракції даних

З рисунку можна побачити що така структура складається з трьох рівнів, а саме прикладного, драйверу та апартного.

Верхнім рівнем є прикладний, де користувач оперує базовими сутностями як текстові рядки або числа. На такому рівні відбувається виклик API-функцій, для виконання основного завдання програмного засобу.

Рівень драйвера є ядром розробленого рішення та складається з двох підрівнів: керування сесією та емуляції UART. Перший з них відповідає за логіку сесії, розбиваючи вхідні дані на окремі команди протоколу. Інший відповідає за фізичне перетворення байт команд у послідовність бітів та забезпечує генерацію тактових сигналів для синхронізації часу моделі апаратури.

Апаратний рівень є останнім рівнем, який представлений моделлю розробленого апаратного рішення. Сюди надходять вже сформовані сигнали, ідентичні тим, що були б на фізичних ніжках ПЛІС.

Така структура дозволяє повністю ізолювати користувача від складності апаратного протоколу.

Розроблений API надає набір методів, що покривають повний життєвий цикл роботи з пристроєм. Функціональність драйвера поділяється на низькорівневі та високорівневі методи.

Метод ініціалізації є низькорівневим та виконує функцію налаштування захищеного каналу. Алгоритм його роботи описується кроками:

Крок 1. Формування заголовка.

Крок 2. Обробка ключа.

Крок 3. Рукостискання.

На першому кроці драйвер відправляє командний байт початку, за яким рухаються байти режиму (шифрування/розшифрування) та селектора квазігрупи.

Після чого, наступним кроком, вхідний 64-бітний ключ розбивається на вісім 8-бітних фрагментів. Передача здійснюється послідовно, починаючи з молодшого байта.

Після передачі останнього байта ключа драйвер переходить у режим очікування відповіді. Успішна ініціалізація підтверджується отриманням байта

ACK (0x80). Будь-яка інша відповідь або відсутність відповіді інтерпретується як помилка ініціалізації.

Метод побайтової обробки є низькорівневим і забезпечує процес зашифрування або розшифрування вхідних даних. Такий метод працює за принципом запит – відповідь та описується станами:

– Надсилання команди обробки (0x02).
– Передавання байту вхідних даних M(C).
– Очікування готовності результату від апаратної частини.
– Зчитування обробленого байту.

Виклики методу та обробка даних відбувається до моменту вичерпування вхідних даних.

Для зручності інтеграції розроблено високорівневі методи encrypt та decrypt які забезпечують абстракцію та працюють зі стандартними типами даних не змушуючи вдаватись до глибокого розуміння роботи апаратури. Такі методи автоматизують роботу всієї системи, шляхом ініціалізації сесії, ітерації по вхідних масивах даних, виклику методів обробки даних та формування вихідного буферу.

Однією з головних проблем роботи з асинхронними інтерфейсами, включно з UART, є існуючий ризик блокування роботи програми у випадку збою апаратури або втрати даних. Така ситуація призводить до порушення роботи програмного потоку, в наслідок чого він може нескінченно очікувати на подію, яка ніколи не відбудеться. Для вирішення цієї проблеми у драйвері реалізовано механізм обмеження за часом. Головна ідея полягає в тому, що кожна операція очікування події обмежується за часом. Якщо протягом заданого інтервалу очікувана подія не відбувається, драйвер припиняє очікування та повертає помилку, дозволяючи програмі коректно продовжити виконання, а не зациклюватись у вічному очікуванні.

Розроблений метод очікування відповіді, замість нескінченного циклу, використовує лічильник тактів симуляції. Якщо апаратний модуль не виставляє прапор готовності даних протягом заданої кількості тактів, функція примусово

завершує очікування та повертає код помилки. Це запобігає зависанню стенду та дозволяє коректно обробляти аварійні ситуації.

Таким чином інтеграція розробленого драйверу в програму починається зі створення екземпляру класу драйвера, який автоматично виконує ініціалізацію та скидання віртуальної апаратури. Після підготовки вхідних даних M(C) та ключа шифрування K, основний функціонал забезпечується викликом методу encrypt для режиму зашифрування та decrypt для режиму розшифрування. Реалізований підхід забезпечує повну ізоляцію логіки шифрування від деталей апаратної реалізації, що відповідає принципам абстракції та сучасним вимогам до розробки вбудованих систем.

3.3 Опис моделі апаратно-програмного стенду

Для перевірки коректності функціонування розробленої системи в умовах, максимально наближених до реальних, було створено модель апаратно-програмного стенду. Оскільки фізична імплементація на ПЛІС на етапі розробки може бути ресурсозатратною та складною для налагодження, було обрано підхід симуляції в контурі (Hardware-in-the-Loop), що дозволяє проводити тестування апаратної логіки без її безпосереднього розгортання на реальному обладнанні.

Метод Hardware-in-the-Loop передбачає інтеграцію симульованої апаратної частини з реальним або симульованим програмним забезпеченням у єдиному контурі керування. У такому режимі програмний драйвер взаємодіє з високоточною програмною моделлю апаратної логіки так само, як і з фізичним пристроєм, що забезпечує раннє виявлення помилок, прискорює перевірку функціональності та зменшує витрати часу на налагодження [23].

Розроблений стенд реалізовано за допомогою інструменту Verilator, який дозволяє транслювати синтезований Verilog код апаратної частини у високопродуктивну C++ модель . Це дає можливість об'єднати апаратну логіку та програмний драйвер в єдиний виконуваний файл.

Verilator є високопродуктивним компілятором апаратних описів на Verilog або SystemVerilog, що перетворює синтезовані апаратні модулі на статичну C++

модель, придатну для швидкого моделювання. На відміну від традиційних інтерпретуючих симуляторів, Verilator здійснює саме компіляцію апаратної логіки, що забезпечує суттєве прискорення симуляції та спрощує інтеграцію з програмними драйверами або тестовими фреймворками. Завдяки цьому апаратну частину та програмний модуль керування можливо об’єднати у єдиний виконуваний файл, що підвищує ефективність апаратно-програмних систем [24].

Використання принципу емуляції апаратної частини значно пришвидшує розробку та тестування отриманого рішення, шляхом зменшення часових витрат на купівлю та освоєння реальної апаратної плати. Розроблений стенд працює за ідентичними принципами як і фізичні прилади, що забезпечує повну інтеграцію та перенесення реалізації на реальні палати.

Апаратно-програмний стенд побудовано за модульним принципом, де кожен компонент відповідає за свій рівень абстракції. Структурна схема взаємодії компонентів стенду наведена на рисунку 3.4

Рисунок 3.4 – Схема взаємодії компонентів стенду

Таким чином основними елементами архітектури є програмна частина, інтерфейс та апаратна частина.

Програмна частина є найвищим рівнем у такій ієрархії, для керування стендом, та складається з двох блоків. Головний блок такої частини керує ходом виконання програми, забезпечуючи правильність виконання всіх функцій. Блок драйвера виконує функцію комунікації програмної частини з апаратурою шляхом формування та відправки відповідних команд управління.

Інтерфейс – це проміжний шар абстракції, який забезпечує зв’язок між програмним кодом та апаратною моделлю, та складається з двох блоків. Тактовий генератор синхронізує роботу моделі, керуючи перемиканнями сигналу тактової частоти (clk) та викликом методу оцінки її стану. Блок емуляції

UART виконує перетворення байтів від драйвера у послідовність бітів для входу (RX) моделі та навпаки, шляхом емуляції фізичної лінії передачі даних.

Апаратна частина є нижнім рівнем ієрархії. Такий блок представлений об’єктом відповідного класу, який скомпільований у C++ на основі оригінальної моделі реалізації Verilog. Такий об'єкт поводиться ідентично до реальної ПЛІС, реагуючи на зміни вхідних сигналів на кожному такті.

Такий підхід дозволяє досягти тактової точності, гарантуючи, що часові параметри протоколу UART у симуляції повністю відповідатимуть поведінці реальної апаратури на частоті 50 МГц.

Для керування процесом компіляції та запуску стенду розроблено систему автоматизації на базі утиліти Make, яка є одним із базових інструментів автоматизації збірки програмних проектів. Make забезпечує формальний опис залежностей між елементами проекту та визначає послідовність операцій, необхідних для отримання цільових артефактів. На основі цього опису утиліта автоматично визначає, які компоненти потребують повторної збірки, що мінімізує людський фактор, скорочує час розробки та підвищує відтворюваність результатів [25]. Разроблений сценарій збірки виконує наступні три етапи для досягнення стану працюючої моделі.

Першим етапом є трансляція вхідних файлів логіки реалізованої на апаратній мові Verilog у вихідний код на мові C++ для подальшої роботи, за допомогою утиліти Verilator. На цьому етапі також відбувається статичний аналіз коду для визначення потенційних помилок проектування.

Наступним етапом є компіляція згенерованого C++ коду моделі та написаного драйвера за допомогою компілятора GCC з прапорами для оптимізації збірки. Компілятор GCC забезпечує перетворення вихідного коду на машинний, завдяки вбудованим механізмам оптимізації та підтримці стандартів C/C++, застосування GCC гарантує, що отриманий виконуваний файл буде ефективним за продуктивністю та сумісним зі стандартами, що є критичним для коректної роботи стенду.

Фінальним етапом є об’єднання всіх отриманих файлів у один виконуваний файл для запуску розробленого апаратно-програмного комплексу.

Такі етапи збірки дозволяють проводити швидке ітеративне тестування, коли внесення будь-яких змін у апаратний Verilog код або драйвер на C++, автоматично враховуються при наступній збірці проекту, що значно прискорює процес налагодження та інтеграції.

3.4 Тестування та аналіз моделі апаратно-програмного стенду

Після розробки моделі апаратно-програмного комплексу необхідно провести комплексні випробування отриманого стенду. Метою такого тестування є підтвердження функціональної відповідності системи заданим вимогам, перевірка коректності криптографічних перетворень на різних наборах даних та аналіз часових характеристик роботи протоколу.

Програма тестувань складається з різних сценаріїв для охоплення різних аспектів роботи системи і включає п’ять етапів тестування.

Першим етапом є базовий тест для роботи з вхідним коротким повідомленням “hi”. Метою такого тесту є перевірка базової працездатності передачі та обробки даних. Успішним тестуванняv вважається повний збіг отриманого результату після проходження процесу зашифрування та розшифрування. Результат першого етапу тестування наведено на рисунку 3.5

```
================================================
TEST 1: Encrypt/Decrypt 'hi'
================================================

Plaintext: "hi"
Starting session: mode=ENCRYPT, ls=0, key=0x123456789abcdef
Session started (ACK received)
Ciphertext: 8f bf
Starting session: mode=DECRYPT, ls=0, key=0x123456789abcdef
Session started (ACK received)
Decrypted: "hi"

>>> TEST 1: PASSED <<<
```

Рисунок 3.5 – Результат першого етапу тестування

Отримані результати демонструють успішність проходження тестування відтворюючи вхідний текст після операції зашифрування та розшифрування.

Другим етапом є тестування коректності обробки простих рядків з вхідним повідомленням “Bohdan”. Метою такого тесту є перевірка стабільності роботи FSM при обробці послідовності байтів. Тестування перевіряє, чи коректно система збирає два бітні пари назад у байти без втрати синхронізації між символами. Тому успішним тестуванням вважається повний збіг отриманого результату після проходження процесу зашифрування та розшифрування. Результат другого етапу тестування наведено на рисунку 3.6

```
==================================================
TEST 2: Encrypt/Decrypt 'Bohdan'
==================================================

Plaintext: "Bohdan"
Starting session: mode=ENCRYPT, ls=0, key=0x123456789abcdef
Session started (ACK received)
Ciphertext: 9a b2 cb f0 00 39
Starting session: mode=DECRYPT, ls=0, key=0x123456789abcdef
Session started (ACK received)
Decrypted: "Bohdan"

>>> TEST 2: PASSED <<<
```

Рисунок 3.6 – Результат другого етапу тестування

Отримані результати демонструють успішність проходження тестування, система збирає два бітні пари назад у байти без втрати синхронізації, при тому відтворюючи вхідний текст після основних кроків шифрування.

Третім етапом є навантажувальний тест потоку. Метою такого тестування є перевірка стабільності роботи алгоритму при роботі з великими послідовностями шляхом перевірки відсутності накопичених помилок та переповнення буферу. Вхідними даними такого тестування є довге текстове повідомлення “The quick brown fox jumps over the lazy dog”, довжиною понад 40 байт. Аналогічно успішним тестуванням вважається відсутність артефактів або зміщення даних у результуючому рядку після проходження процесу

зашифрування та розшифрування. Результат третього етапу тестування наведено на рисунку 3.7

```
================================================
TEST 3: Long Message
================================================

Plaintext: "The quick brown fox jumps over the lazy dog"
Length: 43 bytes
Starting session: mode=ENCRYPT, ls=0, key=0x123456789abcdef
Session started (ACK received)
Ciphertext: a7 bc c0 3c 30 0f 40 76 d9 10 21 ed 7a 79 a5 55
            3b fa 76 ee 9a a7 c1 85 6a dd b0 33 fa de e7 f9
            8f 32 15 58 19 08 30 d9 ef 5a fb
Starting session: mode=DECRYPT, ls=0, key=0x123456789abcdef
Session started (ACK received)

Decrypted: "The quick brown fox jumps over the lazy dog"

>>> TEST 3: PASSED <<<
```

Рисунок 3.7 – Результат третього етапу тестування

Результати зашифрування та розшифрування повідомлення великої довжини є позитивними. Система відпрацювала у штатному режимі успішно зашифрувавши текст, після чого відтворивши його у повному обсязі шляхом розшифрування шифротексту.

Четвертим етапом є перевірка механізму відновлення сесії. Під час такої перевірки виконується серія перетворень із динамічною зміною параметрів сесії, а саме 64-бітного ключа *K* та квазігрупи *LS*. Метою такого тестування є перевірка коректності роботи команди початку сесії (0x01) яка скидає внутрішній стан шифратора та завантажує нові параметри незалежно від попереднього стану системи. Тест вважається успішним за умови відсутності артефактів або зміщення даних у результуючому рядку після проходження процесу зашифрування та розшифрування. Результат третього етапу тестування наведено на рисунку 3.8

```
================================================
TEST 4: Different Keys & LS Pairs
================================================

--- Key=0x0123456789ABCDEF, LS=0 ---
Starting session: mode=ENCRYPT, ls=0, key=0x123456789abcdef
Session started (ACK received)
Starting session: mode=DECRYPT, ls=0, key=0x123456789abcdef
Session started (ACK received)
Result: PASS

--- Key=0xFEDCBA9876543210, LS=1 ---
Starting session: mode=ENCRYPT, ls=1, key=0xfedcba9876543210
Session started (ACK received)
Starting session: mode=DECRYPT, ls=1, key=0xfedcba9876543210
Session started (ACK received)
Result: PASS

--- Key=0xAAAAAAAAAAAAAAAA, LS=2 ---
Starting session: mode=ENCRYPT, ls=2, key=0xaaaaaaaaaaaaaaaa
Session started (ACK received)
Starting session: mode=DECRYPT, ls=2, key=0xaaaaaaaaaaaaaaaa
Session started (ACK received)
Result: PASS

>>> TEST 4: PASSED <<<
```

Рисунок 3.8 – Результат четвертого етапу тестування

Експеримент передбачав послідовну ініціалізацію трьох незалежних сесій шифрування з кардинально відмінними параметрами без перезавантаження апаратної платформи. Як можна побачити з рисунку, для кожного вектора ітерації, драйвер успішно отримав сигнал підтвердження від апаратного контролера як при встановленні режиму зашифрування, так і при перемиканні в режим розшифрування. Позитивний результат тестування підтверджує, що механізм динамічної реконфігурації функціонує коректно і не призводить до збоїв синхронізації FSM.

П’ятий етап тестування є повна перевірка системи у режимі реального часу з симуляцією дій користувача. Метою такого тестування є відтворення типового сценарію використання апаратно-програмного комплексу користувачам, що включає встановлення ключа, зашифрування та розшифрування вхідного повідомлення. Результат тестування вважається позитивним за умови повного відтворення вхідних даних введених користувачем після процесу зашифрування та розшифрування.

```
> k a1b2c3d4e5f60718
Key set to 0xa1b2c3d4e5f60718
> e Дедлайн midnight

Encrypting: "Дедлайн midnight"
Key: 0xa1b2c3d4e5f60718, LS: 0
Starting session: mode=ENCRYPT, ls=0, key=0xa1b2c3d4e5f60718
Session started (ACK received)
Ciphertext: 4b 85 29 f0 0f d1 ed 35 9b 2d 79 65 5f 4a 9d 13
            8c 21 e0 6e c1 48 41

> d 4b 85 29 f0 0f d1 ed 35 9b 2d 79 65 5f 4a 9d 13 8c 21 e0 6e c1 48 41

Decrypting hex bytes...
Key: 0xa1b2c3d4e5f60718, LS: 0
Input: 4b 85 29 f0 0f d1 ed 35 9b 2d 79 65 5f 4a 9d 13
            8c 21 e0 6e c1 48 41
Starting session: mode=DECRYPT, ls=0, key=0xa1b2c3d4e5f60718
Session started (ACK received)
Decrypted: "Дедлайн midnight"
```

Рисунок 3.9 – Результат п’ятого етапу тестування

Отримані результати демонструють позитивний результат проходження тестування. Процес встановлення ключа пройшов успішно. Вхідне повідомлення було зашифровано, після чого отриманий шифротекст було подано на вхід для режиму розшифрування. Після чого було отримано ідентичний оригінальному результуючий текст.

З отриманих результатів видно, що всі етапи тестування були проведені з позитивними результатами, що свідчить про коректність реалізації та стабільність роботи розробленої моделі апаратно-програмного комплексу.

Після тестування комплексу доцільно проаналізувати характеристики розробленого апаратно-програмного рішення. Тактова частота системи F_{clk} = 50 МГц, швидкість інтерфейсу UART складає 115200 бод., кількість тактів на один біт встановлено як 434. Таким чином можна визначити час передачі одного байту з урахуванням бітів початку та завершення, перемноживши таких 10 біт на визначені раніше 434 такти для його передачі. Тоді можна отримати T_{UART} = 10 × 434 = 4340 тактів.

Операційний блок обробляє одну два бітну пару за 1 такт з урахуванням плюс затримки FSM на перемикання станів. Тому обробка повного байту займає менше 20 тактів внутрішньої логіки. Таким чином час криптографічного перетворення є надзвичайно малим у порівнянні з часом передачі даних. Це свідчить про високу ефективність розробленого операційного блоку. Введена

ним затримка не впливає на загальну швидкість потоку даних при використанні інтерфейсу UART. Такий запас продуктивності дозволяє в майбутньому інтегрувати розроблений блок з більш швидкісними інтерфейсами без зміни внутрішньої архітектури шифратора.

Для підтвердження можливості фізичної реалізації розробленого засобу та оцінки його апаратних витрат, було проведено синтез проекту у реальному середовищі. Такий аналіз виконано за підтримки партнерів ВНТУ, та отримано доступ до середовища Quartus Prime для роботи з платами ПЛІС. Як цільову платформу обрано ПЛІС сімейства Cyclone V, якої цілком вистачить для реалізації такого проекту. Результат синтезу розробленого рішення наведено на рисунку 3.10

Flow Status	Successful - Sat Nov 1 20:07:29 2025
Quartus Prime Version	23.1std.1 Build 993 05/14/2024 SC Lite Edition
Revision Name	CipherTest
Top-level Entity Name	CipherTest
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	169 / 32,070 (< 1 %)
Total registers	316
Total pins	5 / 457 (1 %)
Total virtual pins	0
Total block memory bits	0 / 4,065,280 (0 %)
Total DSP Blocks	0 / 87 (0 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

Рисунок 3.10 – Результат синтезу проекту на ПЛІС

Синтез пройшов успішно та його результати демонструють компанктність розробленого рішення. Звіт про використання ресурсів наведено у таблиці 3.

Таблиця 3.1 – Використання ресурсів ПЛІС Cyclone V

Параметр	Використано	Всього доступно	Відсоток
Logic utilization (ALMs)	169	32070	< 1 %
Total registers	316	-	-
Total block memory bits	0	4065280	0 %
Total DSP Blocks	0	87	0 %
Total PLLs	0	6	0 %

Після успішного синтезу на ПЛІС доречно виконати тестування працездатності такого рішення на реальній системі. Для цього реалізовано канал зв’язку за допомогою інтерфейсу UART. Зовнішній керуючий пристрій відправляє команди та дані згідно з розробленим проколом у пункті 3.1.

Першим етапом тестування є перевірка режиму зашифрування для поданих даних. Результат зашифрування вхідного повідомлення на ПЛІС з вихідного логу роботи наведено на рисунку 3.11

```
[TX]: 0x01 0x00 0x00 0x4B 0x65 0x79 0x5F 0x53 0x74 0x72 0x00
[RX]: 0x80 0x00
[TX]: 0x02 0xAA
[TX]: 0x02 0xBB
[TX]: 0x02 0xCC
[TX]: 0x02 0xDD
[TX]: 0x02 0xEE
[TX]: 0x02 0xFF
[RX]: 0x01 0x74 0xA2 0x00
[RX]: 0xFA 0xA3 0x6C 0x00
[TX]: 0x02 0xCC
[TX]: 0x02 0xCC
[TX]: 0x02 0xDD
[TX]: 0x02 0xDD
[TX]: 0x02 0xCC
[TX]: 0x02 0xCC
[RX]: 0xD9 0xFD 0xF7 0xBE 0x00
[RX]: 0x26 0x50 0x00
```

Рисунок 3.11 – Результат зашифрування на ПЛІС

З отриманих результатів можна побачити, що система разом з протоколом працює стабільно та успішно. Видно, що присутні сигнали 0x01 та 0x00 що свідчать про початок сесії з режимом зашифрування. Відповідно отримана відповідь 0x80 та 0x00, що демонструє успішність початку сесії. Далі відправлена послідовність байтів “0xAA 0xBB 0xCC 0xDD 0xEE 0xFF” була зашифрована у “0x01 0x74 0xA2 0xFA 0xA3 0x6C”. Також було перевірено

обробку патернів. Вхідна послідовність “0xCC 0xCC 0xDD 0xDD 0xCC 0xCC” була перетворена у псевдовипадкову послідовність “0xD9 0xFD 0xF7 0xBE 0x26 0x50”, що свідчить про розсіювальні властивості шифру навіть при монотонних вхідних даних. Таким чином процес зашифрування можна вважати успішним.

Другим етапом є перевірка режиму розшифрування для зашифрованих на попередньому етапі даних. Результат розшифрування вхідного повідомлення на ПЛІС з вихідного логу роботи наведено на рисунку 3.12

```
[TX]: 0x01 0x01 0x00 0x4B 0x65 0x79 0x5F 0x53 0x74 0x72 0x00
[RX]: 0x80 0x00
[TX]: 0x02 0x01
[TX]: 0x02 0x74
[TX]: 0x02 0xA2
[TX]: 0x02 0xFA
[TX]: 0x02 0xA3
[TX]: 0x02 0x6C
[RX]: 0xAA 0xBB 0xCC 0xDD 0xEE 0x00
[RX]: 0xFF 0x00
[TX]: 0x02 0xD9
[TX]: 0x02 0xFD
[TX]: 0x02 0xF7
[TX]: 0x02 0xBE
[TX]: 0x02 0x26
[TX]: 0x02 0x50
[RX]: 0xCC 0xCC 0xDD 0xDD 0xCC 0xCC 0x00
```

Рисунок 3.12 – Результат розшифрування на ПЛІС

Після успішного зашифрування було подано шифротекст у якості вхідного тексту для режиму розшифрування. На рисунку видно, що вхідні дані були успішно розшифровані та є ідентичними початковому вхідному повідомленні. Зашифровані байти “0x01 0x74 0xA2 0xFA 0xA3 0x6C” було відновлено до початкових “0xAA 0xBB 0xCC 0xDD 0xEE 0xFF”. Аналогічно для перевірки патернів, вхідне зашифроване значення “0xD9 0xFD 0xF7 0xBE 0x26 0x50”, було відновлено до початкового “0xCC 0xCC 0xDD 0xDD 0xCC 0xCC”. Таким чином процес розшифрування можна вважати успішним.

Синтез та фізичні випробування довели, що розроблений проект, на основі квазігруп, є не лише коректним з точки зору симуляції, але й ефективним з точки зору використання апаратних ресурсів, що робить його перспективним для використання у вбудованих системах з обмеженими ресурсами.

3.5 Висновки до розділу

На основі розробленого алгоритму потокового шифрування на основі квазігруп та моделі операційного блоку, було розроблено модель апаратно-програмного комплексу для практичної реалізації алгоритму для реального середовища.

Після реалізації операційного блоку на мові апаратури, було розроблено середовище для його інтеграції у комплексну систему з можливістю подальшого перенесення на реальну плату.

Розроблено апаратний інтерфейсний модуль для ядра шифрування, який адаптує його специфічний два бітний інтерфейс до стандартного байт-орієнтованого протоколу UART. Це дозволило абстрагувати внутрішню складність криптографічного алгоритму від зовнішнього середовища.

Після чого створено програмний драйвер, який реалізує повний стек протоколу керування. Драйвер забезпечує високорівневий API для прикладного програмного забезпечення, приховуючи низькорівневі деталі бітових операцій та синхронізації. Впроваджено механізми контролю помилок та затримок, що гарантує стійкість системи до збоїв у каналі зв'язку.

Після розробки ключових елементів, було розроблено стенд, який дозволяє виконувати симуляцію апаратної моделі та програмного драйвера в єдиному процесі. Це забезпечило потактову точність верифікації та дозволило налагодити взаємодію компонентів до етапу фізичного перенесення на пристрій.

Проведено тестування розробленого рішення. Виконано серію функціональних тестів, які підтвердили коректність зашифрування та розшифрування даних різного об'єму. Аналіз показників стенду показав, що швидкодія розробленого ядра перевищує пропускну здатність інтерфейсу UART, що свідчить про високий запас продуктивності.

Після тестування стенду, виконано синтез проекту для ПЛІС Intel Cyclone V. Тестування на реальному обладнанні продемонстрували низьке споживання ресурсів та коректність виконання функціоналу.

4 ЕКОНОМІЧНА ЧАТИНА

Під час проведення наукових досліджень необхідно враховувати як потенційні витрати на проведення дослідницького процесу, так і безпосередні результати, які визначають доцільність проведення дослідження. Отримані результати характеризують створений кінцевий продукт, а також наукові знання, які можуть бути застосовані для подальшого розвитку науки і техніки.

Комплексна магістерська кваліфікаційна робота на тему “Метод та засіб потокового шифрування на основі квазігруп”, а саме її друга частина “Операційний блок”, відноситься до науково-технічних робіт, які орієнтовані на виведення на ринок (або рішення про виведення науково-технічної розробки на ринок може бути прийнято у процесі проведення самої роботи), тобто коли відбувається так звана комерціалізація науково-технічної розробки. Цей напрямок є пріоритетним, оскільки результатами розробки можуть користуватися інші споживачі, отримуючи при цьому певний економічний ефект. Але для цього потрібно знайти потенційного інвестора, який би взявся за реалізацію цього проекту і переконати його в економічній доцільності такого кроку.

Для розрахунку доцільності та ефективності інвестицій, вкладених в дану роботу, необхідно провести такі етапи робіт:

1. проведення комерційного та технологічного аудиту науково-технічної розробки, тобто встановлення її науково-технічного рівня та комерційного потенціалу;

2. розрахунок витрат на здійснення науково-технічної розробки;

3. розрахунок економічної ефективності науково-технічної розробки у випадку її можливої комерціалізації потенційним інвестором та обґрунтування економічної доцільності такої комерціалізації.

4.1 Проведення комерційного та технологічного аудиту науково-технічної розробки

Метою проведення аудиту є оцінювання науково-технічного рівня та комерційного потенціалу розробки, створеної в результаті виконання магістерської роботи. Оцінювання здійснюється за 5-бальною шкалою за 12 критеріями, які характеризують технічну досконалість, ринкові переваги та практичну реалізацію проекту [26]. Критерії для оцінки науково-технічного рівня і комерційного потенціалу розробки наведені у таблиці 4.1.

Таблиця 4.1 – Рекомендовані критерії оцінювання науково-технічного рівня і комерційного потенціалу розробки та бальна оцінка

Бали (за 5-ти бальною шкалою)					
Кри-терій	1	2	3	4	5
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено працездатність продукту в реальних умовах
Ринкові переваги (недоліки)					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі влас-тивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів

Продовження таблиці 4.1

Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь- які регламентні обмеження на виробництво та реалізацію продукту

Для проведення оцінювання необхідно визначити оцінки відповідно кожного критерія і обчислити середню арифметичну оцінку, яка визначатиме науково-технічний рівень та комерційний потенціал розробки.

Результати оцінювання науково-технічного рівня та комерційного потенціалу науково-технічної розробки наведено у таблиці 4.2.

Таблиця 4.2 – Результати оцінювання науково-технічного рівня і комерційного потенціалу розробки

Критерії	Експерт (ПІБ, посада)		
	1	2	3
	Бали:		
1. Технічна здійсненність концепції	5	5	5
2. Ринкові переваги (наявність аналогів)	5	4	4
3. Ринкові переваги (ціна продукту)	5	5	5
4. Ринкові переваги (технічні властивості)	4	4	4
5. Ринкові переваги (експлуатаційні витрати)	5	5	5
6. Ринкові перспективи (розмір ринку)	3	3	3
7. Ринкові перспективи (конкуренція)	3	3	3
8. Практична здійсненність (наявність фахівців)	4	4	4
9. Практична здійсненність (наявність фінансів)	2	3	2
10. Практична здійсненність (необхідність нових матеріалів)	4	4	4
11. Практична здійсненність (термін реалізації)	4	4	4
12. Практична здійсненність (розробка документів)	3	3	3
Сума балів:	47	47	46
Середньоарифметична сума балів $СБ_с$	46.67		

Середньоарифметична сума балів оцінюється за наступною формулою:

$$\text{СБ}_\text{с} = \frac{\sum_{i=1}^{3} \text{СБ}_i}{3} = \frac{47 + 47 + 46}{3} = 46.67 \qquad (4.1)$$

де $СБ_с$ – середньоарифметична сума балів; $СБ_i$ – сума балів і-го експерта.

За результатами розрахунків, наведених в таблиці 4.2, можна зробити висновок щодо науково-технічного рівня і рівня комерційного потенціалу розробки [26]. Висновок ґрунтується на оцінках кожного рівня, що наведені в таблиці 4.3

Таблиця 4.3 – Науково-технічні рівні та комерційні потенціали розробки

Середньоарифметична сума балів СБ, розрахована на основі висновків експертів	Науково-технічний рівень та комерційний потенціал розробки
41…48	Високий
31…40	Вищий середнього
21…30	Середній
11…20	Нижчий середнього
0…10	Низький

Відповідно до шкали оцінювання наведені у таблиці 4.3, отриманий результат 46.67 бала свідчить про те, що науково-технічний рівень та комерційний потенціал розробки є високим.

Такий високий рівень досягнуто за рахунок використання квазігруп, що дозволило досягти високої криптографічної стійкості при наднизькій апаратній складності, що є критичною перевагою для ринку IoT. Не менш важливим фактором є ступінь готовності рішенні, оскільки навіть зараз розроблено не лише теоретичний метод, а й повноцінну модель апаратно-програмного комплексу, верифікованого на ПЛІС. При цьому відсутні потреби у дорогому спеціалізованому обладнанні для впровадження та низькі експлуатаційні витрати кінцевого пристрою.

4.2 Розрахунок витрат на здійснення науково-дослідної роботи

Витрати, пов'язані з проведенням науково-дослідної роботи (НДР) щодо створення апаратно-програмного комплексу потокового шифрування, під час планування, обліку і калькулювання собівартості групуються за такими статтями:

- витрати на оплату праці;
- відрахування на соціальні заходи;
- сировина та матеріали;
- витрати на комплектуючі;
- спецустаткування для наукових (експериментальних) робіт;
- програмне забезпечення для наукових (експериментальних) робіт;
- амортизація обладнання, програмних засобів та приміщень;

– паливо та енергія для науково-виробничих цілей;

– витрати на службові відрядження;

– витрати на роботи, які виконують сторонні підприємства, установи і організації;

– інші витрати;

– накладні (загальновиробничі) витрати.

Необхідно проаналізувати кожну з наведених статей.

4.2.1 Витрати на оплату праці

До такої статті витрат належать витрати на виплату основної та додаткової заробітної плати працівникам, безпосередньо зайнятим виконанням конкретної теми, обчисленої за посадовими окладами, відрядними розцінками, тарифними ставками згідно з чинними в організаціях системами оплати праці [26].

Витрати на заробітну плату дослідників розраховуються на основі формули:

$$З_о = \sum_{i=1}^{k} \frac{M_{ni} \cdot t_i}{T_p}, \qquad (4.2)$$

де k – кількість посад дослідників, залучених до процесу досліджень;

M_{ni} – місячний посадовий оклад конкретного дослідника, грн;

t_i – кількість днів роботи конкретного дослідника, дн.;

T_p – середня кількість робочих днів в місяці, T_p = 22 дні.

Розрахунки витрат на заробітну плату дослідників наведено у таблиці 4.4

Таблиця 4.4 – Витрати на заробітну плату дослідників.

Посада	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Кількість днів роботи	Витрати на заробітну плату, грн
Керівник проекту	25000	1136.36	10	11363.63
Провідний спеціаліст	23000	1045.45	22	23000
Розробник	22000	1000	22	22000
Тестувальник	18000	818.18	10	8181.81
Всього				65545.44

Витрати на основну заробітну плату робітників $З_р$ за відповідними найменуваннями робіт розраховують за формулою:

$$З_{\text{р}} = \sum_{i=1}^{n} C_i \cdot t_i \,, \tag{4.3}$$

де C_i – погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год;

t_i – час роботи робітника на виконання певної роботи, год.

Для визначення тарифної ставки робітника відповідного розряду C_i можна скористатись формулою:

$$C_i = \frac{M_M \cdot K_i \cdot K_c}{T_p \cdot t_{\text{ЗМ}}}, \tag{4.4}$$

де M_M – розмір прожиткового мінімуму працездатної особи або мінімальної місячної заробітної плати (залежно від діючого законодавства), для розрахунків прийнято M_M = 8000 грн;

K_i – коефіцієнт міжкваліфікаційного співвідношення для встановлення тарифної ставки робітнику відповідного розряду [26];

K_c – мінімальний коефіцієнт співвідношень місячних тарифних ставок робітників першого розряду з нормальними умовами праці виробничих об’єднань і підприємств до законодавчо встановленого розміру мінімальної заробітної плати.

T_p – середнє число робочих днів в місяці, приблизно T_p = 22 дн;

$t_{зм}$ – тривалість зміни, год.

Результати розрахунків погодинних тарифних ставок для кожного виду робіт наведено разом з розрахунками витрат в таблиці 4.5

Таблиця 4.5 – Величина витрат на основну заробітну плату робітників

Найменування робіт	Тривалість роботи, год	Розряд роботи	Тарифний коефіцієнт	Погодинна тарифна ставка, грн	Величина оплати на робітника грн
Підготовка робочого місця	2	2	1.1	57.50	115
Установка ПЗ та середовищ розробки	4	3	1.35	70.57	282.28
Підготовка дослідження	10	7	2.2	115	1150
Розробка моделі	16	7	2.2	115	1840
Розробка апаратно-програмного стенду	12	6	2	104.55	1254.6

Продовження таблиці 4.5

Реалізація проекту на ПЛІС	6	5	1.7	88.86	533.16
Тестування	4	2	1.1	57.50	230
Всього					5405.04

Після обчислення витрат на основну заробітну плату робітників необхідно також визначити величину витрат на додаткову заробітну плату дослідників та робітників, що складає 10 … 12% від суми витрат за основну заробітну плату дослідників та робітників. Для розрахунків взято середнє значення 11%. Величину витрат на додаткову заробітну плату дослідників та робітників можна визначити за формулою:

$$З_{\text{дод}} = \left(З_{\text{о}} + З_{p}\right) \cdot \frac{Н_{\text{дод}}}{100\%} = (65545.44 + 5405.04) \cdot \frac{11}{100}\% = 7804.56 \quad (4.5)$$

4.2.2 Витрати на соціальні заходи

Відрахування на соціальні заходи включає в себе відрахування внеску на загальнообов'язкове державне соціальне страхування та для здійснення заходів щодо соціального захисту населення. Цей внесок обраховується як 22% від суми основної та додаткової заробітної плати дослідників та робітників за формулою:

$$З_{\text{Н}} = \left(З_{\text{о}} + З_{p} + З_{\text{дод}}\right) \cdot \frac{Н_{\text{ЗП}}}{100\%}, \quad (4.6)$$

Таким чином відрахування на соціальні заходи можна обрахувати як:

$$З_{\text{Н}} = (65545.44 + 5405.04 + 7804.56) \cdot \frac{22}{100}\% = 17326.11$$

де $Н_{зп}$ – норма нарахування на заробітну плату.

Таким чином, бачимо, що сума відрахувань на соціальні заходи становить 17326.11 грн.

4.2.3 Сировина та матеріали

До статті "Сировина та матеріали" належать витрати на сировину, основні та допоміжні матеріали, інструменти, пристрої та інші засоби й предмети праці, які придбані у сторонніх підприємств, установ і організацій та витрачені на проведення досліджень за темою дослідження "Метод та засіб потокового шифрування на основі квазігруп".

Витрати на матеріали *М* у вартісному вираженні розраховуються окремо для кожного виду матеріалів за формулою:

$$\text{М} = \sum_{j=1}^{n} \text{Н}_j \cdot \text{Ц}_j \cdot \text{К}_j - \sum_{j=1}^{n} \text{В}_j \cdot \text{Ц}_{\text{в}j}, \quad (4.7)$$

де H_j – норма витрат матеріалу j-го найменування, кг;

n – кількість видів матеріалів;

$Ц_j$ – вартість матеріалу j-го найменування, грн/кг;

$К_j$ – коефіцієнт транспортних витрат, ($К_j$ = 1,1 … 1,15);

$В_j$ – маса відходів j-го найменування, кг;

$Ц_{вj}$ – вартість відходів j-го найменування, грн/кг.

Таким чином отримані розрахунки наведено у таблиці 4.6

Таблиця 4.6 – Витрати на матеріали

Найменування матеріалу, марка, тип, сорт	Ціна за 1 кг (од.), грн	Норма витрат, кг (шт.)	Величина відходів, кг	Ціна від ходів, грн/кг	Вартість ви траченого ма теріалу, грн
Папір офісний А4 Maestro (500 арк.)	72	2.5	0	0	198
Канцелярський набір приладдя Buromax	150	1	0	0	165
USB-накопичувач Kingston 64GB	250	1	0	0	275
Тонер для принтера HP 1010	1750	0.2	0	0	402.5
Всього					1040.5

Тоді, бачимо, що сума відрахувань на сировину та матеріали становить 1040.5 грн

4.2.4 Розрахунок витрат на комплектуючі

Витрати на комплектуючі вироби $К_в$, які використовують при дослідженні нового технічного рішення, розраховуються, згідно з їхньою номенклатурою, за формулою:

$$\text{К}_\text{в} = \sum_{j=1}^{n} \text{Н}_j \cdot \text{Ц}_j \cdot \text{К}_j, \quad (4.8)$$

де H_j – кількість комплектуючих j-го виду, шт.;

$Ц_j$ – покупна ціна комплектуючих j-го виду, грн;

$К_j$ – коефіцієнт транспортних витрат, ($К_j$ = 1,1 … 1,15).

Таким чином отримані розрахунки наведено у таблиці 4.7

Таблиця 4.7 – Витрати на комплектуючі

Найменування комплектуючих	Кількість, шт.	Ціна за штуку, грн	Сума, грн
Відлагоджувальна плата FPGA Altera Intel Cyclone V	1	8400	9240
Інтерфейсний кабель USB Cablexpert Type-A to Micro-B	1	150	165
З'єднувальні провідники Dupont M-F	1	100	110
Блок живлення лабораторний 5V/2.5A DC	1	150	165
Всього			9680

Тоді, бачимо, що сума відрахувань на комплектуючі становить 9680 грн

4.2.5 Спецустаткування для наукових (експериментальних) робіт

До цієї статті належать витрати на виготовлення або придбання спеціального обладнання, стендів, пристроїв, інструментів та приладдя, необхідних для проведення наукових експериментів.

Оскільки для розробки та тестування операційного блоку потокового шифрування використовується стандартна комп'ютерна техніка та комплектуючі вироби, розраховані раніше, витрати за статтею "Спецустаткування" не передбачаються.

4.2.6 Програмне забезпечення для наукових (експериментальних) робіт

До даної статті витрат належать витрати на розробку та/або придбання спеціального програмного забезпечення, що необхідно для проведення досліджень та розробку продукту, а також витрати на проектування, формування та встановлення. До балансової вартості програмного забезпечення входять витрати на його інсталяцію, тому ці витрати беруться додатково в розмірі 10 … 12% від вартості програмного забезпечення.

Балансова вартість програмного забезпечення обчислюється за наступною формулою:

$$В_{прг} = \sum_{i=1}^{k} Ц_{iпрг} \cdot С_{пргi} \cdot К_i, \quad (4.9)$$

де $Ц_{iпрг}$ – ціна придбання одиниці програмного засобу цього виду, грн;

$С_{пргі}$ – кількість одиниць програмного забезпечення відповідного найменування, які придбані для проведення досліджень, шт.;

$К_i$ – коефіцієнт, що враховує інсталяцію, налагодження програмного засобу тощо, ($К_i$ = 1,10…1,12). Для обчислень візьмемо $К_i$ = 1.10;

k – кількість найменувань програмних засобів.

Таким чином, результати обчислень наведено у таблиці 4.8

Таблиця 4.8 – Витрати на придбання програмних засобів по кожному виду

Найменування програмного засобу	Кількість, шт.	Ціна за штуку, грн	Сума, грн
Система математичного моделювання криптографічних перетворень MATLAB	1	5000	5500
Середовище розробки	1	2000	2200
Всього			7700

Таким чином, отримано вартість програмного забезпечення, що складає 7700 грн.

4.2.7 Амортизація обладнання, програмних засобів та приміщень

До статті “Амортизація обладнання, програмних засобів та приміщень” відносять амортизаційні відрахування по кожному виду обладнання, устаткування та інших приладів і пристроїв, а також програмного забезпечення для проведення науково-дослідної роботи, за його наявності в дослідній організації або на підприємстві.

В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо можуть бути розраховані з використанням прямолінійного методу амортизації за формулою:

$$А_{обл} = \frac{Ц_б}{Т_в} \cdot \frac{t_{вик}}{12}, \quad (4.10)$$

де $Ц_б$ – балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вик}$ – термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

$T_в$ – строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

Розрахунки за такою статтею наведено у таблиці 4.9

Таблиця 4.9 – Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаціні відрахування, грн
Ноутбук ASUS	23000	3	1	638.9
Ноутбук HP	22000	3	1	611.1
Плата FPGA	8400	2	1	350
Програмне забезпечення	7700	3	1	213.8
Приміщення лабораторії досліджень	150000	30	1	416.6
Принтер	15000	2	1	625
Всього				2855.4

Таким чином, вартість амортизації обладнання, програмних засобів та приміщень складає 2855.4 грн.

4.2.8 Паливо та енергія для науково-виробничих цілей

До статті “Паливо та енергія для науково-виробничих цілей” належать витрати на придбання енергії, що витрачається з технологічною метою на проведення досліджень. Оскільки робота виконується з використанням комп'ютерної техніки та спеціалізованого обладнання, розраховуються витрати на силову електроенергію.

Витрати на силову електроенергію B_e розраховують за формулою:

$$\mathrm{В_е} = \sum_{i=1}^{n} \frac{W_{yi} \cdot t_i \cdot \mathrm{Ц_е} \cdot \mathrm{К_{вп}}i}{\eta_i}, \qquad (4.11)$$

де W_{yt} – встановлена потужність обладнання на визначеному етапі розробки, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_е$ – вартість 1 кВт-години електроенергії, грн; (вартість електроенергії визначається за даними енергопостачальної компанії), приймемо $Ц_е$ = 8.00 грн;

$К_{впі}$ – коефіцієнт, що враховує використання потужності, $К_{впі} < 1$, приймемо $К_{впі} = 0.8$;

η_i – коефіцієнт корисної дії обладнання, $\eta_i < 1$, приймемо $\eta_i = 0.9$.

Таблиця 4.10 – Витрати на паливо та енергію для науково-виробничих цілей

Найменування обладнання	Встановлена потужність, кВт	Тривалість роботи, год	Сума, грн
Ноутбук ASUS	0.15	176	187.73
Ноутбук HP	0.12	176	150.19
Плата FPGA	0.02	176	25.03
Принтер	0.25	5	8.8
			371.75

Таким чином, витрати на енергію для науково-виробничих цілей становлять 371.75 грн.

4.2.9 Службові відрядження

До статті “Службові відрядження” належать витрати на відрядження штатних працівників, працівників організацій, які працюють за договорами цивільно-правового характеру, аспірантів, зайнятих розробленням досліджень, відрядження, пов’язані з проведенням випробувань машин та при ладів, а також витрати на відрядження на наукові з’їзди, конференції, наради, пов’язані з виконанням конкретних досліджень.

Витрати за статтею “Службові відрядження” розраховуються як 20…25% від суми основної заробітної плати дослідників та робітників за формулою:

$$В_{св} = (З_О + З_Р) \cdot \frac{Н_{св}}{100\%}, \quad (4.12)$$

де $Н_{св}$ – норма нарахування за статтею “Службові відрядження”, прийнято $Н_{св} = 0\%$.

Тоді $Н_{св} = 0$ грн, витрати за цією статтею відсутні.

4.2.10 Витрати на роботи, які виконують сторонні підприємства, установи і організації

До цієї статті належать витрати на проведення робіт, які не можуть бути виконані внутрішніми ресурсами, і для виконання яких залучаються сторонні організації.

У даній роботі до таких витрат віднесено використання спеціалізованого лабораторного обладнання партнерів університету для фінального синтезу проекту на ПЛІС (FPGA) та проведення апаратної верифікації розробленого криптографічного модуля, що вимагало специфічних ліцензійних засобів та апаратних стендів.

Витрати за статтею “Витрати на роботи, які виконують сторонні підприємства, установи і організації” розраховуються як 30…45% від суми основної заробітної плати дослідників та робітників за формулою:

$$\text{В}_{\text{сп}} = (\text{З}_{\text{О}} + \text{З}_{\text{Р}}) \cdot \frac{\text{Н}_{\text{сп}}}{100\%}, \quad (4.13)$$

де $H_{сп}$ – норма нарахування за статтею “Витрати на роботи, які виконують сторонні підприємства, установи і організації”, приймемо $H_{сп}$ = 30%.

Підставляючи необхідні значення, розрахувати норму нарахування за такою статтею можна за формулою:

$$\text{В}_{\text{сп}} = (\,65545.44 + \,5405.04) \cdot \frac{30\%}{100\%} = 21285.14$$

Таким чином, витрати на роботи, які виконують сторонні підприємства, установи і організації становлять 21285.14 грн.

4.2.11 Інші витрати

До статті “Інші витрати” належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками.

Витрати за статтею «Інші витрати» розраховуються як 50…100% від суми основної заробітної плати дослідників та робітників за формулою:

$$I_{\text{в}} = (\text{З}_{\text{О}} + \text{З}_{\text{Р}}) \cdot \frac{\text{Н}_{\text{ів}}}{100\%}, \quad (4.14)$$

де $H_{ів}$ – норма нарахування за статтею “Інші витрати”, приймемо $H_{ів}$ = 50%.

Підставляючи необхідні значення, розрахувати норму нарахування за такою статтею можна за формулою:

$$I_{\text{в}} = (65545.44 + 5405.04) \cdot \frac{50\%}{100\%} = 35475.24$$

Таким чином, інші витрати становлять 35475.24 грн

4.2.12 Накладні (загальновиробничі) витрати

Накладні витрати розраховуються виходячи з основної заробітної плати дослідників та робітників, становлячи 100 … 150% від суми цих витрат. Сюди входять витрати, пов'язані з управлінням організацією, витрати на винахідництво та раціоналізацію, витрати на підготовку (перепідготовку) та навчання кадрів, витрати, пов'язані з набором робочої сили, витрати на оплату послуг банків, витрати, пов'язані з освоєнням виробництва продукції, витрати на науково-технічну інформацію та рекламу та ін.

Такі витрати обраховуються за формулою:

$$\text{В}_{\text{нзв}} = (\text{З}_{\text{О}} + \text{З}_{\text{Р}}) \cdot \frac{\text{Н}_{\text{нзв}}}{100\%} = (65545.44 + 5405.04) \cdot \frac{100\%}{100\%} = 70950.48 \quad (4.15)$$

де $H_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати», прийнято $H_{нзв}$ = 100%.

Тоді, витрати на статтю, накладні (загальновиробничі) витрати, становлять 70950.48 грн.

Витрати на проведення науково-дослідної роботи розраховуються як сума всіх попередніх статей витрат за формулою:

$$\text{В}_{\text{заг}} = \text{З}_{\text{О}} + \text{З}_{\text{Р}} + \text{З}_{\text{дод}} + \text{З}_{\text{Н}} + \text{М} + \text{К}_{\text{В}} + \text{В}_{\text{спец}} + \text{В}_{\text{прг}} + \text{А}_{\text{обл}} + \text{В}_{\text{е}} + \text{В}_{\text{св}} + \text{В}_{\text{сп}} + \text{І}_{\text{в}} + \text{В}_{\text{нзв}} \quad (4.16)$$

Підставляючи необхідні значення, розрахувати витрати на проведення науково-дослідної роботи можна за формулою:

$B_{заг}$ = (65545.44 + 5405.04 + 7804.56 + 17326.11 + 1040.5 + 9680 + 0 + 7700 + 2855.4 + 371.75 + 0 + 21285.14 + 35475.24 + 70950.48) = 245439.66 грн

Для визначення кінцевої вартості завершення науково-технічної роботи необхідно визначити етап виконання науково-технічної роботи. Дана науково технічна розробка знаходиться на останніх етапах стадії розробки промислового зразка, тому буде застосовано коефіцієнт η = 0,9. Таким чином, застосовуючи формулу отримано:

$$ЗВ = \frac{В_{заг}}{\eta} = \frac{245439.66}{0.9} = 272710.73 \quad (4.17)$$

Таким чином, кінцева вартість науково-технічної роботи становить 272710.73 грн.

4.3 Розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором

Для оцінювання економічної доцільності вкладення коштів у комерціалізацію розробленого апаратно-програмного комплексу потокового шифрування на основі квазігруп необхідно розрахувати показники економічної ефективності для потенційного інвестора.

Результати дослідження проведені за темою “Метод та засіб потокового шифрування на основі квазігруп” передбачають комерціалізацію протягом 4-х років реалізації на ринку.

При розрахунку необхідно врахувати такі показники як:

ΔN – збільшення кількості споживачів пристрою, в аналізовані періоди часу, від покращення його певних характеристик. Значення відповідного показника за розрахунковими роками наведено у таблиці 4.11.

Таблиця 4.11 – Збільшення кількості споживачів пристрою, в аналізовані періоди часу

Показник	1-й рік	2-й рік	3-й рік	4-й рік
Збільшення кількості споживачів пристрою, осіб	250	400	600	800

N – кількість споживачів, які використовували аналогічний пристрій у році до впровадження результатів нової науково-технічної розробки, приймемо N = 200 осіб.

$Ц_o$ – вартість пристрою (машини, механізму) у році до впровадження результатів розробки, приймемо 5000 грн.

$\pm\Delta Ц_o$ – зміни вартості пристрою (зростання чи зниження) від впровадження результатів науково-технічної розробки в аналізовані періоди часу, приймемо 1000 грн.

Враховуючи високу актуальність кіберзахисту для малоресурсних систем, прогнозується стійкий попит з боку інтеграторів розумних мереж та промислової автоматизації.

Можливе збільшення чистого прибутку у потенційного інвестора $\Delta П_i$ для кожного із 4-х років, протягом яких очікується отримання позитивних результатів від можливого впровадження та комерціалізації науково-технічної розробки, розраховується за формулою [26]:

$$\Delta П_{\text{і}} = (\pm \Delta Ц_{\text{о}} \cdot N + Ц_{\text{о}} \cdot \Delta N)_{\text{і}} \cdot \lambda \cdot \rho \cdot \left(1 - \frac{\vartheta}{100}\right), \qquad (4.18)$$

де λ – коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2025 році ставка податку на додану вартість складає 20%, а коефіцієнт = 0,8333.

ρ – коефіцієнт, який враховує рентабельність інноваційного продукту, приймемо 30%, ρ = 0.3;

ϑ – ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2025 році ϑ = 18%;

З наведеної формули можна розрахувати збільшення чистого прибутку для кожного з років.

Розрахунок збільшення чистого прибутку 1-го року:

$$\Delta П_1 = (1000 \cdot 200 + 5000 \cdot 250)_1 \cdot 0.8333 \cdot 0.3 \cdot \left(1 - \frac{18}{100}\right) = 297238.11 \text{ грн}$$

Розрахунок збільшення чистого прибутку 2-го року:

$$\Delta П_2 = (1000 \cdot 200 + 5000 \cdot 400)_2 \cdot 0.8333 \cdot 0.3 \cdot \left(1 - \frac{18}{100}\right) = 450981.96 \text{ грн}$$

Розрахунок збільшення чистого прибутку 3-го року:

$$\Delta П_3 = (1000 \cdot 200 + 5000 \cdot 600)_3 \cdot 0.8333 \cdot 0.3 \cdot \left(1 - \frac{18}{100}\right) = 655973.76 \text{ грн}$$

Розрахунок збільшення чистого прибутку 4-го року:

$$\Delta П_4 = (1000 \cdot 200 + 5000 \cdot 800)_4 \cdot 0.8333 \cdot 0.3 \cdot \left(1 - \frac{18}{100}\right) = 860965.56 \text{ грн}$$

Приведена вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки можна розрахувати за формулою:

$$\text{ПП} = \sum_{i=1}^{T} \frac{\Delta \text{П}_i}{(1+\tau)^i}, \quad (4.19)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн;

T – період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, приймемо $\tau = 0.15$;

t – період часу (в роках) від моменту початку впровадження науково-технічної розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

З наведеної формули можна розрахувати приведену вартість збільшення всіх чистих прибутків на основі отриманих значень:

$$\text{ПП} = \frac{297238.11}{1.15} + \frac{450981.96}{1.15^2} + \frac{655973.76}{1.15^3} + \frac{860965.56}{1.15^4} = 1523048.32 \quad (4.20)$$

Далі необхідно розрахувати приведену величину початкових інвестицій *PV* за формулою:

$$PV = k_{\text{інв}} \cdot \text{ЗВ} = 1.1 \cdot 272710.73 = 299981.80, \quad (4.21)$$

Де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію, приймаємо $k_{інв} = 1.1$;

ЗВ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, приймаємо 272710.73 грн

Також необхідно розрахувати абсолютний економічний ефект $\text{Е}_{\text{абс}}$ для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки за формулою:

$$\text{Е}_{\text{абс}} = \text{ПП} - PV = 1523048.32 - 299981.80 = 1223066.52, \quad (4.22)$$

де *ПП* – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, розраховано 1523048.32 грн;

PV – теперішня вартість початкових інвестицій, розраховано 299981.80 грн.

Оскільки величина $E_{абс}$ має велике позитивне значення, це свідчить про потенційну зацікавленість інвесторів у впровадженні та комерціалізації такої науково-технічної роботи. Але для остаточного прийняття рішення з цього питання необхідно розрахувати внутрішню економічну дохідність $E_в$ або показник внутрішньої норми дохідності (IRR, Internal Rate of Return) вкладених інвестицій та порівняти їх з так званою бар'єрною ставкою дисконтування, яка визначає ту мінімальну внутрішню економічну дохідність, нижче якої інвестиції в будь-яку науково-технічну розробку вкладати буде економічно недоцільно.

Розрахувати внутрішню економічну дохідність $E_в$ можна за формулою:

$$E_в = \sqrt[Т_ж]{1 + \frac{E_{абс}}{PV}} - 1 = \sqrt[4]{1 + \frac{1223066.52}{299981.80}} - 1 = 0.50, \qquad (4.23)$$

де $E_{абс}$ – абсолютний економічний ефект вкладених інвестицій, розраховано 1223066.52 грн;

PV – теперішня вартість початкових інвестицій, розраховано 299981.80 грн;

$T_ж$ – життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, 4 роки.

Розрахувати мінімальну внутрішню економічну дохідність вкладених інвестицій τ_{min} можна за формулою:

$$\tau_{min} = d + f = 0.12 + 0.3 = 0.42, \qquad (4.24)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2025 році в Україні d = 0,12;

f – показник, що характеризує ризикованість вкладення інвестицій, приймемо f = 0,3.

Оскільки внутрішня економічна дохідність перевищує мінімальну, потенційний інвестор може бути зацікавлений в даній розробці.

Далі необхідно розрахувати період окупності інвестицій $T_{ок}$ (DPP, Discounted Payback Period), які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки за формулою:

$$T_{ок} = \frac{1}{E_в} = \frac{1}{0.50} = 2, \quad (4.25)$$

де $E_в$ – внутрішня економічна дохідність вкладених інвестицій.

Таким чином, термін окупності складає 2 роки. Отриманий термін є меншим ніж три роки, це свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження цієї розробки та виведення її на ринок.

Узагальнені результати розрахунків основних техніко-економічних показників проекту наведено у таблиці 4.12.

Таблиця 4.11 – Збільшення кількості споживачів пристрою, в аналізовані періоди часу

Найменування показника	Умовне позначення	Значення
Загальна вартість науково-технічної розробки	ЗВ	272710.73 грн
Початкові інвестиції (з урахуванням впровадження)	PV	299981.80 грн
Прогнозований сумарний приріст чистого прибутку	ПП	1523048.32 грн
Абсолютний економічний ефект	$E_{абс}$	1223066.52 грн
Внутрішня норма дохідності (IRR)	$E_В$	50%
Мінімальна бар'єрна ставка	τ_{min}	42%
Термін окупності інвестицій	$T_{ок}$	2 роки
Рівень комерційного потенціалу	-	46,67 (Високий)

Аналіз отриманих показників, наведених у таблиці 4.12, свідчить про високу економічну доцільність проекту. Зокрема, додатне значення чистого приведеного доходу $E_{абс} > 0$ та значне перевищення внутрішньої норми дохідності $E_В = 50\%$ над бар'єрною ставкою $\tau_{min} = 42\%$ гарантують ефективність вкладених коштів. Враховуючи короткий термін окупності (2 роки) та високий рівень комерційного потенціалу розробки, проект можна вважати стійким до ризиків та доцільним для реалізації за участі потенційного інвестора.

4.4 Висновки до розділу

На основі проведених досліджень, рівень комерційного потенціалу розробки за темою “Метод та засіб потокового шифрування на основі квазігруп” становить 46.67 бала, що, свідчить про високий рівень комерційного потенціалу розробки.

Після оцінки всіх очікуваних витрат на реалізацію проекту було отримано фінальну суму 272710.73 грн., що є хорошим показником враховуючи те, що розробка виконується у сфері криптографії та має на меті підвищення рівня захисту інформації від несанкціонованого доступу.

Розрахований термін окупності становить 2 роки, що менше 3-х років та свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

На основі отриманих результатів, можна вважати доцільним проведення науково-дослідної роботи за темою “Метод та засіб потокового шифрування на основі квазігруп”.

ВИСНОВКИ

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. Такий підхід сприяє підвищенню рівня безпеки інформаційних систем.

Проведений аналіз показав, що потокове шифрування є хорошим інструментом для забезпечення конфіденційності даних. Високий рівень стійкості до атак досягається завдяки генерації псевдовипадкових послідовностей, що є основним елементом цього процесу.

Під час аналізу області LW-криптографії визначено, що зі зростанням обсягів даних існує нагальна потреба в ефективних та масштабованих алгоритмах шифрування. Квазігрупи, завдяки своїм властивостям, пропонують перспективні рішення для задоволення цих вимог, забезпечуючи високу ентропію та стійкість до криптоаналітичних атак.

Квазігрупи виявились перспективним інструментом для побудови криптографічних алгоритмів завдяки своїм унікальним математичним властивостям. Вони можуть забезпечити значну стійкість до атак завдяки високій ентропії та складності.

Використовуючи отримані результати, було розроблено метод потокового шифрування на основі квазігруп. Описано основні кроки виконання такого алгоритму, для шифрування вхідних повідомлень.

На основі описаного методу, реалізовано операційний блок такого шифру. Такий апаратний блок продемонстрував працездатність описаного методу, шляхом зашифрування та розшифрування вхідних повідомлень. Таким чином

доведено коректність та можливість інтеграції та використання розробленого апаратного блоку.

На основі реалізованого та протестованого операційного блоку розроблено модель апаратно-програмного комплексу потокового шифрування. Структура апаратно-програмного комплексу засобу складається з розробленого апаратного блоку, інтерфейсу взаємодії та програмного драйверу. Тестування розробленого рішення демонструє успішність інтергації апаратного блоку у прогамне середовище, шляхом проходження тестування та успішного зашифрування та розшифрування вхідних повідомлень.

Розроблений операційний блок потокового шифру на основі квазігруп має складність 701.73 GE, що є прийнятним для сфери малоресурсної криптографії та кращим у порівнянні з аналогами.

Розроблений апаратний операційний блок потокового шифру був успішно втілений в апаратній реалізації на ПЛІС сімейства Cyclone V та протестований на тестовому наборі даних. Це підтверджує працездатність та функціональну коректність запропонованого апаратного рішення.

Таким чином було виконано всі поставлені завдання і досягнуто головну мету дослідження, яка формується як зменшення апаратної складності засобу потокового шифрування на основі квазігруп.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Status Report on the Final Round of the NIST Lightweight Cryptography Standardization Process. *NIST Internal Report 8454*. URL: https://nvlpubs.nist.gov/nistpubs/ir/2023/NIST.IR.8454.pdf (date of access: 08.09.2025).

2. Al-Garadi M. A. та ін. A Survey of Machine and Deep Learning Methods for Internet of Things (IoT) Security. *IEEE Communications Surveys & Tutorials*. 2020. Vol. 22, Iss. 3. P. 1646–1685. DOI: 10.1109/COMST.2020.2988293 (date of access: 08.09.2025).

3. Conti M., Losiouk E., Poovendran R., Spolaor R. Side-channel attacks on mobile and IoT devices for Cyber–Physical systems. *Computer Networks*. 2022. Vol. 202. P. 108582. DOI: 10.1016/j.comnet.2021.108582 (date of access: 08.09.2025).

4. Maes R. A Review of Fault-Resistant Logic and Fault Attacks. *Fault-Resistant Design Methodologies for Nanoscale CMOS VLSI*. Cham : Springer, 2023. P. 15–43. DOI: 10.1007/978-3-031-15585-1_2 (date of access: 08.09.2025).

5. Dey S., Singh J. K., Singh S. K. A comprehensive survey of lightweight block ciphers for resource-constrained IoT devices. *Journal of Information Security and Applications*. 2023. Vol. 72. P. 103407. DOI: 10.1016/j.jisa.2022.103407 (date of access: 08.09.2025).

6. Jeyashankher R. K. C. G. та ін. A Comprehensive Survey on the Implementations, Attacks, and Countermeasures of the Current NIST Lightweight Cryptography Standard. *arXiv preprint arXiv:2304.06222*. 2023. DOI: 10.48550/arXiv.2304.06222 (date of access: 08.09.2025).

7. Pericherla S. S., Rallabandi V., Lee S. A Survey on Lightweight Cryptographic Algorithms in IoT. *Cybernetics and Information Technologies*. 2024. Vol. 24, Iss. 1. P. 21–43. DOI: 10.2478/cait-2024-0002 (date of access: 08.09.2025).

8. Abead S. A., Ali N. H. M. Lightweight Block and Stream Cipher Algorithm: A Review. *Journal of Applied Engineering and Technological Science (JAETS)*. 2024. Vol. 5(2). P. 860–874. DOI: 10.37385/jaets.v5i2.3966 (date of access: 08.09.2025).

9. Robshaw M. J. Stream Ciphers. *In: van Tilborg H., Jajodia S. (eds) Encyclopedia of Cryptography and Security*. New York, NY : Springer, 2011. P. 1261–1266. DOI: 10.1007/978-1-4419-5906-5_423 (date of access: 08.09.2025).

10. Paar C., Pelzl J. Stream Ciphers. *In: Understanding Cryptography: A Textbook for Students and Practitioners*. Berlin, Heidelberg : Springer, 2010. P. 107–134. DOI: 10.1007/98-3-642-04101-3_5 (date of access: 16.09.2025).

11. Practical Cryptography. LFSRs and the Berlekamp–Massey Algorithm. *Modern Cryptanalysis*. 2016. URL: http://practicalcryptography.com/cryptanalysis-/modern-cryptanalysis/lfsrs-and-berlekampmassey-algorithm/ (date of access: 16.09.2025).

12. 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: 16.09.2025).

13. (Li X., Zhang B., Wang K., Li F. Research Status of Nonlinear Feedback Shift Register Based on Semi-Tensor Product. *Mathematics*. 2022. Vol. 10(19). P. 3538. DOI: 10.3390/math10193538 (date of access: 22.09.2025).

14. Hell M., Johansson T., Maximov A., Meier W. A Stream Cipher Grains of Salt. *In: New Stream Cipher Designs. Lecture Notes in Computer Science*. Berlin, Heidelberg : Springer, 2008. Vol 4986. P. 1–17. DOI: 10.1007/978-3-540-68351-3_1 (date of access: 22.09.2025).

15. Rashidi B. Lightweight Cryptographic S-Boxes Based on Efficient Hardware Structures for Block Ciphers. *The ISC International Journal of Information Security*. 2023. Vol. 15(1). P. 137–151. DOI: 10.22042/isecure.2022.275268.646 (date of access: 22.09.2025).

16. Hua Z. та ін. Design and application of an S-box using complete Latin square. *Nonlinear Dynamics*. 2021. Vol. 104, Iss. 2. P. 807–825. DOI: 10.1007/s11071-021-06308-3 (date of access: 22.09.2025).

17. Shcherbacov V. Quasigroups in cryptology. *Elements of Quasigroup Theory and Applications*. 2017. pp. 471–508. DOI: 10.1201/9781315120058-14 (date of access: 22.09.2025).

18. Shannon C. E. Communication Theory of Secrecy Systems. *Bell System Technical Journal*. 1949. Vol. 28, Iss. 4. P. 656–715. DOI: 10.1002/j.1538-7305.1949.tb00928.x (date of access: 22.09.2025).

19. Dénes J., Keedwell A. D. *Latin Squares and their Applications*. 2nd ed. Amsterdam : Elsevier, 2015. 642 p (date of access: 22.09.2025).

20. Wang H., Yu S., Chen C., Turhan B., Zhu X. Beyond Accuracy: An Empirical Study on Unit Testing in Open-source Deep Learning Projects // ACM Transactions on Software Engineering and Methodology. – New York: Association for Computing Machinery (ACM), 2024. – Vol. 33, No. 4. – Article 104. DOI: 10.1145/3638245 (date of access: 22.09.2025).

21. Wild N., Lichter H., Mensendiek C. Expectation-Based Integration Testing of Unidirectional Interactions in Component-Based Software Systems // Proceedings of the 19th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2024). – SciTePress, 2024. – P. 341–348. DOI: 10.5220/0012725900003687 (date of access: 22.09.2025).

22. Li, Y. Research of Improving Universal Asynchronous Receiver/Transmitter Speed / Y. Li // Proceedings of the 2025 2nd International Conference on Mechanics, Electronics Engineering and Automation (ICMEEA 2025) : Advances in Engineering Research, vol. 272. – 2025. – P. 406–411. – DOI: 10.2991/978-94-6463-821-9_42. (date of access: 22.11.2025).

23. Abboush M., Veiga J., García D., Gutiérrez-Moreno E., Valera A. A Virtual Testing Framework for Real-Time Validation of Safety-Critical Systems Based on Hardware-in-the-Loop and Fault Injection. Sensors. 2024. Vol. 24(12). P. 3733. DOI: 10.3390/s24123733 (date of access: 27.11.2025).

24. Ham E., Jeon Y., Lim J., Kim J. H. Verilator-based Fast Verification Methodology for BLE MAC Hardware. In: 2023 International Conference on Electronics, Information, and Communication (ICEIC 2023). Singapore, 5–8 Feb. 2023. IEEE, 2023. DOI: 10.1109/ICEIC57457.2023.10049940 (date of access: 27.11.2025).

25. Lamb C., Zacchiroli S. Reproducible Builds: Increasing the Integrity of Software Supply Chains. IEEE Software. 2021. Vol. 39(2). Pp. 62–70. DOI: 10.1109/MS.2021.3073045 (date of access: 27.11.2025).

26. Козловський В. О. , Лесько О. Й., Кавецький В. В. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт. Вінниця . ВНТУ. 2021. 42 с.

27. Каплун В. А., Куперштейн Л. М., Баришев Ю. В., Войтович О. П. Методичні вказівки до виконання магістерських кваліфікаційних робіт зі спеціальності «Кібербезпека та захист інформації» освітньо-професійна програма «Безпека інформаційних і комунікаційних систем». Вінниця . ВНТУ. 2024. 96 с.

28. 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.

29. Загирняк Б. Д. Крайнічук (Шелепало) Г. В. Алгоритм потокового шифрування на основі латинських квадратів. *SMICS: Безпека сучасних інформаційно-комунікаційних систем: матеріали міжнар. наук.-техн. конф., м. Львів, 16-18 жовтня 2025 р.* ЛНУ ім. І. Франка, 2025, С. 246 – 250.

ДОДАТКИ

Додаток А

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Метод та засіб потокового шифрування на основі квазігруп. Частина 2. Операційний блок

Автор роботи: Загирняк Богдан Дмитрович

Тип роботи: магістерська кваліфікаційна робота

Підрозділ кафедра захисту інформації ФІТКІ , група 1 БС-24м

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 1,55 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

√ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

В. о. зав. кафедри ЗІ д. т. н., проф. _____ Володимир ЛУЖЕЦЬКИЙ

Гарант освітньої програми «Безпека інформаційних і комунікаційних систем» к.т.н., доцент _____ Олеся ВОЙТОВИЧ

Особа, відповідальна за перевірку _____ Валентина КАПЛУН

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Галина ШЕЛЕПАЛО

Здобувач _____ Богдан ЗАГИРНЯК

Додаток Б
КОД ПРОГРАМНОГО ЗАСОБУ

cipher_core.v

```
// Модуль Ядра Шифратора (Cipher Core) - Версія 5.0 (Оптимізована)
// Логіка повністю ідентична версії 4.1, але реалізована
// через булеві вирази замість великих LUT (case).
module cipher_core (
    input  wire [1:0] i_key_bits,       // Y
    input  wire [1:0] i_keystream_bits, // X
    input  wire [1:0] i_data_bits,      // Z (encrypt) or M (decrypt)
    input  wire       i_mode,           // 0 = encrypt, 1 = decrypt
    output wire [1:0] o_data_bits,      // M (encrypt) or Z (decrypt)
    output wire [1:0] o_new_key_bits
);
    // --- Проміжні сигнали ---
    wire [1:0] encrypted_result;
    wire [1:0] decrypted_result;
    wire [1:0] data_for_key_update;

    // --- 1. Логіка Шифрування ---
    // Формула: M = LS1(Z, X) ^ Y
    // LS1(P,K) = {P[0]^K[1]^K[0], P[1]^P[0]^K[1]}
    // Заміна: P -> i_data_bits (Z), K -> i_keystream_bits (X)

    wire [1:0] M_temp;
    assign M_temp[1] = i_data_bits[0] ^ i_keystream_bits[1] ^
i_keystream_bits[0];
    assign M_temp[0] = i_data_bits[1] ^ i_data_bits[0] ^ i_keystream_bits[1];

    assign encrypted_result = M_temp ^ i_key_bits; // M = M_temp ^ Y

    // --- 2. Логіка Розшифрування ---
    // Формула: Z = LS1_inv(M ^ Y, X)
    // LS1_inv(K',K) = {K'[1]^K'[0]^K[0], K'[1]^K[1]^K[0]}
    // Заміна: M -> i_data_bits, Y -> i_key_bits, X -> i_keystream_bits

    wire [1:0] M_pre;
    assign M_pre = i_data_bits ^ i_key_bits; // K' = M ^ Y

    // Заміна: K' -> M_pre, K -> i_keystream_bits (X)
    assign decrypted_result[1] = M_pre[1] ^ M_pre[0] ^ i_keystream_bits[0];
    assign decrypted_result[0] = M_pre[1] ^ i_keystream_bits[1] ^
i_keystream_bits[0];

    // --- 3. Вибір вихідних даних ---
    assign o_data_bits = i_mode ? decrypted_result : encrypted_result;

    // --- 4. Логіка Оновлення Ключа ---
    // Формула: K' = LS1(P, K)
    // LS1(P,K) = {P[0]^K[1]^K[0], P[1]^P[0]^K[1]}
    // Вхід P (plaintext) обирається залежно від режиму
    assign data_for_key_update = i_mode ? decrypted_result : i_data_bits;
    // Заміна: P -> data_for_key_update, K -> i_key_bits
    assign o_new_key_bits[1] = i_key_bits[1] ^ data_for_key_update[0];
    assign o_new_key_bits[0] = i_key_bits[0] ^ data_for_key_update[1] ^
(i_key_bits[1] & ~data_for_key_update[0]);
endmodule
```

keystream_generator.v

```
module keystream_generator (
    input wire clk,
    input wire rst_n,
    input wire i_start,
    input wire [1:0] i_ls_select,
    output reg [1:0] o_keystream_bits,
    output reg o_ready
);

    // --- Оголошення регістрів ---
    reg [1:0] u1_reg, u2_reg, u3_reg;

    // --- Оголошення ВСІХ Wires ---
    wire [1:0] ls0_idx, ls1_idx;
    wire [1:0] intermediate;
    wire [1:0] next_val;

    // Wires для логічного блоку A (обчислення intermediate)
    wire [1:0] ls1_out_A, ls2_out_A, ls3_out_A, ls4_out_A;

    // Wires для логічного блоку B (обчислення next_val)
    wire [1:0] ls1_out_B, ls2_out_B, ls3_out_B, ls4_out_B;
    // --- Початок логіки ---
    // Вибір пари матриць
    assign {ls1_idx, ls0_idx} =
        (i_ls_select == 2'b00) ? {2'd1, 2'd0} :  // LS1/LS2
        (i_ls_select == 2'b01) ? {2'd0, 2'd1} :  // LS2/LS1
        (i_ls_select == 2'b10) ? {2'd3, 2'd2} :  // LS3/LS4
        {2'd3, 2'd0};                            // default LS1/LS4


    // --- Логічний Блок A: Обчислення 'intermediate' ---
    // (Логіка на основі u1_reg, u2_reg)

    // LS1 (sel=00): Скручений XOR
    assign ls1_out_A = {u1_reg[0] ^ u2_reg[1], u1_reg[1] ^ u2_reg[0]};

    // LS2 (sel=01): Прямий XOR
    assign ls2_out_A = u1_reg ^ u2_reg;

    // LS3 (sel=10): Мінімізована логіка
    assign ls3_out_A = {
        u1_reg[1] ^ (u1_reg[0] ? u2_reg[0] : u2_reg[1]), // Bit 1
        u1_reg[1] ^ u2_reg[1] ^ u2_reg[0]              // Bit 0
    };
    // LS4 (sel=11): Мінімізована логіка
    assign ls4_out_A = {
        // Bit 1:
        u1_reg[1] ? (u1_reg[0] ? ~u2_reg[1] : u2_reg[0]) : (u1_reg[0] ?
~u2_reg[0] : u2_reg[1]),
        // Bit 0:
        u1_reg[1] ? (u1_reg[0] ? u2_reg[0] : u2_reg[1]) : (u1_reg[0] ?
~u2_reg[1] : ~u2_reg[0])
    };

    // Фінальний мультиплексор для 'intermediate'
    assign intermediate = (ls0_idx == 2'b00) ? ls1_out_A :
                          (ls0_idx == 2'b01) ? ls2_out_A :
                          (ls0_idx == 2'b10) ? ls3_out_A :
                          ls4_out_A; // (ls0_idx == 2'b11)
```

```
    // --- Логічний Блок B: Обчислення 'next_val' ---
    // (Логіка на основі intermediate, u3_reg)

    // LS1 (sel=00): Скручений XOR
    assign ls1_out_B = {intermediate[0] ^ u3_reg[1], intermediate[1] ^
u3_reg[0]};

    // LS2 (sel=01): Прямий XOR
    assign ls2_out_B = intermediate ^ u3_reg;

    // LS3 (sel=10): Мінімізована логіка
    assign ls3_out_B = {
        intermediate[1] ^ (intermediate[0] ? u3_reg[0] : u3_reg[1]), // Bit 1
        intermediate[1] ^ u3_reg[1] ^ u3_reg[0]                      // Bit 0
    };
    // LS4 (sel=11): Мінімізована логіка
    assign ls4_out_B = {
        // Bit 1:
        intermediate[1] ? (intermediate[0] ? ~u3_reg[1] : u3_reg[0]) :
(intermediate[0] ? ~u3_reg[0] : u3_reg[1]),
        // Bit 0:
        intermediate[1] ? (intermediate[0] ? u3_reg[0] : u3_reg[1]) :
(intermediate[0] ? ~u3_reg[1] : ~u3_reg[0])
    };

    // Фінальний мультиплексор для 'next_val'
    assign next_val = (ls1_idx == 2'b00) ? ls1_out_B :
                      (ls1_idx == 2'b01) ? ls2_out_B :
                      (ls1_idx == 2'b10) ? ls3_out_B :
                      ls4_out_B; // (ls1_idx == 2'b11)

    // Послідовна логіка (без змін)
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            u1_reg <= 2'd1;
            u2_reg <= 2'd2; u3_reg <= 2'd3;
            o_keystream_bits <= 2'd0; o_ready <= 1'b0;
        end else begin
            if (i_start) begin
                u1_reg <= u2_reg;
                u2_reg <= u3_reg; u3_reg <= next_val;
                o_keystream_bits <= u3_reg;
                o_ready <= 1'b1;
            end else begin
                o_ready <= 1'b0;
            end
        end
    end
endmodule
```

top_controller.v

```
module cipher_top (
    input wire clk,
    input wire rst_n,

    // Control
    input wire i_start,              // Почати нову сесію шифрування
    input wire i_mode,               // 0 = encrypt, 1 = decrypt
    input wire [1:0] i_ls_select,    // Вибір Latin Squares
    input wire [63:0] i_key_init,    // Початковий ключ (64 бити)

    // Stream interface
```

```
    input wire i_data_valid,         // Вхідні дані готові
    input wire [1:0] i_data_pair,    // Вхідна пара бітів (2 біти/такт)
    output reg o_data_valid,         // Вихідні дані готові
    output reg [1:0] o_data_pair,    // Вихідна пара бітів

    // Status
    output reg o_ready,              // Готовий приймати дані
    output reg o_busy                // В процесі обробки
);

    // ========================
    // Внутрішні сигнали
    // ========================

    // Keystream Generator
    reg kg_start;
    reg kg_rst_n;
    wire [1:0] kg_keystream_bits;
    wire kg_ready;

    // Cipher Core
    reg [1:0] cc_key_bits;
    reg [1:0] cc_keystream_bits;
    reg [1:0] cc_data_bits;
    reg cc_mode;
    wire [1:0] cc_data_out;
    wire [1:0] cc_new_key_bits;

    // Проміжні регістри
    reg [1:0] ls_select_reg;
    reg [63:0] key_register;         // 64-bit ключ (змінено з 8-bit)
    reg mode_reg;

    // Стани FSM
    localparam IDLE       = 3'd0;
    localparam INIT       = 3'd1;
    localparam WAIT_DATA  = 3'd2;
    localparam GEN_KS     = 3'd3;
    localparam WAIT_KS    = 3'd4;
    localparam PROCESS    = 3'd5;
    localparam OUTPUT     = 3'd6;

    reg [2:0] state, next_state;
    // ========================
    // Підключення модулів
    // ========================
    keystream_generator kg_inst (
        .clk(clk),
        .rst_n(kg_rst_n),
        .i_start(kg_start),
        .i_ls_select(ls_select_reg),
        .o_keystream_bits(kg_keystream_bits),
        .o_ready(kg_ready)
    );

    cipher_core cc_inst (
        .i_key_bits(cc_key_bits),
        .i_keystream_bits(cc_keystream_bits),
        .i_data_bits(cc_data_bits),
        .i_mode(cc_mode),
        .o_data_bits(cc_data_out),
        .o_new_key_bits(cc_new_key_bits)
    );
```

```
// ========================
// FSM - Sequential Logic
// ========================

always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        state <= IDLE;
    end else begin
        state <= next_state;
    end
end

// ========================
// FSM - Combinational Logic
// ========================

always @(*) begin
    next_state = state;
    if (i_start) begin
        next_state = INIT;
    end else begin
        case (state)
            IDLE: next_state = IDLE;
            INIT: next_state = WAIT_DATA;
            WAIT_DATA: begin
                if (i_data_valid) begin
                    next_state = GEN_KS;
                end else begin
                    next_state = WAIT_DATA;
                end
            end
            GEN_KS: next_state = WAIT_KS;
            WAIT_KS: begin
                if (kg_ready) begin
                    next_state = PROCESS;
                end else begin
                    next_state = WAIT_KS;
                end
            end
            PROCESS: next_state = OUTPUT;
            OUTPUT: next_state = WAIT_DATA;
            default: next_state = IDLE;
        endcase
    end
end

// ========================
// Основна логіка обробки
// ========================

always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        // Скидання всіх регістрів
        key_register <= 64'd0;          // 64-bit ключ (змінено)
        o_data_pair <= 2'd0;
        o_data_valid <= 1'b0;
        o_ready <= 1'b0;
        o_busy <= 1'b0;
        kg_start <= 1'b0;
        kg_rst_n <= 1'b0;
        cc_key_bits <= 2'd0;
        cc_keystream_bits <= 2'd0;
        cc_data_bits <= 2'd0;
        cc_mode <= 1'b0;
```

```
            ls_select_reg <= 2'd0;
            mode_reg <= 1'b0;
        end else begin
            // Уніфіковане скидання генератора keystream при i_start
            kg_rst_n <= i_start ? 1'b0 : 1'b1;

            case (state)
                IDLE: begin
                    o_data_valid <= 1'b0;
                    o_ready <= 1'b0;
                    o_busy <= 1'b0;
                    kg_start <= 1'b0;
                end

                INIT: begin
                    // Ініціалізація для нової сесії
                    key_register <= i_key_init;   // Завантажити 64-bit ключ
                    ls_select_reg <= i_ls_select;
                    mode_reg <= i_mode;
                    cc_mode <= i_mode;
                    o_ready <= 1'b1;                    // Готові приймати дані
                    o_busy <= 1'b0;
                    o_data_valid <= 1'b0;
                end

                WAIT_DATA: begin
                    o_ready <= 1'b1;
                    o_busy <= 1'b0;
                    o_data_valid <= 1'b0;
                    kg_start <= 1'b0;

                    // Коли отримали дані, готуємось до обробки
                    if (i_data_valid) begin
                        cc_data_bits <= i_data_pair;
                        cc_key_bits <= key_register[63:62];// Беремо ТОП 2 біти
ключа
                    end
                end

                GEN_KS: begin
                    // Запуск генератора keystream
                    kg_start <= 1'b1;
                    o_ready <= 1'b0;
                    o_busy <= 1'b1;
                    o_data_valid <= 1'b0;
                end

                WAIT_KS: begin
                    kg_start <= 1'b0;
                    o_ready <= 1'b0;
                    o_busy <= 1'b1;

                    if (kg_ready) begin
                        cc_keystream_bits <= kg_keystream_bits;
                    end
                end

                PROCESS: begin
                    // Результат доступний комбінаційно
                    // Циклічно зсуваємо 64-bit ключ на 2 біти
                    key_register <= {key_register[61:0], cc_new_key_bits};
                    o_busy <= 1'b1;
                    o_ready <= 1'b0;
                end
```

```
                OUTPUT: begin
                    // Виведення результату
                    o_data_pair <= cc_data_out;
                    o_data_valid <= 1'b1;
                    o_ready <= 1'b0;                    // Не приймаємо під час
виводу
                    o_busy <= 1'b0;
                end

                default: begin
                    o_data_valid <= 1'b0;
                    o_ready <= 1'b0;
                    o_busy <= 1'b0;
                end
            endcase
        end
    end

endmodule
```

top_controller.v

```
// Cipher with UART Interface - 64-bit Key Version
// Protocol:
//   CMD 0x01: START_SESSION [mode] [ls_select] [key_byte0-7]  (11 байтів)
//   CMD 0x02: PROCESS_DATA [data_byte]
//   CMD 0x03: GET_STATUS
// Responses:
//   0x80: ACK
//   0x81: DATA [result_byte]
//   0x82: STATUS [flags]
//   0xFF: ERROR

module cipher_with_uart #(
    parameter CLK_FREQ = 50_000_000,
    parameter BAUD_RATE = 115200
)(
    input wire clk,
    input wire rst_n,

    // UART interface
    input wire uart_rx,
    output wire uart_tx,

    // Optional status LEDs
    output wire led_ready,
    output wire led_busy
);

    localparam CLKS_PER_BIT = CLK_FREQ / BAUD_RATE;

    // UART RX signals
    wire [7:0] rx_byte;
    wire rx_valid;

    // UART TX signals
    reg [7:0] tx_byte;
    reg tx_valid;
    wire tx_ready;

    // Cipher signals
    reg cipher_start;
    reg cipher_mode;
```

```
reg [1:0] cipher_ls_select;
reg [63:0] cipher_key_init;       // 64-bit ключ (змінено з 8-bit)
reg cipher_data_valid;
reg [1:0] cipher_data_pair;
wire cipher_data_out_valid;
wire [1:0] cipher_data_out_pair;
wire cipher_ready;
wire cipher_busy;

// Protocol FSM states
localparam IDLE              = 4'd0;
localparam CMD_START_1       = 4'd1;  // Receive mode
localparam CMD_START_2       = 4'd2;  // Receive ls_select
localparam CMD_START_KEY     = 4'd3;  // Receive key bytes (0-7)
localparam CMD_START_ACK     = 4'd4;  // Send ACK
localparam CMD_DATA          = 4'd5;  // Receive data byte
localparam PROCESS_INIT      = 4'd6;  // Initialize cipher processing
localparam PROCESS_PAIR      = 4'd7;  // Process one pair
localparam WAIT_CIPHER       = 4'd8;  // Wait for cipher output
localparam COLLECT_PAIR      = 4'd9;  // Collect output pair
localparam SEND_RESULT       = 4'd10; // Send result byte
localparam SEND_STATUS       = 4'd11; // Send status

reg [3:0] state, next_state;
reg [2:0] key_byte_counter;        // Лічильник для 8 байтів ключа (змінено)
reg [7:0] data_buffer;
reg [7:0] result_buffer;
reg [2:0] pair_counter;            // 0-3 for 4 pairs per byte

// UART modules
uart_rx #(.CLKS_PER_BIT(CLKS_PER_BIT)) rx_inst (
    .clk(clk),
    .rst_n(rst_n),
    .i_uart_rx(uart_rx),
    .o_rx_byte(rx_byte),
    .o_rx_valid(rx_valid)
);

uart_tx #(.CLKS_PER_BIT(CLKS_PER_BIT)) tx_inst (
    .clk(clk),
    .rst_n(rst_n),
    .i_tx_byte(tx_byte),
    .i_tx_valid(tx_valid),
    .o_tx_ready(tx_ready),
    .o_uart_tx(uart_tx)
);

// Cipher core
cipher_top cipher_inst (
    .clk(clk),
    .rst_n(rst_n),
    .i_start(cipher_start),
    .i_mode(cipher_mode),
    .i_ls_select(cipher_ls_select),
    .i_key_init(cipher_key_init),  // 64-bit ключ
    .i_data_valid(cipher_data_valid),
    .i_data_pair(cipher_data_pair),
    .o_data_valid(cipher_data_out_valid),
    .o_data_pair(cipher_data_out_pair),
    .o_ready(cipher_ready),
    .o_busy(cipher_busy)
);

// LED outputs
```

```
    assign led_ready = cipher_ready;
    assign led_busy = cipher_busy;

    // FSM Sequential Logic
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            state <= IDLE;
        end else begin
            state <= next_state;
        end
    end

    // FSM Combinational Logic
    always @(*) begin
        next_state = state;

        case (state)
            IDLE: begin
                if (rx_valid) begin
                    case (rx_byte)
                        8'h01: next_state = CMD_START_1;
                        8'h02: next_state = CMD_DATA;
                        8'h03: next_state = SEND_STATUS;
                        default: next_state = IDLE;
                    endcase
                end
            end

            CMD_START_1: if (rx_valid) next_state = CMD_START_2;
            CMD_START_2: if (rx_valid) next_state = CMD_START_KEY;

            CMD_START_KEY: begin
                if (rx_valid) begin
                    // Якщо отримали останній (8-й) байт ключа, переходимо до
ACK
                    if (key_byte_counter == 3'd7)
                        next_state = CMD_START_ACK;
                    // Інакше залишаємось у CMD_START_KEY для наступного байта
                end
            end

            CMD_START_ACK: if (tx_ready) next_state = IDLE;

            CMD_DATA: if (rx_valid) next_state = PROCESS_INIT;

            PROCESS_INIT: next_state = PROCESS_PAIR;

            PROCESS_PAIR: if (cipher_ready) next_state = WAIT_CIPHER;

            WAIT_CIPHER: if (cipher_data_out_valid) next_state = COLLECT_PAIR;

            COLLECT_PAIR: begin
                if (pair_counter == 3'd3)
                    next_state = SEND_RESULT;
                else
                    next_state = PROCESS_PAIR;
            end

            SEND_RESULT: if (tx_ready) next_state = IDLE;
            SEND_STATUS: if (tx_ready) next_state = IDLE;

            default: next_state = IDLE;
        endcase
    end
```

```
    // FSM Output Logic
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            key_byte_counter <= 3'd0;
            data_buffer <= 8'd0;
            result_buffer <= 8'd0;
            pair_counter <= 3'd0;
            cipher_key_init <= 64'd0;      // 64-bit ключ

            cipher_start <= 1'b0;
            cipher_mode <= 1'b0;
            cipher_ls_select <= 2'd0;
            cipher_data_valid <= 1'b0;
            cipher_data_pair <= 2'd0;

            tx_byte <= 8'd0;
            tx_valid <= 1'b0;
        end else begin
            // Defaults
            cipher_start <= 1'b0;
            cipher_data_valid <= 1'b0;
            tx_valid <= 1'b0;

            case (state)
                IDLE: begin
                    key_byte_counter <= 3'd0;
                    pair_counter <= 3'd0;
                    result_buffer <= 8'd0;
                end

                CMD_START_1: begin
                    if (rx_valid) begin
                        cipher_mode <= rx_byte[0];
                    end
                end

                CMD_START_2: begin
                    if (rx_valid) begin
                        cipher_ls_select <= rx_byte[1:0];
                    end
                end

                CMD_START_KEY: begin
                    if (rx_valid) begin
                        // Записуємо байт у відповідну позицію 64-бітного ключа
                        case (key_byte_counter)
                            3'd0: cipher_key_init[7:0]   <= rx_byte;
                            3'd1: cipher_key_init[15:8]  <= rx_byte;
                            3'd2: cipher_key_init[23:16] <= rx_byte;
                            3'd3: cipher_key_init[31:24] <= rx_byte;
                            3'd4: cipher_key_init[39:32] <= rx_byte;
                            3'd5: cipher_key_init[47:40] <= rx_byte;
                            3'd6: cipher_key_init[55:48] <= rx_byte;
                            3'd7: cipher_key_init[63:56] <= rx_byte;
                        endcase

                        // Якщо це 8-й байт, не інкрементуємо (для ACK)
                        if (key_byte_counter < 3'd7)
                            key_byte_counter <= key_byte_counter + 1;
                        else
                            cipher_start <= 1'b1;  // Запуск шифру після
завантаження ключа
                    end
```

```
                end

                CMD_START_ACK: begin
                    tx_byte <= 8'h80;  // ACK
                    tx_valid <= 1'b1;
                end

                CMD_DATA: begin
                    if (rx_valid) begin
                        data_buffer <= rx_byte;
                    end
                end

                PROCESS_INIT: begin
                    pair_counter <= 3'd0;
                    result_buffer <= 8'd0;
                end

                PROCESS_PAIR: begin
                    if (cipher_ready) begin
                        // Extract pair from MSB first: bits [7:6], [5:4],
[3:2], [1:0]
                        case (pair_counter)
                            3'd0: cipher_data_pair <= data_buffer[7:6];
                            3'd1: cipher_data_pair <= data_buffer[5:4];
                            3'd2: cipher_data_pair <= data_buffer[3:2];
                            3'd3: cipher_data_pair <= data_buffer[1:0];
                            default: cipher_data_pair <= 2'd0;
                        endcase
                        cipher_data_valid <= 1'b1;
                    end
                end

                COLLECT_PAIR: begin
                    // Collect pairs into result_buffer (MSB first)
                    case (pair_counter)
                        3'd0: result_buffer[7:6] <= cipher_data_out_pair;
                        3'd1: result_buffer[5:4] <= cipher_data_out_pair;
                        3'd2: result_buffer[3:2] <= cipher_data_out_pair;
                        3'd3: result_buffer[1:0] <= cipher_data_out_pair;
                    endcase
                    pair_counter <= pair_counter + 1;
                end

                SEND_RESULT: begin
                    tx_byte <= result_buffer;
                    tx_valid <= 1'b1;
                end

                SEND_STATUS: begin
                    tx_byte <= {6'd0, cipher_busy, cipher_ready};
                    tx_valid <= 1'b1;
                end
            endcase
        end
    end

endmodule
```

uart_rx.v

```
// UART Receiver Module
// Baudrate configurable via CLKS_PER_BIT parameter
// Default: 50MHz clock, 115200 baud => 434 clocks/bit
```

```
module uart_rx #(
    parameter CLKS_PER_BIT = 434  // 50MHz / 115200 = 434
)(
    input wire clk,
    input wire rst_n,

    // UART line
    input wire i_uart_rx,          // UART RX line

    // Data interface
    output reg [7:0] o_rx_byte,    // Received byte
    output reg o_rx_valid          // Data valid pulse
);

    // States
    localparam IDLE  = 3'd0;
    localparam START = 3'd1;
    localparam DATA  = 3'd2;
    localparam STOP  = 3'd3;

    reg [2:0] state;
    reg [15:0] clk_count;
    reg [2:0] bit_index;
    reg [7:0] rx_data;

    // Input synchronizer (prevent metastability)
    reg rx_sync1, rx_sync2;

    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            rx_sync1 <= 1'b1;
            rx_sync2 <= 1'b1;
        end else begin
            rx_sync1 <= i_uart_rx;
            rx_sync2 <= rx_sync1;
        end
    end

    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            state <= IDLE;
            o_rx_valid <= 1'b0;
            clk_count <= 16'd0;
            bit_index <= 3'd0;
            rx_data <= 8'd0;
            o_rx_byte <= 8'd0;
        end else begin
            o_rx_valid <= 1'b0;  // Default: pulse only one cycle

            case (state)
                IDLE: begin
                    clk_count <= 16'd0;
                    bit_index <= 3'd0;

                    if (rx_sync2 == 1'b0) begin  // Start bit detected
                        state <= START;
                    end
                end

                START: begin
                    // Sample in the middle of start bit
                    if (clk_count < (CLKS_PER_BIT / 2)) begin
                        clk_count <= clk_count + 1;
```

```
                    end else begin
                        if (rx_sync2 == 1'b0) begin  // Valid start bit
                            clk_count <= 16'd0;
                            state <= DATA;
                        end else begin
                            state <= IDLE;  // False start
                        end
                    end
                end

                DATA: begin
                    if (clk_count < CLKS_PER_BIT - 1) begin
                        clk_count <= clk_count + 1;
                    end else begin
                        clk_count <= 16'd0;
                        rx_data[bit_index] <= rx_sync2;

                        if (bit_index < 7) begin
                            bit_index <= bit_index + 1;
                        end else begin
                            bit_index <= 3'd0;
                            state <= STOP;
                        end
                    end
                end

                STOP: begin
                    if (clk_count < CLKS_PER_BIT - 1) begin
                        clk_count <= clk_count + 1;
                    end else begin
                        clk_count <= 16'd0;

                        if (rx_sync2 == 1'b1) begin  // Valid stop bit
                            o_rx_byte <= rx_data;
                            o_rx_valid <= 1'b1;
                        end

                        state <= IDLE;
                    end
                end

                default: state <= IDLE;
            endcase
        end
    end

endmodule
```

uart_tx.v

```
// UART Transmitter Module
// Baudrate configurable via CLKS_PER_BIT parameter
// Default: 50MHz clock, 115200 baud => 434 clocks/bit

module uart_tx #(
    parameter CLKS_PER_BIT = 434  // 50MHz / 115200 = 434
)(
    input wire clk,
    input wire rst_n,

    // Data interface
    input wire [7:0] i_tx_byte,   // Byte to transmit
    input wire i_tx_valid,         // Start transmission
    output reg o_tx_ready,         // Ready for new byte
```

```
    // UART line
    output reg o_uart_tx           // UART TX line
);

    // States
    localparam IDLE  = 3'd0;
    localparam START = 3'd1;
    localparam DATA  = 3'd2;
    localparam STOP  = 3'd3;

    reg [2:0] state;
    reg [15:0] clk_count;
    reg [2:0] bit_index;
    reg [7:0] tx_data;

    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            state <= IDLE;
            o_uart_tx <= 1'b1;  // Idle high
            o_tx_ready <= 1'b1;
            clk_count <= 16'd0;
            bit_index <= 3'd0;
            tx_data <= 8'd0;
        end else begin
            case (state)
                IDLE: begin
                    o_uart_tx <= 1'b1;
                    o_tx_ready <= 1'b1;
                    clk_count <= 16'd0;
                    bit_index <= 3'd0;

                    if (i_tx_valid) begin
                        tx_data <= i_tx_byte;
                        o_tx_ready <= 1'b0;
                        state <= START;
                    end
                end

                START: begin
                    o_uart_tx <= 1'b0;  // Start bit

                    if (clk_count < CLKS_PER_BIT - 1) begin
                        clk_count <= clk_count + 1;
                    end else begin
                        clk_count <= 16'd0;
                        state <= DATA;
                    end
                end

                DATA: begin
                    o_uart_tx <= tx_data[bit_index];

                    if (clk_count < CLKS_PER_BIT - 1) begin
                        clk_count <= clk_count + 1;
                    end else begin
                        clk_count <= 16'd0;

                        if (bit_index < 7) begin
                            bit_index <= bit_index + 1;
                        end else begin
                            bit_index <= 3'd0;
                            state <= STOP;
                        end
```

```
                    end
                end

                STOP: begin
                    o_uart_tx <= 1'b1;  // Stop bit

                    if (clk_count < CLKS_PER_BIT - 1) begin
                        clk_count <= clk_count + 1;
                    end else begin
                        clk_count <= 16'd0;
                        state <= IDLE;
                    end
                end

                default: state <= IDLE;
            endcase
        end
    end

endmodule
```

sim_main.cpp

```
// sim_main.cpp - Main Hardware-in-the-Loop simulation program
#include "cipher_hil.h"
#include <iostream>
#include <iomanip>
#include <string>
#include <vector>
#include <cstring>
#include <sstream>

void print_hex(const std::vector<uint8_t>& data, const std::string& label = "")
{
    if (!label.empty()) {
        std::cout << label << ": ";
    }

    for (size_t i = 0; i < data.size(); i++) {
        std::cout << std::hex << std::setfill('0') << std::setw(2)
                  << (int)data[i] << " ";
        if ((i + 1) % 16 == 0 && i + 1 < data.size()) {
            std::cout << "\n           ";
        }
    }
    std::cout << std::dec << "\n";
}

bool test_hi(CipherHIL& hil) {
    std::cout << "\n" << std::string(50, '=') << "\n";
    std::cout << "TEST 1: Encrypt/Decrypt 'hi'\n";
    std::cout << std::string(50, '=') << "\n";

    std::string plaintext = "hi";
    uint64_t key = 0x0123456789ABCDEFULL;  // 64-bit ключ
    uint8_t ls_sel = 0;

    std::cout << "\nPlaintext: \"" << plaintext << "\"\n";
    auto ciphertext = hil.encrypt(plaintext, ls_sel, key);
    print_hex(ciphertext, "Ciphertext");

    std::string decrypted = hil.decrypt(ciphertext, ls_sel, key);
    std::cout << "Decrypted: \"" << decrypted << "\"\n";
```

```
    bool passed = (decrypted == plaintext);
    std::cout << "\n>>> TEST 1: " << (passed ? "PASSED" : "FAILED") << " <<<\n";

    return passed;
}

bool test_hello(CipherHIL& hil) {
    std::cout << "\n" << std::string(50, '=') << "\n";
    std::cout << "TEST 2: Encrypt/Decrypt 'hello'\n";
    std::cout << std::string(50, '=') << "\n";

    std::string plaintext = "hello";
    uint64_t key = 0x0123456789ABCDEFULL;  // 64-bit ключ
    uint8_t ls_sel = 0;

    std::cout << "\nPlaintext: \"" << plaintext << "\"\n";
    auto ciphertext = hil.encrypt(plaintext, ls_sel, key);
    print_hex(ciphertext, "Ciphertext");

    std::string decrypted = hil.decrypt(ciphertext, ls_sel, key);
    std::cout << "Decrypted: \"" << decrypted << "\"\n";

    bool passed = (decrypted == plaintext);
    std::cout << "\n>>> TEST 2: " << (passed ? "PASSED" : "FAILED") << " <<<\n";

    return passed;
}

bool test_long_message(CipherHIL& hil) {
    std::cout << "\n" << std::string(50, '=') << "\n";
    std::cout << "TEST 3: Long Message\n";
    std::cout << std::string(50, '=') << "\n";

    std::string plaintext = "The quick brown fox jumps over the lazy dog";
    uint64_t key = 0x0123456789ABCDEFULL;  // 64-bit ключ
    uint8_t ls_sel = 0;

    std::cout << "\nPlaintext: \"" << plaintext << "\"\n";
    std::cout << "Length: " << plaintext.length() << " bytes\n";

    auto ciphertext = hil.encrypt(plaintext, ls_sel, key);
    print_hex(ciphertext, "Ciphertext");

    std::string decrypted = hil.decrypt(ciphertext, ls_sel, key);
    std::cout << "\nDecrypted: \"" << decrypted << "\"\n";

    bool passed = (decrypted == plaintext);
    std::cout << "\n>>> TEST 3: " << (passed ? "PASSED" : "FAILED") << " <<<\n";

    return passed;
}

bool test_different_keys(CipherHIL& hil) {
    std::cout << "\n" << std::string(50, '=') << "\n";
    std::cout << "TEST 4: Different Keys & LS Pairs\n";
    std::cout << std::string(50, '=') << "\n";

    std::string plaintext = "test";

    struct TestCase {
        uint64_t key;
        uint8_t ls_sel;
        std::string name;
```

```
    };

    TestCase cases[] = {
        {0x0123456789ABCDEFULL, 0, "Key=0x0123456789ABCDEF, LS=0"},
        {0xFEDCBA9876543210ULL, 1, "Key=0xFEDCBA9876543210, LS=1"},
        {0xAAAAAAAAAAAAAAAAULL, 2, "Key=0xAAAAAAAAAAAAAAAA, LS=2"},
    };

    bool all_passed = true;

    for (const auto& test : cases) {
        std::cout << "\n--- " << test.name << " ---\n";

        auto ciphertext = hil.encrypt(plaintext, test.ls_sel, test.key);
        std::string decrypted = hil.decrypt(ciphertext, test.ls_sel, test.key);

        bool passed = (decrypted == plaintext);
        std::cout << "Result: " << (passed ? "PASS" : "FAIL") << "\n";

        all_passed = all_passed && passed;
    }

    std::cout << "\n>>> TEST 4: " << (all_passed ? "PASSED" : "FAILED") << "
<<<\n";
    return all_passed;
}

std::vector<uint8_t> parse_hex_string(const std::string& hex_str) {
    std::vector<uint8_t> result;
    std::istringstream iss(hex_str);
    std::string token;

    while (iss >> token) {
        unsigned int byte_val;
        if (sscanf(token.c_str(), "%x", &byte_val) == 1 && byte_val <= 0xFF) {
            result.push_back((uint8_t)byte_val);
        } else {
            std::cerr << "Invalid hex byte: " << token << "\n";
            return std::vector<uint8_t>();
        }
    }

    return result;
}

void interactive_mode(CipherHIL& hil) {
    std::cout << "\n" << std::string(60, '=') << "\n";
    std::cout << "INTERACTIVE MODE\n";
    std::cout << std::string(60, '=') << "\n";
    std::cout << "\nCommands:\n";
    std::cout << "  e <text>           - Encrypt text\n";
    std::cout << "  d [hex bytes]      - Decrypt last ciphertext or hex
bytes\n";
    std::cout << "  k <hex>            - Set key (64-bit hex, e.g., k
0123456789ABCDEF)\n";
    std::cout << "  l <0-3>            - Set LS pair (0, 1, 2, or 3)\n";
    std::cout << "  help               - Show this help\n";
    std::cout << "  q                  - Quit\n\n";
    std::cout << "Examples:\n";
    std::cout << "  > e Hello          - Encrypt \"Hello\"\n";
    std::cout << "  > d                - Decrypt last result\n";
    std::cout << "  > d ab 66 37 c0    - Decrypt hex bytes\n";
    std::cout << "  > k FEDCBA9876543210  - Set 64-bit key\n";
    std::cout << "  > l 2              - Set LS pair to 2\n\n";
```

```
    uint64_t key = 0x0123456789ABCDEFULL;  // 64-bit ключ за замовчуванням
    uint8_t ls_sel = 0;
    std::vector<uint8_t> last_ciphertext;

    std::cout << "Current settings: key=0x" << std::hex << std::setfill('0') <<
std::setw(16) << key
              << std::dec << ", ls=" << (int)ls_sel << "\n\n";

    std::string line;
    while (true) {
        std::cout << "> ";
        std::cout.flush();

        if (!std::getline(std::cin, line)) break;

        // Trim leading whitespace
        size_t start = line.find_first_not_of(" \t");
        if (start == std::string::npos) {
            std::cout << "Empty input. Type 'help' for commands.\n";
            continue;
        }
        line = line.substr(start);

        char cmd = line[0];
        std::string args = (line.length() > 2) ? line.substr(2) : "";

        if (cmd == 'q') {
            std::cout << "Exiting...\n";
            break;
        }
        else if (cmd == 'h' && line.substr(0, 4) == "help") {
            std::cout << "\nCommands:\n";
            std::cout << "  e <text>           - Encrypt text\n";
            std::cout << "  d [hex bytes]      - Decrypt last ciphertext or hex
bytes\n";
            std::cout << "  k <hex>            - Set key (64-bit hex, e.g., k
0123456789ABCDEF)\n";
            std::cout << "  l <0-3>            - Set LS pair (0, 1, 2, or 3)\n";
            std::cout << "  help               - Show this help\n";
            std::cout << "  q                  - Quit\n\n";
        }
        else if (cmd == 'k') {
            if (args.empty()) {
                std::cout << "Current key: 0x" << std::hex << key << std::dec <<
"\n";
            } else {
                unsigned long long new_key;
                if (sscanf(args.c_str(), "%llx", &new_key) == 1) {
                    key = new_key;
                    std::cout << "Key set to 0x" << std::hex << key << std::dec
<< "\n";
                } else {
                    std::cout << "Invalid key. Use 64-bit hex format (e.g.,
0123456789ABCDEF)\n";
                }
            }
        }
        else if (cmd == 'l') {
            if (args.empty()) {
                std::cout << "Current LS pair: " << (int)ls_sel << "\n";
            } else {
                unsigned int new_ls;
                if (sscanf(args.c_str(), "%u", &new_ls) == 1 && new_ls <= 3) {
```

```
                    ls_sel = (uint8_t)new_ls;
                    std::cout << "LS pair set to " << (int)ls_sel << "\n";
                } else {
                    std::cout << "Invalid LS pair. Use 0, 1, 2, or 3\n";
                }
            }
        }
        else if (cmd == 'e') {
            if (args.empty()) {
                std::cout << "Usage: e <text>\n";
            } else {
                std::cout << "\nEncrypting: \"" << args << "\"\n";
                std::cout << "Key: 0x" << std::hex << key << std::dec
                          << ", LS: " << (int)ls_sel << "\n";

                last_ciphertext = hil.encrypt(args, ls_sel, key);
                print_hex(last_ciphertext, "Ciphertext");
                std::cout << "\n";
            }
        }
        else if (cmd == 'd') {
            if (!args.empty()) {
                // Decrypt provided hex bytes
                auto ciphertext = parse_hex_string(args);
                if (ciphertext.empty()) {
                    std::cout << "Failed to parse hex bytes. Format: d ab 66 37
c0\n";
                } else {
                    std::cout << "\nDecrypting hex bytes...\n";
                    std::cout << "Key: 0x" << std::hex << key << std::dec
                              << ", LS: " << (int)ls_sel << "\n";
                    print_hex(ciphertext, "Input");

                    std::string decrypted = hil.decrypt(ciphertext, ls_sel,
key);
                    std::cout << "Decrypted: \"" << decrypted << "\"\n\n";
                }
            } else if (!last_ciphertext.empty()) {
                // Decrypt last ciphertext
                std::cout << "\nDecrypting last ciphertext...\n";
                std::cout << "Key: 0x" << std::hex << key << std::dec
                          << ", LS: " << (int)ls_sel << "\n";

                std::string decrypted = hil.decrypt(last_ciphertext, ls_sel,
key);
                std::cout << "Decrypted: \"" << decrypted << "\"\n\n";
            } else {
                std::cout << "No ciphertext to decrypt. Encrypt something first
or provide hex bytes.\n";
                std::cout << "Usage: d [hex bytes] (e.g., d ab 66 37 c0)\n";
            }
        }
        else {
            std::cout << "Unknown command. Type 'help' for commands.\n";
        }
    }
}

int main(int argc, char** argv) {
    Verilated::commandArgs(argc, argv);

    std::cout << "\n";
    std::cout << "╔═══════════════════════════════════════════════╗\n";
    std::cout << "║  Stream Cipher Hardware-in-the-Loop Simulator ║\n";
```

```
    std::cout << "╠  Verilator + UART Protocol                                ╣\n";
    std::cout << "╚═══════════════════════════════════════════════════════════╝\n";

    bool interactive = false;

    for (int i = 1; i < argc; i++) {
        if (strcmp(argv[i], "--interactive") == 0 || strcmp(argv[i], "-i") == 0)
{
            interactive = true;
        } else if (strcmp(argv[i], "--help") == 0 || strcmp(argv[i], "-h") == 0)
{
            std::cout << "\nUsage: " << argv[0] << " [options]\n";
            std::cout << "Options:\n";
            std::cout << "  -i, --interactive   Run in interactive mode\n";
            std::cout << "  -h, --help          Show this help\n\n";
            return 0;
        }
    }

    CipherHIL hil;

    if (interactive) {
        interactive_mode(hil);
    } else {
        int passed = 0, total = 0;

        if (test_hi(hil)) passed++;
        total++;

        if (test_hello(hil)) passed++;
        total++;

        if (test_long_message(hil)) passed++;
        total++;

        if (test_different_keys(hil)) passed++;
        total++;

        std::cout << "\n" << std::string(50, '=') << "\n";
        std::cout << "SUMMARY: " << passed << "/" << total << " tests passed\n";
        std::cout << std::string(50, '=') << "\n\n";

        hil.print_stats();
    }

    return 0;
}
```

ІЛЮСТРАТИВНА ЧАСТИНА

МЕТОД ТА ЗАСІБ ПОТОКОВОГО ШИФРУВАННЯ НА ОСНОВІ КВАЗІГРУП. ЧАСТИНА 2. ОПЕРАЦІЙНИЙ БЛОК.

Виконав: студент групи 1БС-24м
спеціальності 125 Кібербезпека та захист інформації
_____ Богдан ЗАГИРНЯК
16 грудня 2025 р.
Керівник: к.ф.-м.н., доцент каф. ЗІ
_____ Галина ШЕЛЕПАЛО
16 грудня 2025 р.

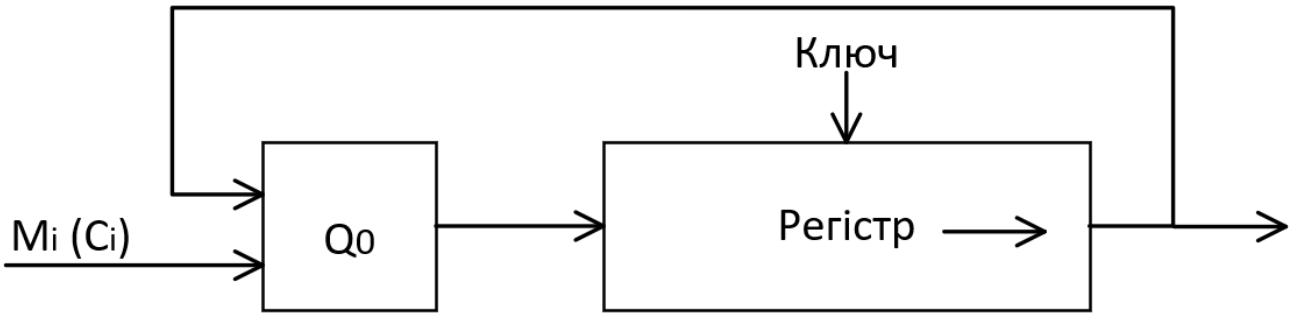
Порівняльні характеристики досліджених потокових шифрів

Алгоритм	Ключ (біт)	Апаратна складність (GE)
ASCON-128	128	2650 - 3700
Xoodyak	128	8100
SPARKLE	128	7400
PHOTON-Beetle	128	2380
ISAP	128	2750
GIFT-COFB	128	2400
Romulus-M	128	4200
Elephant	128	5300
TinyJambu-128	128	1960 - 2700
Grain-128AEAD	128	2400

Таблиця Келі 4-го порядку для генератора гами

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

Схема генератора гами



Таблиці Келі 4-го порядку для операції зашифрування 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

Таблиці Келі 4-го порядку для 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

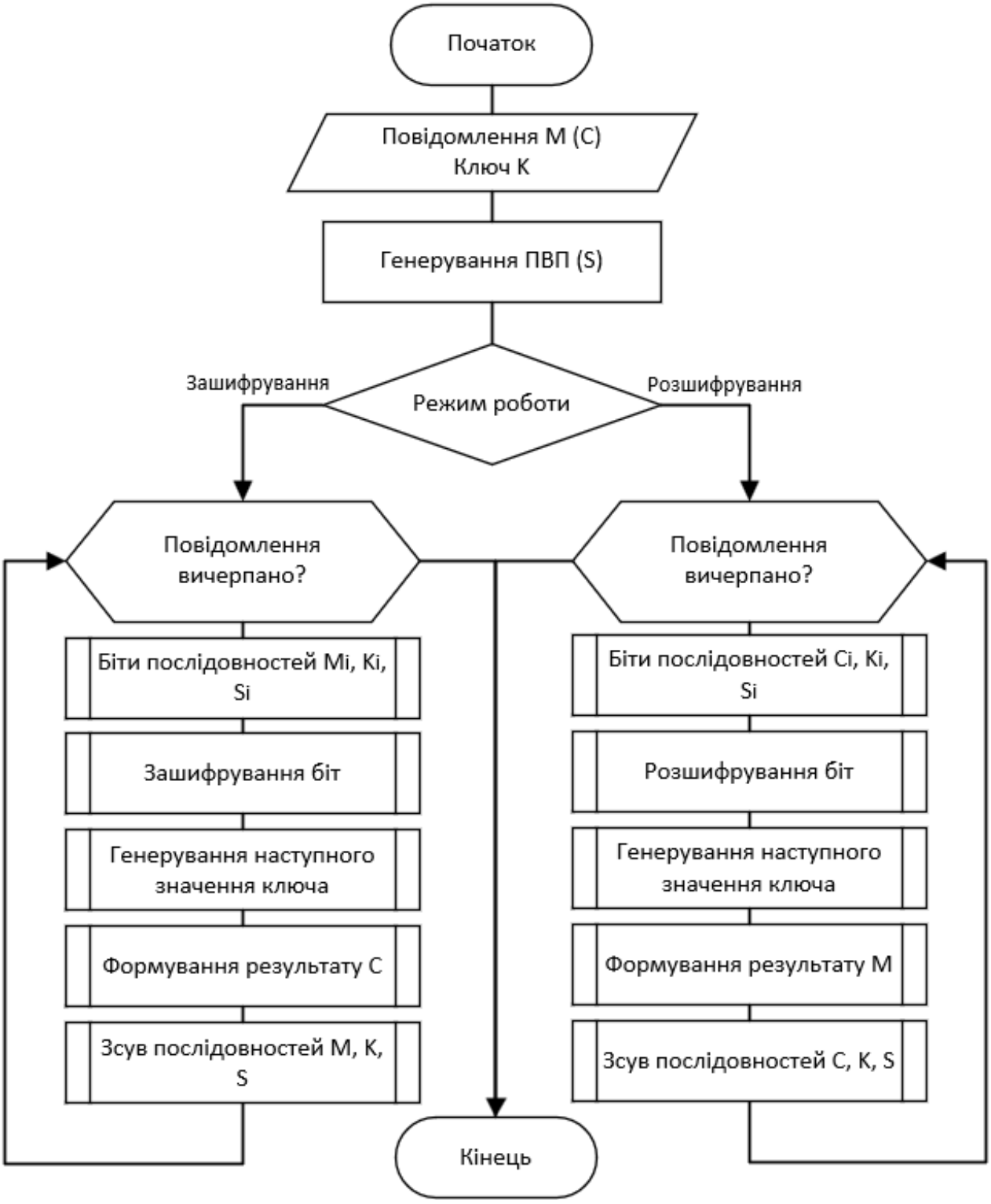
Q7:

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

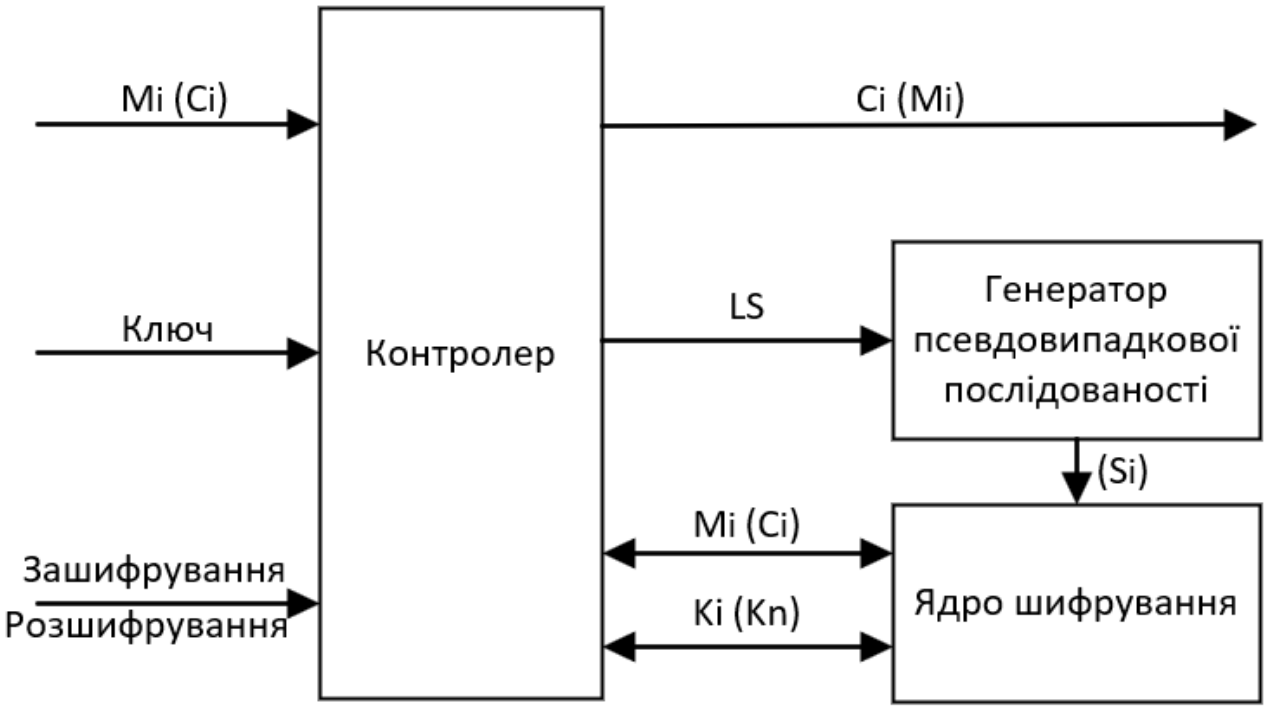
Q8:

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

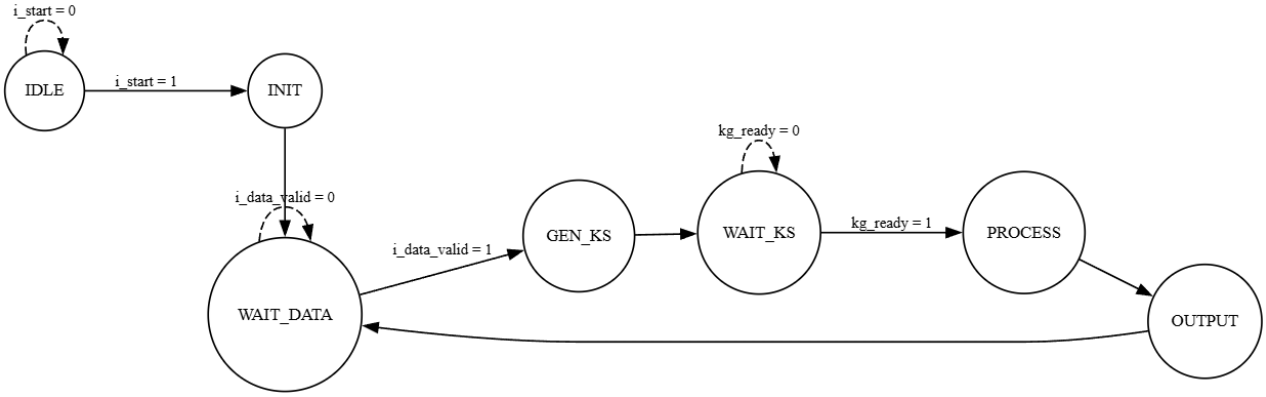
Схема алгоритму апаратного засобу



Загальна структурна схема операційного блоку



Діаграма станів керуючого автомату (FSM)



Загальна структурна схема апаратного інтерфейсу

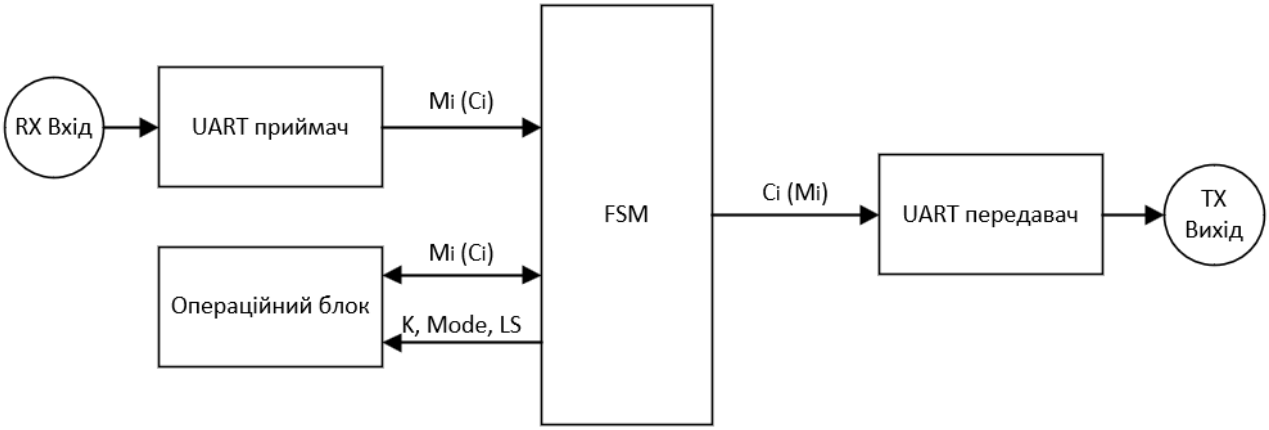


Схема роботи автомату FSM розробленого протоколу

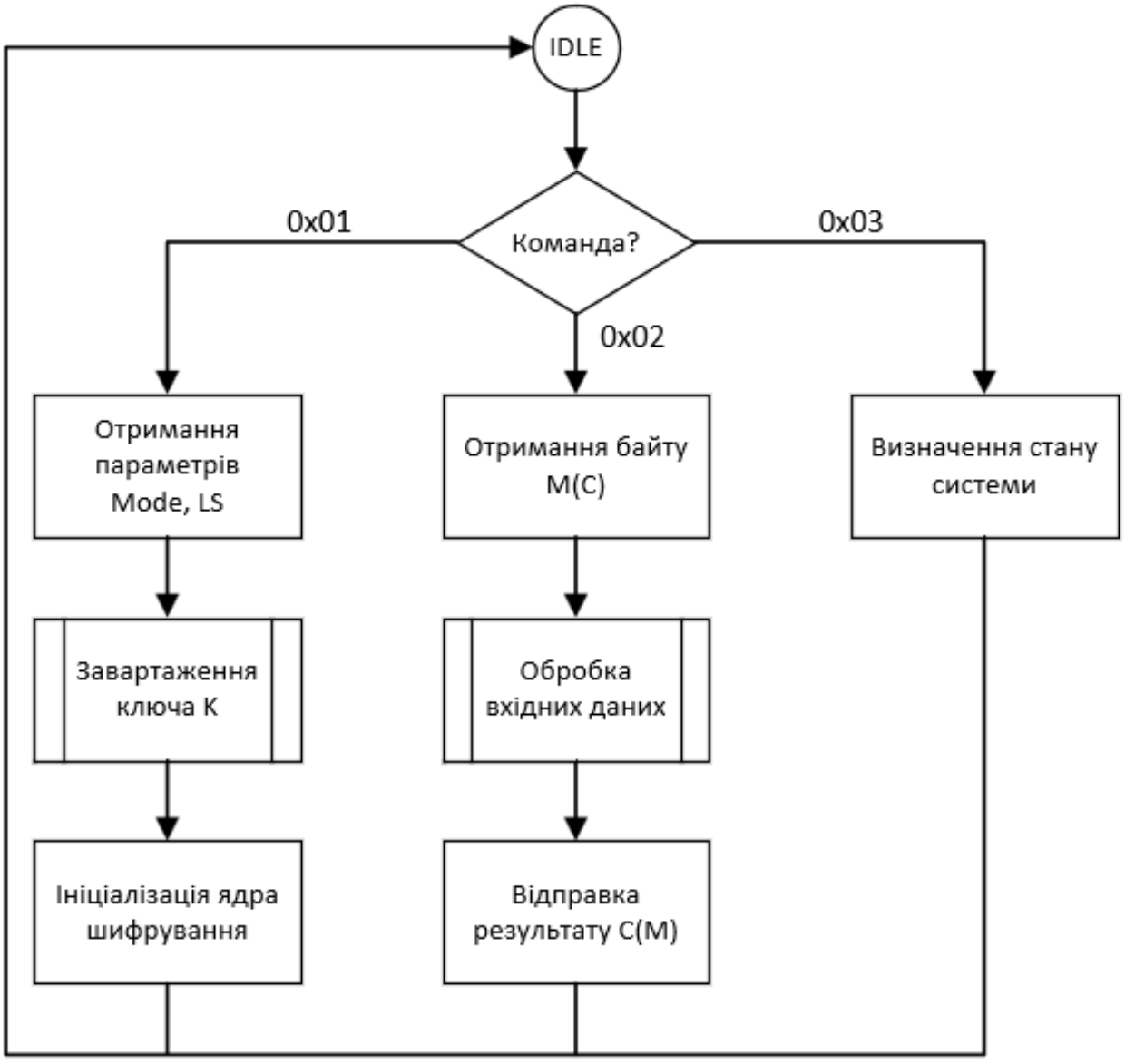


Схема ієрархічної структури рівнів абстракції даних

Схема взаємодії компонентів стенду

ТОВ «КАСКАД-БЕЗПЕКА»
вул. Київська 60, кв. 50, м. Вінниця, 21032
ЄДРПОУ 39256397

+38 (095) 388 74 99
support@kaskadb.com.ua

АКТ
ВПРОВАДЖЕННЯ

Цим актом підтверджується, що результати магістерської кваліфікаційної роботи тему «Метод та засіб потокового шифрування на основі квазігруп. Частина 2. ераційний блок.», яку виконано Загирняком Богданом Дмитровичем за еціальністю 125 «Кібербезпека та захист інформації» (Освітня програма «Безпека ормаційних і комунікаційних систем»), яка виконувалася протягом 24 вересня 2025 ку по 16 грудня 2025 року, впроваджені у апаратні технології на виробництві ТОВ АСКАД-БЕЗПЕКА»

Цей документ не є підставою для фінансових розрахунків.

Директор
ТОВ «Каскад-Безпека»

Товариство з обмеженою відповідальністю «КАСКАД-БЕЗПЕКА» 39256397 Україна, м.Вінниця

Артем КОВАЛЬ