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

Бакалаврська кваліфікаційна робота на тему:

«Засіб для блокового шифрування на основі перетворення Уолша»

Виконав: студент 4 курсу групи 1БС-21б
спеціальності 125 Кібербезпека

_____Євгеній КЕСАРЧУК

Керівник бакалаврської кваліфікаційної
роботи: завідувач кафедри ЗІ д.т.н., проф.

_____ Володимир ЛУЖЕЦЬКИЙ
«16» червня 2025 р.

Рецензент: завідувач кафедри ПЗ д.т.н., проф.

_____Олександр РОМАНЮК
«16» червня 2025 р.

Допущено до захисту

Завідувач кафедри ЗІ д. т. н., проф.
_____ Володимир ЛУЖЕЦЬКИЙ
«16» червня 2025р.

Вінниця ВНТУ – 2025 року

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

ЗАТВЕРДЖУЮ
Завідувач кафедри ЗІ ,д.т.н., проф.
Володимир ЛУЖЕЦЬКИЙ
«21» березня 2025 року

ЗАВДАННЯ
на бакалаврську кваліфікаційну роботу студенту
Євгенію Юрійовичу Кесарчуку

1. Тема роботи: «Засіб для блокового шифрування на основі перетворення Уолша»
   керівник роботи: Володимир ЛУЖЕЦЬКИЙ, д. т. н., проф. зав. кафедри ЗІ, затверджені наказом ректора ВНТУ від 20 березня 2025 року №97.
2. Строк подання студентом роботи 16 червня 2025 р.
3. Вихідні дані до роботи:
   - розрядність блоку даних – 128 біт ;
   - розрядність секретного ключа – 128 біт;
   - раундові перетворення – на основі перетворень Уолша;
   - розмірності перетворень Уолша – 2, 4, 8.
4. Зміст текстової частини: Вступ. 1. Аналіз підходів щодо побудови блокових шифрів. 2. Розробка методу блокового шифрування на основі перетворення Уолша. 3. Розробка програмного засобу. Висновки. Список використаних джерел.
5. Перелік ілюстративного матеріалу: 1. Підходи щодо побудови блокових шифрів (2 плакати). 2.Схема процесу зашифрування. 3. Метод зашифрування. 4. Схема процесу розшифрування . 5. Метод процесу розшифрування. 6. Метод розгортання ключа.7. Оцінка алгоритмічної складності шифрування блоку даних.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Лужецький В.А, д. т. н зав. каф. ЗІ	21.03.25	11.06.25
2	Лужецький В.А, д. т. н зав. каф. ЗІ	15.04.25	11.06.25
3	Лужецький В.А, д. т. н зав. каф. ЗІ	20.05.25	11.06.25

7. Дата видачі завдання 21 березня 2025 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз завдання. Вступ	21.03.25 – 25.03.25	
2	Аналіз інформаційних джерел за напрямком бакалаврської кваліфікаційної роботи	26.03.25 – 10.04.25	
3	Розробка моделей, алгоритмів	11.04.22 – 30.04.25	
4	Практична реалізація, експериментування, результати	01.05.25 – 20.05.25	
5	Висновки	20.05.25 – 23.05.25	
6	Оформлення пояснювальної записки	23.05.25 – 30.05.25	
7	Попередній захист БКР	30.05.25 – 31.05.25	
8	Виправлення зауважень, підготовка ілюстративного матеріалу	01.06.25 – 12.06.25	
9	Представлення БКР до захисту, рецензування	12.06.25 – 19.06.25	

Студент Євгеній КЕСАРЧУК

Керівник роботи Володимир ЛУЖЕЦЬКИЙ

АНОТАЦІЯ

Бакалаврська кваліфікаційна робота складається з 61 сторінок формату А4, на яких є 28 рисунків, список використаних джерел містить 23 найменування.

Бакалаврська кваліфікаційна робота присвячена розробці нового підходу до побудови симетричних блокових шифрів на основі швидкого перетворення Уолша різних розмірностей та його реалізація.

В результаті проаналізовано існуючі підходи до побудови блокових шифрів та описано власний підхід до блокового шифрування. Після розробки схем функціонування програмного засобу в цілому і алгоритмів його окремих складових здійснено програмну реалізацію.

Ключові слова: криптографія, шифрування, блоковий шифр, перетворення Уолша.

ABSTRACT

The bachelor's qualification work consists of 61 pages of A4 format, on which there are 28 drawings, the list of used sources contains 23 names.

The bachelor's qualification work is devoted to the development of a new approach to the construction of symmetric block ciphers based on the fast Walsh transform of different dimensions and its implementation.

As a result, existing approaches to the construction of block ciphers are analyzed and our own approach to block encryption is described. After developing the scheme of the functioning of the software violation as a whole and the algorithms of its individual components, the software implementation was carried out using the latest programming technologies.

Keywords: cryptography, encryption, block cipher, Walsh transform.

ЗМІСТ

ВСТУП

Сучасний розвиток інформаційних технологій супроводжується стрімким зростанням обсягу цифрової інформації, що передається та зберігається. Через це актуалізується потреба в надійних криптографічних механізмах захисту даних. Одним із фундаментальних інструментів такої безпеки є блокові шифри — алгоритми, які перетворюють фіксовані за довжиною блоки відкритого тексту у шифротекст за допомогою секретного ключа. Вони становлять ядро більшості протоколів шифрування, автентифікації та контролю цілісності у сучасних комп'ютерних мережах, мобільних системах і хмарних сервісах.

Попри десятиліття досліджень, конструювання стійких блокових шифрів залишається відкритою науковою та інженерною проблемою. Найпоширенішими підходами до реалізації симетричних блокових шифрів є мережі Фейстеля з двома та чотирма гілками, SP-мережі, структури «квадрат» та операцій за модулем. Кожен із цих методів пропонує власний баланс між стійкістю, швидкодією та апаратною/програмною реалізованістю. Водночас для глибокого аналізу криптографічної стійкості критично важливо оцінити нелінійність і кореляційну чутливість будівельних блоків шифру — передусім S-блоків. Одним із найпотужніших математичних інструментів такого аналізу є перетворення Уолша, яке дозволяє кількісно вимірювати невідхильність від лінійних та афінних апроксимацій, оцінювати ймовірність диференціалів та визначити потенційні вразливості до лінійного й диференційного криптографічного аналізу [1].

Метою бакалаврської кваліфікаційної роботи є пришвидшення процесу шифрування даних завдяки новому підходу до реалізації блокового шифру, який буде використовувати перетворення Уолша для шифрування.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- проаналізувати існуючі підходи щодо побудови блокових шифрів;
- розробити методи зашифрування і розшифрування даних;

- реалізувати програмний засіб для шифрування.

Наукова новизна роботи полягає у тому, що запропоновано використовувати як раундові перетворення блокового шифру перетворення Уолша, що забезпечує пришвидшення процесу шифрування даних.

Практична цінність роботи полягає у тому, що пропонується програмний засіб для пришвидшеного шифрування даних.

Структурно робота складається з трьох розділів, висновків і списку використаних джерел. У першому розділі наведено аналіз сучасних підходів щодо побудови блокових шифрів та сформульовані задачі бакалаврської кваліфікаційної роботи. Другий розділ присвячено розробці методів зашифрування та розшифрування, методу розгортання секретного ключа та оцінка алгоритмічної складності реалізації процесів зашифрування та розшифрування. Третій розділ присвячено розробці програмного засобу, що реалізує запропоновані методи шифрування.

Результати бакалаврської кваліфікаційної роботи доповідались на міжнародній науково-практичній конференції «Інформаційні технології та комп’ютерне моделювання» ІТКМ-2025, 20-22 травня 2025 року, м. Івано-Франківськ.

1 АНАЛІЗ ПІДХОДІВ ЩОДО ПОБУДОВИ БЛОКОВИХ ШИФРІВ

1.1 Основні поняття

У сучасному світі інформація є одним із найцінніших ресурсів, а її захист від несанкціонованого доступу та збереження цілісності стає одним з основних завдань для держав, підприємств та окремих користувачів. Одним із найбільш ефективних інструментів для забезпечення цієї безпеки є шифрування, яке дозволяє перетворювати відкритий текст у нерозбірливий для сторонніх осіб варіант, доступний лише тим, хто володіє відповідним ключем. Це забезпечує конфіденційність і цілісність даних, захищаючи їх від можливих атак та витоків інформації.

Процес шифрування базується на використанні математичних алгоритмів, які застосовують певний ключ для перетворення даних. Ключ є критично важливим елементом, оскільки він визначає безпеку шифрування та здатність розшифрувати дані. Ключі можуть бути статичними або динамічними, а їх захист є невід'ємною частиною криптографічних систем, оскільки злом ключа дозволяє отримати доступ до захищеної інформації.

Історія шифрування бере свій початок ще в античності, коли для захисту важливих повідомлень використовувалися прості методи, такі як шифр Цезаря або шифр Віженера, де літери або символи тексту замінювалися згідно з певними правилами. Проте, з розвитком технологій і зростанням складності атак, ці методи виявилися недостатньо надійними для забезпечення сучасної безпеки.

У двадцятому столітті криптографія зазнала значного розвитку, особливо під час Другої світової війни, коли стали широко відомими успішні спроби розшифрувати складні шифри, такі як Enigma. Це подія не тільки стимулювала розвиток нових криптографічних методів, але й поставила перед науковцями завдання створення більш стійких до атак алгоритмів, здатних витримати нові техніки криптографічного аналізу.

Сьогодні шифрування є основою для захисту даних у різних сферах: від персональних пристроїв до глобальних інформаційних систем. Важливість ефективних методів шифрування особливо відчутна в умовах зростаючої кіберзлочинності, де шахраї намагаються отримати доступ до фінансових і персональних даних користувачів, державних і корпоративних ресурсів. Несанкціонований доступ до цих даних може призвести до серйозних наслідків, таких як крадіжка особистої інформації, фінансових ресурсів, а також до порушення конфіденційності та приватності.

Шифрування стало ключовим інструментом для забезпечення інформаційної безпеки, особливо у зв’язку з розвитком хмарних технологій, інтернет речей та розширеного використання мобільних пристроїв. Безпека в інформаційних системах значною мірою залежить від того, наскільки ефективно застосовуються криптографічні методи для захисту від несанкціонованих атак.

Враховуючи важливість шифрування для збереження цілісності та конфіденційності даних, розробка нових криптографічних алгоритмів та вдосконалення існуючих методів залишаються одними з пріоритетних напрямків у галузі інформаційної безпеки. Безумовно, шифрування даних є важливою складовою частиною глобальної системи безпеки інформації, де приватність та захист даних стали основою для довіри та ефективної роботи в цифровому середовищі.

Блокові шифри — це різновид симетричних алгоритмів, де відкритий текст спершу розбивається на блоки фіксованої довжини в бітах. Кожен блок шифрується окремо за допомогою визначеного алгоритму, причому для кожного блоку застосовується свій ключ. Сьогодні саме блокові шифри лежать в основі більшості сучасних криптографічних рішень. Всі повідомлення, зашифровані дані та ключі подаються як послідовності бітів. Формально блоковий шифр описують як перетворення такого вигляду:

$$C = E_k(P)$$

де P – відкритий текст (Plaintext),

C – зашифрований текст (Ciphertext),

E – зашифрування (Encription),

k – ключ (key)

Зазвичай симетричні блокові шифри мають схожий алгоритм шифрування, який складається з вхідного забілювання, раундів шифрування та вихідного забілювання. Узагальнена схема процесу шифрування показана на рис.1.1.

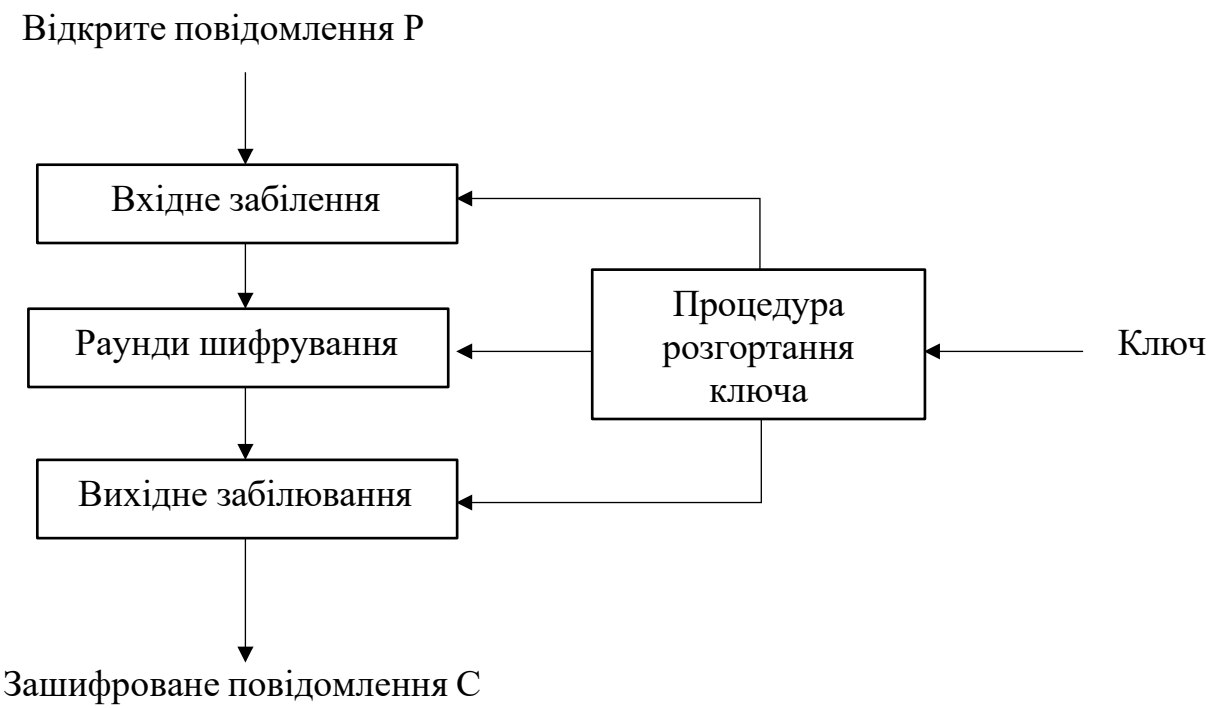


Рисунок 1.1 – Узагальнена схема процесу шифрування

1.2 Підходи до реалізації блокових шифрів

Блокові шифри діляться на різні групи. За підходом до реалізації існують чотири основні типи симетричних блокових шифрів:

- мережі Фейстеля
- SP-мережі
- Структура типу «квадрат»
- На основі операцій за модулем.

1.2.1 Мережі Фейстеля

Мережа Фейстеля є одним з основних будівельних блоків блокових шифрів та Мережа Фейстеля — це універсальна, раундова схема побудови блокових шифрів, що ділить дані на дві однакові частини й у кожному раунді змінює їх за ключем так, аби шифрування й розшифрування виконувалися тим самим алгоритмом у зворотному порядку ключів. У класичному варіанті з двома гілками [2] кожен раунд лише переганяє дані між лівою та правою половинами, поступово нарощуючи дифузію й плутанину, що дає стійку, але водночас просту для реалізації конструкцію [3]. Схема мережі Фейстеля з двома гілками показана на рис. 1.2.

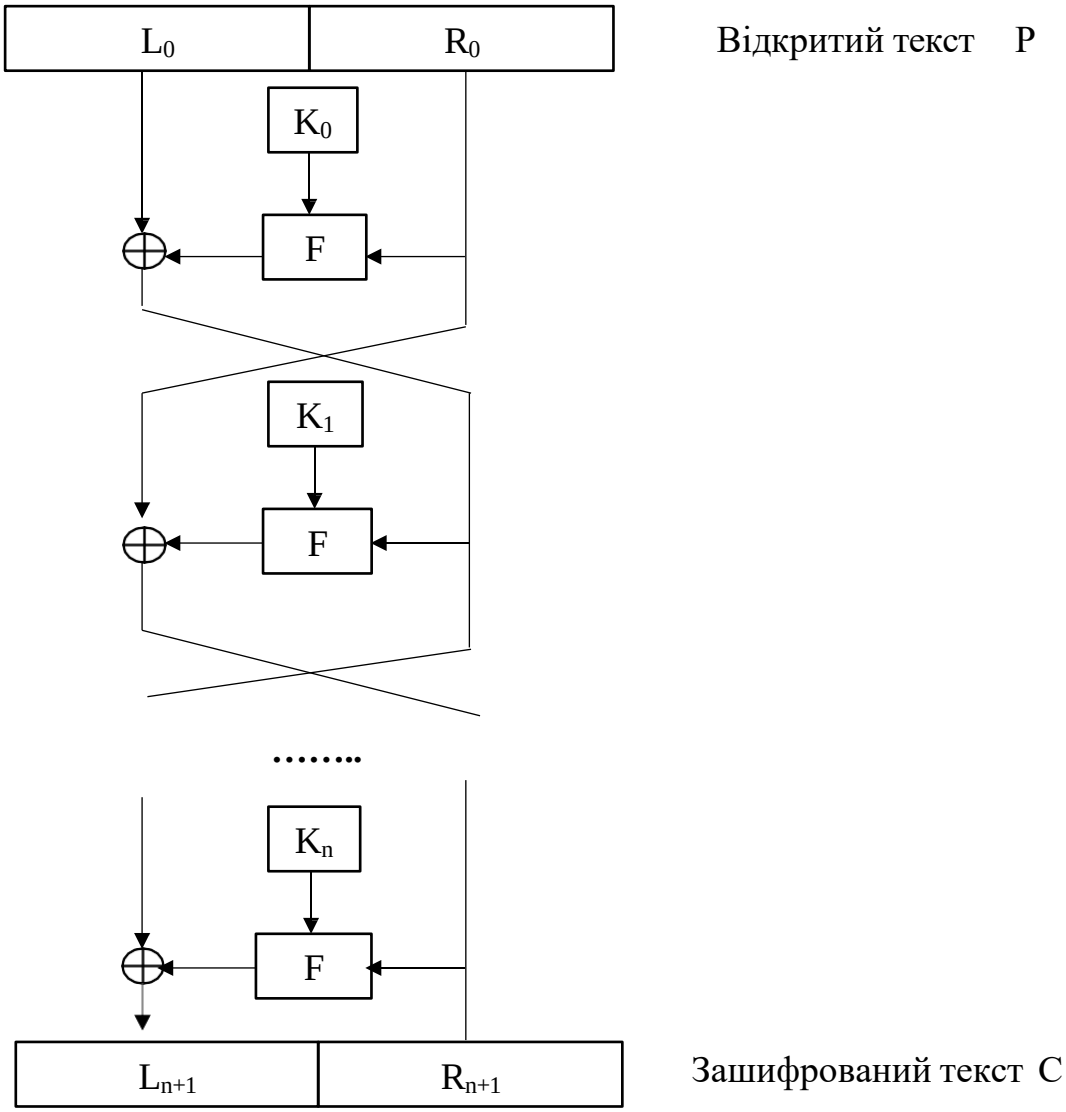


Рисунок 1.2 – Структура мережі Фейстеля з двома гілками

Фундаментальним будівельним блоком мережі Фейстеля є F-функція: залежне від ключа відображення вхідного рядка на вихідний рядок. Функція F завжди нелінійна та, можливо, несюр'єктивна :

$$F : \{0, 1\}^{n/2} \times \{0, 1\}^{N} \rightarrow \{0, 1\}^{n/2}$$

де n – розмір блоку мережі Фейстеля, а F – це функція, яка приймає $n/2$ бітів блоку та N бітів ключа як вхідні дані та створює вихідний результат довжиною $n/2$ біти. У кожному раунді «вихідний блок» є входом до F, а вихід F обробляється з «цільовим блоком», після чого ці два блоки міняються місцями для наступного раунду. Ідея полягає в тому, щоб взяти функцію F, яка сама по собі може бути слабким алгоритмом шифрування, та повторити її, щоб створити сильний алгоритм шифрування.

Два раунди мережі Фейстеля називаються «циклом». В одному циклі кожен біт текстового блоку був змінений один раз [4].

Мережу Фейстеля з двома гілками є основою великої кількості алгоритмів шифрування. Найбільш відомі з них : 3DES, Blowfish, Camellia, RC6, TEA та інші.

Одним з найперших шифрів побудованих за допомогою мережі Фейстеля був DES (Data Encryption Standard). Він був затверджений у 1976 році та був світовим стандартом протягом 25 років. Але він мав дуже короткий ключ – 56 біт, що робить його вразливим до атак методу brute force, через це був не одноразово зламаний [5].

На сьогоднішній день використовуються модернізований шифр 3DES (Triple Data Encryption Standard). Його ідея полягає у потрійному використанні алгоритму DES як це показано на рис. 1.3.

У даного алгоритму блокового шифрування є декілька режимів роботи DES – EEE3, схема роботи цього режиму показана на рис. 1.4, та DES – EDE3, схема цього режиму роботи продемонстрована на рис. 1.5 [6].

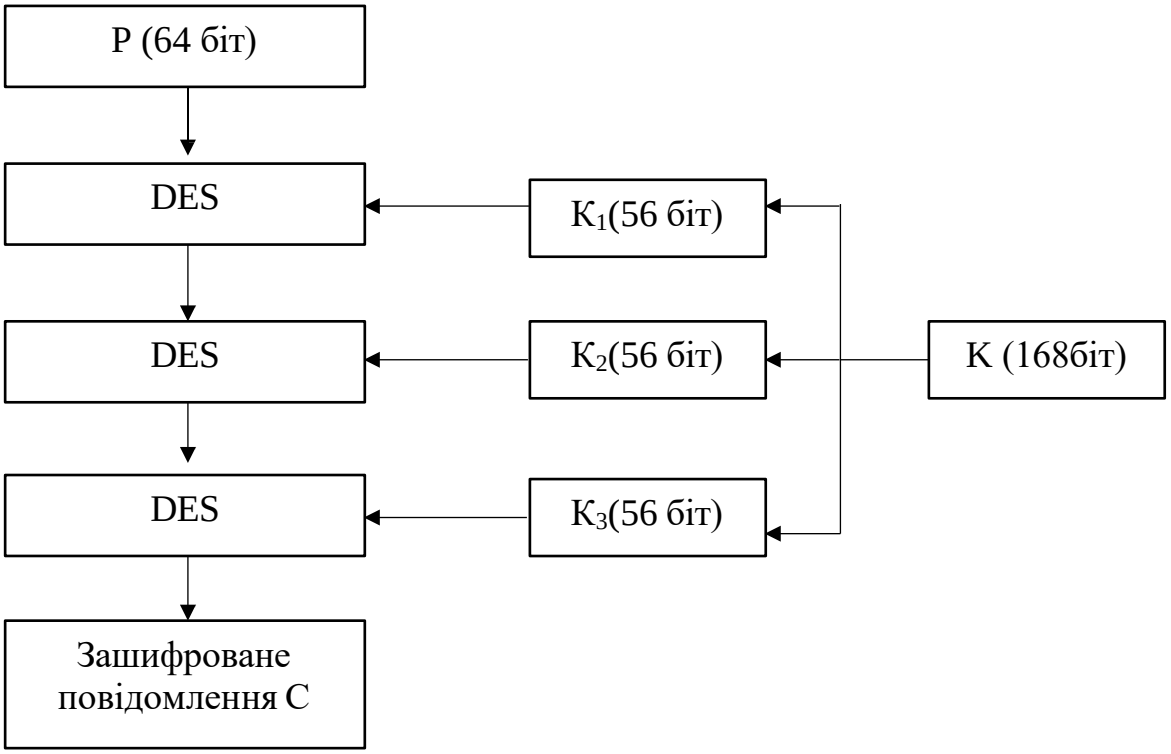


Рисунок 1.3 – Узагальнена схема алгоритму шифрування 3DES

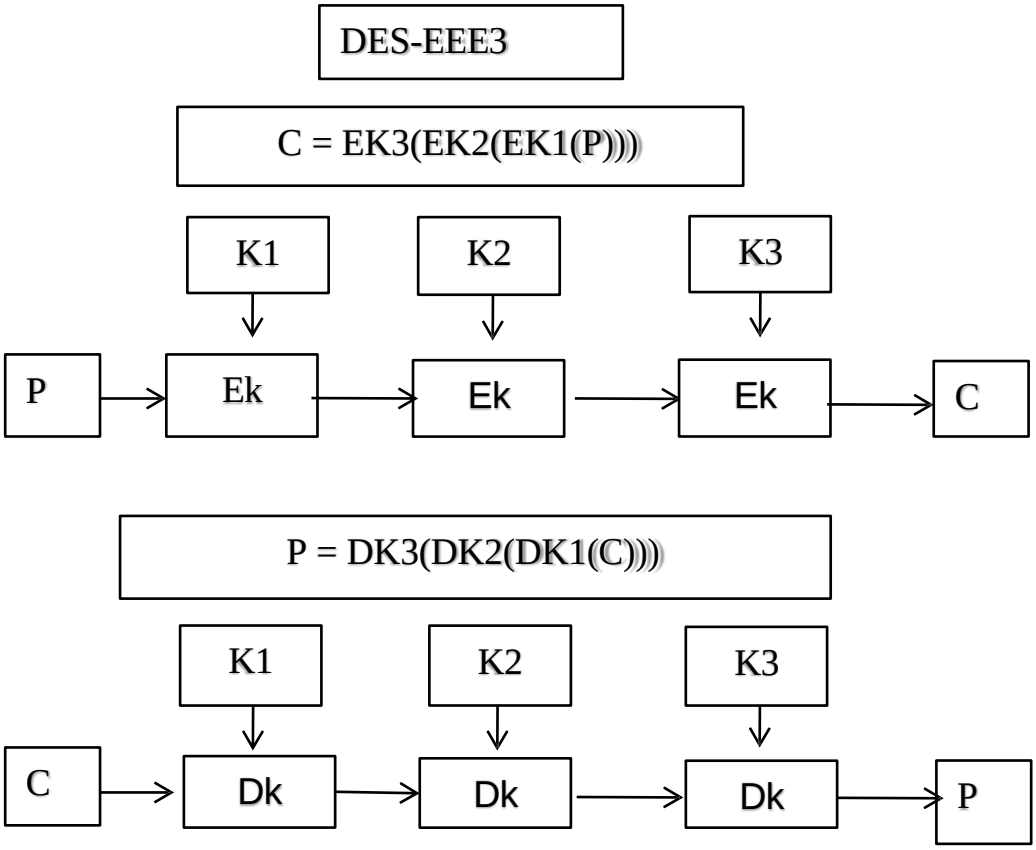


Рисунок 1.4 – Схема режиму роботи DES – EEE3

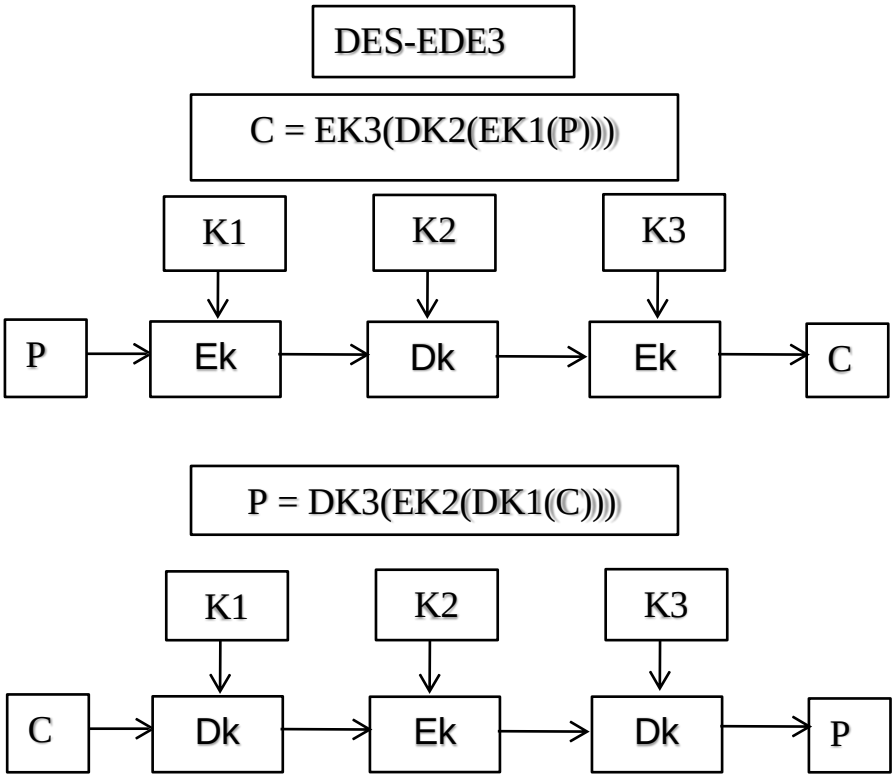
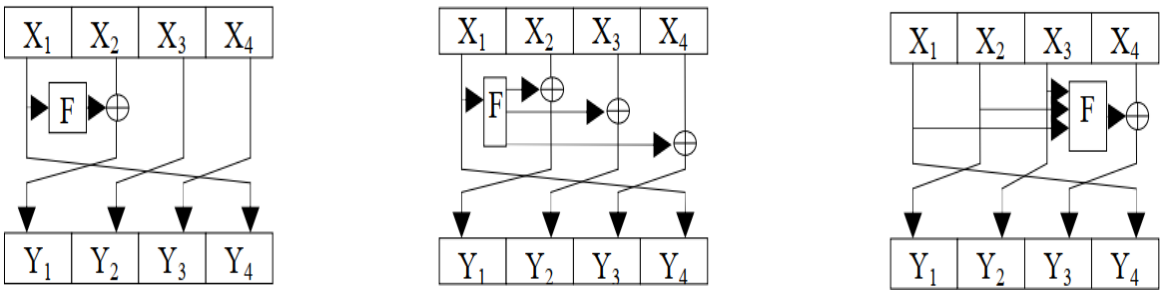


Рисунок 1.5 – Схема режиму роботи DES – EDE3

Тут описано найбезпечніший варіант коли для кожного DES використовується різний ключ K_1, K_2, K_3, ключ у такому варіанті 168 біт. Також є варіанти використання цього шифру з двома раундовими ключами, тоді $K_1=K_3\neq K_2$ і розрядність ключа 112 біт. Третій варіант коли раундові ключі однакові $K_1=K_2=K_3$. Але такий варіант не бажаний для використання.

Ще один різновид мереж Фейстеля це мережі Фейстеля з чотирма гілками. Такі мережі мають структури, що наведено на рис. 1.6 [7].



а) б) в)

Рисунок 1.6 – Типи мережі Фейстеля з чотирма гілками

Перший тип, показано на рис. 1.6.а, передбачає, що вхідні дані діляться на чотири рівні частини: B1, B2, B3 та B4. У кожному раунді функція F застосовується до блоку B1, а результат операції XOR додається до блоку B2. Після цього відбувається циклічне переміщення блоків, де B2 переходить на місце B1, B3 на місце B2, B4 на місце B3, а B1 стає на місце B4. Така схема забезпечує поступову дифузію даних між блоками протягом кожного раунду, що підвищує стійкість до криптографічного аналізу. Прикладом алгоритму, який використовує подібну структуру, є CAST-256 [8].

CAST-256 підтримує довжини ключів 128, 160, 192, 224 та 256 біт і оперує блоками даних розміром 128 біт. Він складається з 48 раундів, організованих у 12 "четвіркових раундів" (quad-rounds). У кожному раунді дані розбиваються на чотири 32-бітні підблоки: A, B, C і D. Функція F застосовується до одного з підблоків, а результат операції XOR додається до іншого підблоку. Після цього відбувається циклічне переміщення підблоків. Ця структура забезпечує поступову дифузію даних між підблоками протягом кожного раунду, що підвищує стійкість до криптографічного аналізу [8].

Другий тип, показано на рис. 1.6.б, має більш складну структуру обробки. Дані також діляться на чотири блоки, але в цьому випадку функції F1 та F2 застосовуються одночасно до двох різних блоків: F1 до B1 і F2 до B3. Результати обчислень функцій додаються через XOR до блоків B2 і B4 відповідно. Після завершення обчислень відбувається циклічний зсув, аналогічний до першого типу: B2 переходить на місце B1, B3 на місце B2, B4 на місце B3, а B1 стає на місце B4. Це ускладнення структури дозволяє досягти більшої дифузії за меншу кількість раундів. Відомим прикладом шифру, що використовує цю структуру, є RC6 [9].

RC6 працює з блоками даних розміром 128 біт і підтримує довжини ключів 128, 192 або 256 біт. Основою структури є узагальнена мережа Фейстеля з чотирма гілками другого типу, де блоки даних діляться на чотири частини: A, B, C, D.

Алгоритм складається з 20 раундів, де відбувається послідовне застосування наступних операцій: додавання за модулем 2^{32}, додавання за модулем 2(XOR), Побітове обертання.

Ключова особливість RC6 — це використання динамічного обертання на основі значень з інших частин блоку. Це дозволяє забезпечити сильну дифузію інформації навіть за невелику кількість раундів [9].

Третій тип, показано на рис. 1.6.в, є найбільш складним з усіх трьох. Тут використовуються три різні функції — F1, F2 та F3, які одночасно застосовуються до трьох блоків: B1, B2 і B3. Результати цих обчислень додаються через XOR до сусідніх блоків: результат F1 додається до B2, результат F2 — до B3, а результат F3 — до B4. Після цього також відбувається циклічне переміщення блоків за тим самим принципом, що і в попередніх типах. Така складна структура підвищує рівень заплутування даних, забезпечуючи високу стійкість до атак. Одним із прикладів алгоритмів, що використовують цю структуру, є TwoFish [4].

Twofish оперує з блоками даних розміром 128 біт та підтримує ключі довжиною 128, 192 або 256 біт. Основою його структури є узагальнена мережа Фейстеля з чотирма гілками третього типу. У кожному раунді дані діляться на чотири частини, і дві з них обробляються функцією F, яка включає в себе нелінійні перетворення (S-блоки), MDS-матриці для дифузії та операції XOR.

Відмінною особливістю Twofish є використання: залежних від ключів S-боксів, MDS-матриць, які забезпечують максимальну дифузію між байтами, та псевдо-перетворення Адамара — додаткове перетворення, яке розподіляє вплив змін одного байту на інші [4].

1.2.2 SP-мережа

Ще один підхід до побудови симетричних блокових шифрів на основі SP-мереж(мережа підстановки-перестановки) Шифр, розроблений на SP-мережі, складається з S-блоків та P-блоків. Схема такого шифру наведена на рис. 1.7.

S-блок (substitution box) замінює блок вхідних бітів іншим блоком вихідних бітів. Ця підстановка має бути один до одного, щоб забезпечити її

оборотність. Оскільки S-блок реалізує нелінійне перетворення, це дозволяє шифру витримувати лінійний криптографічний аналіз.

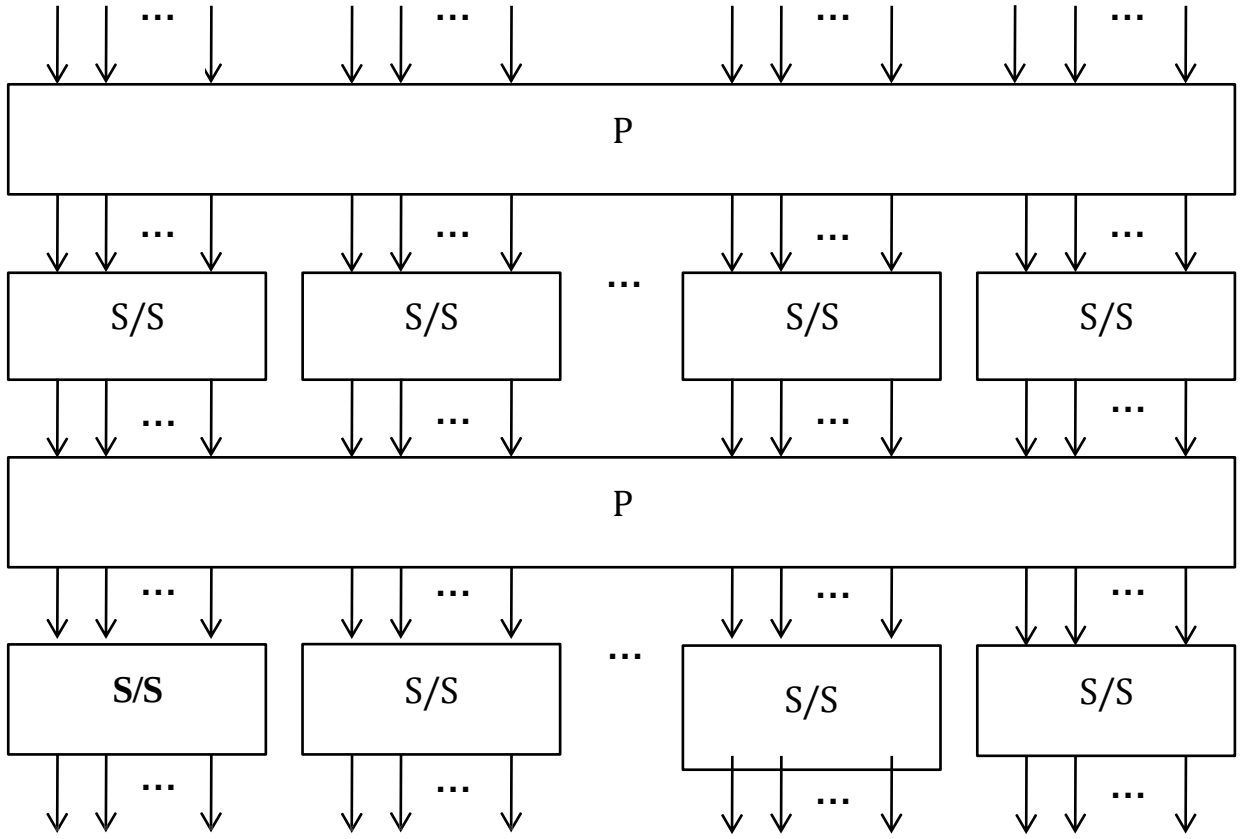


Рисунок 1.7 – Загальна схема блокових шифрів розроблених на SP-мережі

P-блок (permutation box) – це перестановка всіх бітів блоку [10].

Для того, щоб краще проаналізувати цей підхід до створення блокових шифрів розглянемо досить відомий шифр цього типу – Serpent [11].

Serpent — симетричний блочний шифр із блоком 128 біт і довжинами ключа 128, 192 або 256 біт. Він проходить крізь 32 раунди, що забезпечує значний запас стійкості: навіть половини цієї кількості було б досить, аби відбити стандартні лінійні й диференційні атаки, тому подвоєне число раундів залишає великий «захисний буфер». Стан шифру представлено у вигляді чотирьох 32-бітних слів; такий підхід дає змогу реалізувати всі раундові перетворення як паралельні логічні операції XOR, AND і NOT, що добре для сучасних процесорів та конвеєрів апаратних реалізацій.

У кожному раунді спершу додається підключ (проста операція XOR), далі застосовується один із восьми 4×4 S-боксів, причому їх циклічно чергують за номером раунду, після чого виконується лінійна трансформація: набір циклічних зсувів і XOR, який швидко «розмазує» зміну будь-якого біта по всьому блоку і створює сильний ефект лавини. У фінальному, тридцять другому, раунді лінійну частину опускають і одразу додають заключний підключ, що дозволяє робити дешифрування тим самим кодом, лише S-бокси проходять у зворотному порядку [11].

1.2.3 Структура типу «квадрат»

Ще один підхід до реалізації симетричних блокових шифрів це структура типу «Квадрат».

Структура "квадрат" – це підхід до реалізації симетричних блокових шифрів схожий на SP – мережу, тому, що він використовує операції перестановки та підстановки, але всі перетворення відбуваються в матрицях. На цьому підході працює всім відомий американський національний стандарт блокового шифрування AES . Але AES не перший в цьому сімействі шифрів. Класифікація шифрів з цією структурою показана на рис. 1.8.

Перший з цього сімейства був шифр Shark (Secure Hash Algorithm Regenerative Keys) [12] – симетричний алгоритм блокового шифрування. В теорії дозволяє використовувати блоки і ключі різної довжини, проте авторська реалізація використовує 128-бітний ключ і 64-бітові блоки [12].

Наступний був шифр SQUARE [13], він використовує ключ довжиною 128 біт, дані шифруються 128-бітними блоками, проте модульний підхід до побудови шифру дозволяє легко розширити до великих розмірів довжину ключа і довжину блоку даних. Дані і ключ представляються байтовим квадратом розміру 4x4. Кожен з 8 раундів SQUARE складається з чотирьох окремих перетворень. На його основі був створений всесвітньо відомий алгоритм симетричного блокового шифрування AES.

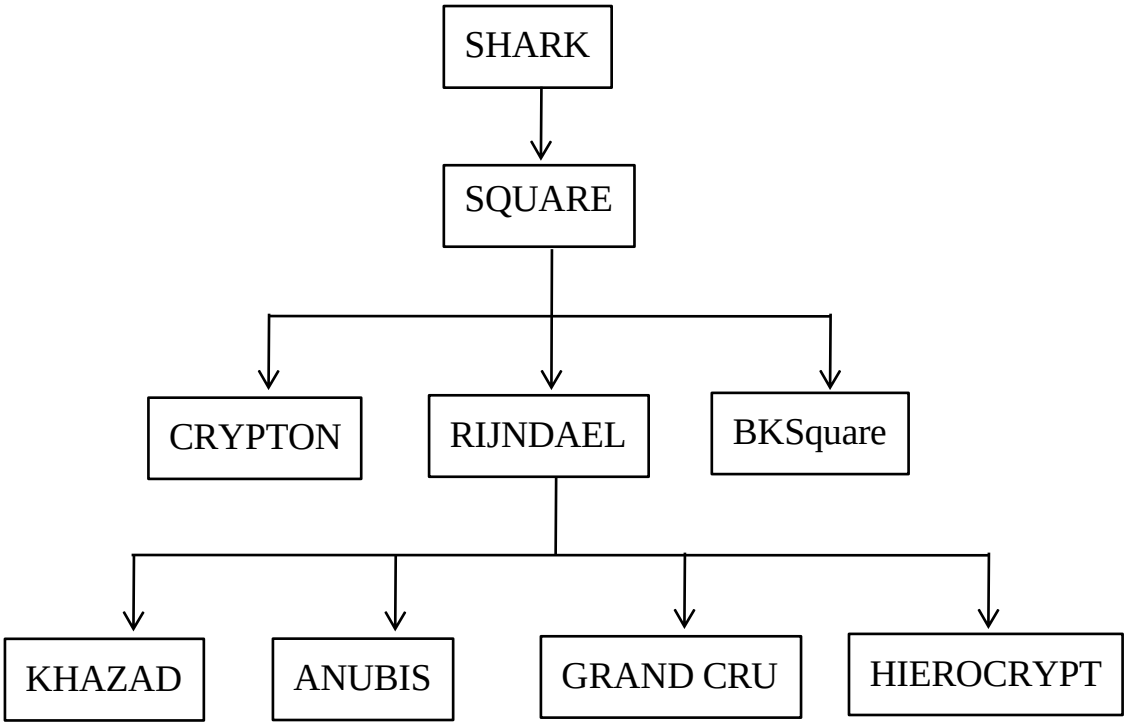


Рисунок 1.8 – Сімейство шифрів структури типу «квадрат»

AES(Advance Encryption Standard) – це симетричний блочний шифр, який був обраний стандартом США у 2001 році (FIPS PUB 197) для заміни старого DES. AES широко використовується в усьому світі для захисту даних завдяки високій безпеці та ефективності [14].

AES має різні версії такі як AES-128, AES-192 та AES-256. У всіх них розмір блоку даних рівний 128 бітам, але змінюється розмір ключа та кількість раундів перетворень. Довжина ключа підтримується 128, 192 та 256 біт, а кількість раундів 10, 12 та 14 відповідно. Схема алгоритму AES-128 показана на рис. 1.9.

Усі раунди, крім останнього, ідентичні. В останньому раунді операція перемішування стовбців відсутня, що робить усю послідовність операцій симетричною. Схема раундового перетворення показана на рис. 1.10.

З вище наведеного рисунку, AES-128 має 4 операції перетворення: SubBite, ShiftRow, MixColumn, AddRoundKey. Схеми перетворень кожної операції наведено в рис. 1.11 [14,15].

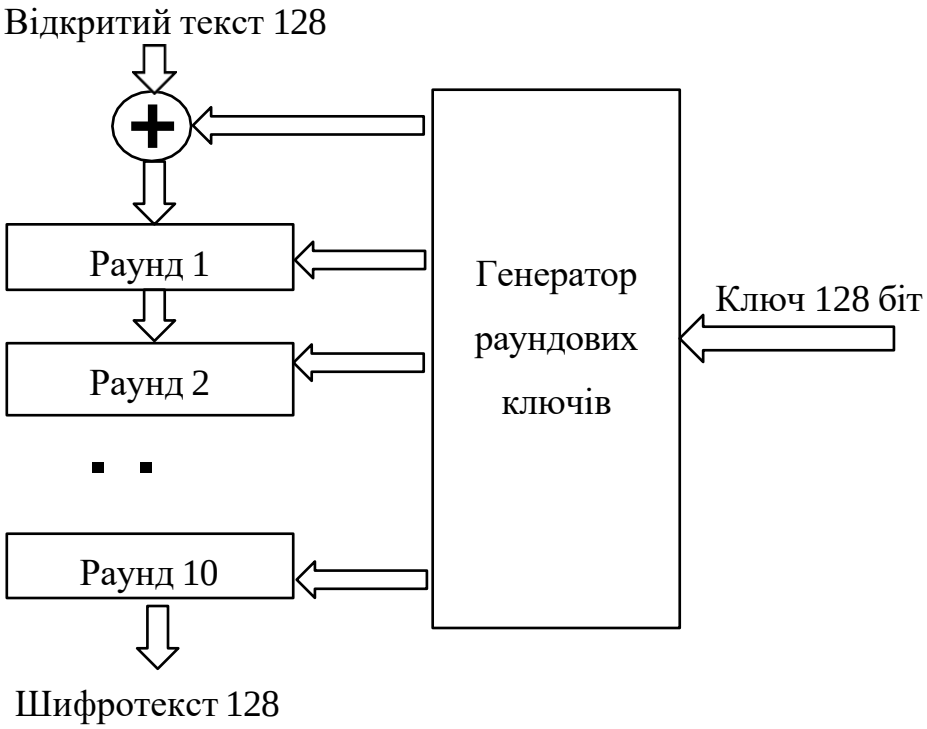


Рисунок 1.9 – Загальна схема перетворення блокового шифру AES-128

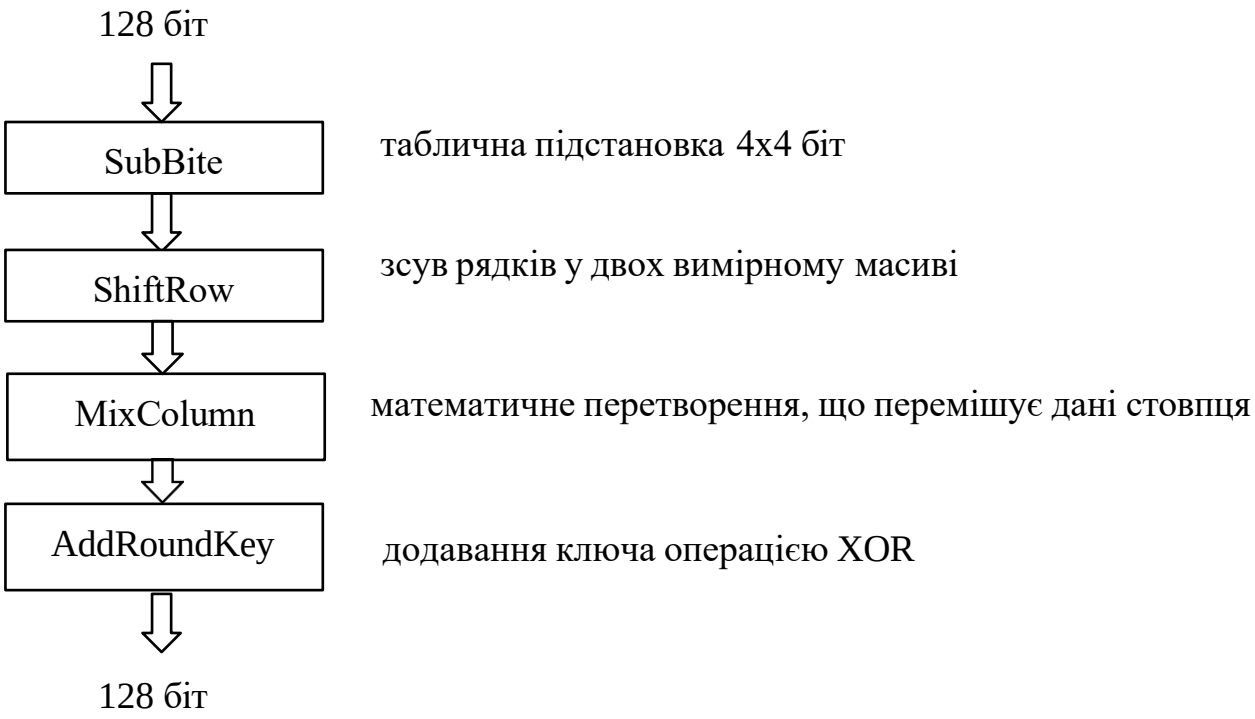
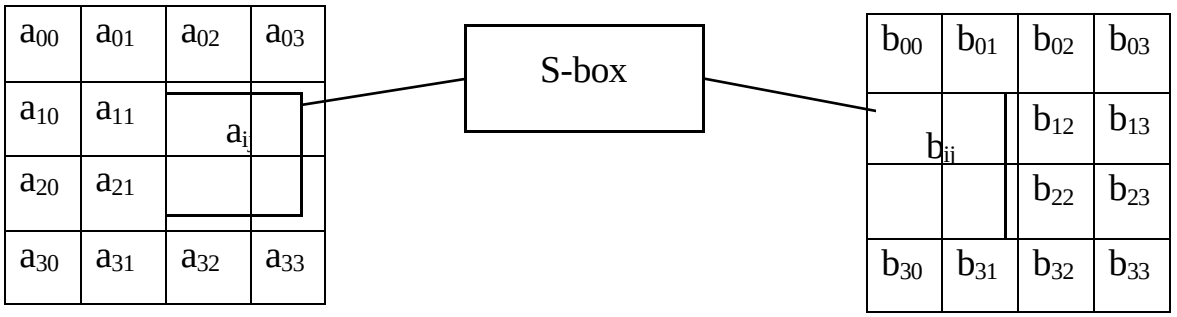
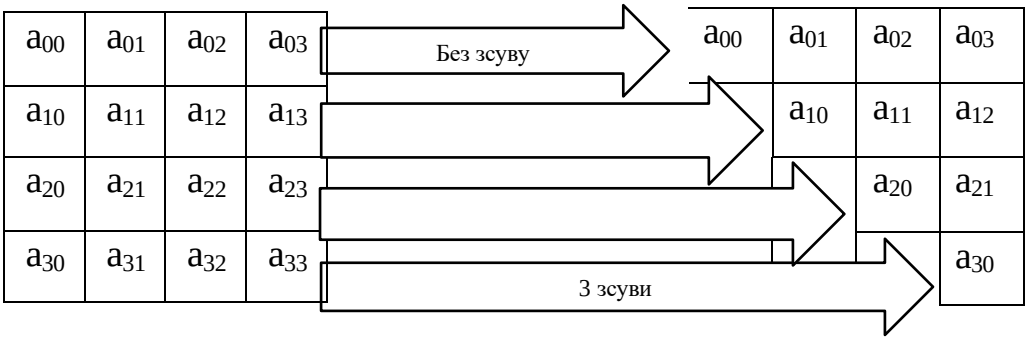


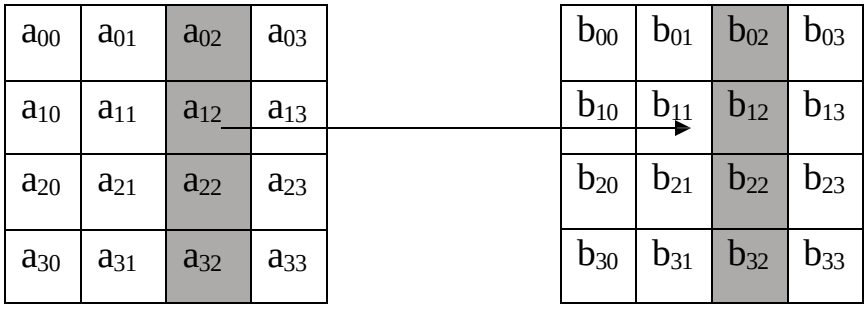
Рисунок 1.10 – Схема раундового перетворення AES-128



а)

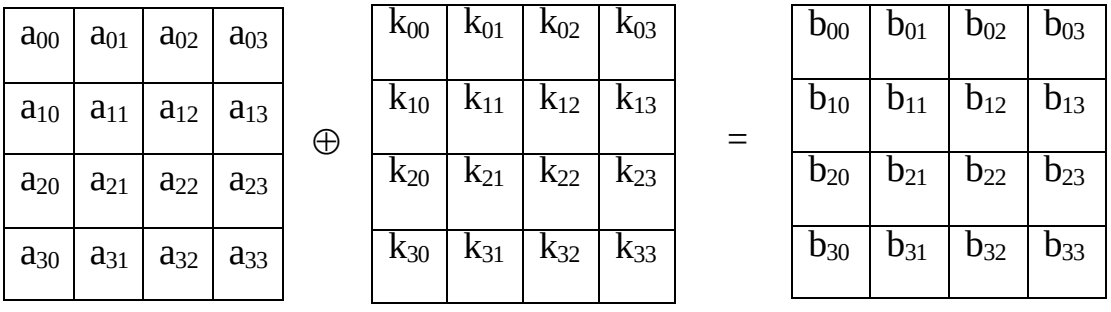


б)



в)

4



г)

Рисунок 1.11 – Схеми перетворень в AES: а) – SubByte, б) – ShiftRow, в)– MixColumn, г) – AddRoundKey

1.2.4 На основі операцій за модулем

Ще один підхід до реалізації блокових шифрів це використання операцій за модулем. Їх існує багато типів але для прикладу проаналізовано основні, такі як шифрування з відомим модулем, шифрування із секретним модулем та інші.

Шифрування з відомим модулем наприклад використання в якості модуля числа 2^n [16].

Нехай M – блок відкритого тексту, *C* – блок шифротексту і *K* – секретний ключ, при чому $M, C, K \in Z_2^n$. Записана процедуру зашифрування у випадку використання однієї операції множення за модулем:

$$C = (M \times K) \, mod \, 2^n$$

Криптографічна стійкість такого алгоритму базується на тому факті, що якщо *M* та *K* великі складені цілі числа, то відновити їх за відомим значенням *C* дуже складно і при великих значеннях *M* практично неможливо. Такий алгоритм може використовуватись в тих випадках, коли вимоги щодо криптографічної стійкості незначні, режим записної книжки не є необхідним, а вірогідність отримання зловмисником пар відкритих та зашифрованих текстів незначна. Якщо відомі одна або декілька пар шифротекстів і відповідних до них відкритих текстів, знаходження секретного ключа стає задачею тривіальною, оскільки в цьому випадку можна обчислити ключ розділивши число *C* на число *M* за модулем . Головна перевага запропонованого підходу – швидкість, з якою виконується шифрування [16].

Недоліком попереднього методу шифрування можна віднести те, що значення модуля на наступному раунді шифрування має бути більшим за попереднє, що зменшує криптографічну стійкість шифру. Окрім того, розрядність модулів повинна бути більшою за розрядність множеного, що ускладнює реалізацію метода та висуваються доволі складні вимоги до формування підключів шифрування, складовими яких є модуль та множник. Для вирішення проблеми надлишковості вихідного блоку даних та з метою усунення

залежностей між модулями пропонується використовувати таке множення за секретним значенням модуля [17]:

$$X \times A \bmod m = \begin{cases} X \cdot f(m') \bmod m' \\ (X - m' \cdot f(m'') \bmod m'' + m' \end{cases}$$

де *m′, m″* – n-бітні модулі, *m′ = 2ⁿ⁻²; 3*2ⁿ⁻², m″ = 2n – m′; f(m′), f(m″)* – функція, що визначає взаємно прості числа з m′ і m″ відповідно.

Оскільки значення модуля залишається невідомим, то це по-перше, дозволяє протидіяти криптографічному аналізу на основі мультиплікативних диференціалів і по-друге, використання двох послідовних множень за різними модулями дозволяє отримувати групи несумісних операцій.

1.3 Перетворення Уолша

Перетворення Уолша є фундаментальним інструментом у криптографії, має дуже багато застосувань : швидкі перетворення сигналів, підбір S-боксів, пошук bent-функцій [18] та інші [19]. Крім того, за допомогою перетворення Уолша аналізують статистичні відхилення потокових шифрів і шукають б'яс-функції, що детально показано в роботах Lu та Desmedt [20, 21]. Так як перетворення Уолша тісно пов'язане з матрицями Адамара.

Класична матриця Адамара виглядає так:

$$\boldsymbol{H}_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

За допомогою добутку Кронекера було побудовано матриці Адамара більших розмірностей.

$$H_{2^k} = \begin{bmatrix} H_{2^{k-1}} & H_{2^{k-1}} \\ H_{2^{k-1}} & H_{2^{k-1}} \end{bmatrix} = H_2 \otimes H_{2^{k-1}}.$$

Наприклад, щоб побудувати матрицю Адамара четвертого порядку H_4

необхідно матрицю H_2 помножити саму на себе. Виглядає це так :

$$\boldsymbol{H}_4 = \boldsymbol{H}_2 \otimes \boldsymbol{H}_2 = \begin{bmatrix} \boldsymbol{H}_2 & \boldsymbol{H}_2 \\ \boldsymbol{H}_2 & -\boldsymbol{H}_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}.$$

Натомість, для побудови матриці Адамара восьмого порядку H_8 необхідно помножити матрицю четвертого порядку H_4 на класичну матрицю Адамара другого порядку H_2. Побудова матриці Адамара восьмого порядку H_8 виглядає так [19] :

$$\boldsymbol{H}_8 = \boldsymbol{H}_4 \otimes \boldsymbol{H}_2 = \begin{bmatrix} \boldsymbol{H}_2 & \boldsymbol{H}_2 & \boldsymbol{H}_2 & \boldsymbol{H}_2 \\ \boldsymbol{H}_2 & -\boldsymbol{H}_2 & \boldsymbol{H}_2 & -\boldsymbol{H}_2 \\ \boldsymbol{H}_2 & \boldsymbol{H}_2 & -\boldsymbol{H}_2 & -\boldsymbol{H}_2 \\ \boldsymbol{H}_2 & -\boldsymbol{H}_2 & -\boldsymbol{H}_2 & \boldsymbol{H}_2 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix}$$

Перетворення Уолша базується на множені вектора на матрицю Адамара. Це перетворення у математичній формі виглядає так : $X = H_N \times x$. Наприклад, Множення на матрицю Адамара другого порядку H_2 , виглядає так :

$$\boldsymbol{H}_2 \times X = \begin{bmatrix} X_0 \\ X_1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \times \begin{bmatrix} x_0 \\ x_1 \end{bmatrix}.$$

Операція для обрахування цього виразу називається «метелик». Вона виглядає так :

$$\begin{matrix} A^* = A + B \\ B^* = A - B \end{matrix} ; \quad \text{з цього} \quad \begin{matrix} X_0 = x_0 + x_1 \\ X_1 = x_0 - x_1 \end{matrix};$$

Схематичне представлення операції для обрахування множення на

матрицю Адамара другого порядку H_2 показане на рис. 1.12.

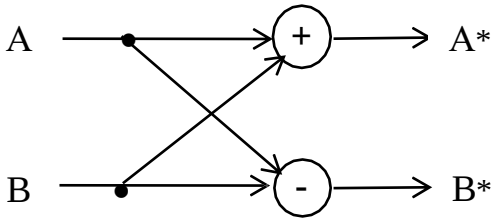


Рисунок 1.12 – Схематичне представлення множення вектора на матрицю Адамара

Для матриць більших розмірностей існує алгоритм швидкого перетворення Уолша. Наприклад для матриці Адамара четвертого порядку H_4 це перетворення можна виконати поступово виконавши дії, або за допомогою швидкого перетворення. Нижче показано множення на вектор та швидке перетворення Уолша.

$$\boldsymbol{H}_4 \times X = \begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \times \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}.$$

$$X_0 = x_0 + x_1 + x_2 + x_3;$$
$$X_1 = x_0 - x_1 + x_2 - x_3;$$
$$X_2 = x_0 + x_1 - x_2 - x_3;$$
$$X_3 = x_0 - x_1 - x_2 + x_3;$$

$$a_0 = x_0 + x_2;\ a_1 = x_1 + x_3;$$
$$a_2 = x_0 - x_2;\ a_3 = x_1 - x_3;$$
$$X_0 = a_0 + a_1; X_1 = a_2 + a_3;$$
$$X_3 = a_0 - a_1; X_4 = a_2 - a_3.$$

Зліва наведено приклад множення на вектор для обчислення якого необхідно виконати 12 дій додавання або віднімання. З права наведене швидке перетворення Уолша, для якого необхідно всього 8 дій. Схематично швидке перетворення Уолша показано на рис. 1.13 [19]. Це найпростіше швидке перетворення, яке зменшує обчислення на 4 дії.

Для матриць Адамара більших розмірностей це перетворення відбувається складніше, але скорочення обчислення при цьому суттєвіші. Так для матриці Адамара восьмого порядку це перетворення відбувається майже в двічі швидше. Оскільки для множення матриці на вектор це обчислення відбувається так:

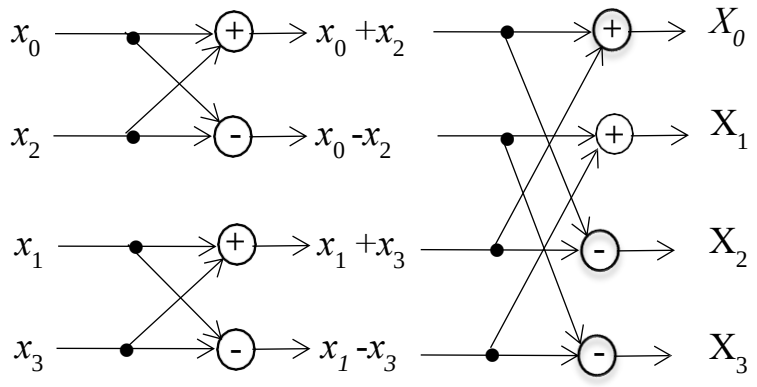


Рисунок 1.13 – Схематичне представлення швидкого перетворення Уолша для матриці четвертого порядку H_4

$$\boldsymbol{H}_4 \times X = \begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \\ [X_7] \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ [1 & -1 & -1 & 1 & -1 & 1 & 1 & -1] \end{bmatrix} \times \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ [x_7] \end{bmatrix}.$$

$$X_0 = x_0 + x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7;$$
$$X_1 = x_0 - x_1 + x_2 - x_3 + x_4 - x_5 + x_6 - x_7;$$
$$X_2 = x_0 + x_1 - x_2 - x_3 + x_4 + x_5 - x_6 - x_7;$$
$$X_3 = x_0 - x_1 - x_2 + x_3 + x_4 - x_5 - x_6 + x_7;$$
$$X_4 = x_0 + x_1 + x_2 + x_3 - x_4 - x_5 - x_6 - x_7;$$
$$X_5 = x_0 - x_1 + x_2 - x_3 - x_4 + x_5 - x_6 + x_7;$$
$$X_6 = x_0 + x_1 - x_2 - x_3 - x_4 - x_5 + x_6 + x_7;$$
$$X_7 = x_0 - x_1 - x_2 + x_3 - x_4 + x_5 + x_6 - x_7;$$

Для того щоб порахувати це перетворення необхідно виконати 56 операцій, тоді як алгоритм швидкого перетворення Уолша дає змогу виконати це за 24 дії. Нижче наведено алгоритм швидкого перетворення Уолша [19]:

Крок 1

$a_0 = x_0 + x_1;\quad a_1 = x_0 - x_1;$

Крок 2

$b_0 = a_0 + a_2; b_1 = a_1 + a_3;$

$a_0 = x_2 + x_3;\ a_3 = x_2 - x_3;$ $b_0 = a_0 + a_2; b_1 = a_1 + a_3;$

$a_4 = x_4 + x_5\ ; a_5 = x_4 - x_5;$ $b_4 = a_4 + a_6; b_5 = a_5 + a_7;$

$a_6 = x_6 + x_7; a_7 = x_6 - x_7;$ $b_6 = a_4 - a_6; b_7 = a_5 - a_7;$

Крок 3

$$X_0 = b_0 + b_4; X_1 = b_1 + b_5;$$
$$X_2 = b_2 + b_6; X_3 = b_3 + b_7;$$
$$X_4 = b_0 - b_4; X_5 = b_1 - b_5;$$
$$X_6 = b_2 - b_6; X_7 = b_3 - b_7;$$

Висновки до розділу 1

Наведений аналіз підходів до побудови блокових шифрів показує, що головна ідея полягає в тому, що багаторазово виконуються перетворення блоків даних з використанням раундових ключів, які формуються шляхом перетворень секретного ключа шифрування.

Швидкість шифрування залежить від складності раундових перетворень та їх кількості. Зменшення складності раундових перетворень, з одного боку, приводить до пришвидшення процесу шифрування, а з іншого боку, призводить до зменшення криптографічної стійкості. Тому для забезпечення потрібної стійкості змушені збільшувати кількість раундів. Тобто існує певна залежність між складністю реалізації раундів та їх кількістю.

Однак, існуючі методи шифрування мають свої обмеження та недоліки, які є наслідками нових викликів у сфері кібербезпеки. Через невпинне зростання обсягів даних, що передаються і зберігаються, надзвичайно актуальним стає питання про пришвидшення процесів шифрування зберігаючи при цьому достатньо високу криптографічну стійкість. Тому продовжуються пошуки нових підходів, що базуються на використанні властивостей певних математичних об’єктів. Зокрема такими об’єктами є функції Уолша та перетворення Уолша.

2 АЛГОРИТМ ШИФРУВАННЯ НА ОСНОВІ ПЕРЕТВОРЕНЬ УОЛША

2.1 Процес зашифрування блоку даних на основі швидкого перетворення Уолша

Метод блокового шифрування, що розробляється, використовує раундові перетворення, які базуються на прямому та оберненому перетвореннях Уолша [19]. Особливість перетворення Уолша полягає в тому, що використовуються лише операції додавання та віднімання. Крім того, існує алгоритм швидкого множення на матрицю Адамара-Уолша.

Схему процесу зашифрування наведено на рис. 2.1.

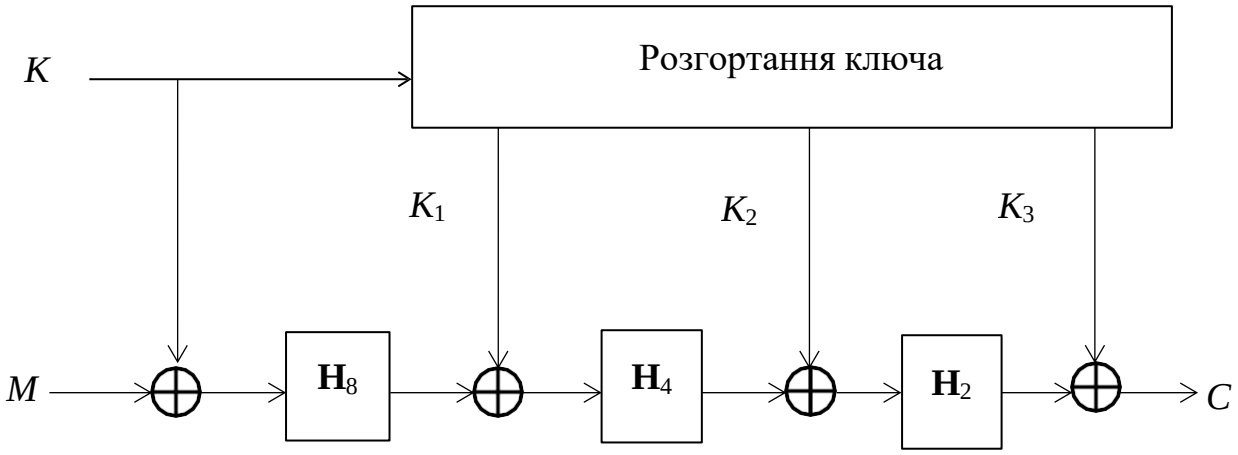


Рисунок 2.1 – Схема процесу зашифрування

Метод зашифрування будемо розглядати для блоку відкритого повідомлення *M* довжиною 128 біт з використанням початкового секретного ключа *K* довжиною 128 біт. Однак цей метод може бути застосований до блоків даних та секретних ключів інших розрядностей.

Цей метод передбачає виконання такої послідовності дій. Результат додавання за модулем 2 блоку відкритого повідомлення *M* і початкового секретного ключа *K* подається як вектор, що складається з 8 елементів по 16 біт кожен:

$$\boldsymbol{V}_1 = (v_{11}\ \ v_{12}\ \ v_{13}\ \ v_{14}\ \ v_{15}\ \ v_{16}\ \ v_{17}\ \ v_{18}).$$

Потім виконується множення цього вектору на матрицю Адамара-Уолша

$$\boldsymbol{H}_8 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix}$$

за алгоритмом швидкого перетворення. При цьому операції додавання і віднімання виконуються за модулем $2^{16}+1$. Результатом перетворення є вектор, що складається з елементів довжиною 17 бітів кожен.

$$\boldsymbol{C}_1 = (c_{11}\ \ c_{12}\ \ c_{13}\ \ c_{14}\ \ c_{15}\ \ c_{16}\ \ c_{17}\ \ c_{18}).$$

Цей вектор подається як 136-розрядний код

$$C_1 = (c_{11} || c_{12} || c_{13} || c_{14} || c_{15} || c_{16} || c_{17} || c_{18}).$$

Формується 136-розрядний раундовий ключ K_1 і виконується додавання за модулем 2 цього ключа і коду C_1 . Результат подається як вектор, що складається з 4 елементів по 34 біти кожен:

$$\boldsymbol{V}_2 = (v_{21}\ \ v_{22}\ \ v_{23}\ \ v_{24}).$$

Далі виконується множення цього вектору на матрицю Адамара-Уолша

$$\boldsymbol{H}_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

за алгоритмом швидкого перетворення. При цьому операції додавання і віднімання виконуються за модулем $2^{34}+1$. Результатом перетворення є вектор,

що складається з елементів довжиною 35 бітів кожен.

$$\boldsymbol{C}_2 = (c_{21}\ \ c_{22}\ \ c_{23}\ \ c_{24}).$$

Цей вектор подається як 140-розрядний код

$$C_2 = (c_{21}||\ c_{22}\ ||\ c_{23}\ ||\ c_{24}\ ).$$

Формується 140-розрядний раундовий ключ K_2 і виконується додавання за модулем 2 цього ключа і коду C_2 . Результат подається як вектор, що складається з 2 елементів по 70 бітів кожен:

$$\boldsymbol{V}_3 = (v_{31}\ \ v_{32}).$$

Далі виконується множення цього вектору на матрицю Адамара-Уолша

$$\boldsymbol{H}_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

При цьому операції додавання і віднімання виконуються за модулем $2^{70}+1$. Результатом перетворення є вектор, що складається з елементів довжиною 71 біт кожен.

$$\boldsymbol{C}_3 = (c_{31}\ \ c_{32}).$$

Цей вектор подається як 142-розрядний код

$$C_3 = (c_{31}||\ c_{32}).$$

Формується 142-розрядний раундовий ключ K_3 і виконується додавання за модулем 2 цього ключа і коду C_3 . Результатом цієї операції є код зашифрованого повідомлення C.

2.2 Процес розшифрування блоку даних на основі швидкого перетворення Уолша

Процес розшифрування полягає у виконанні у зворотному порядку операцій процесу зашифрування, що показано на рис. 2.2.

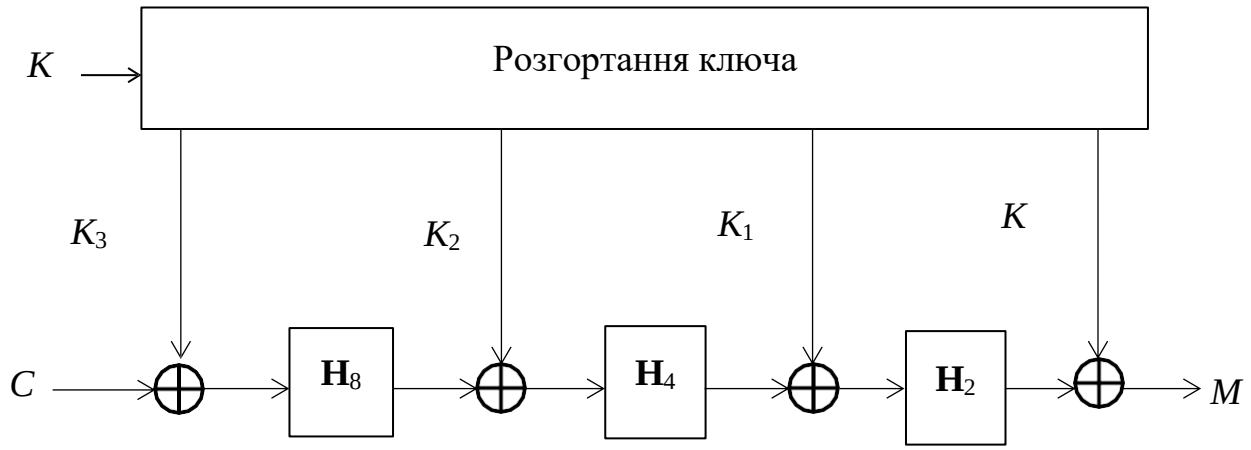


Рисунок 2.2 – Схема процесу розшифрування

Метод розшифрування передбачає виконання такої послідовності дій. Результат додавання за модулем 2 блоку зашифрованого повідомлення C і раундового ключа K_3 подається як вектор, що складається з 2 елементів по 71 біту кожен:

$$\boldsymbol{V}_3 = (v_{31} \;\; v_{32}).$$

Далі виконується обернене перетворення Уолша розмірності 2:

$$\boldsymbol{M}_3 = (-2^{70}) \cdot \boldsymbol{V}_3 \cdot \boldsymbol{H}_2.$$

Тут $(-2^{70}) = (\frac{1}{2}) \, mod(2^{71} + 1)$.

Результатом перетворення є вектор, що складається з 2 елементів довжиною 70 біт кожен.

$$\boldsymbol{M}_2 = (m_{21} \;\; m_{22}).$$

Цей вектор подається як 140-розрядний код

$$M_2 = (m_{21} || \, m_{22}).$$

Результат додавання за модулем 2 кодів M_2 і K_2 подається як вектор, що складається з 4 елементів по 35 бітів кожен:

$$\boldsymbol{V}_2 = (v_{21} \;\; v_{22} \;\; v_{23} \;\; v_{24}).$$

Виконується обернене перетворення Уолша розмірності 4:

$$\boldsymbol{M}_1 = (-2^{33}) \cdot \boldsymbol{V}_2 \cdot \boldsymbol{H}_4.$$

Тут $(-2^{33}) = \binom{1}{4} mod(2^{35} + 1)$.

Результатом перетворення є вектор, що складається з 4 елементів довжиною 34 біти кожен.

$$\boldsymbol{M}_1 = (m_{11} \;\; m_{12} \;\; m_{13} \;\; m_{14}).$$

Цей вектор подається як 136-розрядний код

$$M_1 = (m_{11} || \, m_{12} || m_{13} || \, m_{14}).$$

Результат додавання за модулем 2 кодів M_1 і K_1 подається як вектор, що складається з 8 елементів по 17 бітів кожен:

$$\boldsymbol{V}_1 = (v_{11} \;\; v_{12} \;\; v_{13} \;\; v_{14} \;\; v_{15} \;\; v_{16} \;\; v_{17} \;\; v_{18}).$$

Виконується обернене перетворення Уолша розмірності 8:

$$\boldsymbol{M}_0 = (-2^{14}) \cdot \boldsymbol{V}_1 \cdot \boldsymbol{H}_8.$$

Тут $(-2^{14}) = \binom{1}{-8} mod(2^{17} + 1)$.

Результатом перетворення є вектор, що складається з 8 елементів довжиною 16 біт кожен.

$$\boldsymbol{M}_0 = (m_{01}\ \ m_{02}\ \ m_{03}\ \ m_{04}\ \ m_{05}\ \ m_{06}\ \ m_{07}\ \ m_{08}).$$

Цей вектор подається як 128-розрядний код

$$M_0 = (m_{01}||\ m_{02}||m_{03}||\ m_{04}||m_{05}||\ m_{06}||m_{07}||\ m_{08}).$$

Результат додавання за модулем 2 кодів M_0 і K є кодом блоку відкритих даних M.

2.3 Процедура розгортання ключа

Процедура розгортання ключа передбачає виконання таких дій :

$$K_1 = K \oplus \overset{\leftarrow 8}{K};\quad K_2 = K_1 \oplus \overset{\leftarrow 4}{K_1};\ K_3 = K_2 \oplus \overset{\leftarrow 2}{K_2}.$$

Тут $\leftarrow n$ означає зсув коду на n розрядів уліво.

Розглянемо на прикладі процедуру розгортання ключа довжиною 16 біт.

Нехай K= 0101 1011 0110 1100.

$K_1 = K \oplus \overset{\leftarrow 8}{K}$, K= 0101 1011 0110 1100, $\overset{\leftarrow 8}{K}$ = 0101 1011 0110 1100 0000 0000

$$\begin{array}{rl} & 0000\ 0000\ 0101\ 1011\ 0110\ 1100 \\ \oplus & 0101\ 1011\ 0110\ 1100\ 0000\ 0000 \\ \hline & 0101\ 1011\ 0011\ 0111\ 0110\ 1100 \end{array}$$

K_1= 0101 1011 0011 0111 0110 1100

$K_2 = K_1 \oplus \overset{\leftarrow 4}{K_1}$, тоді $\overset{\leftarrow 4}{K_1}$ = 0101 1011 0011 0111 0110 1100 0000,

K_1=0101 1011 0010 0111 0110 1100

$$\begin{array}{rl} & 0101\ 1011\ 0011\ 0111\ 0110\ 1100\ 0000 \\ \oplus & 0000\ 0101\ 1011\ 0011\ 0111\ 0110\ 1100 \\ \hline & 0101\ 1110\ 1000\ 0100\ 0001\ 1010\ 1100 \end{array}$$

K_2= 0101 1110 1000 0100 0001 1010 1100

$K_3 = K_2 \oplus \overleftarrow{K_2}^{2}$, тоді K_2= 0101 1110 1000 0100 0001 1010 1100,
$\overleftarrow{K_2}^{2}$ = 0101 1110 1000 0100 0001 1010 1100 00

$$\begin{array}{rl} & 0101\ 1110\ 1000\ 0100\ 0001\ 1010\ 1100\ 00 \\ \oplus & 0001\ 0111\ 1010\ 0001\ 0000\ 0110\ 1011\ 00 \\ \hline & 0100\ 1001\ 0010\ 0101\ 0001\ 1100\ 0111\ 00 \end{array}$$

K_3 = 0100 1001 0010 0101 0001 1100 0111 00

Розгортання ключа завершене, отримано 3 раундових ключі та основний ключ K= 0101 1011 0110 1100, K_1= 0101 1011 0011 0111 0110 1100, K_2= 0101 1110 1000 0100 0001 1010 1100, K_3 = 0100 1001 0010 0101 0001 1100 0111 00.

2.4 Приклад зашифрування

Для того щоб продемонструвати алгоритм роботи блокового шифру на основі швидких перетворень Уолша наведено приклад зашифрування і розшифрування блоку даних розміром 16 біт.

Для цього взято довільний 16 бітний блок даних M = 1011 0100 1010 0110. Відповідно до схеми процесу зашифрування (рис.2.1), першим етапом виконується вхідне забілювання. Для цього на повідомлення M накладається ключ K операцією сума за модулем 2: $M_1=M \oplus K$, тоді :

$$\begin{array}{rl} & 1011\ 0100\ 1010\ 0110 \\ \oplus & 0101\ 1011\ 0110\ 1100 \\ \hline & 1110\ 1111\ 1100\ 1010 \end{array}$$

Отримано забілене повідомлення M_1 = 1110 1111 1100 1010, яке розділяється на 8 частин

$$v_{11} = (11)_2 = (3)_{10};\ v_{12} = (10)_2 = (2)_{10};\ v_{13} = (11)_2 = (3)_{10};$$
$$v_{14} = (11)_2 = (3)_{10};\ v_{15} = (11)_2 = (3)_{10};\ v_{16} = (00)_2 = (0)_{10};$$

$$v_{17} = (10)_2 = (2)_{10};\ v_{18} = (10)_2 = (2)_{10}$$

та подається як вектор V_1 = (3, 2, 3, 3, 3, 0, 2, 2).

Потім виконується множення цього вектору на матрицю Адамара-Уолша

$$\boldsymbol{H}_8 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix}$$

Раніше було згадано, що модуль за яким проводиться обрахунок дорівнює $2^n + 1$.Виходячи з цього, подальше обчислення будуть проводитись за модулем $2^2 + 1 = 5$

Крок 1

$$a_1 = v_{11} + v_{12} = 3 + 2 = 5(mod\ 5) = 0;$$
$$a_2 = v_{11} - v_{12} = 3 - 2 = 1(mod\ 5) = 1;$$
$$a_3 = v_{13} + v_{14} = 3 + 3 = 6(mod\ 5) = 1;$$
$$a_4 = v_{13} - v_{14} = 3 - 3 = 0(mod\ 5) = 0;$$
$$a_5 = v_{15} + v_{16} = 3 + 0 = 3(mod\ 5) = 3;$$
$$a_6 = v_{15} - v_{16} = 3 - 0 = 3(mod\ 5) = 3;$$
$$a_7 = v_{17} + v_{18} = 2 + 2 = 4(mod\ 5) = 4;$$
$$a_8 = v_{17} - v_{18} = 2 - 2 = 0(mod\ 5) = 0.$$

Крок 2

$$b_1 = a_1 + a_3 = 0 + 1 = 1(mod\ 5) = 1;$$
$$b_2 = a_2 + a_4 = 1 + 0 = 1(mod\ 5) = 1;$$
$$b_3 = a_1 - a_3 = 0 - 1 = 4(mod\ 5) = 4;$$
$$b_4 = a_2 - a_4 = 1 - 0 = 1(mod\ 5) = 1;$$
$$b_5 = a_5 + a_7 = 3 + 4 = 7(mod\ 5) = 2;$$
$$b_6 = a_6 + a_8 = 3 + 0 = 3(mod\ 5) = 3;$$

$$b_7 = a_5 - a_7 = 3 - 4 = -1(mod\ 5) = 4;$$
$$b_8 = a_6 - a_8 = 3 - 0 = 3(mod\ 5) = 3.$$

Крок 3

$$c_1 = b_1 + b_5 = 1 + 2 = 3(mod\ 5) = 3;$$
$$c_2 = b_2 + b_6 = 1 + 3 = 4(mod\ 5) = 4;$$
$$c_3 = b_3 + b_7 = 4 + 4 = 8(mod\ 5) = 3;$$
$$c_4 = b_4 + b_8 = 1 + 3 = 4(mod\ 5) = 4;$$
$$c_5 = b_1 - b_5 = 1 - 2 = -1(mod\ 5) = 4;$$
$$c_6 = b_2 - b_6 = 1 - 3 = -2(mod\ 5) = 3;$$
$$c_7 = b_3 - b_7 = 4 - 4 = 0(mod\ 5) = 0;$$
$$c_8 = b_4 - b_8 = 1 - 3 = -2(mod\ 5) = 3.$$

У ході обчислень кожне значення було збільшене на один розряд. Отримано значення вектора $C_1 = (3, 4, 3, 4, 4, 3, 0, 3)$.

Цей вектор подається як 24– бітний код

$$C_1 = c_{11}||c_{12}||c_{13}||c_{14}||c_{15}||c_{16}||c_{17}||c_{18} = 0111\ 0001\ 1100\ 1000\ 1100\ 0011.$$

Після першого раунду на блок даних C_1 накладається раундовий ключ K_1, операцією сума за модулем 2.

C_1= 0111 0001 1100 1000 1100 0011, K_1= 0101 1011 0011 0111 0110 1100

$$\begin{array}{rl} & 0111\ 0001\ 1100\ 1000\ 1100\ 0011 \\ \oplus & 0101\ 1011\ 0011\ 0111\ 0110\ 1100 \\ \hline & 0010\ 1010\ 1111\ 1111\ 1010\ 1111 \end{array}$$

Результат накладання раундового ключа подається як вектор, що складається з 4 елементів по 6 бітів кожен:

$$v_{21} = (001010)_2 = (10)_{10};\ v_{22} = (101111)_2 = (47)_{10};$$
$$v_{23} = (111110)_2 = (62)_{10};\ v_{24} = (101111)_2 = (47)_{10}$$

та подається як вектор V_2=(10, 47, 62 47).

Далі виконується множення цього вектору на матрицю Адамара-Уолша

$$\boldsymbol{H}_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

за алгоритмом швидкого перетворення. При цьому операції додавання і віднімання виконуються за модулем $2^6+1 = 65$.

Крок 1

$$a_1 = v_{21} + v_{22} = 10 + 47 = 57(mod\ 65) = 57;$$
$$a_2 = v_{21} - v_{22} = 10 - 47 = -37(mod\ 65) = 28;$$
$$a_3 = v_{23} + v_{24} = 62 + 47 = 109(mod\ 65) = 44;$$
$$a_4 = v_{23} - v_{24} = 62 - 47 = 15(mod\ 65) = 15.$$

Крок 2

$$c_1 = a_1 + a_3 = 57 + 44 = 101(mod\ 65) = 36;$$
$$c_2 = a_2 + a_4 = 28 + 15 = 43(mod\ 65) = 43;$$
$$c_3 = a_1 - a_3 = 57 - 44 = 13(mod\ 65) = 13;$$
$$c_4 = a_2 - a_4 = 28 - 15 = 13(mod\ 65) = 13.$$

У ході обчислень кожне значення було збільшене на один розряд. Отримано значення вектора $C_2 = (36, 43, 13, 13)$.

Цей вектор подається як 28– бітний код

$C_2 = c_{21}||c_{22}||c_{23}||c_{24}$=0100 1000 1010 1100 0110 1000 1101.

Після другого раунду на блок даних C_2 накладається раундовий ключ K_2, операцією сума за модулем 2.

C_2=0100 1000 1010 1100 0110 1000 1101;

K_2=0101 1110 1000 0100 0001 1010 1100;

$$\begin{array}{r} 0100\ 1000\ 1010\ 1100\ 0110\ 1000\ 1101 \\ \oplus \quad 0101\ 1110\ 1000\ 0100\ 0001\ 1010\ 1100 \\ \hline 0001\ 0110\ 0010\ 1000\ 0111\ 1010\ 0001 \end{array}$$

Результат накладання раундового ключа подається як вектор, що складається з 2 елементів по 14 бітів кожен:

$v_{31} = (0001\ 0110\ 0010\ 10)_2 = (1418)_{10}$; $v_{32} = (00\ 0111\ 1010\ 0001)_2 = (1953)_{10}$

та подається як вектор V_3=(1418, 1953).

Далі виконується множення цього вектору на матрицю Адамара-Уолша

$$\boldsymbol{H}_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

При цьому операції додавання і віднімання виконуються за модулем 2^{14}+1= 16385.

$$c_1 = a_1 + a_2 = 1418 + 1953 = 3371(mod\ 16385) = 3371;$$
$$c_2 = a_1 - a_2 = 1418 - 1953 = -535(mod\ 16385) = 15850.$$

У ході обчислень кожне значення було збільшене на один розряд. Отримано значення вектора $C_3 = (3371, 15850)$.

Цей вектор подається як 30– бітний код

$C_3 = c_{31}||c_{32}$ =00 0110 1001 0101 1011 1101 1110 1010.

Після останнього раунду виконується вихідне забілення. На блок даних C_3 накладається раундовий ключ K_3, операцією сума за модулем 2.

C_3=00 0110 1001 0101 1011 1101 1110 1010,

K_3=01 0010 0100 1001 0100 0111 0001 1100.

$$\begin{array}{r} 00\ 0110\ 1001\ 0101\ 1011\ 1101\ 1110\ 1010 \\ \oplus \quad 01\ 0010\ 0100\ 1001\ 0100\ 0111\ 0001\ 1100 \\ \hline 01\ 0100\ 1101\ 1100\ 1111\ 1010\ 1111\ 0110 \end{array}$$

Отримано зашифроване повідомлення C = 01 0100 1101 1100 1111 1010 1111 0110.

2.5 Приклад розшифрування

Процес розшифрування полягає у виконанні у зворотному порядку операцій процесу зашифрування (рис. 2). Спочатку виконується розгортання ключа та вхідне забілювання, шляхом накладання ключа K_3 на блок даних зашифрованого повідомлення C – 30 біт .

C= 01 0100 1101 1100 1111 1010 1111 0110;

K_3=01 0010 0100 1001 0100 0111 0001 1100,

$$\begin{array}{rl} & 01\ 0100\ 1101\ 1100\ 1111\ 1010\ 1111\ 0110 \\ \oplus & 01\ 0010\ 0100\ 1001\ 0100\ 0111\ 0001\ 1100 \\ \hline & 00\ 0110\ 1001\ 0101\ 1011\ 1101\ 1110\ 1010 \end{array}$$

Результат накладання раундового ключа ділиться на дві частини по 15 біт

$$v_{31}=(000\ 1101\ 0010\ 1011)_2=(3371)_{10};$$

$$v_{32}=(011\ 1101\ 1110\ 1010)_2=(15850)_{10};$$

та подається, як вектор $V_3 = (3371; 15850)$.

Далі виконується обернене перетворення Уолша для матриці розмірністю 2:

$$\boldsymbol{M}_3 = \boldsymbol{k}_8 \cdot \boldsymbol{V}_3 \cdot \boldsymbol{H}_2.$$

Для обчислення константи k_8 для матриці другого порядку H_2 , необхідно обернене значення розмірності матриці піднести до модуля за яким відбувалось зашифрування, тобто $2^{n-1}+1$.

Тут $(k_8) = \binom{1}{-2} mod(2^{15}+1) = 8193$.

При цьому операції додавання, віднімання та множення виконуються за модулем 2^{14}+1= 16385 .

$$m_1 = (a_1 + a_2) * k_8 = (3371 + 15850) * 8193 = 157477653(mod\ 16385) = 1418$$

$$m_2 = (a_1 - a_2) * k_8 = (3371 - 15850) * 8193 = -102240447(mod\ 16385) = 1953$$

Результатом перетворення є вектор $M_3(1418; 1953)$.

Цей вектор подається як 28-розрядний код

$$M_3 = m_{31}||m_{32} = 0001\ 0110\ 0010\ 1000\ 0111\ 1010\ 0001.$$

Після першого раунду розшифрування накладається раундовий ключ K_2 на зашифрований блок даних M_3 за допомогою операції сума за модулем 2 .

M_3=0001 0110 0010 1000 0111 1010 0001;

K_2=0101 1110 1000 0100 0001 1010 1100;

$$\begin{array}{ll} & 0001\ 0110\ 0010\ 1000\ 0111\ 1010\ 0001 \\ \oplus & 0101\ 1110\ 1000\ 0100\ 0001\ 1010\ 1100 \\ \hline & 0100\ 1000\ 1010\ 1100\ 0110\ 1000\ 1101 \end{array}$$

Результат додавання за модулем 2 кодів M_3 і K_2 ділиться на 4 частини

$$v_{21} = (0100100)_2 = (36)_{10};\ v_{22} = (0101011)_2 = (43)_{10};$$
$$v_{23} = (0001101)_2 = (13)_{10};\ v_{24} = (0001101)_2 = (13)_{10};$$

Та подається як вектор V_2=(36, 43, 13, 13).

Далі виконується обернене перетворення Уолша для матриці розмірності 4:

$$\boldsymbol{M}_2 = \boldsymbol{k}_4 \cdot \boldsymbol{V}_2 \cdot \boldsymbol{H}_4.$$

Для обчислення константи k_4 для матриці четвертого порядку H_4 , необхідно обернене значення розмірності матриці піднести до модуля за яким відбувалось зашифрування, тобто $2^{n-1} + 1$

Тут $(k_4) = \overset{1}{(-)}_4 mod(2^6 + 1) = 49$.

за алгоритмом швидкого перетворення. При цьому операції додавання, віднімання та множення виконуються за модулем $2^6+1 = 65$.

Крок 1

$$a_1 = v_{21} + v_{22} = 36 + 43 = 79(mod\ 65) = 14;$$
$$a_2 = v_{21} - v_{22} = 36 - 43 = -7(mod\ 65) = 58;$$
$$a_3 = v_{23} + v_{24} = 13 + 13 = 26(mod\ 65) = 26;$$
$$a_4 = v_{23} - v_{24} = 13 - 13 = 0(mod\ 65) = 0;$$

Крок 2

$$c_1 = a_1 + a_3 * k_4 = (14 + 26) * 49 = 1960(mod\ 65) = 10;$$
$$c_2 = a_2 + a_4 * k_4 = (58 + 0) * 49 = 2842(mod\ 65) = 44;$$
$$c_3 = a_1 - a_3 * k_4 = (14 - 26) * 49 = -588(mod\ 65) = 62;$$
$$c_4 = a_2 - a_4 * k_4 = (58 - 0) * 49 = 2842(mod\ 65) = 47;$$

Результатом перетворення є вектор $M_2(10, 47, 62, 47)$.

Цей вектор подається як 24-розрядний код

$$M_2 = m_{21}||m_{22}||m_{23}||m_{24} = 0010\ 1010\ 1111\ 1111\ 1010\ 1111.$$

Після другого раунду розшифрування накладається раундовий ключ K_1 на зашифрований блок даних M_1 за допомогою операції сума за модулем 2

M_1=0010 1010 1111 1111 1010 1111 ; K_1= 0101 1011 0010 0111 0110 1100

$$\begin{array}{rl} & 0010\ 1010\ 1111\ 1111\ 1010\ 1111 \\ \oplus & 0101\ 1011\ 0010\ 0111\ 0110\ 1100 \\ \hline & 0111\ 0001\ 1100\ 1000\ 1100\ 0011 \end{array}$$

Результат додавання за модулем 2 кодів M_1 і K_1, ділиться на 8 частин

$$v_{11} = (011)_2 = (3)_{10};\ v_{12} = (100)_2 = (4)_{10};\ v_{13} = (011)_2 = (3)_{10};$$
$$v_{14} = (100)_2 = (4)_{10};\ v_{15} = (100)_2 = (4)_{10};\ v_{16} = (011)_2 = (3)_{10};$$
$$v_{17} = (000)_2 = (0)_{10};\ v_{18} = (011)_2 = (3)_{10}$$

Та подається як вектор V_1 = (3, 4, 3, 4, 4, 3, 0, 3).

Наступним кроком виконується обернене перетворення Уолша для матриці розмірностю 8:

$$\boldsymbol{M}_1 = \boldsymbol{k}_8 \cdot \boldsymbol{V}_2 \cdot \boldsymbol{H}_8.$$

Для обчислення константи k_8 для матриці восьмого порядку H_8 , необхідно обернене значення розмірності матриці піднести до модуля за яким відбувалось зашифрування, тобто $2^{n-1}+1$

Тут $(k_8) = (-)^{1}_{8}\ mod(2^2+1) = 2$.

за алгоритмом швидкого перетворення. При цьому операції додавання, віднімання та множення виконуються за модулем 2^2+1= 5.

Крок 1

$$a_1 = v_{11} + v_{12} = 3 + 4 = 7(mod\ 5) = 2;$$
$$a_2 = v_{11} - v_{12} = 3 - 4 = -1(mod\ 5) = 4;$$
$$a_3 = v_{13} + v_{14} = 3 + 4 = 7(mod\ 5) = 2;$$
$$a_4 = v_{13} - v_{14} = 3 - 4 = -1(mod\ 5) = 4;$$
$$a_5 = v_{15} + v_{16} = 4 + 3 = 7(mod\ 5) = 2;$$
$$a_6 = v_{15} - v_{16} = 4 - 3 = 1(mod\ 5) = 1;$$
$$a_7 = v_{17} + v_{18} = 0 + 3 = 3(mod\ 5) = 3;$$
$$a_8 = v_{17} - v_{18} = 0 - 3 = -3(mod\ 5) = 2;$$

Крок 2

$$b_1 = a_1 + a_3 = 2 + 2 = 4(mod\ 5) = 4;$$
$$b_2 = a_2 + a_4 = 4 + 4 = 8(mod\ 5) = 3;$$
$$b_3 = a_1 - a_3 = 2 - 2 = 0(mod\ 5) = 0;$$
$$b_4 = a_2 - a_4 = 4 - 4 = 0(mod\ 5) = 0;$$
$$b_5 = a_5 + a_7 = 2 + 3 = 5(mod\ 5) = 0;$$
$$b_6 = a_6 + a_8 = 1 + 2 = 3(mod\ 5) = 3;$$
$$b_7 = a_5 - a_7 = 2 - 3 = -1(mod\ 5) = 4;$$
$$b_8 = a_6 - a_8 = 1 - 2 = -1(mod\ 5) = 4;$$

Крок 3

$$m_1 = (b_1 + b_5) * k_8 = (4 + 0) * 2 = 8(mod\ 5) = 3;$$
$$m_2 = (b_2 + b_6) * k_8 = (3 + 3) * 2 = 12(mod\ 5) = 2;$$
$$m_3 = (b_3 + b_7) * k_8 = (0 + 4) * 2 = 8(mod\ 5) = 3;$$
$$m_4 = (b_4 + b_8) * k_8 = (0 + 4) * 2 = 8(mod\ 5) = 3;$$
$$m_5 = (b_1 - b_5) * k_8 = (4 - 0) * 2 = 8(mod\ 5) = 3;$$
$$m_6 = (b_2 - b_6) * k_8 = (3 - 3) * 2 = 0(mod\ 5) = 0;$$
$$m_7 = (b_3 - b_7) * k_8 = (0 - 4) * 2 = -8(mod\ 5) = 2;$$
$$m_8 = (b_4 - b_8) * k_8 = (0 - 4) * 2 = -8(mod\ 5) = 2;$$

Результатом перетворення є вектор $(M_0(3, 2, 3, 3, 3, 0, 2, 2)$.

Цей вектор подається як 24– бітний код

$$M_0 = (m_{01}||\ m_{02}||m_{03}||\ m_{04}||m_{05}||\ m_{06}||m_{07}||\ m_{08}) = 1110\ 1111\ 1100\ 1010.$$

Після фінального раунду на зашифрований блок даних M_0 накладається раундовий ключ K, операцією сума за модулем 2, таким чином виконується вихідне забілювання.

M_0=1110 1111 1100 1010, K= 0101 1011 0110 1100 ;

$$\begin{array}{rl} & 1110\ 1111\ 1100\ 1010 \\ \oplus & 0101\ 1011\ 0110\ 1100 \\ \hline & 1011\ 0100\ 1010\ 0110 \end{array}$$

Отримано розшифроване повідомлення M = 1011 0100 1010 0110

2.6 Оцінки алгоритмічної складності шифрування блоку даних

Алгоритмічна складність процесу зашифрування обчислюється за формулою:

$$S_E = S_{\oplus} + S_{W2} + S_{W4} + S_{W8} + S_K.$$

Умовною одиницею складності є арифметична операція за модулем $2^n + 1$.

$S_{\oplus} = 4$.

Складність перетворень Уолша розмірності 2 дорівнює $S_{W2} = 2$, розмірності 4 - $S_{W4} = 8$, а розмірності 8 - $S_{W8} = 24$.

Виходячи з цього, складність розгортання ключа дорівнює $S_K = 17$.

Таким чином, алгоритмічна складність процесу зашифрування блоку даних довжиною 128 біт дорівнює $S_E = 55$ ум. од.. Тобто для зашифрування одного байту даних потрібно виконати 3,4 операцій.

Для порівняння, зашифрування алгоритмом AES блоку даних довжиною 128 біт потрібно виконати 420 операцій [15].

Отже, запропонований метод блокового шифрування забезпечує пришвидшення процесу зашифрування в 7,6 разів.

Алгоритмічна складність процесу розшифрування дорівнює $S_D = 69$ ум. од.. Вона є більшою за складність процесу за шифрування через те, що обернене перетворення Уолша передбачає виконання множення певну константу.

3 РОЗРОБКА ЗАСОБУ ДЛЯ ШИФРУВАННЯ

3.1 Обґрунтування мови програмування

Вибір мови програмування Python для втілення блочного шифрування визначається кількома вагомими аспектами, які роблять її найкращим вибором для таких завдань. Одним із основних факторів є відмінна читабельність коду, що дозволяє розробникам, навіть з середнім рівнем досвіду, швидко розуміти структуру та логіку програми. Python [22], за своєю природою, орієнтований на легкість сприйняття коду, полегшуючи його налагодження та перевірку, що є критично важливим у розробці криптографічних алгоритмів, де кожна помилка може мати серйозні наслідки. Читабельність коду також вигідна в академічних дослідженнях і реалізаціях, адже дозволяє чітко продемонструвати алгоритмічні кроки без зайвої складності.

Важливим аспектом є також динамічна типізація, що дає змогу ефективно працювати з різними типами даних та змінювати структуру програми без потреби численних змін у коді. Наприклад, при реалізації блокових шифрів, де використовуються великі числа та операції з ними, Python дозволяє працювати з цілими числами довільної довжини, що значно спрощує реалізацію криптографічних операцій без необхідності додаткових обчислень з розрядністю. Це зменшує вірогідність виникнення помилок переповнення чи неточностей у роботі з великими числами, що можуть виникати при використанні інших мов з фіксованими типами даних, таких як C або Java.

Іншим чинником, який обґрунтовує вибір Python, є його висока продуктивність на етапах розробки та тестування. Python дозволяє швидко створювати прототипи, що є критичним для криптографічних досліджень, де важливо мати можливість швидко змінювати параметри шифрування, кількість раундів, схеми доповнення та інші аспекти алгоритму для їхньої перевірки та оптимізації. Завдяки цьому розробник може зосередитися на оптимізації самих

алгоритмів, а не на деталях реалізації, що є значною перевагою під час тестування криптографічних рішень.

Щодо безпеки, Python дає можливість легко інтегрувати та застосовувати перевірки безпеки через наявні вбудовані механізми для аудиту та валідації коду. Відкритість Python дозволяє перевіряти кожен етап розробки, а також впроваджувати найкращі практики в реалізації криптографічних рішень. Це включає забезпечення відповідності міжнародним стандартам криптографії, що є важливим аспектом для наукових та практичних застосувань.

Окрім цього, Python володіє дуже розвиненою підтримкою для паралельних обчислень та багатозадачності, що може бути корисним при реалізації паралельних шифрувальних операцій або тестуванні великих обсягів даних. Хоча для простих задач Python може не бути таким швидким, як C чи C++, завдяки можливості впровадження розширень або використанню технологій на базі Cython чи інших інтерфейсів, Python дозволяє оптимізувати найкритичніші ділянки коду без необхідності переписувати всю програму іншою мовою.

Додатково, Python є мультиплатформенною мовою, що дозволяє запускати розроблені програми на різних операційних системах без значних змін у коді. Це забезпечує високу переносимість програми, що важливо в умовах реального застосування криптографічних алгоритмів у різноманітних системах.

Отже, Python є оптимальним вибором для розробки програми, що реалізує блокове шифрування, завдяки своїй простоті, потужним можливостям для швидкої розробки, високій читабельності коду, а також здатності підтримувати великі обсяги даних та забезпечувати необхідний рівень безпеки.

3.2 Алгоритм роботи програмного засобу для блокового шифрування даних

Програмний засіб для шифрування даних на основі швидкого перетворення Уолша складається з модулів, показаних на рис 3.1.

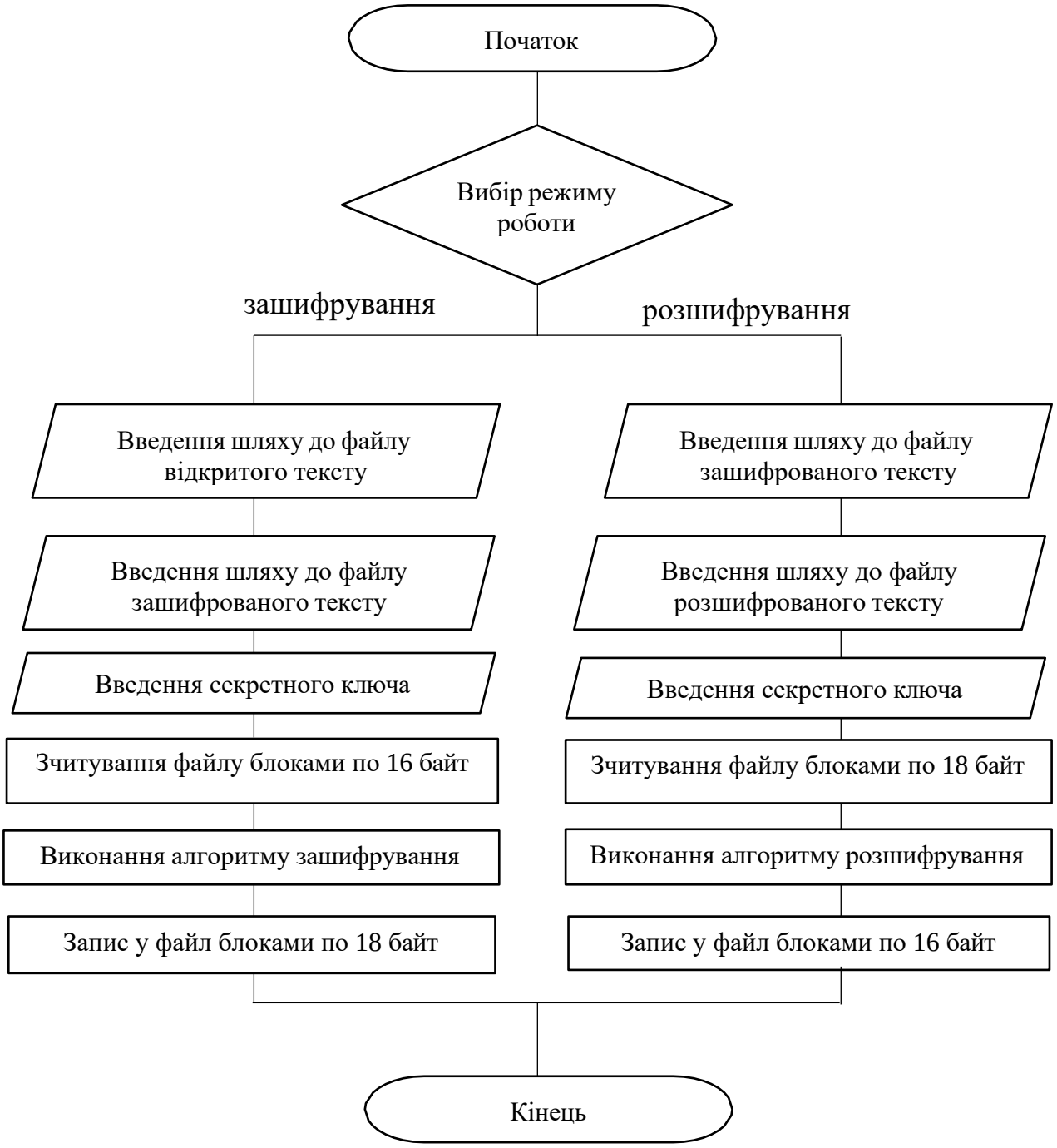


Рисунок 3.1 – Алгоритм роботи програмного засобу для шифрування даних на основі швидкого перетворення Уолша

В свою чергу виконання алгоритму зашифрування складається з модулів, показаних на рис. 3.2, а розшифрування з модулів, вказаних на рис. 3.3.

Після запуску програми необхідно обрати режим роботи, який необхідний користувачу: зашифрування – e або розшифрування – d, якщо користувач обирає не коректний режим роботи програма видає помилку та виведе «Помилка: введіть 'e' або 'd'.»

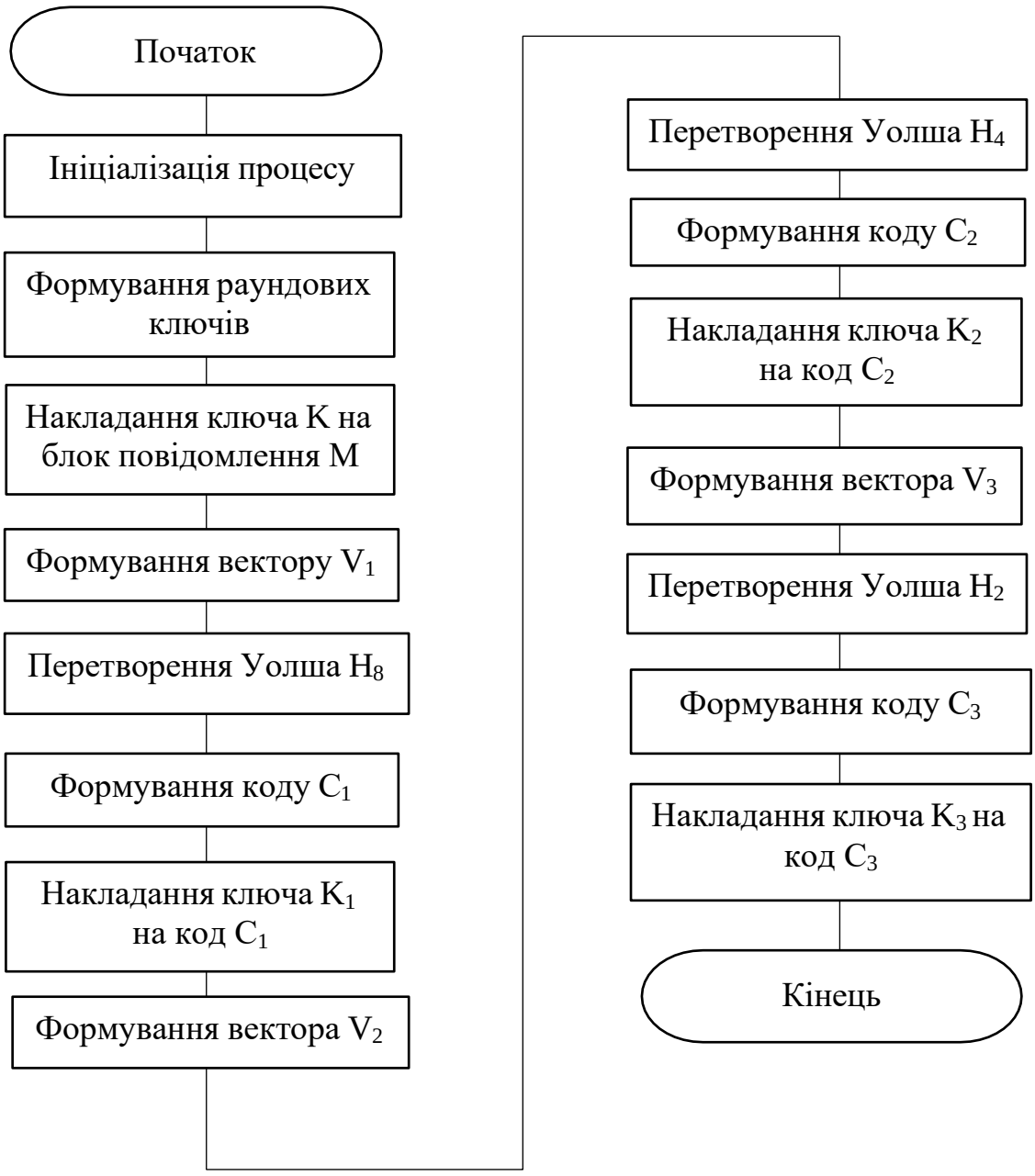


Рисунок 3.2 – Алгоритм зашифрування на основі швидкого перетворення Уолша

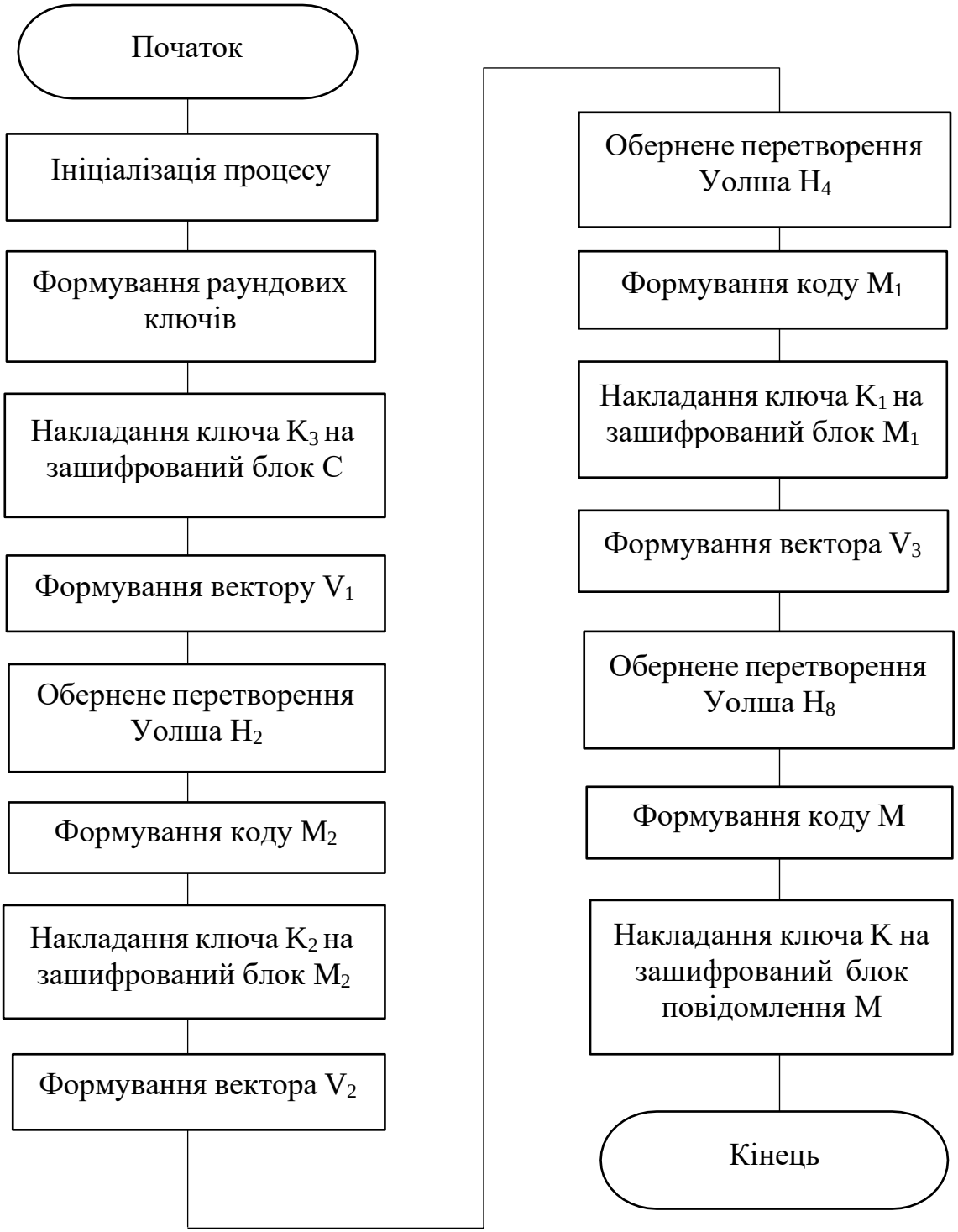


Рисунок 3.3 – Алгоритм розшифрування на основі швидкого перетворення Уолша

```
print("Оберіть режим: 'e' – зашифрування, 'd' – розшифрування'")
    mode = input("Введіть 'e' або 'd': ").strip().lower()
    if mode not in ('e', 'd'):
        print("Помилка: введіть 'e' або 'd'.")
        return
```

Наступним кроком є введення шляху до файлу який користувач хоче зашифрувати чи розшифрувати

```
input_path = input("Введіть шлях до вхідного файлу: ").strip()
```

і шлях до файлу у який буде записано зашифрований чи розшифрований файл.

```
output_path = input("Введіть шлях до вихідного файлу: ").strip()
```

Після цього необхідно ввести секретний ключ, який є цілим числом не більшим за 128 біт.

```
key_str = input("Введіть 128-бітний секретний ключ у десятковому
форматі: ").strip()
    try:
        K = int(key_str)
    except ValueError:
        print("Помилка: ключ має бути десятковим цілим числом.")
        return
```

Після цього викликається функція розгортання раундових ключів .

```
K, K1, K2, K3 = expand_key(K)
round_keys = (K, K1, K2, K3)
```

Функція розгортання ключа реалізована наступним чином

```
def expand_key(K: int) -> tuple[int, int, int, int]:
    K1 = (K << 8) ^ K
    K2 = (K1 << 4) ^ K1
    K3 = (K2 << 2) ^ K2
    return K, K1, K2, K3
```

Після цього викликається функція `process_stream`.

```
process_stream(input_path, output_path, round_keys, mode)
```

Вона обробляє всі дані, які ввів користувач

```
def process_stream(
    input_path: str,
    output_path: str,
    round_keys: tuple[int, int, int, int],
    mode: str
) -> None:
```

Та в залежності від режиму роботи починає зчитувати інформацію в байтах з вхідного файлу блоками по 16 або18 байт та відкриває вихідний файл для запису блоками 18 або 16 байт. Це необхідно оскільки під час виконання зашифрування блок повідомлення збільшується на 14 біт.

```
 with open(input_path, 'rb') as fin, open(output_path, 'wb') as
fout:
        if mode == 'e':
            read_size = 16
            write_size = 18
        else:
            read_size = 18
            write_size = 16
         while True:
            chunk = fin.read(read_size)
            if not chunk:
                break
```

після того як отримано блок даних в байтах він переводиться в ціле число для виконання функції зашифрування чи розшифрування.

```
            block_int = int.from_bytes(chunk, byteorder='big')
            if mode == 'e':
                out_int = encrypt_block(block_int, round_keys)
            else:
                out_int = decrypt_block(block_int, round_keys)
```

коли блок повідомлення зашифровано він переводиться в байти та записується в вихідний файл

```
out_bytes = out_int.to_bytes(write_size, byteorder='big')
            fout.write(out_bytes)
```

Коли програма завершає роботу в залежності від режиму роботи програми виводиться повідомлення пр. те що файл зашифровано чи розшифровано у файл який ввів користувач.

```
 if mode == 'e':
        print(f"Файл зашифровано, результат записано у
{output_path}")
```

Функція зашифрування відповідно до рис 3.2 описана наступним чином

```
def encrypt_block(block_int: int, round_keys: tuple[int, int, int,
int]) -> int:
    K, K1, K2, K3 = round_keys
```

Функція отримує блок повідомлення, який є цілим числом та раундові ключі. Після цього операцією сума за модулем 2 накладається на блок повідомлення накладається секретний ключ.

```
tmp128 = block_int ^ K
```

Після цього формується вектор v для перетворення Уолша H_8 за допомогою зсуву та накладання маски. Кожне значення вектору 16 біт.

```
v = [(tmp128 >> (16 * (7 - i))) & 0xFFFF for i in range(8)]
   v0, v1, v2, v3, v4, v5, v6, v7 = v
```

Наступним кроком обраховується модуль, за яким будуть проводитись обчислення

```
mod16p1 = (1 << 16) + 1
```

та перетворення Уолша для матриці восьмого порядку :

Перший крок :

```
a0 = (v0 + v4) % mod16p1
a1 = (v1 + v5) % mod16p1
a2 = (v2 + v6) % mod16p1
a3 = (v3 + v7) % mod16p1
a4 = (v0 - v4) % mod16p1
a5 = (v1 - v5) % mod16p1
a6 = (v2 - v6) % mod16p1
a7 = (v3 - v7) % mod16p1
```

Другий крок :

```
b0 = (a0 + a4) % mod16p1
b1 = (a1 + a5) % mod16p1
b2 = (a2 + a6) % mod16p1
b3 = (a3 + a7) % mod16p1
b4 = (a0 - a4) % mod16p1
b5 = (a1 - a5) % mod16p1
b6 = (a2 - a6) % mod16p1
b7 = (a3 - a7) % mod16p1
```

Третій крок :

```
c0 = (b0 + b4) % mod16p1
c1 = (b1 + b5) % mod16p1
c2 = (b2 + b6) % mod16p1
```

```
    c3 = (b3 + b7) % mod16p1
    c4 = (b0 - b4) % mod16p1
    c5 = (b1 - b5) % mod16p1
    c6 = (b2 - b6) % mod16p1
    c7 = (b3 - b7) % mod16p1
```

Після цього формується вектор `C_stage1` та формується результат перетворення H_8 `C1.`

```
C_stage1 = [c0, c1, c2, c3, c4, c5, c6, c7]
    C1 = 0
    mask17 = (1 << 17) - 1
    for i, ci in enumerate(C_stage1):
        shift17 = 17 * (7 - i)
        C1 |= (ci & mask17) << shift17
```

Наступним кроком на результат `C1` накладається раундовий ключ K_1.

```
result136 = C1 ^ K1
```

та формується вектор V2 з чотирьох елементів по 34 біти для перетворення Уолша H_4.

```
V2 = []
mask34 = (1 << 34) - 1
for i in range(4):
    shift34 = 34 * (3 - i)
    seg34 = (result136 >> shift34) & mask34
    V2.append(seg34)
vv0, vv1, vv2, vv3 = V2
```

Після цього обраховується модуль за яким буде проводитись швидке перетворення Уолша для матриці четвертого порядку H_4

```
mod34p1 = (1 << 34) + 1
a0p = (vv0 + vv2) % mod34p1
a1p = (vv1 + vv3) % mod34p1
a2p = (vv0 - vv2) % mod34p1
a3p = (vv1 - vv3) % mod34p1
c0p = (a0p + a2p) % mod34p1
c1p = (a1p + a3p) % mod34p1
c2p = (a0p - a2p) % mod34p1
c3p = (a1p - a3p) % mod34p1
C_stage2 = [c0p, c1p, c2p, c3p]
```

Після чого формується вектор `C2`

```
C2 = 0
 mask35 = (1 << 35) - 1
 for i, ci in enumerate(C_stage2):
     shift35 = 35 * (3 - i)
     C2 |= (ci & mask35) << shift35
```

Та накладається раундовий ключ K_2

```
 result140 = C2 ^ K2
```

Наступним кроком формується вектор V3 з двох елементів по 70 біт.

```
 V3 = []
 mask70 = (1 << 70) - 1
 for i in range(2):
     shift70 = 70 * (1 - i)
     seg70 = (result140 >> shift70) & mask70
     V3.append(seg70)
 w0, w1 = V3
```

Далі обраховується модуль за яким будуть виконуватись перетворення та виконується перетворення Уолша для матриці другого порядку H_2.

```
 mod70p1 = (1 << 70) + 1
 d0 = (w0 + w1) % mod70p1
 d1 = (w0 - w1) % mod70p1
 C_stage3 = [d0, d1]
```

Після чого формується результат перетворення, як ціле 142 розрядне число

.

```
 C3 = 0
 mask71 = (1 << 71) - 1
 for i, ci in enumerate(C_stage3):
     shift71 = 71 * (1 - i)
     C3 |= (ci & mask71) << shift71
```

Та накладається раундовий ключ K_3 після чого повертається результат, який є цілим числом.

```
 result142 = C3 ^ K3
 return result142
```

Функція розшифрування показана на рис. 3. З описано наступним чином

```
def decrypt_block(block_int: int, round_keys: tuple[int, int, int,
int]) -> int:
    K, K1, K2, K3 = round_keys
```

Функція має на вході зашифрований блок повідомлення розмірністю 142 біта та значення раундових ключів. У функції зашифрування все буде виконуватись обернено до функції зашифрування. Перша операція сума за модулем 2 зашифрованого блоку і ключа K_3.

```
C3 = block_int ^ K3
```

Після цього формується вектор для зворотного швидкого перетворення Уолша для матриці H_2.

```
mask71 = (1 << 71) - 1
shift_d0 = 71
d0 = (C3 >> shift_d0) & mask71
d1 = C3 & mask71
```

Далі обраховується модуль за яким буде виконуватись перетворення та значення на яке необхідно буде множити для оберненого перетворення Уолша.

```
mod70p1 = (1 << 70) + 1
inv2_70 = (1 << 69) + 1
```

та виконання перетворення Уолша для матриці другого порядку.

```
w0 = ((d0 + d1) * inv2_70) % mod70p1
w1 = ((d0 - d1) * inv2_70) % mod70p1
```

обраховані значення об’єднуються

```
result140 = (w0 << 70) | w1
```

Наступним кроком на результат накладається ключ K_2

```
C2 = result140 ^ K2
```

Далі зашифрований блок повідомлення розділяється на 4 частини по 35 біт.

```
mask35 = (1 << 35) - 1
c0p = (C2 >> (35 * 3)) & mask35
c1p = (C2 >> (35 * 2)) & mask35
c2p = (C2 >> (35 * 1)) & mask35
c3p = C2 & mask35
```

Після чого обраховується модуль за яким буде виконуватись перетворення та значення на яке необхідно буде множити для оберненого перетворення Уолша.

```
mod34p1 = (1 << 34) + 1
```

```
inv2_34 = (1 << 33) + 1
a0p = ((c0p + c2p) * inv2_34) % mod34p1
a2p = ((c0p - c2p) * inv2_34) % mod34p1
a1p = ((c1p + c3p) * inv2_34) % mod34p1
a3p = ((c1p - c3p) * inv2_34) % mod34p1
v0p = ((a0p + a2p) * inv2_34) % mod34p1
v2p = ((a0p - a2p) * inv2_34) % mod34p1
v1p = ((a1p + a3p) * inv2_34) % mod34p1
v3p = ((a1p - a3p) * inv2_34) % mod34p1
```

Значення отримані в результати перетворення Уолша для матриці четвертого порядку H_4 об'єднуються в одне значення.

```
result136 = (v0p << (34 * 3)) | (v1p << (34 * 2)) | (v2p << (34
* 1)) | v3p
```

Після цього на зашифрований блок повідомлення накладається раундовий ключ K_1.

```
C1 = result136 ^ K1
```

Наступним кроком блок повідомлення розділяється на 8 частин по 17 біт для швидкого оберненого перетворення Уолша для матриці H_8

```
mask17 = (1 << 17) - 1
c0 = (C1 >> (17 * 7)) & mask17
c1 = (C1 >> (17 * 6)) & mask17
c2 = (C1 >> (17 * 5)) & mask17
c3 = (C1 >> (17 * 4)) & mask17
c4 = (C1 >> (17 * 3)) & mask17
c5 = (C1 >> (17 * 2)) & mask17
c6 = (C1 >> (17 * 1)) & mask17
c7 = C1 & mask17
```

Після чого обраховується модуль за яким буде виконуватись перетворення та значення на яке необхідно буде множити для оберненого перетворення Уолша.

```
mod16p1 = (1 << 16) + 1
inv2_16 = (1 << 15) + 1
```

та виконується швидке обернене перетворення Уолша для матриці H_8:

Перший крок:

```
b0 = ((c0 + c4) * inv2_16) % mod16p1
b1 = ((c1 + c5) * inv2_16) % mod16p1
b2 = ((c2 + c6) * inv2_16) % mod16p1
```

```
    b3 = ((c3 + c7) * inv2_16) % mod16p1
    b4 = ((c0 - c4) * inv2_16) % mod16p1
    b5 = ((c1 - c5) * inv2_16) % mod16p1
    b6 = ((c2 - c6) * inv2_16) % mod16p1
    b7 = ((c3 - c7) * inv2_16) % mod16p1
```

Другий крок:

```
    a0 = ((b0 + b4) * inv2_16) % mod16p1
    a1 = ((b1 + b5) * inv2_16) % mod16p1
    a2 = ((b2 + b6) * inv2_16) % mod16p1
    a3 = ((b3 + b7) * inv2_16) % mod16p1
    a4 = ((b0 - b4) * inv2_16) % mod16p1
    a5 = ((b1 - b5) * inv2_16) % mod16p1
    a6 = ((b2 - b6) * inv2_16) % mod16p1
    a7 = ((b3 - b7) * inv2_16) % mod16p1
```

Третій крок:

```
    v0 = ((a0 + a4) * inv2_16) % mod16p1
    v1 = ((a1 + a5) * inv2_16) % mod16p1
    v2 = ((a2 + a6) * inv2_16) % mod16p1
    v3 = ((a3 + a7) * inv2_16) % mod16p1
    v4 = ((a0 - a4) * inv2_16) % mod16p1
    v5 = ((a1 - a5) * inv2_16) % mod16p1
    v6 = ((a2 - a6) * inv2_16) % mod16p1
    v7 = ((a3 - a7) * inv2_16) % mod16p1
```

Після чого отримані значення об’єднуються в одне

```
tmp128 = (
        (v0 << (16 * 7)) | (v1 << (16 * 6)) | (v2 << (16 * 5)) |
     (v3 << (16 * 4)) |
        (v4 << (16 * 3)) | (v5 << (16 * 2)) | (v6 << (16 * 1)) | v7
)
```

На отримане значення накладається секретний ключ K, після чого функція повертає розшифрований блок повідомлення.

```
    plaintext = tmp128 ^ K
    return plaintext
```

3.3 Приклад роботи програмного засобу

Для того щоб показати як працює програмний засіб на основі швидкого перетворення Уолша буде показано процедуру та результати на прикладі двох

контейнерів перший це текстовий файл з розширенням .txt та графічний файл з розширенням .bmp.

Для зашифрування написано довільний текст та збережено в текстовому документі. Приклад тексту, який буде зашифровано, показано на рис.3.4.

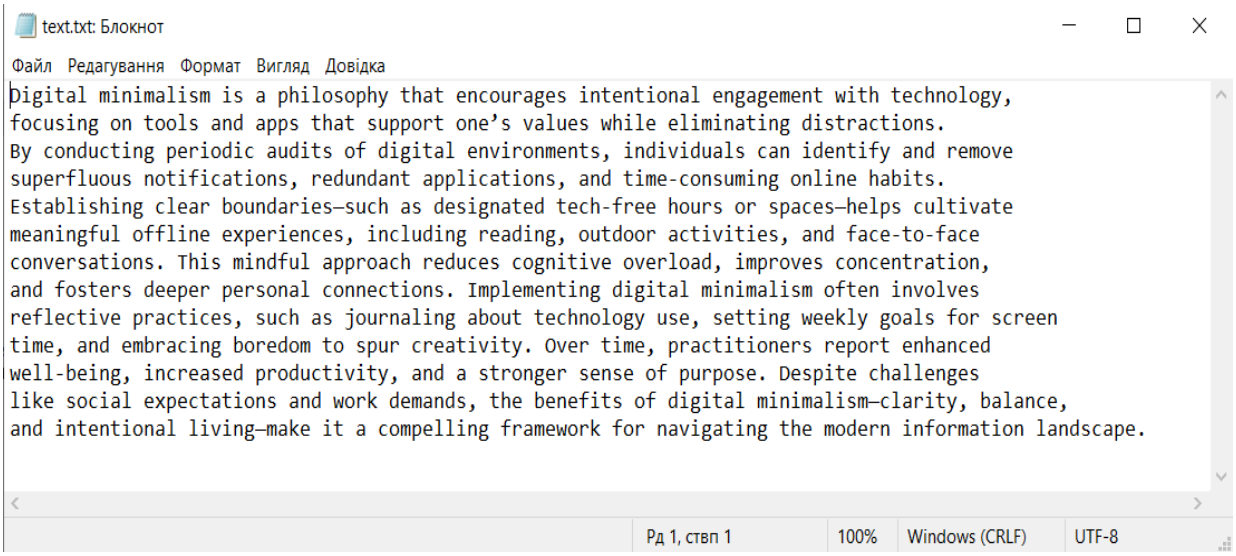


Рисунок 3.4 – Текстовий документ, який буде зашифровано

Приклад зашифрування даних, показаний на рис. 3.5.

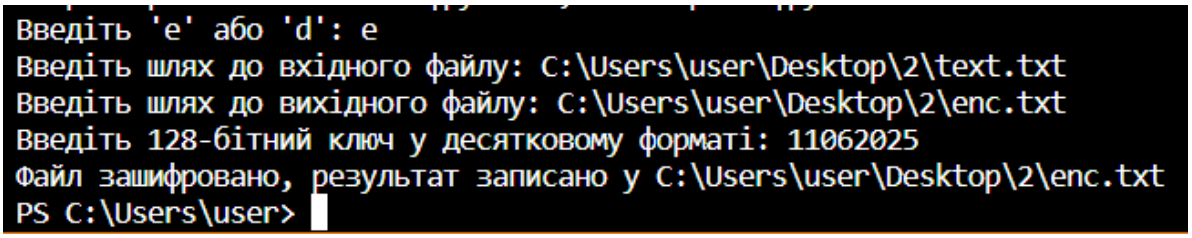


Рисунок 3.5 – Приклад роботи програми в режимі зашифрування

Результат зашифрування записаний у файл enc.txt, показаний на рис. 3.6

Зашифрування пройшло успішно. Приклад розшифрування даних, показано на рис. 3.7.

Результат розшифрування записаний у текстовий файл dec.txt, показаний на рис. 3.8.

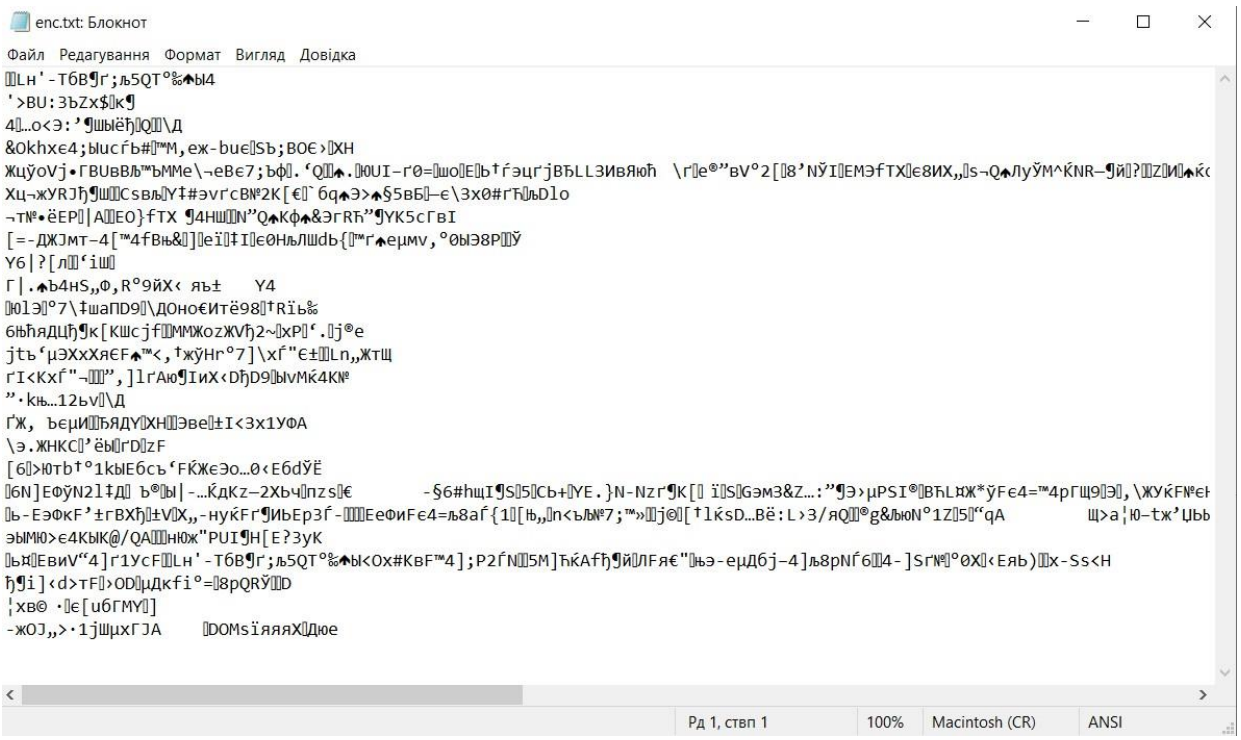


Рисунок 3.6 – Зашифрований файл

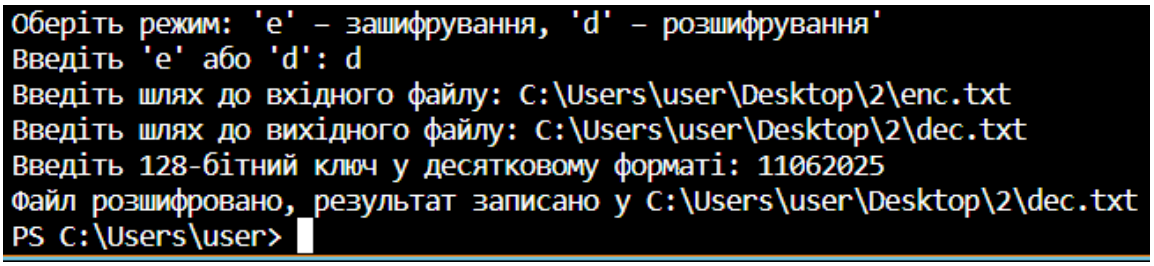


Рисунок 3.7 – Приклад роботи програми в режимі зашифрування

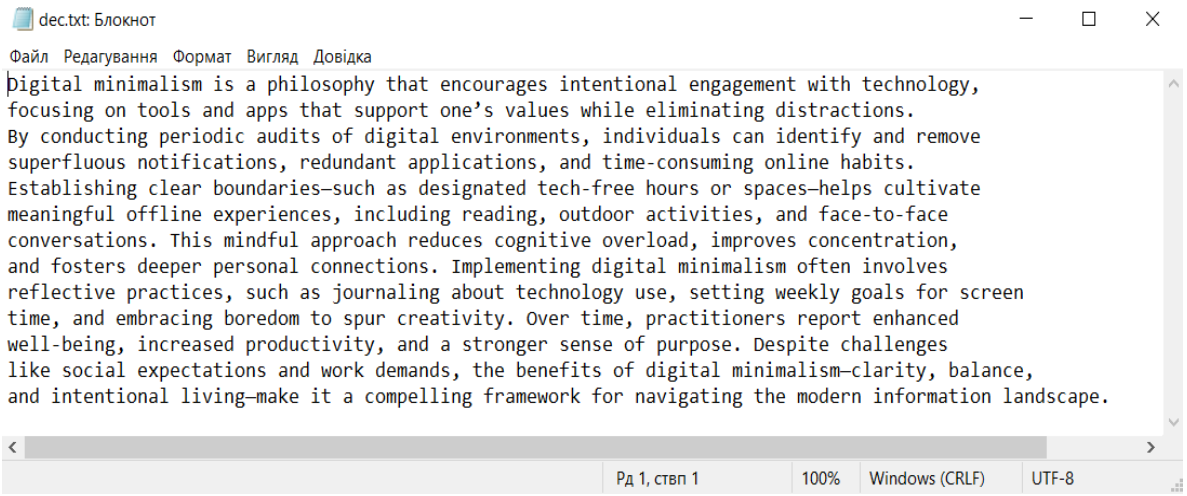


Рисунок 3.8 – Розшифрований файл

Розшифрування пройшло успішно текст ідентичний з вхідним значенням.

Наступним кроком буде перевірено програму на складнішому файлі. Буде зашифровано та розшифровано графічний файл. Файл показаний на рис. 3.9.

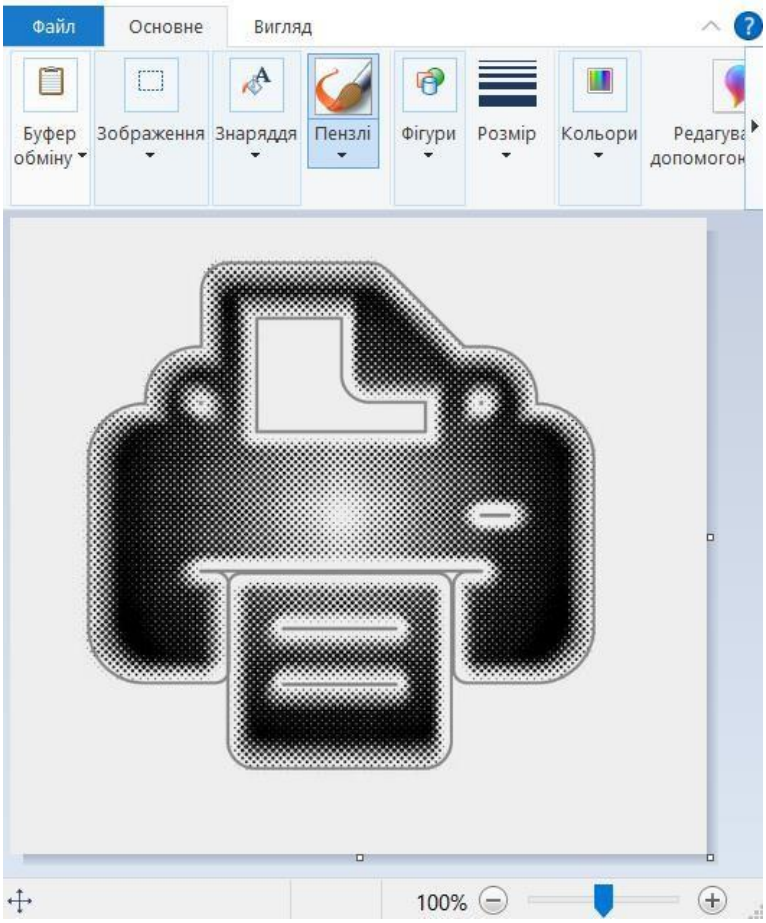


Рисунок 3.9 – Зображення графічного файлу, яке буде зашифровано

Приклад зашифрування, показано на рис. 3.10.

```
Оберіть режим: 'e' – зашифрування, 'd' – розшифрування'
Введіть шлях до вхідного файлу: C:\Users\user\Desktop\2\b.bmp
Введіть шлях до вихідного файлу: C:\Users\user\Desktop\2\e.bmp
Введіть 128-бітний секретний ключ у десятковому форматі: 12345
```

Рисунок 3.10 – Приклад зашифрування графічного файлу

При зашифруванні графічного файлу отриманий файл відкрити не вийшло, це показано на рис. 3.11.

Приклад розшифрування графічного фалу, показано на рис. 3.12.

Результат розшифрування графічного файлу записаний в d.bmp, показаний на рис. 3.13.

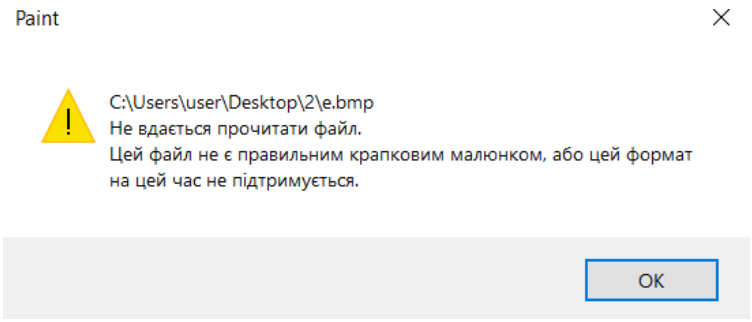


Рисунок 3.11 – Результат зашифрування графічного файлу

```
Оберіть режим: 'e' – зашифрування, 'd' – розшифрування'
Введіть 'e' або 'd': d
Введіть шлях до вхідного файлу: C:\Users\user\Desktop\2\e.bmp
Введіть шлях до вихідного файлу: C:\Users\user\Desktop\2\d.bmp
Введіть 128-бітний ключ у десятковому форматі: 12345
Файл розшифровано, результат записано у C:\Users\user\Desktop\2\d.bmp
```

Рисунок 3.12 – Приклад розшифрування графічного файлу

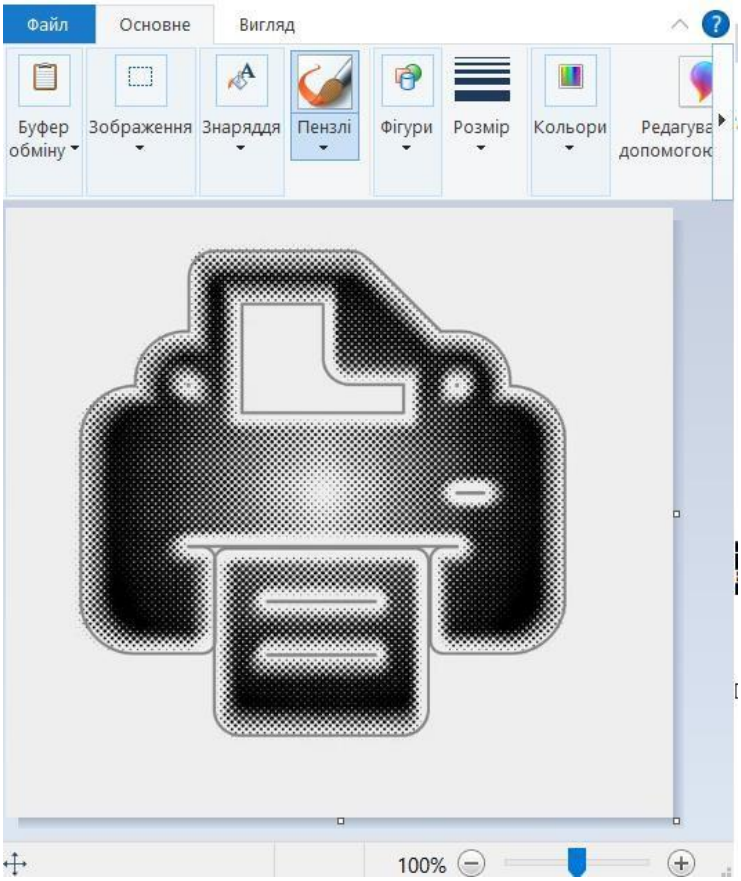


Рисунок 3.13 – Розшифрований графічний файл

Файл вдало розшифрований. Можна зробити висновок, що програма працює коректно.

ВИСНОВКИ

Наведений у першому розділі аналіз показав, що загальні підходи до побудови блокових шифрів, розроблені таким чином, що в кожному раунді виконується багато операцій підстановки та перестановки, а також різних математичних операцій. Оглянуто загальний підхід до шифрування даних.

У другому розділі розроблено методи зашифрування та розшифрування, особливістю яких є використання швидких перетворень Уолша, завдяки чому забезпечується одночасно перестановка і підстановка. Наведено схеми методів зашифрування та розшифрування, та показано на прикладі 16 бітного блоку даних як відбувається процедура шифрування та розгортання секретного ключа. Особливістю алгоритму є те що у ході зашифруваня збільшується розрядність блоку даних після перетворення Уолша розмірності 8 збільшується на 8 біт, після перетворення розмірності 4 на 4 біти та розмірності 2 на два біти, загалом розмірність блоку дих після зашифрування збільшується на 14 біт. Алгоритм розгортання ключів враховує цю особливість алгоритму. Він виконується за допомогою накладання операцією сума за модулем два. Секретний ключ накладається на самого себе зсунутого на вісім позицій в ліво. Наступний раундовий ключ отримується за допомогою накладання першого раундовго ключа на самого себе зсунутого на 4 позиції вліво і третій ключ аналогічно до попередні накладається на результат попереднього розгортання зсунутий на дві позиції вліво. Це забезпечує підходящі розмірності до блоків даних, які утворюються у результаті виконання алгоритму.

У третьому розділі роботі було розроблено програмний засіб для шифрування даних, що реалізує описані в розділі два алгоритми. Для цього було використано мову програмування Python. Результати тестування програмного засобу показують працездатність програмного засобу. У ході тестування було здійснено шифрування різних типів даних на різних ключах.

Оцінка складності реалізації алгоритму шифрування на основі швидких перетворень Уолша показала, що розроблений метод шифрування даних

використовує 3,4 операцій для зашифрування одного байту даних. Тоді як алгоритм блокового шифрування AES витрачає 26,25 операцій на байт. Тобто пришвидшення процесу зашифрування досягає 7,6 разів.

Таким чином, сформульовані задачі бакалаврської кваліфікаційної роботи повністю розв’язані і досягнута мета роботи, а саме пришвидшення процесу шифрування даних.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Chabaud F., Vaudenay S. Links between differential and linear cryptanalysis / F. Chabaud, S. Vaudenay // EUROCRYPT 1994. — 1994. — LNCS vol. 950. — P. 356–365. — Springer-Verlag.
2. Anderson R., Biham E. Two practical and provably secure block ciphers: BEARS and LION / R. Anderson, E. Biham // In Gollmann. — 1999. — P. 113–120.
3. Anderson R., Biham E., Knudsen L. Serpent: a proposal for the advanced encryption standard / R. Anderson, E. Biham, L. Knudsen // Cambridge University, England. — 1998. — URL: https://www.cl.cam.ac.uk/archive/rja14/Papers/serpent.pdf (дата звернення: 21.05.2025).
4. Schneier B., Kelsey J., Whiting D., Wagner D., Hall C., Ferguson N. Twofish: a 128-bit block cipher / B. Schneier та ін. — 1998. — URL: https://www.schneier.com/wp-content/uploads/2016/02/paper-twofish-paper.pdf (дата звернення: 21.05.2025).
5. Камінський О. Є., Мамонова Г. В., Меднікова М. В. Криптографічний аналіз алгоритму DES / О. Є. Камінський, Г. В. Мамонова, М. В. Меднікова. — Київ, 2015. — С. 146–157.
6. Barker E., Mouha N. NIST Special Publication 800-67 Revision 2 Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher / E. Barker, N. Mouha. — 2017. — URL: https://doi.org/10.6028/NIST.SP.800-67r2 (дата звернення: 21.05.2025).
7. Hoang V. T., Rogaway P. On Generalized Feistel Networks / V. T. Hoang, P. Rogaway // Dept. of Computer Science, University of California, Davis, USA. — 2018. — URL: https://eprint.iacr.org/2010/301.pdf (дата звернення: 21.05.2025).

8. Adams C., Gilchrist J. The CAST-256 Encryption Algorithm / C. Adams, J. Gilchrist. — 1999. — URL: https://www.rfc-editor.org/rfc/rfc2612.html (дата звернення: 21.05.2025).
9. Rivest R. L., Robshaw M. J. B., Sidney R., Yin Y. L. The RC6 Block Cipher / R. L. Rivest та ін. — 1998. — URL: https://people.csail.mit.edu/rivest/pubs/RRSY98.pdf (дата звернення: 21.05.2025).
10. Kapalova N., Haumen A. The model of encryption algorithm based on non-positional polynomial notations and constructed on an SP-network / N. Kapalova, A. Haumen. — 2018. — URL: https://www.degruyterbrill.com/document/doi/10.1515/eng-2018-0013/html (дата звернення: 21.05.2025).
11. Agrawal M., Chang D., Ghosh M., Sanadhya S. K. Collision Attack on 4-branch, Type-2 GFN based Hash Functions using Sliced Biclique Cryptanalysis Technique / M. Agrawal та ін. — 2018.
12. Rijmen V., Daemen J., Preneel B., Bosselaers A., De Win E. The Cipher SHARK / V. Rijmen та ін. — 1996. — URL: https://cs.ru.nl/~joan/papers/VRI_JDA_BPE_ABO_EDW_SHARK_1996.pdf (дата звернення: 21.05.2025).
13. Daemen J., Knudsen L., Rijmen V. The Block Cipher Square / J. Daemen та ін. — 1997. — URL: https://www.researchgate.net/publication/2646816 (дата звернення: 21.05.2025).
14. National Institute of Standards and Technology. FIPS PUB 197 — Advanced Encryption Standard (AES). — 2001. — URL: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197-upd1.pdf (дата звернення: 21.05.2025).
15. James Nechvatal; Elaine Barker; Lawrence Bassham; William Burr; Morris Dworkin; James Foti; Edward Roback (October 2000), Report on the Development of the Advanced Encryption Standard (AES), National Institute of Standards and Technology (NIST).

16. Сокирук В., Лужецький В. Блоковий симетричний шифр на основі модульної арифметики / В. Сокирук, В. Лужецький. — Вінниця, 2010.
17. Лужецький В. А., Дмитришин О. В. Використання операції множення за модулем в симетричних блокових шифрах / В. А. Лужецький, О. В. Дмитришин. — Вінниця, 2010. — URL: https://ir.lib.vntu.edu.ua/bitstream/handle/123456789/14575/soi_2010_5_4.pdf (дата звернення: 21.05.2025).
18. Rothaus O. S. On 'Bent' functions / O. S. Rothaus // Journal of Combinatorial Theory, Series A. — 1976. — Vol. 20, № 3. — P. 300–305.
19. Henning F. Harmuth. Sequency Theory (Foundations and Applications) In the series “Advances in Electronics and Electron Physics” (L.Marton, editor), Supplement 9. Academic Press, New York 1977.
20. Lu Y., Desmedt Y. Bias analysis of a certain problem with applications to E0 and Shannon cipher / Y. Lu, Y. Desmedt // ICISC 2010. — 2011. — LNCS vol. 6829. — P. 16–28. — Springer-Verlag.
21. Lu Y., Desmedt Y. Walsh-Hadamard Transform and Cryptographic Applications in Bias Computing / Y. Lu, Y. Desmedt. — 2016. — URL: https://eprint.iacr.org/2016/419.pdf (дата звернення: 21.05.2025).
22. Python Software Foundation. Python Documentation / Python Software Foundation. — 2.0. — URL: https://docs.python.org/2/index.html (дата звернення: 10.06.2025).
23. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт зі спеціальності 125 «Кібербезпека» освітньо-професійні програми «Безпека інформаційних і комунікаційних систем» та «Кібербезпека критичних систем» / Уклад. В. А. Каплун, О. П. Войтович. Вінниця: ВНТУ, 2023. 61с.
24. Лужецький В., Кесарчук Є. Метод блокового шифрування на основі перетворень Уолша // Інформаційні технології та комп’ютерне моделювання: матеріали міжнар. наук.-практ. конф. (Івано-Франківськ, 20–23 трав. 2025 р.). – Івано-Франківськ: п. Голіней О.М., 2025. – С. 247–250

Додаток А

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Засіб для блокового шифрування на основі перетворення Уолша

Автор роботи: Кесарчук Євгеній Юрійович

Тип роботи: бакалаврська кваліфікаційна робота

Підрозділ кафедра захисту інформації ФІТКІ, група 1 БС-21б

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism **4,97** %

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:

Завідувач кафедри ЗІ д. т. н., проф. _____ Володимир ЛУЖЕЦЬКИЙ

Гарант освітньої програми «Безпека інформаційних та комунікаційних систем» к. т. н., доц. _____ Леонід КУПЕРШТЕЙН

Особа, відповідальна за перевірку _____ Валентина КАПЛУН

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Володимир ЛУЖЕЦЬКИЙ

Здобувач _____ Євгеній Кесарчук

ІЛЮСТРАТИВНА ЧАСТИНА

ЗАСІБ ДЛЯ БЛОКОВОГО ШИФРУВАННЯ НА ОСНОВІ ПЕРЕТВОРЕННЯ УОЛША

Виконав: студент 4 курсу групи 1БС-21б
спеціальності 125 Кібербезпека

_____ Євгеній КЕСАРЧУК

Керівник бакалаврської кваліфікаційної роботи

_____ Володимир ЛУЖЕЦЬКИЙ

«16» червня 2025 р.

ПІДХОДИ ЩОДО ПОБУДОВИ БЛОКОВИХ ШИФРІВ

- мережі Фейстеля,
- SP-мережі,
- структура типу «квадрат»,
- на основі операцій за модулем.

Узагальнена схема процесу зашифрування

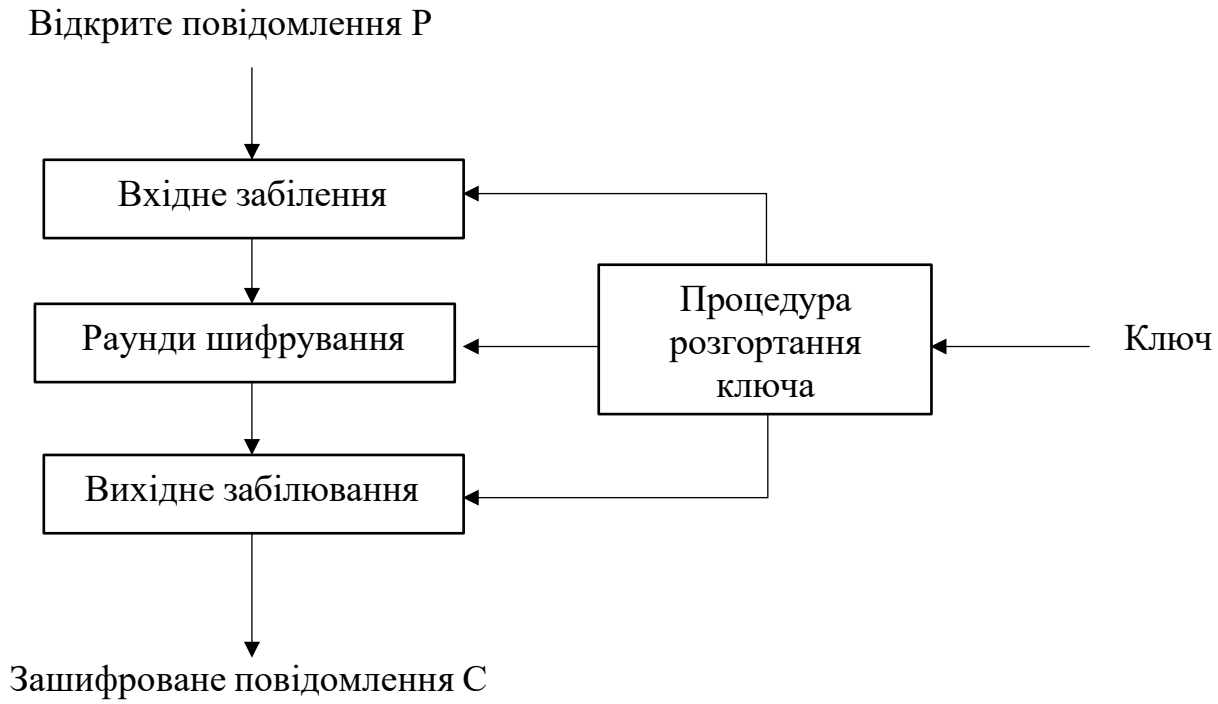
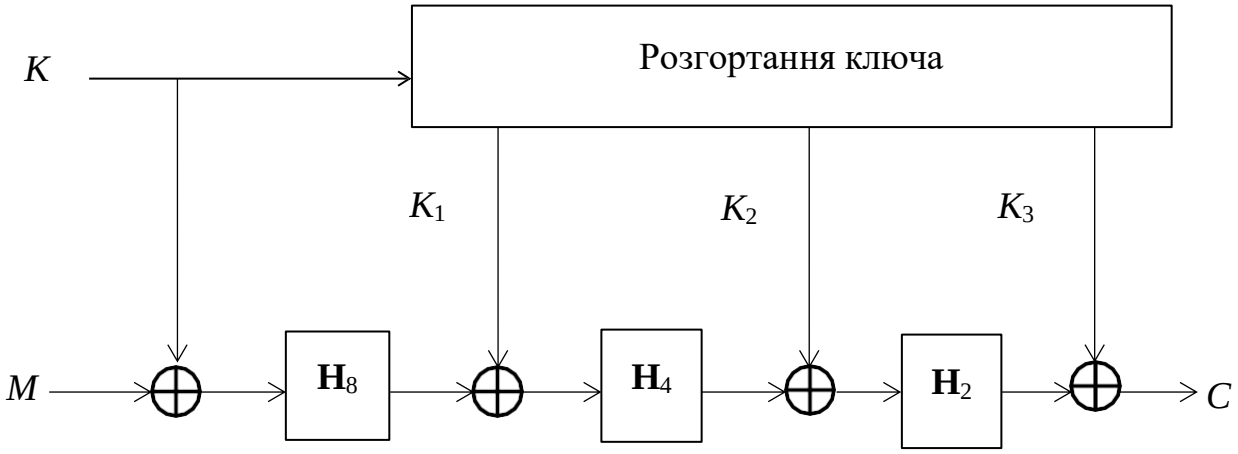
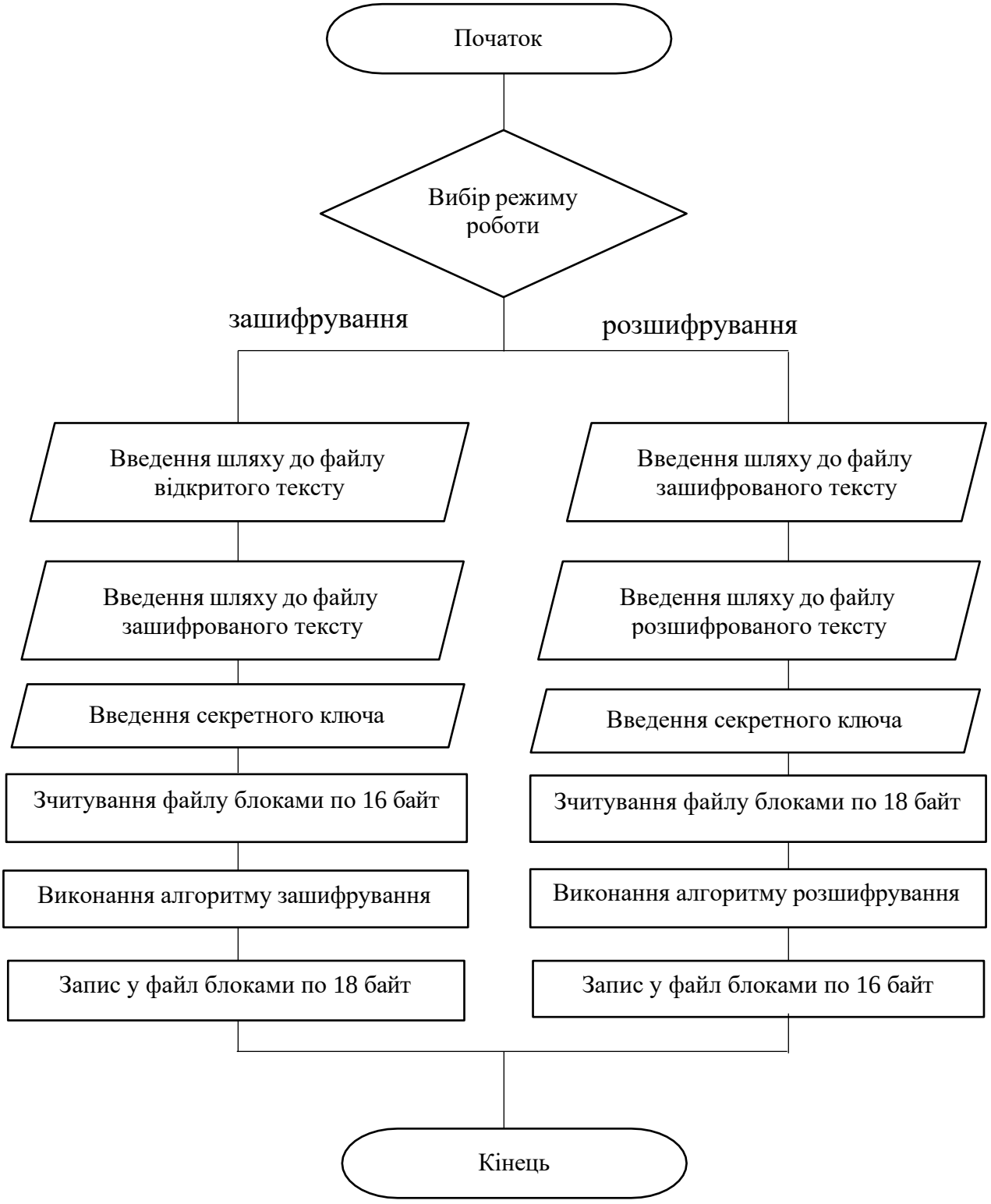


СХЕМА ПРОЦЕСУ ЗАШИФРУВАННЯ



АЛГОРИТМ РОБОТИ ПРОГРАМНОГО ЗАСОБУ ДЛЯ ШИФРУВАННЯ ДАНИХ НА ОСНОВІ ШВИДКОГО ПЕРЕТВОРЕННЯ УОЛША



АЛГОРИТМ ЗАШИФРУВАННЯ НА ОСНОВІ ШВИДКОГО ПЕРЕТВОРЕННЯ УОЛША

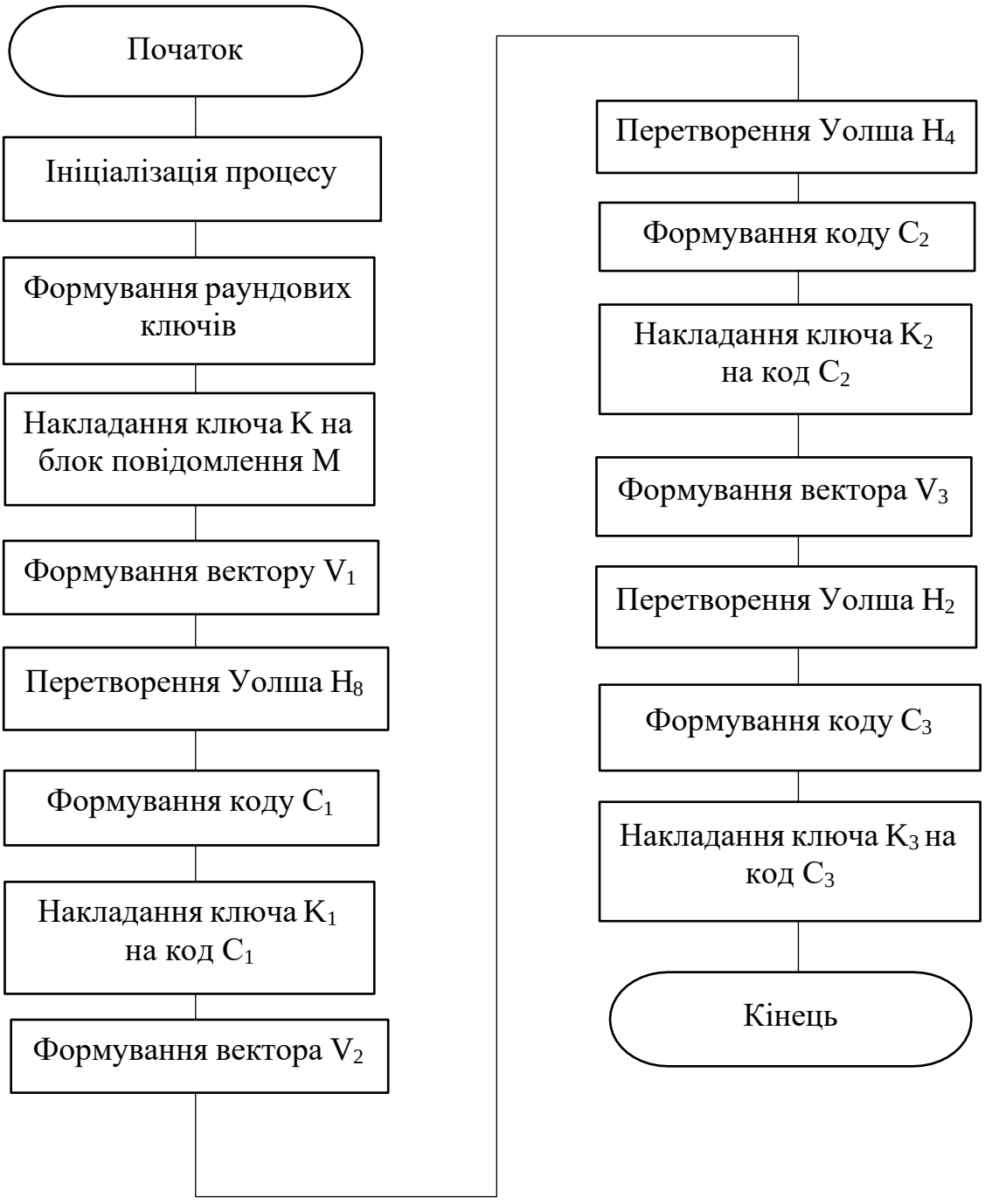
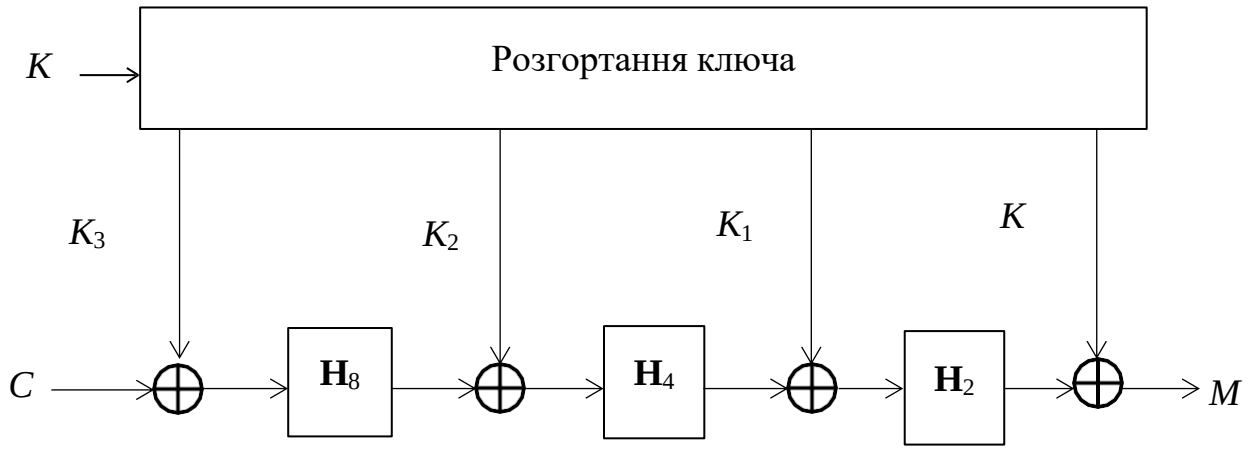
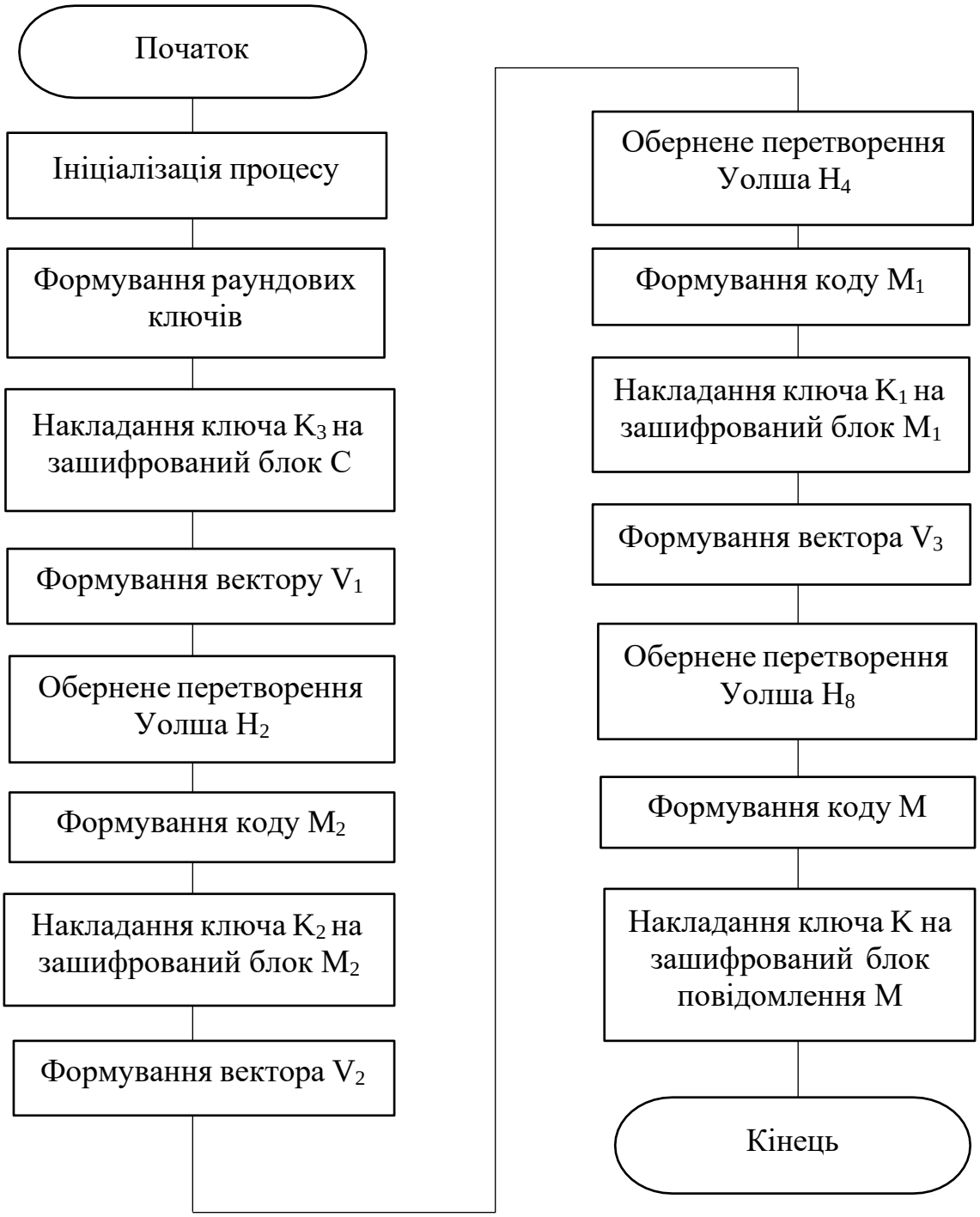


СХЕМА ПРОЦЕСУ РОЗШИФРУВАННЯ



АЛГОРИТМ РОЗШИФРУВАННЯ НА ОСНОВІ ШВИДКОГО ПЕРЕТВОРЕННЯ УОЛША



МЕТОД РОЗГОРТАННЯ СЕКРЕТНОГО КЛЮЧА

$$K_1 = K \oplus \overset{\leftarrow 8}{K}; \quad K_2 = K_1 \oplus \overset{\leftarrow 4}{K_1}; \; K_3 = K_2 \oplus \overset{\leftarrow 2}{K_2}.$$

Тут $\leftarrow n$ означає зсув коду на *n* розрядів уліво.

ОЦІНКА АЛГОРИТМІЧНОЇ СКЛАДНІСТІ ПРОЦЕСУ ШИФРУВАННЯ БЛОКУ ДАНИХ

$$S_E = S_{\oplus} + S_{W2} + S_{W4} + S_{W8} + S_K.$$

$S_{\oplus}$- кількість операцій додавання за модулем 2 ($S_{\oplus}$ = 4);

S_{W2} − складніть перетворення Уолша розмірності 2 ($S_{W2} = 2$);

S_{W4} − складніть перетворення Уолша розмірності 4 ($S_{W4} = 8$);

S_{W8}- складніть перетворення Уолша розмірності 8 ($S_{W8} = 24$);

S_K - складність розгортання ключа ($S_K = 17$).

Умовною одиницею складності є арифметична операція за модулем $2^n + 1$.

Алгоритмічна складність процесу зашифрування блоку даних довжиною 128 біт $S_E = 55$ ум. од.. Тобто для зашифрування одного байту - 3,4 операцій.

Шифр AES - 420 операцій для блоку 128 біт або 26,25 операцій для одного байту.