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 шифрування, що керується даними »

Виконав: студент 2 курсу групи 1БС-21мс
спеціальності 125 Кібербезпека
_____ Ольховий М. В.
Керівник: завідувач каф З.І. д. т. н., професор
_____ Лужецький В. А.
«19» червня 2023 р.
Рецензент: к. т. н. доцент каф. ПЗ
_____ Хошаба О.М.
«19» червня 2023 р.

Допущено до захисту
Завідувач кафедри ЗІ
д. т. н., проф.
_____ Лужецький В. А.
«19» червня 2023 р.

Вінниця ВНТУ – 2023 року

інницький національний технічний університет
акультет інформаційних технологій та комп'ютерної інженерії
афедра захисту інформації
івень вищої освіти І (бакалаврський)
алузь знань – 12 Інформаційні технології
пеціальність – 125 Кібербезпека
світньо-професійна програма – Безпека інформаційних і комунікаційних систем

ЗАТВЕРДЖУЮ

Завідувач кафедри ЗІ,
д. т. н., проф.
_____ В. А. Лужецький
«20» березня 2023 року

ЗАВДАННЯ НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Ольховому Михайлу Вікторовичу

1. Тема роботи: «Засіб для шифрування, що керується даними»
керівник роботи: Лужецький В. А., завідувач кафедри ЗІ д. т. н., професор,
затверджені наказом ректора ВНТУ від 20 березня 2023 року №437.
2. Строк подання студентом роботи 16 червня 2023 р.

3. Вихідні дані до роботи:
Розрядність блоку даних - 8

Розрядність секретного ключа - 64

Набір операцій для шифрування – визначається вмістом даних

4. Зміст пояснювальної записки: Аналіз підходів до побудови блокових шифрів. Розробка методу шифрування. Розробка програмного засобу для шифрування. Висновки. Перелік використаних джерел. Додатки.
5. Перелік ілюстративного матеріалу: Підходи до побудови блокових шифрів (плакат, А4). Метод шифрування(плакат, А4). Алгоритм шифрування (плакат, А4). Структура програмного засобу для шифрування(плакат, А4). Результати тестування (плакат, А4). Порівняльні оцінки алгоритмів шифрування(плакат, А4)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	Лужецький В. А., д.т.н., проф., зав кафедри ЗІ	20.03	19.06
2	Лужецький В. А., д.т.н., проф., зав кафедри ЗІ	20.03	19.06
3	Лужецький В. А., д.т.н., проф., зав кафедри ЗІ	20.03	19.06

7. Дата видачі завдання: 20 березня 2023 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів Бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітки
1	Аналіз завдання. Вступ	20.03.23-24.03.23	
2	Аналіз літературних джерел за напрямком бакалаврської дипломної роботи	27.03.23-07.04.23	
3	Розробка рішень, моделей, алгоритмів	10.04.23-21.04.23	
4	Практична реалізація, моделювання, експериментування, результати	24.04.23-19.05.23	
5	Аналіз результатів виконання завдання	22.05.23-24.05.23	
6	Оформлення пояснювальної записки, підготовка ілюстративного матеріалу	25.05.23-31.05.23	
7	Попередній захист БДР	01.06.23-11.06.23	
8	Виправлення зауважень, перевірка БДР на плагіат	12.06.23-15.06.23	
9	Представлення БДР до захисту, рецензування	16.06.23-19.06.23	
10	Захист БДР	20.06.23-23.06.23	

Студент _____ Ольховий М

Керівник роботи _____ Лужецький В

АНОТАЦІЯ

Бакалаврська дипломна робота складається з 58 сторінок формату А4, на яких є 18 рисунків та 2 таблиці, список використаних джерел містить 19 найменувань.

Бакалаврська робота присвячена розробці програмного засобу для шифрування, що керується даними. В результаті аналізу існуючих методів шифрування обрано симетричний блоковий шифр, що керується даними. У загальному, такий шифр передбачає гнучкий та адаптивний підхід до шифрування, що дозволяє забезпечити безпеку обміну інформацією та зберегти конфіденційність даних. Після розробки методу шифрування та розшифрування в цілому і алгоритмів його окремих складових здійснено програмну реалізацію за допомогою новітніх технологій програмування.

Ключові слова: зашифрування, розшифрування, метод шифрування, регістр зсуву, секретний ключ, блок даних, мережі Фейстеля.

ABSTRACT

The bachelor's thesis consists of 58 A4 pages, which include 18 figures, 2 tables. The list of references contains 19 items. Baccalaureate work is dedicated to the development of a software tool for data-driven encryption.

After analyzing existing encryption methods, a symmetric block cipher controlled by data was chosen. In general, such a cipher provides a flexible and adaptive approach to encryption, ensuring secure information exchange and data confidentiality. Developed encryption and decryption methods, as well as algorithms of their individual components, have been implemented using modern programming technologies.

Keywords: encryption, decryption, encryption method, shift register, secret key, data block, Feistel network.

ЗМІСТ

Вступ

Сучасний світ зі своїм постійним розвитком технологій та зростанням обсягу цифрової інформації ставить перед нами виклик забезпечення безпеки та конфіденційності обміну даними. У цьому контексті шифрування виявляється одним з найефективніших інструментів для захисту інформації від несанкціонованого доступу та зловживань. Однак, існуючі методи шифрування мають свої обмеження та недоліки, особливо в умовах швидкого зростання обсягу даних та появи нових викликів у сфері кібербезпеки.

У сучасному цифровому світі, де обмін та зберігання даних є невід'ємною частиною нашого повсякденного життя, безпека конфіденційної інформації є надзвичайно важливою. Класичні методи шифрування, такі як AES (Advanced Encryption Standard) або RSA (Rivest-Shamir-Adleman), забезпечують деякі гарантії безпеки, але вони часто не враховують контекст та характеристики даних, з якими працюють. Це може призводити до неефективного використання ресурсів та обмежень у швидкості та адаптивності [1].

Коли обсяги передаваних та зберіганих даних невпинно ростуть, пришвидшення процесу шифрування стає надзвичайно актуальним.

В цьому контексті розробка програмного засобу шифрування, що керується даними, є актуальною та доцільною. Такий підхід передбачає гнучке та адаптивне шифрування, здатне забезпечити безпеку обміну інформацією та збереження конфіденційності даних у сучасному цифровому середовищі. Цей підхід дає можливість враховувати специфіку оброблюваних даних та динамічно змінювати методи шифрування залежно від контексту.

Мета роботи – пришвидшення процесу шифрування даних, за рахунок розробки методу шифрування, що керується даними та засобу який його реалізує.

Для досягнення цієї мети потрібно розвязати наступні задачі:

1. Аналіз існуючих методів шифрування та їх обмежень.
2. Розробка методу шифрування, що керується даними.
3. Розробка програмних засобів для шифрування.
4. Тестування розробленого програмного засобу.

За допомогою аналізу існуючих поширених методів симетричного блокового шифрування можна виявити їх переваги, недоліки і вибрати оптимальний алгоритм шифрування для конкретних потреб. Аналіз існуючих методів дозволяє ідентифікувати потенційні проблеми та недоліки, які можуть бути вирішені в новому методі. Цей аналіз може сприяти інноваціям та розвитку нових технологій шифрування, які краще задовольняють сучасні вимоги безпеки та ефективності.

Обєкт дослідження – процес криптографічного захисту інформації.

Предмет дослідження – метод та засіб для шифрування, що керується даними.

Також, не менш важливим завданням виступає документування розробленого програмного засобу, включаючи опис архітектури, функціональних можливостей та інструкцій щодо використання. Це ставить перед собою наступні завдання: опис компонентів, взаємозв'язки та взаємодію між ними. Такий спосіб аналізу дозволить краще зрозуміти структуру програмного засобу та його складові елементи.

Функціональні можливості описують функції та можливості програмного засобу. Це включає перелік основних функцій, які він надає, а також детальніше описує функціональність кожної з них.

Інструкції щодо використання надають користувачеві інформацію про те, як правильно використовувати розроблений програмний засіб. Це включає в себе опис процедури встановлення, запуску та налаштування програмного засобу, а також пояснення використання основних функцій та параметрів.

Документування розробленого програмного засобу є важливим кроком для забезпечення його доступності та використання іншими користувачами або розробниками. Це дозволяє зберегти знання про програмний засіб, спростити процес використання та підтримки, а також забезпечити можливість подальшого розширення та модифікації програмного засобу.

Виконання вищеперелічених завдань дозволить досягти поставленої мети і розробити програмний засіб шифрування, що керується даними, який буде відповідати потребам сучасного цифрового середовища та забезпечувати надійний захист інформації, а також дасть змогу зробити висновки про ефективність

розробленого програмного засобу шифрування, що керується даними, та визначити його потенціал для подальшого застосування та розвитку в галузі кібербезпеки.

1 АНАЛІЗ ПІДХОДІВ ДО ПОБУДОВИ БЛОКОВИХ ШИФРІВ

1.1 Поняття шифрування, що керується даними

Поняття шифрування, що керується даними відноситься до підходу до шифрування, де параметри шифрування визначаються на основі даних, що 2шифруються або розшифровуються. Це означає, що процес шифрування та розшифрування контролюється аналізом і характеристиками цих даних.

Основний принцип шифрування, що керується даними полягає в тому, що шифрування адаптується до властивостей та контексту даних, що обробляються. Замість використання статичних параметрів шифрування, таких як фіксований набір послідовних операцій, використовуються динамічні параметри, які визначаються на підставі характеричтиних ознак даних [6].

Відмінність шифрування, що керується даними від традиційних методів шифрування полягає в тому, що контекст та характеристики даних впливають на параметри шифрування. Це включає в себе визначення операції шифрування для блоку даних вхідного повідомлення. Такий підхід дозволяє досягти більшої адаптивності та ефективності шифрування для різних типів даних та сценаріїв використання.

Такий тип шифрування надає кілька переваг та можливостей. Деякі з них включають:

1. Покращена безпека, за допомогою аналізу даних, шифрування може бути налаштоване для кожного блоку вхідних даних повідомлення, що дозволить забезпечити більшу безпеку. Використання динамічних параметрів шифрування ускладнює атаки та ймовірність зламу шифру.

2. Покращена швидкодія, у зв'язку з тим, що метод шифрування не є складним у реалізації, досягається пришвидшення процесу шифрування, в порівнянні з іншими методами блокового симетричного шифрування.

3. Ефективність на великих масштабах — завдяки своїй гнучкості і адаптивності, метод шифрування, що керується даними, є ефективним рішенням для обробки великих обсягів даних.

Для отримання додаткової інформації та актуальних досліджень з data-driven encryption рекомендується звернутися до наукових статей та публікацій з області криптографії, таких як "Data-Driven Cryptography: Making Modern Encryption Context-Aware" авторів Muhammad Rizwan Asghar, Donghoon Chang, та Gene Tsudik. Дана стаття детально розглядає принципи та застосування шифрування, що керується даними, або data-driven encryption у сучасних системах шифрування [1].

1.2 Огляд сучасних методів симетричного шифрування

1.2.1 Симетричне шифрування

Принцип роботи симетричного шифрування використовує один і той же ключ для шифрування та розшифрування даних. Це означає, що якщо ключом зашифрували дані, то тим самим ключем їх можна розшифрувати.

Переваги такого типу шифрування:

1. Таке шифрування є швидким та ефективним для обробки великих обсягів даних.
2. Воно володіє високим рівнем безпеки при використанні достатньо довгих ключів.

Обмеження: Головним обмеженням симетричного шифрування є проблема обміну ключами між відправником і отримувачем без можливості перехоплення третьою стороною. Кожна пара спілкуючихся сторін повинна мати попередньо обмінені спільні ключі [7].

Приклади методів симетричного шифрування:

- шифрування потоку даних з одноразовим чи нескінченним ключем
- на основі генератора псевдовипадкових чисел
- шифри перестановки (permutation, P-блоки)
- моноалфавітні (код Цезаря)
- КАЛИНА (стандарт України ДСТУ 7624:2014)
- B-Crypt (фірма British Telecom, Великобританія)
- FEAL-1 (Fast Enciphering Algoritm, Японія)
- DES (Data Encryption Standard, США)

1.2.2 Вимоги до сучасних шифрів та криптографічна стійкість

Криптографічна стійкість – здатність криптографічних алгоритмів протистояти атакам зловмисника на нього. Атакуючий криптографічний алгоритм застосовує методи криптоаналізу. Стійкими алгоритмами вважаються ті, успішна атака для котрих потребує від зловмисника неймовірних обсягів обчислювальних ресурсів, значний обсяг перехоплених відкритих та зашифрованих повідомлень або ж потребує стільки часу для розкриття даних, що захищена інформація вже втратить свою актуальність. Тобто вартість зламу шифру та отримання доступу до секретної інформації буде перевищувати вартість зашифрованої інформації у значну кількість разів, у результаті чого, злам буде, щонайменше недоцільним [3].

Існують абсолютно стійки до криптоаналізу алгоритми шифрування. Клод Шеннон доказав їх існування, де зазначив вимоги до таких систем:

— ключ повинен генеруватися для кожного повідомлення (тобто використовується лише один раз)

— довжина ключа може дорівнювати або бути більше довжини повідомлення

— ключ є статистично надійним (тобто символи в ключовій послідовності повинні бути випадковими)

— вхідний (відкритий) текст має деяку надлишковість (є критерієм оцінки правильності розшифрування)

—

Стійкість таких систем не залежить від обчислювальних можливостей зловмисника, а практичне застосування обмежена міркуваннями вартості та зручності користування.

Зазвичай використовуються обчислювально стійкі системи. Стійкість таких систем залежить від того, якими обчислювальними можливостями володіє криптоаналітик. Практична стійкість таких систем базується на теорії складності й оцінюється виключно на певний момент часу та послідовно з двох позицій:

— обчислювальна складність повного перебору

— відомі на даний момент вразливості та їх вплив на обчислювальну складність

У кожному конкретному випадку також існують додаткові критерії оцінки стійкості алгоритму.

Оскільки атака методом грубої сили (повний перебір всіх ключів) можлива для всіх типів криптографічних алгоритмів, для нового алгоритму вона може бути єдиною існуючою. Способи її оцінки засновуються на обчислювальної складності, яка потім може бути конвертована у часовий та грошовий еквівалент, і необхідної продуктивності обчислювальних ресурсів. Така оцінка поки є максимальною та мінімальною одночасно. Подальший аналіз алгоритмів з метою пошуку вразливостей (криптоаналіз) визначає оцінки стійкості по відношенню до відомих видів атак (лінійний криптоаналіз, диференціальний аналіз, атака грубою силою та інші) й можуть знизити відому криптостійкість. Алгоритм буде вважатися практично стійким, тільки тоді, коли на поточний момент його застосування всі знайдені слабкості й вразливості не знижують стійкості алгоритму нижче сучасних обчислювальних можливостей техніки. Тобто це означає, що жодна з відомих нам атак фактично не має будь-якого практичного застосування.

Окрім того, довжина секретного ключа не повинна бути менше, ніж мінімально допустиме значення (наприклад 64 та 128), яке зазначене в національних стандартах. На теперішній час, використання методів шифрування є доволі важливим заходом для забезпечення захисту інформації відповідного рівня.

Необхідно бути впевненим у всіх механізмах та алгоритмах захисту інформації в сферах фінансів, для захисту секретних даних, військових відомостях та державних таємниць.

1.2.3 Загальні відомості про блокові шифри

Криптоалгоритм називають ідеально стійким, якщо прочитати зашифрований блок даних можна лише перебравши всі можливі ключі, доти, поки повідомлення не виявиться осмисленим.

Функція стійкого блокового шифру:

$$\mathrm{C} = E_K\ (\mathrm{P})$$

Функція повинна відповідати наступним вимогам:

— функція повинна бути оборотною

— не повинно існувати інших методів прочитання повідомлення P по відомому блоку C, крім як повним перебором ключів K

— не повинно існувати інших методів визначення яким ключем K було зроблене перетворення відомого повідомлення P у повідомлення C, крім як повним перебором ключів

Сенс роботи криптоаналітика полягає у визначенні ключа K, відкритого тексту P або і того, й іншого.

Однак, його може влаштувати і деяка імовірнісна інформація про P: чи є цей відкритий текст оцифрованим звуком, англійським текстом, даними електронних таблиць або ще чим-небудь.

Криптоаналітики використовують природну надмірність мови для зменшення кількості можливих відкритих текстів. Чим більше надлишковість мови, тим легше її криптоаналізувати [4].

З цієї причини багато криптографічних реалізацій перед зашифруванням використовують програми ущільнення для зменшення розміру тексту.

Ущільнення зменшує надмірність повідомлення разом з обсягом роботи, необхідним для його зашифрування і розшифрування.

Криптоаналітики використовують природну надмірність мови для зменшення кількості можливих відкритих текстів. Чим більше надлишковість мови, тим легше її криптоаналізувати. З цієї причини багато криптографічних реалізацій перед зашифруванням використовують програми ущільнення для зменшення розміру тексту. Ущільнення зменшує надмірність повідомлення разом з обсягом роботи, необхідним для його зашифрування і розшифрування.

Двома основними процедурами маскування надмірності відкритого тексту, згідно Шенону, служать плутанина і дифузія.

Плутанина — маскує зв'язок між відкритим текстом і шифротекстом. Вона ускладнює спроби знайти в шифротексті надмірність і статистичні закономірності. Найпростішим шляхом створити плутанину є підстановка (substitution) .

Дифузія — розсіює надмірність відкритого тексту, поширюючи її на весь шифротекст. Криптоаналітику потрібно чимало часу для пошуку надмірності. Найпростішим способом створити дифузію є перестановка (permutation).

Плутанину і дифузію саму по собі можна зламати, тому при побудові блокових шифрів застосовують і плутанину, і дифузію (підстановки та перестановки) одночасно.

1.2.4 Перспективи розвитку блокових симетричних шифрів

Основним напрямком розвитку блочних симетричних шифрів, безумовно, є збільшення стійкості алгоритмів до різних типів атак. Цього можна досягнути шляхом ускладнення алгоритму застосуванням ємних, з обчислюваної точки зору, операцій. Але такий метод не представляється можливим, бо одним з головних критеріїв до симетричних шифрів є прозорість та простота алгоритму, що досягається при використання операцій, виконання яких відбувається за один такт.

Вважається, що в майбутніх розробках та пропозиціях конструкції перспективних шифрів будуть засновуватися на використанні вже перевіреної часом технології багатоциклового (багаторазового) повторення комбінації лінійного та нелінійного перетворень [5]. Конкуренцію сучасним шифрам можуть скласти композиції перетворень, які мають більш високі криптографічні показники та, зокрема, композиції, які забезпечують (при збереженні високої швидкодії) досягнення багаторазовими шифрувальними перетвореннями властивостей випадкової підстановки при меншому числі циклів зашифрування.

На даний момент часу еталоном серед БСШ можна вважати федеральний стандарт США, тобто AES, тому можна вважати, що майбутнє стоїть за шифрами, які зможуть довести свою надійність при меншому числі раундів зашифрування, ніж у AES (10 раундів для 128-бітного ключа, 12 раундів для 192-бітного ключа, 14 раундів для 256-бітного ключа).

Після всього сказаного вище можна зробити висновок, що симетричні методи шифрування зберігають свою придатність та переваги для використання як в найближчий час, так й в далекій перспективі. Навіть у постквантовий період блокові симетричні шифри (наприклад, AES) будуть зберігати запас стійкості, але це відноситься лише для тих шифрів, які мають розмір ключа 256 біт та більше [3].

Питання їх подальшого вивчення та вдосконалення становлять одне з найважливіших і найактуальніших напрямків для розвитку у сучасній криптографії.

1.3 Механізми керування даними для шифрування

Механізми керування даними для шифрування є важливою складовою шифрування, що керується даними. У цьому пункті розглянемо декілька різних підходів та механізмів, які дозволяють керувати шифруванням на основі даних:

1. Визначення ключів шифрування на основі даних, один з підходів полягає у визначенні ключів шифрування на основі властивостей даних, що шифруються. Наприклад, можна використовувати характеристики даних, такі як тип, розмір, зміст або структура, для генерації ключів шифрування. Це дозволяє створити унікальні ключі для кожного набору даних та підвищити безпеку шифрування.

2. Адаптивні параметри алгоритмів шифрування, деякі алгоритми шифрування дозволяють налаштовувати параметри шифрування на основі властивостей даних. Наприклад, розмір блоків шифрування або кількість ітерацій можуть бути залежними від розміру або складності даних. Це дозволяє оптимізувати процес шифрування для конкретного типу даних і підвищити його ефективність.

3. Керування параметрами шифрування на основі контексту, контекстні дані, які оточують процес шифрування, такі як метадані, інформація про користувача, мережеві умови тощо, можуть використовуватись для керування параметрами шифрування. Наприклад, можна змінювати рівень шифрування або використовувати різні алгоритми шифрування в залежності від поточного контексту. Це дозволяє адаптувати шифрування до змінюваних умов та забезпечити більшу гнучкість.

4. Аналіз даних для виявлення шаблонів, використання методів аналізу даних може допомогти виявити шаблони, тенденції або особливості в даних, які можуть бути використані для керування шифруванням. Наприклад, можна виявити певні типи даних, які потребують більш сильного шифрування, і налаштувати

параметри шифрування відповідно. Це дозволяє забезпечити більшу гранулярність управління шифруванням.

5. Машинне навчання для керування шифруванням, використання методів машинного навчання може допомогти автоматизувати процес визначення параметрів шифрування на основі аналізу великої кількості даних. Моделі машинного навчання можуть виявляти залежності між характеристиками даних та оптимальними параметрами шифрування. Це дозволяє розробляти автономні системи шифрування, які самостійно пристосовуються до змінних умов та забезпечують оптимальний рівень безпеки.

Для отримання додаткової інформації та актуальних досліджень з механізмів керування даними для шифрування рекомендується звернутися до наукових статей та публікацій з області криптографії та безпеки даних.

Наприклад "Data-Driven Encryption: From Secure Channel to Secure Storage". У цій статті розглянуто поняття шифрування, що керується даними та використання контекстуальної інформації [8].

Ці наукові статті та публікації можуть бути використані як початковий пункт для подальшого дослідження та детального вивчення механізмів керування даними для шифрування. Для доступу до повного тексту цих статей рекомендується використовувати наукові бази даних, такі як IEEE Xplore, ACM Digital Library або Google Scholar.

1.4 Підходи до побудови блокових шифрів

Список базових підходів до побудови симетричних блокових шифрів:

- мережі Фейстеля
- на основі SP-мереж
- структура типу «Квадрат»
- на основі арифметичних операцій за модулем

1.4.1 Мережі Фейстеля

Мережі Фейстеля є одним з найпоширеніших криптографічних конструкцій, що використовуються для шифрування та дешифрування даних. Ця конструкція названа на честь Хорста Фейстеля, який вперше її описав.

Мережі Фейстеля використовують блочне шифрування (рис 1.1), де повідомлення поділяється на блоки фіксованої довжини, наприклад, 64 або 128 біт. Кожен блок обробляється у декілька раундів, і кожен раунд включає операції перестановки та заміни даних.

Основні кроки, що виконуються в мережах Фейстеля, наступні:

1. Початковий блок даних розбивається на дві половини.
2. Права половина стає лівою половиною наступного раунду.
3. Ліва половина обробляється шляхом застосування функції, яка приймає на вхід праву половину та ключ раунду.
4. Результат функції XOR-ується з лівою половиною.
5. Ліва та права половини обмінюються місцями для наступного раунду.
6. Кроки 3-5 повторюються декілька разів (зазвичай від 8 до 16 разів) — це називається раундами.
7. На останньому раунді ліва та права половини не обмінюються місцями, результат обробки використовується для отримання зашифрованого або розшифрованого блоку даних.

Ключ раунду використовується для зміни функції шифрування на кожному кроці, що додає безпеку до процесу. Ключ раунду може бути похідним від основного ключа шифрування за допомогою алгоритму генерації ключів.

Важливою властивістю мереж Фейстеля є те, що вони є інверсними - тобто для процесу шифрування та дешифрування використовується одна і та ж сама структура. Це робить їх ефективними та легкими у реалізації [9].

Мережі Фейстеля широко використовуються у багатьох криптографічних алгоритмах, анприклад таких як Triple DES (Data Encryption Standard) та AES (Advanced Encryption Standard), і забезпечують надійний рівень безпеки для шифрування та захисту даних.

Однією з головних переваг мереж Фейстеля є їхній здатність до глибокої та ефективної обробки даних. Вони дозволяють використовувати прості операції, такі як підстановка та перестановка, для створення складних перетворень даних. Крім того, мережі Фейстеля можуть бути ефективно реалізовані апаратно або програмно, що дозволяє їх широке використання у різних системах та пристроях.

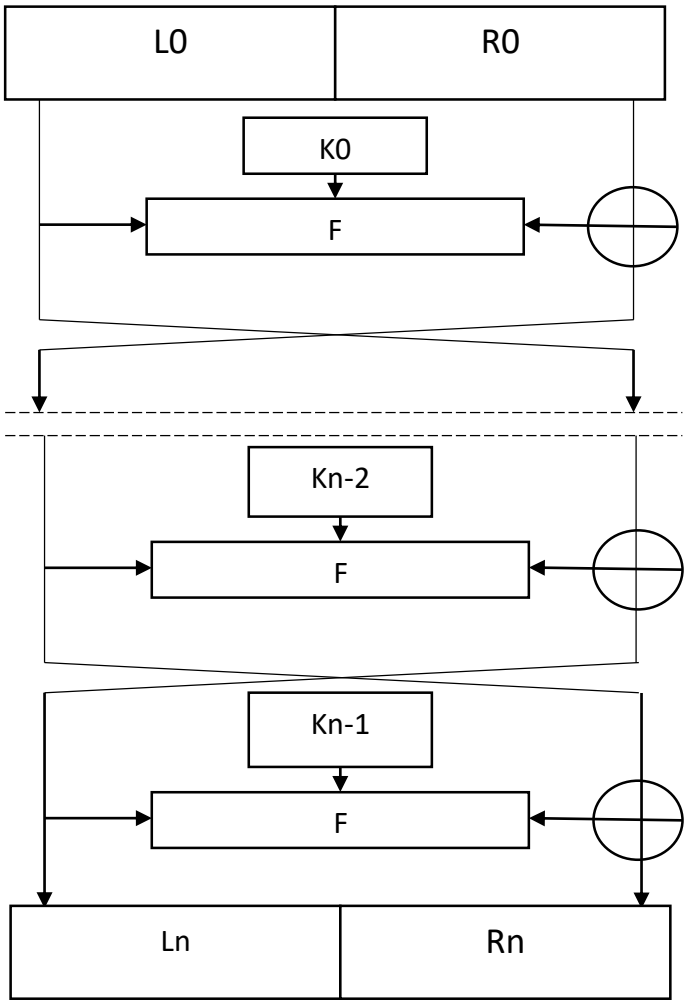


Рисунок 1.1 — Мережа Фейстеля з двума гілками

Мережа Фейстеля симетрична. Використання операції XOR, оборотної своїм повтором, і інверсія останнього обміну гілок уможливлюють розкодування блоку тієї ж мережею Фейштеля, але з інверсним порядком параметрів.

Для оборотності мережі Фейстеля не має значення чи є число раундів парним чи непарним числом. У більшості реалізацій схеми, в яких обидва вищеперелічені умови (операція XOR і знищення останнього обміну) збережені, пряме і зворотне перетворення виробляються однією і тією ж процедурою, якою параметра передається вектор величин або у вихідному, або в інверсному порядку [10].

З незначними доопрацюваннями мережу Фейштеля можна зробити і абсолютно симетричною, тобто виконує функції шифрування та дешифрування одним і тим же набором операцій. Математичною мовою це записується як "Функція EnCrypt тотожно дорівнює функції DeCrypt".

Модифіковані мережі Фейстеля (Modified Feistel Networks): Цей підхід є розширенням мереж Фейстеля, включаючи додаткові операції, такі як заміни і перестановки, в кожному раунді. Це дозволяє розширити можливості шифрування і забезпечити більшу стійкість. Наприклад, шифр DESX є прикладом блокового шифру, побудованого на основі модифікованих мереж Фейстеля.

У класичній мережі Фейстеля кожен раунд включає функцію Фейстеля, яка обробляє одну половину блоку даних і змішує результат з іншою половиною перед наступним раундом. В модифікованих мережах Фейстеля до функції Фейстеля додаються додаткові операції, такі як заміни і перестановки, які роблять шифр більш стійким і надійним [11].

Модифікація мережі Фейстеля, що має подібними властивостями, наведена на рис. 1.2. Як бачимо, основна її хитрість у повторному використанні даних ключа у зворотному порядку в другій половині циклу. Необхідно зауважити, однак, що саме через цю недостатньо досліджену специфіку такої схеми (тобто потенційної можливості ослаблення зашифрованого тексту зворотними перетвореннями) її використовують у криптоалгоритмах з великою обережністю.

Модифікована мережа Фейстеля дозволяє більшу гнучкість у виборі розміру блоку даних, розміру ключа та кількості раундів. Це дає можливість пристосовувати шифрування до конкретних потреб додатків, а також може бути більш стійкою до атак криптоаналітиків.

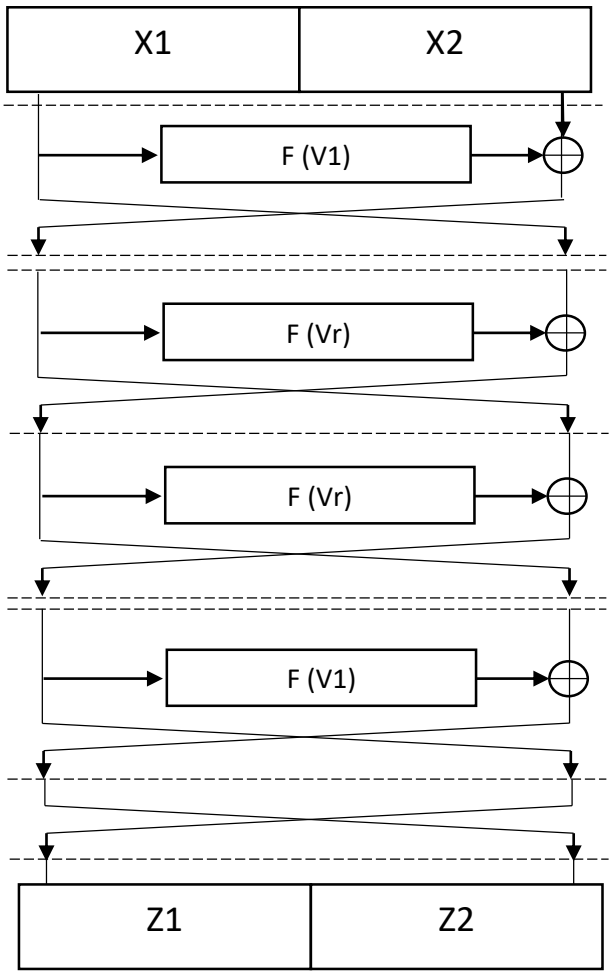


Рисунок 1.2 — Модифікація мережі Фейстеля

1.4.2 Блокові шифри на основі SP-мереж

SP-мережа (Substitution-Permutation network, підстановно-перестановна мережа) – один із підходів щодо побудови блокового шифру.

Шифри на основі SP-мереж - це тип блокових криптографічних алгоритмів, які використовують комбінацію заміни (substitution) та перестановки (permutation) для шифрування даних [12].

У SP-мережах блок даних розбивається на певний розмір блоків, для прикладу, на 64 або 128 бітів, та проходить через деяку серію раундів. Кожен раунд включає в себе дві основні операції: заміну (substitution) та перестановку (permutation) (рис 1.3).

Операція заміни (Substitution): В цій операції застосовуються заміни (substitution), де вхідні біти або байти замінюються на інші значення за певним правилом. Зазвичай для цього використовуються таблиці заміни, які називаються

S-блоками (Substitution boxes) або замінювальними таблицями. S-блоки забезпечують необхідну нелинійність в процесі шифрування.

Операція перестановки (Permutation): В цій операції застосовуються перестановки (permutation), де біти або байти переставляються в певному порядку. Це дозволяє перемішати дані для додаткової конфузії та забезпечення дифузії у процесі шифрування.

Комбінація операцій заміни та перестановки у раундах SP-мережі забезпечує криптографічну стійкість та безпеку шифру. Ці шифри використовуються в багатьох сучасних криптографічних протоколах та системах, таких як AES (Advanced Encryption Standard) і DES (Data Encryption Standard) [13].

Важливими аспектами шифрів на основі SP-мереж безпідставно є їх дифузія (а саме розповсюдження впливу зміни вхідних даних на весь результат шифрування) та конфузія (складність зв'язку між вхідними даними та шифрованим результатом). Шифри SP-мережі добре підходять для реалізації шифрування блоків даних та забезпечують високий рівень безпеки і ефективності.

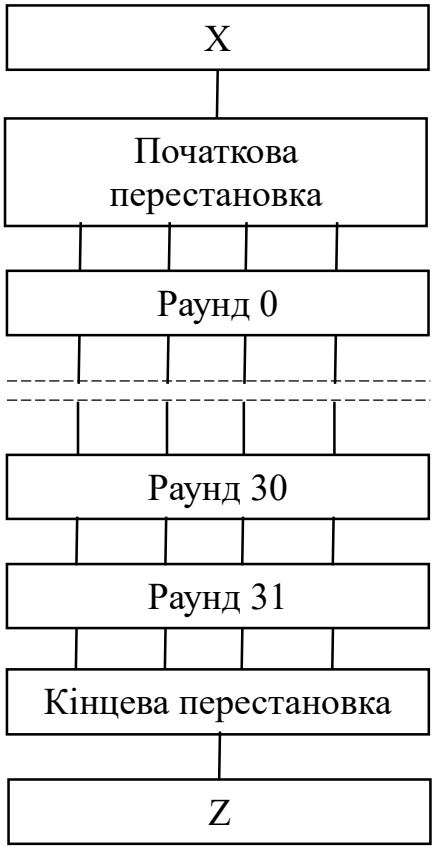


Рисунок 1.3 — Приклад шифру на основі SP-мережі

1.4.3 Блокові шифри зі структурою типу «Квадрат»

Блокові шифри зі структурою типу квадрат — це клас блокових шифрів, що мають структуру подібну квадрату перетворення даних. Вони отримали свою назву через аналогію зі структурою квадрату, де кожен блок даних обробляється незалежно від інших блоків [14].

У шифрах зі структурою типу квадрат зазвичай використовуються дві основні операції: заміна (substitution) та перестановка (permutation). Заміна включає заміну значень байтів або бітів згідно з певними правилами, наприклад, за допомогою таблиці заміни (S-блоків). Перестановка включає переміщення байтів або бітів у певному порядку для додаткового розподілу та змішування даних.

Ці шифри пропонують гнучкий підхід до обробки блоків даних та можуть мати різні конфігурації та розміри блоків. Наприклад, можуть використовуватись квадратні блоки (наприклад, 4x4 або 8x8), але також можуть бути використані і інші форми блоків.

Приклади блокових шифрів зі структурою типу квадрат включають Shark (рис 1.4), Present, Camellia та інші. Кожен з цих шифрів має свою власну структуру, але загальна ідея полягає в тому, що дані розбиваються на блоки, які потім проходять через послідовні етапи заміни та перестановки.

Shark (Serpent Hashing Algorithm for Really Kool) є блоковим шифром, який був розроблений в 1997 році. Цей шифр має блочну структуру і використовує 64-бітні блоки даних та 128-бітний ключ. Shark входить до сімейства шифрів широкого використання і використовується у різних застосуваннях, включаючи захист інформації, криптографічні протоколи та системи безпеки.

Основні особливості методу шифрування Shark включають конфузію та дифузія. Shark використовує швидкі і необоротні операції для розповсюдження інформації по блоку даних, забезпечуючи ефективну зміну шифрованого тексту при навіть невеликих змінах вхідних даних або ключа.

Shark є одним із багатьох шифрів, що використовуються в сучасних криптографічних системах для забезпечення захисту конфіденційності інформації.

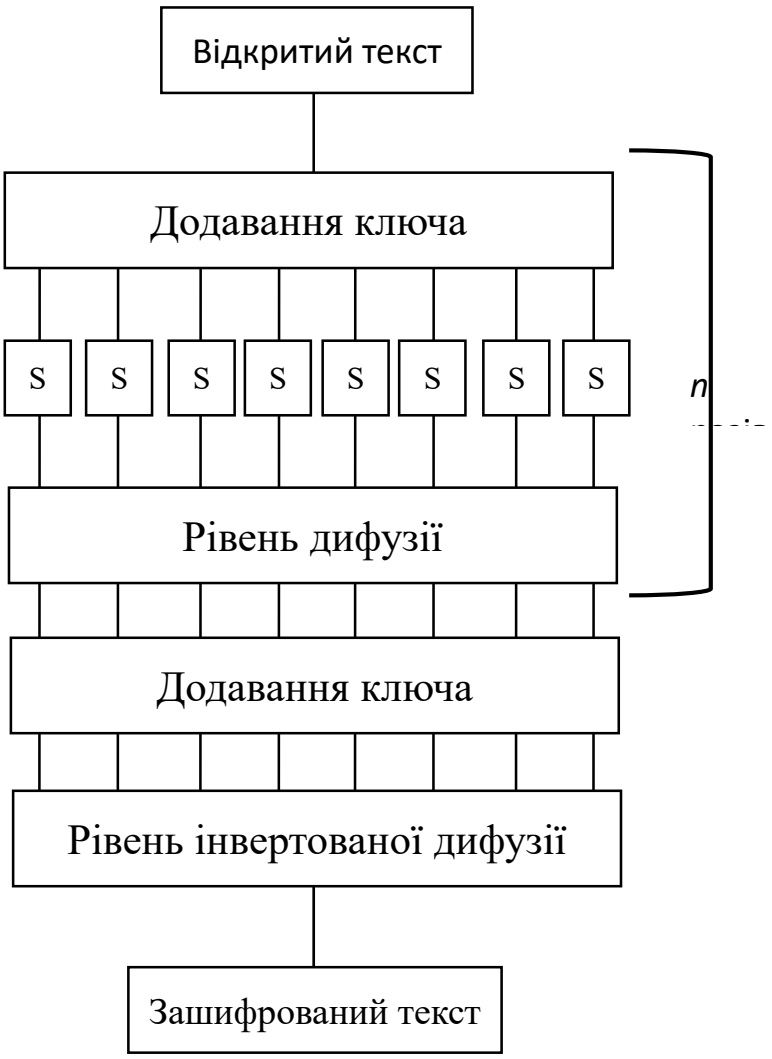


Рисунок 1.4 — Приклад шифру SHARK на основі структури «Квадрат»

1.4.4 Недоліки підходів блокового шифрування

У традиційних блокових симетричних шифрах, таких як AES, Triple DES, та інших, послідовність операцій шифрування чітко визначена і відома. Ця послідовність включає підстановку, перестановку та інші математичні операції, які застосовуються до блоків даних.

Головним недоліком таких підходів, у порівнянні із методом шифрування, що керується даними, є те, що весь процес шифрування має фіксовану послідовність операцій, яка не залежить від самої інформації, що шифрується. Це означає, що навіть якщо дані мають певну структуру або особливості, ці особливості можуть бути помітні у зашифрованому тексті. Знання про фіксовану послідовність операцій також може допомогти зловмисникам при проведенні атак.

У методі шифрування, що керується даними, принцип виконання шифрування інший. Послідовність операцій не є фіксованою, а визначається на основі даних байту ключа. Ключ виступає в ролі керуючого фактора, який визначає, які операції будуть виконуватись. А саме, за відповідальність визначення операції шифрування відповідає характеристична означа ключа.

Це означає, що кожен блок даних може бути оброблений унікальним способом, залежно від значень ключа. Такий підхід дозволяє отримати випадкову послідовність операцій шифрування, що робить аналіз шифрованого тексту складнішим для зловмисників. На відміну від традиційних блокових шифрів, метод шифрування, що керується даними, може забезпечити більшу варіативність та непередбачуваність шифрування, а також меншу кількість дій, в процесі шифрування у порівнянні з такими розповсюдженими методами шифрування як AES.

2 РОЗРОБКА МЕТОДУ ШИФРУВАННЯ

2.1. Метод зашифрування

Розробка методу шифрування включає в себе побудову схеми методу шифрування, побудову схеми методу розшифрування та детальний опис алгоритму шифрування і розшифрування відповідно. Метод шифрування, що керується даними базується на вхідних даних, що можуть мати вигляд файлу, тексту, тощо.

Окрім побудови схем та алгоритму шифрування, є ряд інших важливих аспектів, що необхідно визначити на початку розроби методу шифрування:

1. Вибір криптографічних примітивів: блоковий шифр
2. Розмір ключа: 64 біта
3. Розмір блоку вхідних даних повідомлення: 8 байт

Основний підхід шифрування, що керується даними, полягає у використанні характеристик самого повідомлення або даних для визначення ключа шифрування або самого процесу шифрування. У цьому підході параметри шифрування адаптуються або залежать від вмісту даних, що шифруються, забезпечуючи більшу гнучкість та безпеку [15].

Основна ідея полягає у використанні характеристик повідомлення або даних для визначення послідовності виконування операцій шифрування. Наприклад, можуть бути використані конкретні атрибути повідомлення, такі як його довжина, структура, частота вживання певних символів або будь-які інші властивості, що можуть бути унікальні для даного набору даних.

Шифрування, що керується даними, може забезпечити додатковий рівень безпеки та конфіденційності, оскільки ключі шифрування та параметри алгоритмів можуть бути змінені для кожного повідомлення або набору даних. Це ускладнює завдання зламу шифру та забезпечує більшу стійкість до атак [16].

Підхід шифрування, що керується даними, дозволяє використати гнучкі та динамічні методи шифрування, що адаптуються відповідно вмісту та характеристик повідомлень або даних, що шифруються. Такий підхід використовують в різних криптографічних протоколах та системах, що забезпечують високий рівень безпеки і захисту інформації.

На прикладі алгоритму зображеного на рис. 2.1 даними, що керують шифруванням виступатимуть біти блоку ключа, що буде отримуватись із регістра зсуву і мати значення . Таким чином шифрування буде кероване даними, що приходять у вигляді байту даних секретного ключа g_i. Кількість бітів зі значенням «1» у такому байті буде напряму впливати на вибір операції шифрування, що буде використано для блоку вхідних даних m_i.

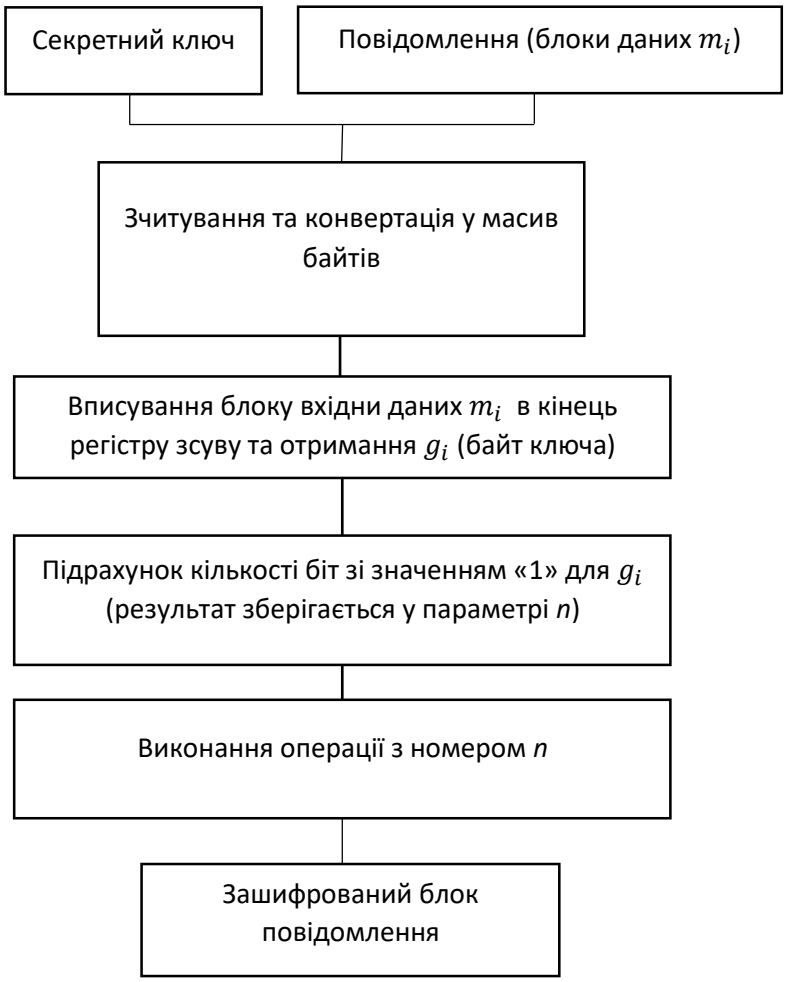


Рисунок 2.1 — Метод зашифрування, що керується даними

На вході у блок-схеми (рис 2.1) маємо вхідні дані (повідомлення, яке необхідно зашифрувати), m_i — блок вхідних даних розміром 8 біт (або 1 байт), k — секретний ключ, g_i — байт даних секретного ключа, m_i — блок вхідних даних повідомлення, що необхідно зашифрувати.

Регістр зсуву відповідає за зсув ключа, на блок даних m_i. блок даних заміщує блок секретного ключа, таким чином ітеративно визначається блок секретного ключа, що направляється далі до блоку керування.

Блок операцій відповідає за визначення 9 логічних, арифметичних побітових операцій, що будуть використовуватись у процесі шифрування.

Останній блок керування несе відповідальність за визначення операції шифрування, на основі аналізу ключа.

Покроковий алгоритм роботи методу:

1. Формування секретного ключа.
2. Формування 9 операцій шифрування (один раз).
3. Визначення блоку вхідних даних (ітеративно).
4. Перша операція, що відбувається у регістрі зсуву, блок даних заміщує блок секретного ключа, таким чином ітеративно визначається блок секретного ключа, що направляється далі до блоку керування.
5. Блок керування, отримує на вході блок даних ключа та проводить аналіз, зокрема підраховує кількість бітів зі значенням «1», яка в подальшому визначає номер операції шифрування, що буде використана для блоку даних ключа та блоку вхідних даних.
6. Блок операцій один раз на початку алгоритму визначив 9 операцій шифрування та отримує на вхід одночасно блок вхідних даних, блок даних ключа та номер операції шифрування. Також він безпосередньо відповідає за проведення операції шифрування над блоком вхідних даних за допомогою блоку даних ключа та операції між цими двома блоками.
7. Нарешті після проходження першої ітерації на виході отримуємо перший блок зашифрованого тексту.
8. Алгоритм повторюється ітеративно, аж допоки у регістрі зсуву не відбудеться зсув останнього байту секретного ключа.
9. Алгоритм завершує дію, на виході ми отримали 8 блоків зашифрованого повідомлення після одного раунду шифрування, отримані блоки складаються та в результаті маємо зашифроване повідомлення методом шифрування, що керується даними.

Для прикладу взято задовільні дані розміром 8 байт. Першим кроком алгоритму буде зчитування байт вхідного повідомлення та ключа, а також

розділення вхідного повідомлення на $m_i = m_0, m_1, m_2, \dots m_8$, секретного ключа на $g_i = g_0, g_1, g_2, \dots g_8$.

Вхідні дані:

1. Повідомлення з розрядністю блоку даних 8 — «message1»
2. Секретний ключ розрядністю 64 біта — «secret12»

Припустивши, що один символ рівнозначний одному байту даних (використовуючи різні мови програмування можна отримати різний розмір повідомлень стосовно величини у байтах), на початку алгоритму вхідне повідомлення розбивається на масив байтів, отже кожен із символів є блоком вхідних даних $m_i =$ «m», «e», «s», «s», «a», «g», «e», «1» та рівний 1 байту. Аналогічна операція відбувається із ключем — секретний ключ розбивається побайтово, маємо $g_i =$ «s», «e», «c», «r», «e», «t», «1», «2», кожен з яких рівнозначний 1 байту (рис 2.2).

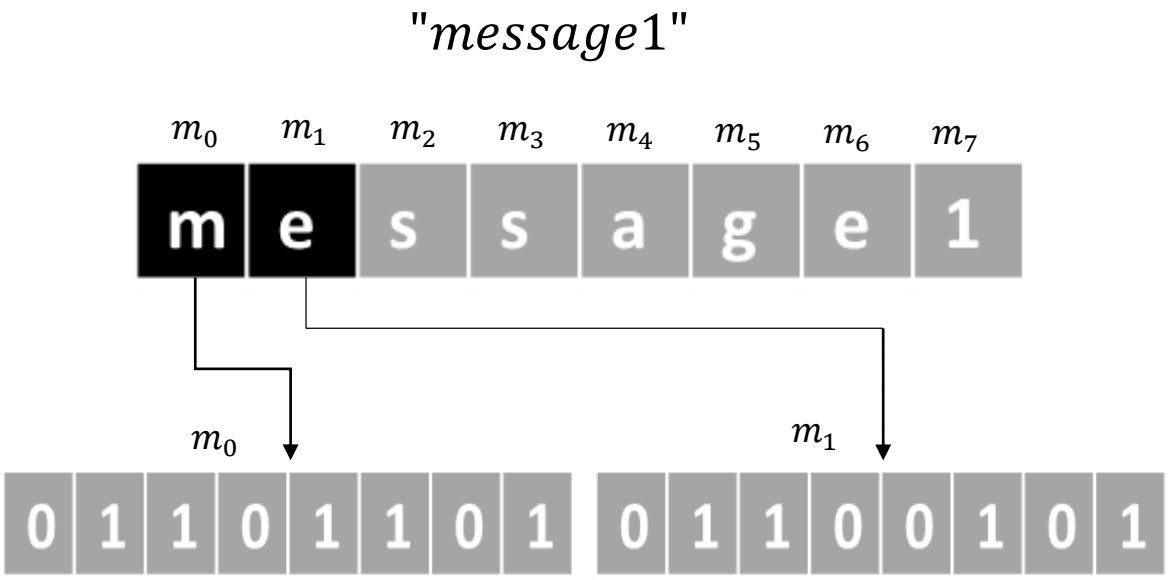


Рисунок 2.2 — Розбиття вхідних даних на блоки

Регістр зсуву є одним із базових компонентів методу шифрування, що керується даними. Він використовується для створення ключів шифрування або для здійснення перестановок бітів у процесі шифрування [17].

Регістр зсуву — це регістр, який містить фіксований набір бітів і виконує циклічне зсування (перестановку) цих бітів на одну або кілька позицій вліво або вправо при кожному такті. Зсування може відбуватись поодинокими бітами або групами бітів, залежно від конфігурації регістра.

У контексті шифрування, регістр зсуву використовується для створення послідовності ключів або для перестановки бітів вхідного повідомлення перед шифруванням. Основні етапи роботи регістра зсуву в методі шифрування, що керується даними, будуть наступні:

1. Ініціалізація: регістр зсуву ініціалізується початковим значенням (секретним ключем шифрування).

2. Зсування: регістр зсуву виконує зсування свого вмісту на одну позицію вправо. Зсування може не циклічне, так як байти ключа не просто зсовуються вправо, а ще й заміняються на m_i, в результаті повного відпрацювання алгоритму замість ключа залишиться повне вхідне повідомлення.

3. Формування ключа: результат зсування регістра зсуву використовується для формування байту g_i ключа шифрування. Цей блок ключа буде використаний для виконання операції шифрування, наприклад, побітового XOR з вхідним повідомленням.

Після розбиття, ключ потрапляє в регістр, а за ним туди починає надходити параметр зі значенням m_i, в регістрі відбувається зсув першого байту ключа $g_i = g_0 =$ «s», за допомогою $m_i = m_0 =$ «m», завдяки чому ми отримуємо 2 параметри g_i та m_i, що передаються далі, відповідно до схеми з рис. 2.1.

Завдяки потраплянню в регістр зсуву блоку даних m_i, увесь ключ зсувається на одну позицію вліво, а останню позицію займає блок вхідних даних m_i, завдяки чому відбувається витіснення g_i, таким чином і формується байт ключа, який далі передається у наступний блок керування, де відбувається аналіз даного байту.

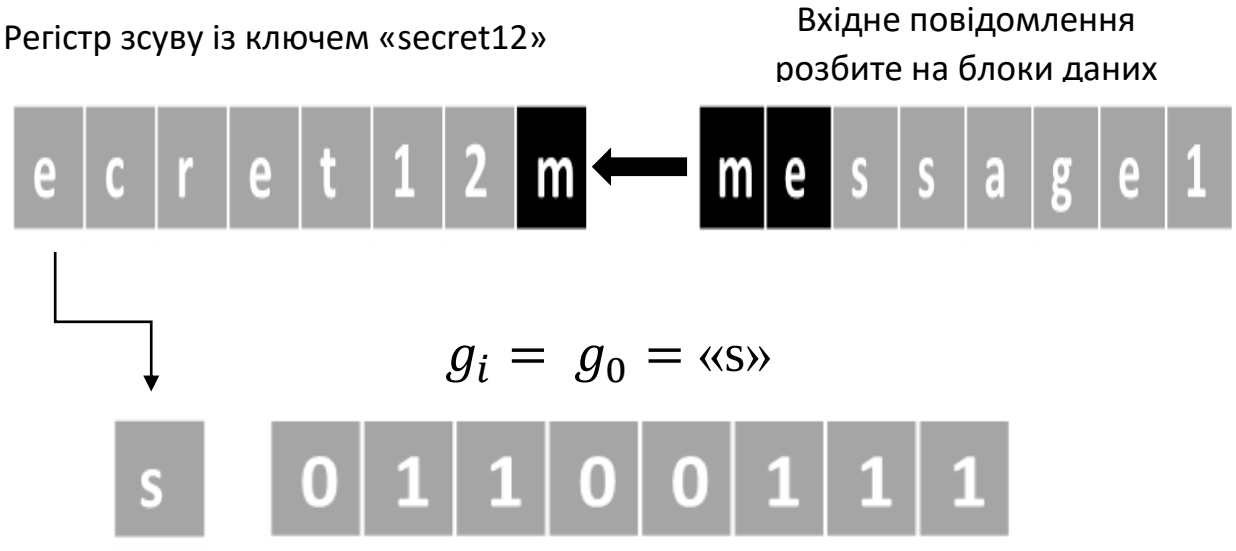


Рисунок 2.3 — Формування g_i

Метою модуля керування є підрахування аналіз байту ключа g_i та передача параметру із результатом у наступний модуль операцій (рис 2.3).

У методі шифрування, що керується даними, підрахунок позитивних бітів у байті ключа відіграє важливу роль у визначенні операції шифрування. Цей підрахунок виконується за допомогою модуля керування.

Перш за все, байт ключа g_i розглядається як послідовність бітів. Кожен біт у цьому байті може мати значення 0 або 1. Аналіз байту ключа полягає у визначенні кількості бітів, які мають значення 1.

Даних підрахунок може бути здійнено шляхом проходження по кожному біту байту ключа g_i та підрахунку кількості позитивних бітів. Наприклад, можна використати цикл, що перевіряє кожен біт і, якщо він має значення «1», збільшується лічильник бітів, цикл повторюється поки не закінчаться біти у g_i.

Після виконання підрахунку кількості бітів зі значенням «1», отримане значення використовуватимесь для вибору відповідної операції шифрування. Наприклад, в залежності від отриманого результату, можна використовувати різні шифрувальні алгоритми або режими роботи, які підходять для даної кількості бітів.

Таким чином, підрахунок бітів зі значенням «1» у байті ключа грає роль у визначенні операції шифрування в методі шифрування, що керується даними. Він

дозволяє адаптувати процес шифрування до вмісту даних і забезпечує гнучкість та безпеку у виборі відповідного алгоритму шифрування.

На даному етапі $g_i = g_0 =$ «s», перетворивши дані g_i у байткод отримаємо наступі значення у цифровому вигляді — 103, та у побітовому вигляді — 01100111 (рис 2.4).

$$g_i = g_0 = \text{«s»}$$

$$n = 5$$

Рисунок 2.4 — Представлення байту ключа

Підрахувавши кількість бітів зі значенням «1» у першому байті секретного ключа $g_i = g_0 =$ «s», зі значенням 01100111, отриманого у результаті першого зсуву, отримали n = 5. Даний параметр буде передаватись як третій параметр в останній блок операцій, де буде виконана п’ята за списком операція шифрування до першого вхідного блоку даних $m_i = m_0 =$ «m».

Після успішного виконання попередніх операцій залишається останній крок до отримання першого зашифрованого блоку даних C_i, а саме виконання операції шифрування.

При першому раунді проходження алгоритму формується статичний список відповідної кількості операцій, що повинні бути обортними та будуть використаними у процесі шифрування. А саме необхідно визначити 9 операцій для усіх можливих випадків значення n. Проаналізувавши вхідні параметри методу шифрування, що керується даними, маємо g_i розміром 1 байт. В одному байті знаходиться 8 бітових позицій, отже n може набувати значення від 1 до 8, але необхідно не забувати про випадок, коли можемо мати 0 бітів зі значенням «1», а це ще один варіант значення змінної n.

Отже, у висновку маємо чітко визначений перелік значень, що може приймати змінна n, усього — 9 варіантів. На основі вище проаналізованих даних

тепер чітко зрозуміло у якій кількості логічних, математичних, операцій є необхідність — 9 операцій (рис 2.5).

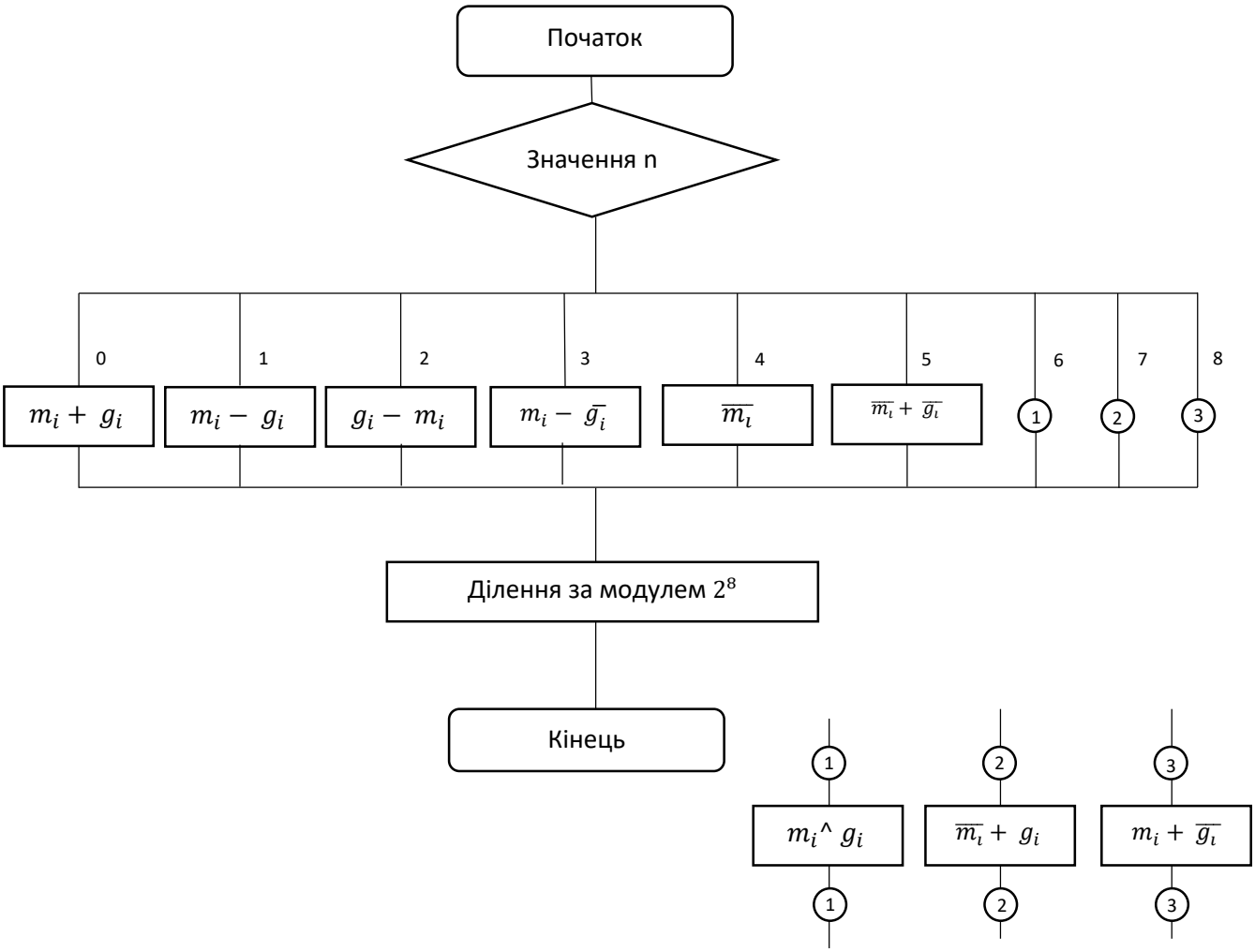


Рисунок 2.5 — Алгоритм роботи блоку операцій при шифруванні

Наступні 9 визначених операцій (з порядком розташування від 0 до 8) будуть використовуватись упродовж алгоритму шифрування (як зашифрування так і розшифрування, оскільки нам необхідно буде знати в правильній послідовності визначена операції, що використовувались для зашифрування блоків вхідних даних), для вхідного незашифрованого повідомлення зображені на табл. 2.1 із назвою операцій, формулою та характеристичною ознакою n, що містить кількість біт зі значенням «1» у поточному байті ключа g_i

Таблиця 2.1 – Операції зашифрування

Характеристична ознака	Назва	Формула

0	Додавання за модулем	$(m_i + g_i)\ mod\ 256$
1	Віднімання за модулем	$(m_i - g_i)\ mod\ 256$
2	Віднімання за модулем	$(g_i - m_i)\ mod\ 256$
3	Віднімання з інверсним значенням	$(m_i - \overline{g_i})\ mod\ 256$
4	Інверсія	$\overline{m_i}\ mod\ 256$
5	Додавання за модулем з інверсними значеннями	$(\overline{m_i} + \overline{g_i})\ mod\ 256$
6	Додавання за модулем 2	$(m_i \hat{}\ g_i)\ mod\ 256$
7	Додавання за модулем з інверсним значенням	$(\overline{m_i} + g_i)\ mod\ 256$
8	Додавання за модулем з інверсним значенням	$(m_i + \overline{g_i})\ mod\ 256$

Визначивши усі операції при першому проходженні циклю шифрування, маючи значення параметру n = 5, $g_i =$ «s» $=$ 01100111 та $m_i =$ «m» $=$ 01101101 виконаємо п’яту операцію зі списку згенерованих операцій у модулі, що відповідає за визначення та виконання операцій шифрування.

Додавання за модулем з інверсією: $\overline{m_i} + \overline{g_i}$ за модулем 2^8, де $m_i =$ 01101101 або 103, $g_i =$ 01100111 або 109.

1. Спершу знаходиться інвертне значення для вхідного блоку даних $\overline{m_i}$ = 10010010.

2. Наступним кроком, по аналогії, відбувається інверсія вхідного байту ключа $\overline{g_i}$ = 10011000.

3. Далі звичайна операція додавання отриманих інверсних значень $\overline{m_i} + \overline{g_i}$ = 10010010 + 10011000 = 100101010 або 298

4. Оскільки результат перевищує максимальне значення 255, виконується додаткова операція ділення по модулю 2^8, щоб отримати коректне значення в межах 0-255.

5. Після виконання ділення по модулю отримуємо результат 42, або 00101010 у бітовому представленні.

Після виконання вказаних операцій ми отримали перший зашифрований блок повідомлення. Цей блок був оброблений шляхом використання додавання за модулем з інверсією, де певні біти були інвертовані та пройшли криптографічні обчислення.

У процесі шифрування, ми спочатку виконали операцію інверсії бітів для вихідних символів повідомлення, змінивши значення кожного біта на протилежне. Це створило новий набір символів зі зворотніми значеннями.

Далі, зі зворотніх символів ми обчислили суму двох блоків, використовуючи додавання за модулем 2^8. Ця операція сприяє змішуванню та поширенню властивостей символів, створюючи нові комбінації та перетворюючи блок даних.

На заключному етапі, отриманий результат був приведений до діапазону від 0 до 255, застосовуючи ділення по модулю 2^8. Це забезпечило правильну і валідну інтерпретацію шифрованого блоку як беззнакового числа.

Таким чином, після виконання цих операцій ми отримали перший зашифрований блок повідомлення, який пройшов через процес інверсії, додавання та модульної арифметики, що забезпечує безпеку та надійність обміну інформацією.

В конкретному випадку проходження першого кола алгоритму по методу шифрування, що керується даними ми отримали перший зашифрований блок даних $C_i = C_0 =$ 00101010 (у вигляді байту даних), зашифрувавши перший вхідний блок даних $m_0 =$ «m» за допомогою $g_0 =$ «s» та операції під номером 5 (додавання за модулем з інверсією, $m_0 + \overline{g_0}$ за модулем 2^8).

Далі за аналогічними кроками усі вищепераховані дії повторюються для усіх m_i при i від 0 до 7.

Після цього кроку, описані дії будуть повторюватись для всіх наступних вхідних блоків даних m_i, де i варіюється від 0 до 7. Це означає, що кожен блок даних буде зашифрований з використанням відповідного ключа g_i та відповідної операції шифрування.

Таким чином, застосовуючи аналогічні кроки для кожного блоку даних, ми здійснюємо процес шифрування всього повідомлення, використовуючи метод шифрування, що керується даними [18].

У результаті отримано 8 блоків зашифрованого тексту.

2.2. Метод розшифрування

На рис 2.6 відображено послідовність та перелік дій, що разом складають метод розшифрування, що керується даними.

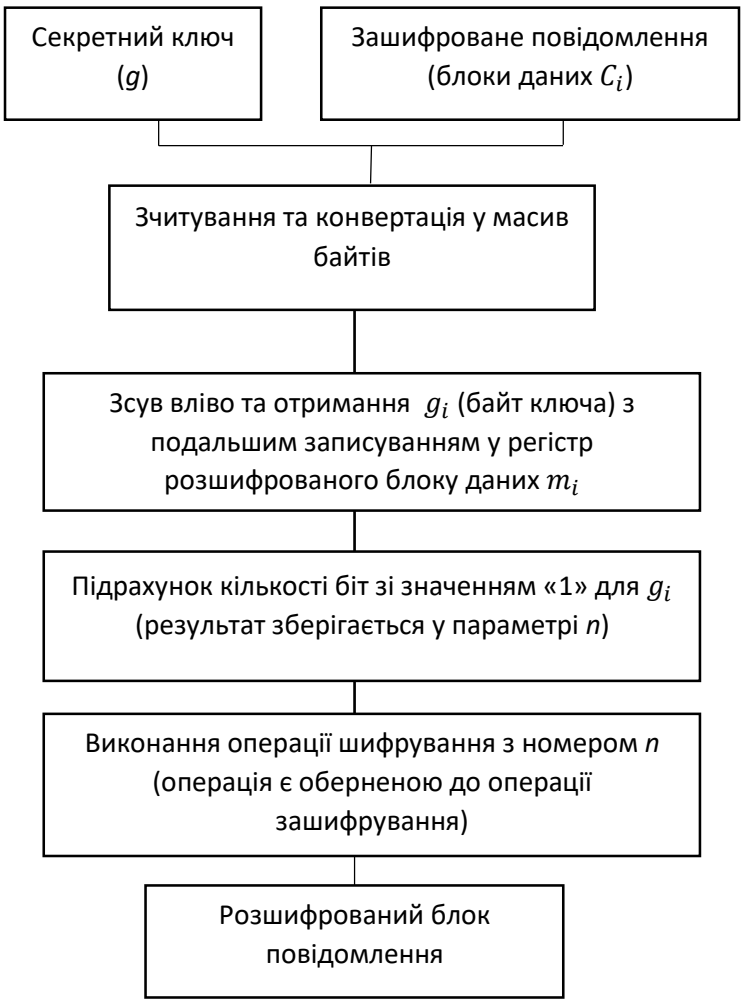


Рисунок 2.6 — Метод розшифрування, що керується даними

На початку схеми (рис 2.6) маємо вхідні дані (повідомлення, яке необхідно розшифрувати), C_i — блок вхідних даних розміром 8 біт (або 1 байт), k — секретний ключ, g_i — байт даних секретного ключа, C_i — блок зашифрованого повідомлення.

Регістр зсуву відповідає за зсув ключа, на 1 байт вліво, таким чином ітеративно визначається блок секретного ключа, що направляється далі до блоку керування.

Блок операцій відповідає за визначення 9 логічних, арифметичних побітових операцій, що будуть використовуватись у процесі шифрування.

Останній блок керування несе відповідальність за визначення операції шифрування, на основі аналізу ключа.

Покроковий алгоритм роботи методу:

1. Отримання секретного ключа.
2. Отримання 9 операцій шифрування та створення обернених операцій, до кожної операції шифрування у визначеному порядку (один раз).
3. Визначення блоку зашифрованих даних (ітеративно).
4. Перша операція, відбувається у регістрі зсуву, за допомогою зсуву вліво на 1 ітеративно визначається блок секретного ключа g_i, що направляється далі до блоку керування, та блоку операцій.
5. Блок керування, отримує на вході блок даних ключа та проводить аналіз, зокрема підрахунок кількості бітів зі значенням «1», яка в подальшому визначає номер операції шифрування, що буде використана для блоку даних ключа та блоку вхідних даних.
6. Блок операцій один раз на початку алгоритму визначив 9 обернених операцій шифрування (таким чином отримали 9 операцій для розшифрування у відповідному порядку) та отримує на вхід одночасно блок вхідних даних, блок даних ключа та номер операції шифрування. Також він безпосередньо відповідає за проведення операції шифрування над блоком вхідних даних за допомогою блоку даних ключа та операції між цими двома блоками.
7. Нарешті після проходження першої ітерації на виході отримуємо перший блок розшифрованого повідомлення.
8. Алгоритм повторюється ітеративно, аж допоки у регістрі зсуву не відбудеться зсув останнього байту секретного ключа.
9. Алгоритм завершує дію, на виході ми отримали 8 блоків розшифрованого повідомлення після одного раунду розшифрування, отримані

блоки складаються та в результаті маємо розшифроване оригінальне повідомлення методом розшифрування, що керується даними.

Для прикладу проведено детально із поясненням кожного кроку операцію розшифрування, попередньо зашифрованого, блоку повідомлення у вигляді байту 00101010.

Першим кроком алгоритму буде отримання секретного ключа та 9 операцій, що були згенеровані та використані у процесі шифрування. В алгоритмі розшифрування обов'язково необхідно мати доступ до секретного ключа та операцій шифрування, які були використані під час шифрування. Ці дані є необхідними для коректного розшифрування зашифрованого повідомлення.

Секретний ключ грає критичну роль у шифруванні та розшифруванні. Він є секретним параметром, який визначає метод шифрування та відтворюється лише власником або особою, яка має право доступу до зашифрованої інформації. Без належного секретного ключа розшифрування стає неможливим, оскільки ключ визначає правильну послідовність операцій та перетворень, які необхідно застосувати до зашифрованих даних [19].

Операції шифрування, які були використані під час процесу шифрування, також є важливими для розшифрування. Кожна операція має свою роль у процесі шифрування та відтворюється у зворотному порядку під час розшифрування. Інформація про операції дозволяє відтворити правильну послідовність перетворень та відновити початковий стан даних.

Збереження операцій та секретного ключа є важливою задачею з точки зору безпеки і забезпечення можливості розшифрування зашифрованої інформації. Без належного збереження цих даних, розшифрування може бути неможливим або привести до неправильних результатів. Тому важливо забезпечити захист і конфіденційність секретного ключа та зберігати операції, необхідні для розшифрування, у безпечному і доступному вигляді.

Таким чином, секретний ключ та операції шифрування відіграють важливу роль у процесі розшифрування, і їх збереження є необхідною передумовою для успішного розшифрування зашифрованої інформації.

Секретний ключ в текстовому форматі: «secret1».

Операції, що були використані для шифрування (зі збереженною нумерацією) :

0. Додавання за модулем: $(m_i + g_i)$ *mod 256*
1. Віднімання за модулем: $(m_i - g_i)$ *mod 256*
2. Віднімання за модулем: $(g_i - m_i)$ *mod 256*
3. Віднімання з інверсією: $(m_i - \overline{g_i})$ *mod 256*
4. Інверсія: $\overline{m_i}$ *mod 256*
5. Додавання за модулем з інверсією: $(\overline{m_i} + \overline{g_i})$ *mod 256*
6. Додавання за модулем 2: $(m_i \wedge g_i)$ *mod 256*
7. Додавання за модулем з інверсією: $(\overline{m_i} + g_i)$ *mod 256*
8. Додавання за модулем з інверсним значенням: $(m_i + \overline{g_i})$ *mod 256*

Наступним кроком в процесі розшифрування є передавання вхідного блоку даних m_i у регістр зсуву, наслідком чого відбувається зсув секретного ключа у регістрі на 1 позицію вліво, таким чином останню позицію в регістрі займає блок вхідних даних m_i витісняючи перший байт даних ключа g_i за межі регістру, таким чином зсування у регістрі зсуву використовується для формування байту g_i ключа шифрування (рис 2.7).

Зсуви у регістрі будуть відбуватись для кожного блоку вхідних даних m_i, поки через регістр зсуву не пройдуть усі блоки повідомлення m_i, в наслідок чого відбувається повне зсування ключа із регістру і наприкінці роботи алгоритму в регістрі зсуву не залишиться жодного байту ключа, а замість секретного ключа регістр буде містити повне вхідне повідомлення із блоків даних m_i.

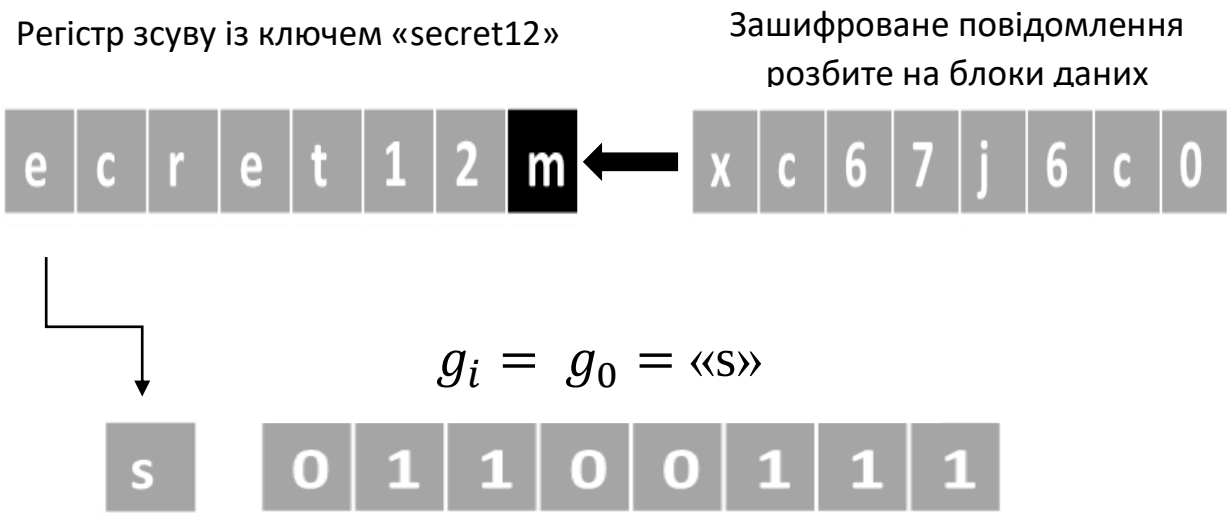


Рисунок 2.7 — Схема роботи регістру зсуву ключа у процесі розшифрування

Після виконання операції у регістрі зсуву отримали перше значення $g_i = g_1 = 01100111$, далі ця інформація передається у модуль керування.

Модуль керування в процесі розшифрування виконує аналогічну роль, як і у процесі шифрування — відповідає за підрахунок бітів зі значенням «1» у ключі, що використовується під час розшифрування.

Кількість таких бітів у ключі використовується для прийняття рішення щодо вибору операції розшифрування, аналогічно до використання цієї інформації у процесі шифрування.

Модуль керування отримує на вхід блок даних ключа та проводить аналіз цього блоку для визначення кількості позитивних бітів. На основі цього аналізу він вибирає відповідну операцію розшифрування, яка буде застосована до блоку зашифрованих даних. Аналогічно до шифрування, правильне функціонування модуля керування в розшифруванні є важливим для досягнення коректного та надійного розшифрування зашифрованої інформації.

Підрахувавши кількість біт зі значенням «1» у першому параметрі $g_i = g_0 =$ «s», зі значенням у вигляді 01100111 отримали наступний результат $n = 5$ (рис. 2.4).

Останньою дією у схемі, що зображено на рис. 2.6 залишається виконання операції розшифрування.

Аналогічно як і в процесі шифрування, в процесі розшифрування на початку алгоритму визначається 9 операцій, але з обо’язковим дотриманням наступних критерій:

1. Операції повинні бути обортиними для початкових, що були застосовані у процесі шифрування
2. Кожна оборотна операція повинна займати позицію в списку операцій відповідно до операції, на основі якої була створена оборотна.

Операція, що застосована під час шифрування, повинна мати відповідну оборотну операцію, яка дозволяє повернутися до початкових даних.

При процесі шифрування застосовуються певні операції, такі як додавання, віднімання, інверсія, XOR тощо, для перетворення блоків даних у зашифровані дані. Щоб розшифрувати такі зашифровані дані та повернути їх до початкового вигляду, необхідно виконати оборотну операцію для кожної використаної під час шифрування операції. Інакше кажучи, оборотна операція відміняє ефект початкової операції, відповідно відміняє ефект шифрування над вхідним блоком даних, дозволяючи відновити початкові дані з зашифрованого вигляду та отримати оригінальне повідомлення.

Наприклад, якщо під час шифрування була використана операція додавання, то оборотною операцією буде віднімання. Якщо під час шифрування використовувалась операція XOR, то оборотною операцією буде знову застосування XOR з тими самими параметрами.

Важливо, щоб кожна операція розшифрування була обортною, тобто виконувала обернену дію відносно початкової операції, щоб розшифрування було можливим. Це дозволить забезпечити відтворюваність і коректність процесу розшифрування, який повинен відновити початкові дані з зашифрованої форми.

Якщо оборотна операція шифрування буде неправильною, то процес розшифрування може бути некоректним або навіть неможливим. Це може призвести до неправильного відновлення початкових даних після шифрування.

Ця операція повинна бути точним оберненням операції шифрування. Якщо оборотна операція неправильно налаштована або має помилки, то наслідками такої проблеми будуть:

1. Втрата цілісності даних
2. Неможливість відновлення початкових даних

Тому важливо правильно реалізувати оборотну операцію шифрування і перевірити її на коректність та відповідність вимогам безпеки. Це допоможе забезпечити правильне розшифрування даних і зберегти їх цілісність.

Оборотні операції для розшифрування зашифрованих блоків даних (від 0 до 8 із збереженням послідовності відповідно до позицій оригінальних операцій шифрування), що будуть використані упродовж процесу розшифрування, для отримання оригінального повідомлення подано у табл 2.2.

Таблиця 2.2 – Операції розшифрування

Характеристична ознака	Назва	Формула
0	Віднімання за модулем	$(C_i + g_i)\ mod\ 256$
1	Додавання за модулем	$(C_i + g_i)\ mod\ 256$
2	Додавання за модулем	$(g_i + C_i)\ mod\ 256$
3	Додавання з інверсним значенням	$(C_i + \overline{g_i})\ mod\ 256$
4	Інверсія	$\overline{C_i}\ mod\ 256$
5	Віднімання за модулем з інверсними значеннями	$\overline{(C_i - \overline{g_i} + 256)}\ mod\ 256$
6	Віднімання за модулем 2	$(C_i \hat{}\ g_i)\ mod\ 256$
7	Віднімання за модулем з інверсним значенням	$\overline{(C_i - g_i + 256)}\ mod\ 256$
8	Віднімання за модулем з інверсним значенням	$(C_i - \overline{g_i})\ mod\ 256$

На рис. 2.8 зображено алгоритм визначення та виконання операції розшифрування, на основі параметру *n*, що містить кількість бітів байту ключа зі значенням «1».

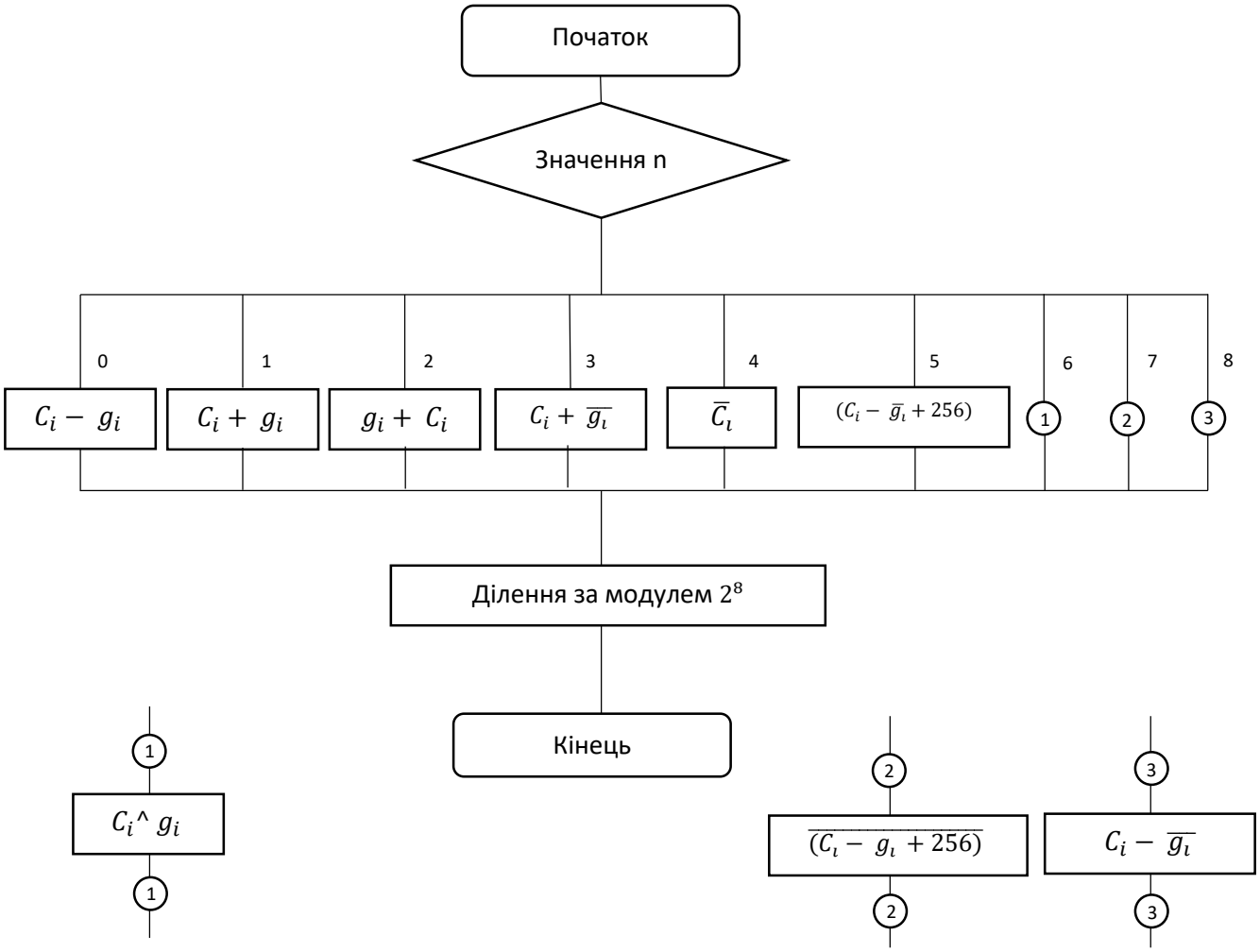


Рисунок 2.8 — Алгоритм визначення операцій при розшифруванні

Взято результат шифрування першого блоку вхідного повідомлення $C_0 =$ 00101010, виконавши операцію підрахунку позитивних бітів $g_0 = 01100111$ отримано результат n = 5. Виходячи з отриманих даних необхідно виконати 5 оборотну операцію розшифрування для отримання оригінального незашифрованого блоку вхідних даних m_0 , а саме, необхідно виконати наступну формулу: віднімання за модулем з інверсією:$\overline{(C_i - \overline{g_i} + 256)}$ за модулем 2^8.

1. Отримуємо зашифрований блок даних, представлений значенням 42 (або 00101010 у бітовому представленні).

2. Виконуємо обернену операцію додавання за модулем з інверсією. Для цього використовуємо інвертне значення блоку зашифрованих даних та інвертне значення ключа.

3. Інвертне значення ключа: $\bar{g}_i = \overline{103} = 152$ (або 10011000 у бітовому представленні).

4. Виконуємо операцію віднімання між отриманими значеннями: $C_i - \bar{g}_i = 42 - 152 = -110$

5. Виконуємо інверсію для отриманого результату віднімання. Отриманий результат 109 (або 01101101 у бітовому представленні).

6. Ділення за модулем не є необхідністю, так як отримане значення входить в межі від 0 до 255.

Після виконання вказаних операцій було отримано перший розшифрований блок повідомлення. Цей блок був оброблений шляхом використання віднімання за модулем з інверсією, де певні біти були інвертовані та пройшли криптографічні обчислення.

Виконавши операцію порівняння оригінального повідомлення $m_0 = \text{"m"} = 109 \text{ або } 01101101$ та отриманого блоку розшифрованого повідомлення $m_0 = 109 \text{ або } 01101101$ робиться висновок, що за допомогою алгоритму розшифрування було вірно розшифровано перший блок вхідних даних після виконання операції шифрування над ним, по методу шифрування, що керується даними.

Для отримання повного розшифрованого повідомлення, необхідно циклічно пройтись по всім елементам ключа та послідовно з'єднати блоки розшифрованого тексту. Детальний процес:

1. Починаємо з першого елемента ключа та першого блоку розшифрованого тексту.

2. Застосовуємо розшифрування до першого блоку розшифрованого тексту за допомогою першого елемента ключа та відповідної оберненої операції шифрування. Отримуємо перший блок оригінального тексту.

3. Переходимо до наступного елемента ключа та наступного блоку розшифрованого тексту.

4. Застосовуємо розшифрування до наступного блоку розшифрованого тексту за допомогою наступного елемента ключа та відповідної оберненої операції шифрування. Отримуємо наступний блок оригінального тексту.

5. Продовжуємо цей процес для всіх елементів ключа та блоків розшифрованого тексту, поки не пройдемо всі елементи ключа та не з'єднаємо всі блоки розшифрованого тексту.

6. Після циклічного проходження по всім елементам ключа та з'єднання всіх блоків розшифрованого тексту, ми отримаємо повне розшифроване повідомлення.

Такий процес дозволяє відновити оригінальний текст з зашифрованого повідомлення за допомогою ключа та обернених операцій шифрування. Кожний елемент ключа використовується для розшифрування відповідного блоку розшифрованого тексту, що забезпечує правильний порядок та повноту розшифрованого повідомлення. Представлений метод шифрування дієвий та має наступні переваги:

— метод забезпечує високий рівень безпеки, оскільки використовує обернені операції та модульну арифметику. Це робить його важким для розшифрування зловмисниками.

— метод може працювати з різними типами даних та ключами, що дозволяє його використання в різних сферах, де необхідне шифрування даних.

— метод шифрування, що керується даними, використовує прості операції, такі як додавання, віднімання та XOR, що робить його швидким та ефективним для використання в реальному часі

— основний принципом є керування шифруванням на основі даних, що мають бути зашифровані. Це дозволяє використовувати різні операції шифрування для різних блоків даних, забезпечуючи більшу безпеку та варіативність.

— кожна операція має свою обернену операцію, що дозволяє розшифрувати зашифровані дані і повернутися до початкових даних. Це забезпечує можливість безпроблемного розшифрування повідомлень

— для дешифрування повідомлення, зашифрованого методом шифрування, що керується даними, необхідно не тільки підібрати ключ, а також підібрати 9 операцій, що ускладнює в рази швидкість виконання дешифрування прирівнює результат успішності зламу шифру до мінімуму

— шифр не вимагає високих технічних чи математичних знань для реалізації, що відповідає принципам Кірхгофа

— шифр використовує мінімум ресурсів для реалізації як процесу шифрування та і дешифрування, що робить його більш зручним у використанні

Хоча метод шифрування, що керується даними, має свої переваги, він також має деякі недоліки:

1. Вразливість до криптоаналітичних атак — застосування фіксованого набору операцій шифрування та ключа може зробити алгоритм вразливим до криптоаналітичних атак, зокрема до атаки з відомим чистим текстом або з відомим шифротекстом.

2. Обмежений набір операцій — метод шифрування, що керується даними, обмежений використанням певного набору операцій, що можуть бути виконані над блоками даних. Це може обмежувати гнучкість та ефективність шифрування в деяких сценаріях.

3 РОЗРОБКА ПРОГРАМНОГО ЗАСОБУ ДЛЯ ШИФРУВАННЯ

3.1 Програмний засіб для зашифрування

При розробці програмного засобу шифрування було використано кілька ключових технологій, що забезпечили ефективну та зручну розробку. Основні технології, які були використані, включають нижче перераховані.

Java 11 — використання Java 11 являється важливим аспектом розробки програмного засобу шифрування. Java 11 забезпечує потужність та надійність для розробки безпечних програмних рішень. Вона пропонує широкі можливості, які полегшують розробку та підтримку програмного засобу шифрування.

IntelliJ IDEA 2023 — IntelliJ IDEA є однією з найпопулярніших інтегрованих середовищ розробки (IDE) для мови Java. Використання IntelliJ IDEA 2023 дозволяє зробити процес розробки більш продуктивним та зручним. Вона надає багато корисних функцій, таких як автодоповнення коду, рефакторинг, налагодження та інструменти для управління проектом.

Криптографічні бібліотеки Java — для реалізації алгоритмів шифрування можна використати готові та розшифрування в програмному засобі були використані криптографічні бібліотеки Java. Ці бібліотеки надають набір функцій та класів для реалізації різноманітних алгоритмів шифрування, генерації ключів та управління криптографічними операціями.

Розробка програмного засобу шифрування, що буде керуватися даними розпочато з визначення модулів, що будуть відповідальними за конкретні кроки в алгоритмі, для цього використано підхід програмування із принципами SOLID.

Принципи SOLID (Single Responsibility, Open-Closed, Liskov Substitution, Interface Segregation, Dependency Inversion) є важливими принципами об'єктно-орієнтованого програмування, які спрямовані на створення гнучких, розширюваних та легко збережених програмних систем.

Один з цих принципів, що використано у програмному застосунку - Single Responsibility Principle (принцип єдиного обов'язку) - стверджує, що кожен клас або модуль повинен мати лише одну причину для зміни. Це означає, що кожен елемент системи має виконувати лише одну функцію або мати одну відповідальність.

Проведемо аналогію між принципом єдиного обов'язку та схемою шифрування, описаною на рис. 2.1. У програмному застосунку структуру коду поділено та реалізовано у вигляді наступних модулів:

Модуль зчитування даних побайтово: його відповідальність полягає у зчитуванні вхідних даних побайтово та передачі їх наступному блоку для обробки. Він виконує одну конкретну функцію і відповідає за неї.

Модуль регістра зсуву: його відповідальність полягає у здійсненні зсуву ключа або блоку даних, що використовується для шифрування. Він виконує окрему функцію, пов'язану з зсувом, і не намагається виконувати інші завдання.

Модуль керування: його відповідальність полягає у визначенні кількості біт зі значенням «1» в ключі. Він аналізує дані та приймає рішення щодо вибору операції шифрування на основі цього аналізу. Це єдине завдання блоку керування.

Модуль операцій: Його відповідальність полягає у генерації операцій шифрування та виконанні відповідної операції над блоком даних. Він не навантажується іншими функціями, крім операцій шифрування.

Кожен з цих блоків виконує лише свою специфічну відповідальність і не перекривається з іншими функціями. Це дозволяє краще організувати код, зробити його більш читабельним, легким у розширенні та підтримці. Крім того, зміна однієї частини схеми не вплине на інші частини, що сприяє гнучкості та модульності програмного засобу шифрування.

Використання принципів SOLID, зокрема принципу єдиного обов'язку, у розробці програмного засобу шифрування допомагає забезпечити чистоту коду, його легку зрозумілість, зручність у розширенні та підтримці. Такий підхід дозволяє знизити залежність між компонентами системи та покращити якість програмного забезпечення загалом.

На рис. 3.1 зображено модульну структуру реалізації програмного засобу шифрування, що керується даними, структура включає в себе 4 основних модулі, котрі відповідають за повну роботу алгоритму шифрування від початку до кінця, влючаючи зчитування, збереження даних, визначення набору статичних операцій для зашифрування та виведення результату.

1. Модуль зчитування та конвертації типу даних

2. Модуль регістру зсуву
3. Модуль керування
4. Модуль операцій

Усі модулі взаємопов'язані один з одним та є важливими компонентами методу шифрування, а сама модульна структура включає в себе частки коду (функції) з програмною реалізацією тієї чи іншої дії.

Рисунок 3.1 — Модулі програмного засобу

Кожен модуль детально розглянуто та пояснено методику його роботи та обсяг роботи, який виконує відповідний модуль.

Модуль зчитування вхідних даних — є важливою частиною програмного засобу шифрування, оскільки він відповідає за зчитування та конвертацію вхідних даних у масив байтів.

Цей модуль може бути більш гнучким і розширеним залежно від ваших потреб. Наприклад, ви можете реалізувати ввод з консолі, де користувач вводить текст з клавіатури, за допомогою класу Scanner або інших засобів читання з консолі. В такому випадку, ви можете використовувати методи з Scanner для зчитування рядка вводу та його подальшої конвертації у масив байтів.

Також, можна передати текстовий файл як вхідні дані у вигляді стрічки. В цьому випадку, можна використати класи Java, такі як FileReader або BufferedReader, для зчитування файлу та отримання його вмісту у вигляді рядка. Після цього, ви можете сконвертувати цей рядок у масив байтів, використовуючи метод getBytes() або інші аналогічні методи.

Метод getBytes() є вбудованим методом у класі String у Java, і використовується для отримання масиву байтів, які представляють символи в рядку.

Робота метода getBytes() полягає в конвертації символів рядка в відповідні байти. При цьому використовується кодування символів, яке залежить від налаштувань системи або вказаного кодування. За замовчуванням, метод getBytes() використовує стандартне кодування системи, тобто використовується кодування, що відповідає налаштуванням операційної системи.

Процес конвертації відбувається наступним чином: кожен символ рядка перетворюється на відповідний байт у масиві байтів. В результаті, отримується масив байтів, де кожен елемент відповідає одному символу у рядку.

Наприклад, якщо ми маємо рядок "Hello", виклик методу getBytes() на цьому рядку поверне масив байтів [72, 101, 108, 108, 111]. У цьому випадку, кожен символ рядка перетворюється відповідно до його ASCII-коду на відповідний байт.

Важливо зазначити, що результатом роботи методу getBytes() є масив байтів, який відображає кодування символів рядка. Залежно від кодування і налаштувань системи, результат може відрізнятися. Також варто враховувати, що певні символи

можуть бути представлені кількома байтами, особливо для мультибайтових кодувань, наприклад, UTF-8.

Отже, метод getBytes() дозволяє отримати байтове представлення рядка в Java, що може бути використане для подальшої обробки, передачі або зберігання даних в байтовому форматі.

Загальний принцип полягає в тому, що вхідні дані можуть бути отримані з різних джерел, таких як консоль або файли, і за допомогою відповідних методів читання і конвертації їх можна перетворити у масив байтів. Ваш код лише надає просту реалізацію зчитування рядка як вхідних даних та його конвертацію у байти.

Отже модуль зчитує інформацію, та конвертує її у масив байт за допомогою влаштованих у Java бібліотек із можливостями конвертації одного типу змінних у інші, а саме за допомогою методу getBytes().

Модуль регістра зсуву — виконує зсув блоку даних або ключа у шифрувальному алгоритмі. Цей модуль відповідає за переміщення бітів у масиві на певну кількість позицій.

У реалізації модуля регістра зсуву використано цикл. Цикл працює на основі вхідних даних, які представлені у вигляді масиву байтів. В залежності від потреби, зсув може бути виконаний вправо або вліво.

Зсув імітується перебором усіх елементів масиву байтів (bytes[]) у Java, за допомогою чого отримується відповідний індекс, по якому дістається необхідний елемент ключа із масиву байтів, а також блок вхідних даних.

Цикл проходить по усьому масиву, поки алгоритм не завершить свою роботу, що означає проходження по усім елементам масиву, або по 8 блокам по 1 байту (64 біта), тоді цикл та відповідно модуль регістру зсуву завершують свою роботу.

Основна мета регістра зсуву полягає в забезпеченні переміщення даних для подальшого шифрування. Він виконується перед подальшими етапами шифрування та впливає на результат шифрування.

У цілому, модуль регістра зсуву виконує важливу функцію у процесі шифрування, забезпечуючи зсув блоку даних або ключа. Це дає можливість забезпечити варіативність шифрування та підвищити його безпеку.

Модуль керування — використовує метод countOnes(), який приймає ціле число (байт) як аргумент та виконує побітовий аналіз ключа. Кількість бітів зі значенням «1» впливає на процес шифрування, зокрема на вибір операції шифрування, яка буде застосована до блоку даних. Це забезпечує варіативність шифрування та підвищує безпеку алгоритму.

У модулі керування, після підрахунку кількості бітів зі значенням «1» у ключі, зазвичай використовується ця кількість для визначення номера операції шифрування. Залежно від кількості відповідних бітів, можуть використовуватись різні операції шифрування, що забезпечує більшу варіативність процесу.

Наприклад, якщо кількість бітів зі значенням «1» дорівнює 0, це означає, що в ключі немає жодного біту зі значенням «1», і використана операція зі списку визначених модулем операцій буде під номером 0. Так само, якщо кількість бітів зі значенням «1» дорівнює 8 (максимально можлива кількість), це означає, що всі біти секретного ключа мають значення «1», і буде застосована остання операція із сформованого списку.

Таким чином, кількість таких бітів включає в себе варіативність шифрування і дозволяє керувати процесом шифрування залежно від вмісту ключа. Це покращує безпеку алгоритму і робить його більш гнучким у використанні.

У методі використовується цикл, який проходить через кожен біт числа зліва направо (від старших до молодших розрядів). Цикл виконується 8 разів, оскільки байт складається з 8 бітів.

На кожній ітерації циклу, за допомогою операції зсуву вправо (>>), число зміщується на і позицій, де і - поточна ітерація циклу. Після зсуву, застосовується побітова операція & з числом 1, щоб отримати найменший біт числа.

Якщо результат побітової операції рівний 1, це означає, що знайдено біт зі значенням «1». У такому випадку, лічильник count збільшується на 1.

По завершенні циклу, метод повертає значення лічильника count, яке відображає загальну кількість бітів зі значенням «1» у вхідному числі.

Наприклад, якщо ми викликаємо метод countOnes(155), то в результаті отримаємо значення 4. Це означає, що в числі 155 (10011011) є 5 бітів (біти, що мають значення 1).

Метод countOnes() є простим та ефективним способом підрахувати кількість бітів зі значенням «1» у вхідному числі, та є головним алгоритмом, що використовується для подальшого аналізу та прийняття рішень в модулі керування шифрування.

Модуль операцій — відповідає за генерацію 9 операцій шифрування, які будуть використовуватись під час процесу шифрування та розшифрування. Цей модуль може бути реалізований як список або статичний набір операцій.

У випадку, коли операції шифрування є фіксовані та не залежать від зовнішніх факторів, можна використати статичний набір операцій. Це означає, що список операцій буде заздалегідь визначений і фіксований протягом всього процесу шифрування. Наприклад, список операцій може включати такі операції, як додавання, віднімання, побітове XOR та інші, які відповідають вимогам конкретного алгоритму шифрування.

У запропонованій програмній реалізації виконано рішення із статичним визначення 9 операцій, але функціонал можна розширити за потребою. З іншого боку, якщо операції шифрування залежать від зовнішніх факторів, можна використати список операцій, який може бути змінюваним і визначатись динамічно. Наприклад, залежно від кількості бітів зі значенням «1» у ключі або інших параметрів, можуть вибиратись різні операції для шифрування. В цьому випадку, список операцій буде генеруватись або модифікуватись на основі цих параметрів.

Обидва підходи мають свої переваги і застосовуються в залежності від вимог конкретного алгоритму шифрування та програмного засобу.

За реалізацію модуля операцій відповідає Java метод performOperation() виконує операцію шифрування або розшифрування на основі вхідних даних, ключа та вказаної операції.

Пояснення щодо кожної операції:

0. $dataByte + keyByte$ & 0xFF, здійснюється додавання вхідного байта даних dataByte до ключового байта keyByte

1. $(dataByte - keyByte + 256)\ \&\ 0xFF$, виконується віднімання ключового байта keyByte від вхідного байта даних dataByte, після чого до

результату додається 256 і виконується ділення по модулю 256. Це забезпечує коректне значення в межах 0-255

2. $(keyByte - dataByte + 256)\ \&\ 0xFF$, проводиться віднімання вхідного байта даних dataByte від ключового байта keyByte, після чого до результату додається 256 і виконується ділення по модулю 256. Це також забезпечує коректне значення в межах 0-255

3. $\overline{dataByte}$, виконується побітова інверсія вхідного байта даних dataByte, де кожен біт, який має значення 0, перетворюється на 1, і навпаки.

4. $dataByte - \overline{keyByte}$, відбувається віднімання побітової інверсії ключового байта keyByte від вхідного байта даних dataByte.

5. ($\overline{keyByte} + \overline{dataByte}$) & 0xFF, побітове додавання побітової інверсії ключового байта keyByte до побітової інверсії вхідного байта даних dataByte. Результат обмежується значеннями в межах 0-255 за допомогою операції побітового І (&) з 0xFF

6. (dataByte ^ keyByte) & 0xFF, виконується операція XOR між вхідним байтом даних dataByte та ключовим байтом keyByte. Результат обмежується значеннями в межах 0-255 за допомогою операції побітового І (&) з 0xFF

7. $\left(keyByte + \overline{dataByte}\right)$ & 0xFF, виконується побітове додавання побітової інверсії вхідного байта даних dataByte до ключового байта keyByte. Результат обмежується значеннями в межах 0-255 за допомогою операції побітового І (&) з 0xFF

8. ($\overline{keyByte} + dataByte$) & 0xFF, відбувається побітове додавання побітової інверсії ключового байта keyByte до вхідного байта даних dataByte. Результат обмежується значеннями в межах 0-255 за допомогою операції побітового І (&) з 0xFF

Якщо передана операція не входить в діапазон від 0 до 8, метод поверне вхідний байт даних без змін.

Модуль операцій виконує основну функцію визначення операцій та застосування їх до вхідних даних для шифрування або розшифрування. Кожна операція має свою специфіку і внесе відповідний вплив на обробку даних, забезпечуючи безпеку та надійність шифрування.

Важливим моментом у реалізації модуля операцій є те, що Java не підтримує роботи із беззнаковими байтами, це ніяким чином не впливає на правильну роботу чи складність реалізації алгоритму, а впливає лише на те, що ми будемо отримувати знакові байти. У такий спосіб Java самостійно, виконуючи лише алгоритм для розшифрування без додаткових налаштувань, лише виконавши оборотні операції успішно повертає розшифроване оригінальне повідомлення.

На рис. 3.2 зображено UML діаграму із переліком Java методів та типами даних, вхідними даними, що приймає метод, та вихідні дані, які отримано у результаті виконання відповідної функції.

UML (Unified Modeling Language) діаграма - це графічний засіб моделювання, що використовується для візуалізації, специфікації, побудови та документування структури та поведінки системи або програмного продукту. Вона надає зручний спосіб відображення ключових аспектів системи за допомогою графічних символів та зв'язків.

UML діаграми дозволяють комунікувати між розробниками, архітекторами та зацікавленими сторонами, надаючи зрозумілу візуальну модель системи. Вони є стандартом в індустрії програмного забезпечення і використовуються для різних цілей, таких як аналіз, проектування, виконання, тестування та документування програмного забезпечення.

Існують різні типи діаграм, на рис. 3.2 зображено UML діаграму Java класу з детальним описом сигнатури методів.

Рисунок 3.2 — UML діаграма для Java класу з реалізацією методу шифрування, що керується даними

Результатами повного виконання алгоритму шифрування, що керується даними, є отримання зашифрованого повідомлення з використанням секретного ключа та визначених операцій шифрування. Після проходження крізь різні модулі, які включають модуль зчитування вхідних даних, регістр зсуву та блок керування, кожен блок даних повідомлення піддається відповідній операції шифрування з використанням ключа.

На кожному етапі модуля операцій виконується певна операція відповідно до вибраної операції шифрування. Кожна операція виконується над блоком даних та відповідним блоком ключа згідно з вибраним методом шифрування.

Після виконання всіх операцій над блоками даних із відповідними блоками ключа отримуємо зашифровані блоки даних (рис 3.3). Ці блоки можуть бути об'єднані разом, утворюючи повне зашифроване повідомлення. Отримані зашифровані дані є надійними та безпечними, оскільки кожен блок даних був підданий операціям шифрування з використанням секретного ключа, що гарантує конфіденційність та недоступність оригінального повідомлення для сторонніх осіб.

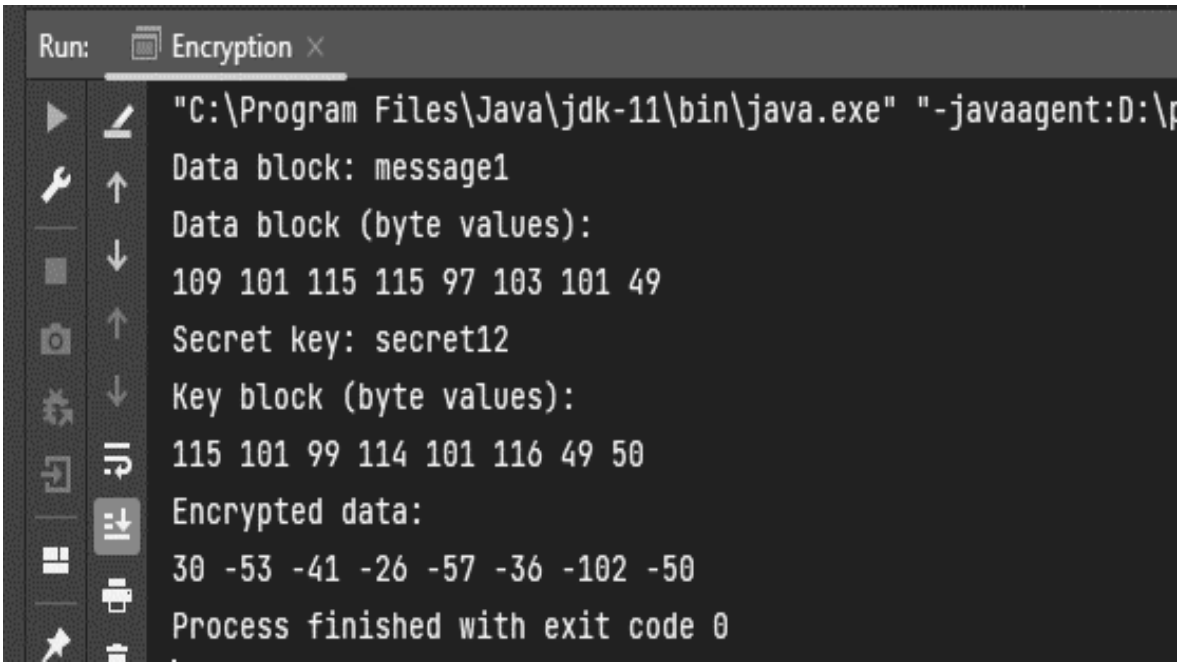


Рисунок 3.3 — Результат тестування методу шифрування

Таким чином, повне виконання алгоритму методу шифрування, що керується даними, забезпечує ефективне та надійне шифрування повідомлення з використанням визначеного ключа та операцій шифрування. Цей підхід дозволяє зберегти конфіденційність та безпеку обміну інформацією, роблячи її недоступною для несанкціонованого доступу.

В результаті тестування методу шифрування було зашифровано повідомлення «message1» із секретним ключем «secret12» та отримано зашифроване повідомлення у вигляді набору байт-даних із наступними значеннями: 30, -53, -41, -26, -57, -36, -102, -50 у знаковому вигляді, або 30, 181, 169, 154, 185, 164, 230, 178

3.2 Програмний засіб для розшифрування

Програмне рішення для розшифрування в цілому ідентичне як і для виконання процесу шифрування, але має деякі відмінності в реалізації.

Реалізація програмного застосунку для розшифрування буде включати наступні кроки:

1. Зчитування зашифрованого повідомлення — при запуску програмного застосунку користувач буде звернутий до вибору способу зчитування

зашифрованого повідомлення. Це може бути зчитування з текстового файлу, введення через консоль або інший визначений спосіб. Зчитане зашифроване повідомлення буде збережено у відповідній змінній.

2. Введення секретного ключа — користувачу буде запропоновано ввести секретний ключ, необхідний для розшифрування повідомлення. Введений ключ буде збережено у відповідній змінній.

3. Визначення операцій шифрування — за допомогою модуля операцій, будуть згенеровані обернені операції для кожної з 9 вихідних операцій шифрування. Це забезпечить правильну розшифрування зашифрованого повідомлення. Згенеровані операції будуть збережені у відповідному списку або масиві.

4. Розшифрування повідомлення: — застосунок буде ітеруватись через кожен блок зашифрованого повідомлення. Для кожного блоку, буде використовуватись обернена операція шифрування з відповідного списку або масиву операцій. Результат розшифрування буде збережений у відповідному списку або масиві розшифрованих блоків.

5. Об'єднання розшифрованих блоків — після розшифрування кожного блоку, розшифровані блоки будуть об'єднані в одне повне розшифроване повідомлення.

6. Виведення розшифрованого повідомлення — останнім кроком буде виведення отриманого розшифрованого повідомлення на екран або збереження його у файл, в залежності від вибору користувача.

Реалізація програмного застосунку написано також за допомогою використання Java 11 та середовища розробки IntelliJ IDEA 2023, а також враховуючи принципи SOLID, зокрема принцип Single Responsibility, при якому кожен модуль відповідає за конкретну функціональність і має лише одну причину зміни.

Імплементація застосунку для розшифрування базується на тому ж підході, що і для шифрування. Структура засобу поділяється на наступні модулі:

1. Модуль зчитування зашифрованого повідомлення та секретного ключа
2. Модуль регістру зсуву

3. Модуль керування

4. Модуль операцій

Усі модулі реалізовано згідно процесу шифрування, відмінність полягає є лише у тому, що у модулі операцій необхідно визначити правильно обернені операції до операцій, що були застосовані при процесі шифрування, а саме:

0. $(encryptedByte - keyByte + 256)\ mod\ 256$, значення нового розшифрованого байту обчислюється шляхом віднімання ключового байту від зашифрованого байту, додавання 256, а потім обчислення остачі від ділення на 256.

1. $(encryptedByte + keyByte)\ mod\ 256$, значення нового розшифрованого байту обчислюється шляхом додавання ключового байту до зашифрованого байту, а потім обчислення остачі від ділення на 256.

2. $((encryptedByte + keyByte - 256)\ mod\ 256 + 256)\ mod\ 256$, з начення нового розшифрованого байту обчислюється шляхом додавання ключового байту до зашифрованого байту, віднімання 256, а потім обчислення остачі від ділення на 256 з подальшим додаванням 256.

3. (encryptedByte ^ 0xFF), значення нового розшифрованого байту отримується шляхом виконання XOR між зашифрованим байтом та значенням 0xFF.

4. $\left(\text{encryptedByte} + \overline{\text{keyByte}}\right) mod\ 256$ значення нового розшифрованого байту обчислюється шляхом додавання до зашифрованого байту інвертованого значення ключового байту.

5. $\overline{\left(\text{encryptedByte} + 256 - \overline{\text{keyByte}}\right)}\ \text{mod}\ 256$, значення нового розшифрованого байту обчислюється шляхом виконання побітової негації суми зашифрованого байту, 256 та інвертованого значення ключового байту, а потім побітового Із з останніми 8 бітами (додаткове забезпечення того, що значення перебуває у межах 8-бітного беззнакового діапазону).

6. (encryptedByte ^ keyByte) & 0xFF, значення нового розшифрованого байту отримується шляхом виконання побітового XOR між зашифрованим байтом та ключовим байтом.

7. ~((encryptedByte + 256 - keyByte)) & 0xFF, значення нового розшифрованого байту обчислюється шляхом виконання побітової негації суми зашифрованого байту, 256 та значення ключового байту.

8. (encryptedByte + 256 - ~keyByte) & 0xFF, значення нового розшифрованого байту обчислюється шляхом виконання побітової суми між зашифрованим байтом та інвертованим значенням ключового байту, а потім обчислення остачі від ділення на 256.

Ці операції виконуються над зашифрованим байтом з метою отримання нового значення розшифрованого байту. Вибір операції залежить від значення "operation", яке передається в метод performOperation(). Якщо значення "operation" не зустрічається в переліку операцій, повертається початкове значення зашифрованого байту без змін.

Модулі у процесі розшифрування такі самі як і при шифруванні та працюють наступним чином:

— модуль зчитування вхідних даних зчитує дані з консолі

— модуль регістру зсуву починає зсув із початку ключа до його кінця.

— модуль керування даними визначає операцію за допомогою того ж самого методу countOnes() із аналогічним алгоритмом роботи.

— модуль операцій містить статично визначені операції для розшифрування (що є оберненим варіантом для усіх операцій шифрування із збереженням послідовності)

На рис. 3.4 зображено UML діаграму Java класу із описом методів, сигнатур методів, по назві методів можна визначити дії, за які відповідає та чи інша частка коду.

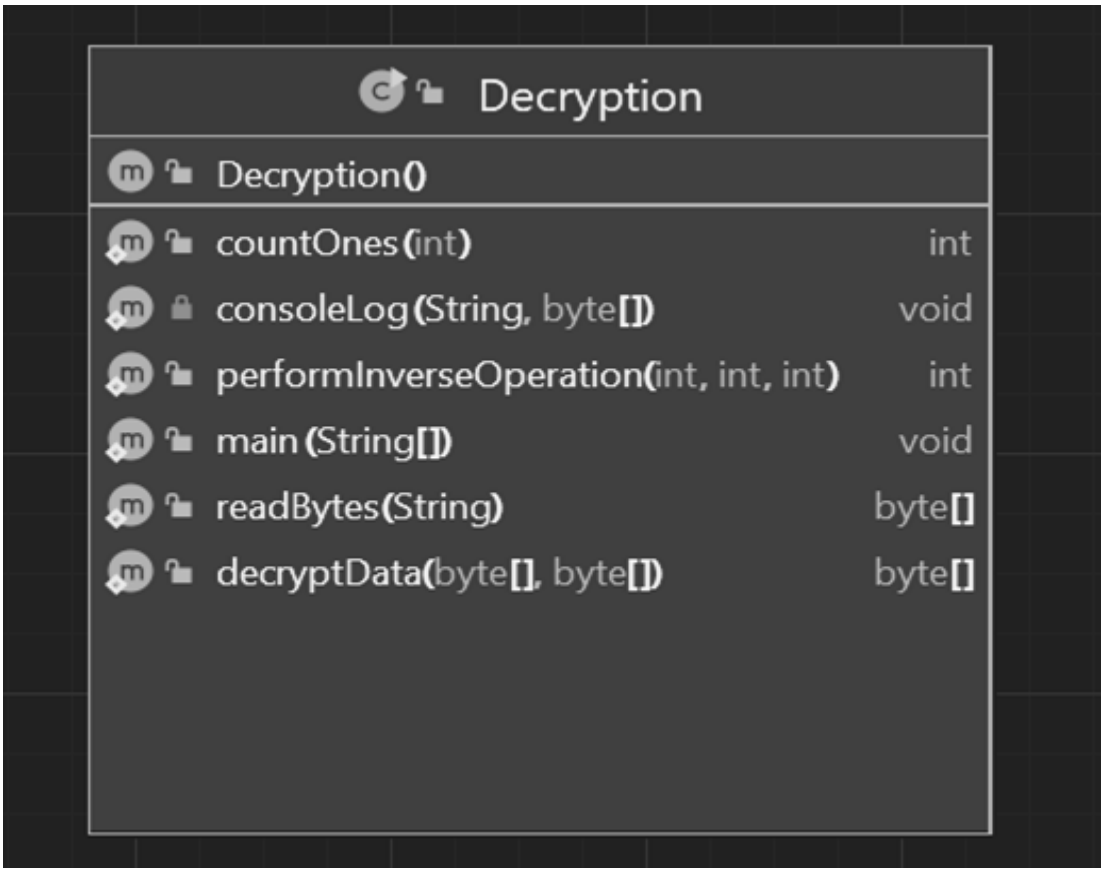


Рисунок 3.4 — UML діаграма для Java класу з реалізацією методу розшифрування, що керується даними

Після розшифрування раніше зашифрованого повідомлення (30, 181, 169, 154, 185, 164, 230, 178) із секретним ключем «secret12» отримано коректний результат, який відповідає оригінальному розшифрованому тексту у байтовому вигляді (109, 101, 115, 115, 97, 103, 101, 49). Всі операції, проведені над зашифрованими даними, вірно відтворили початковий текст, що свідчить про ефективність і правильність методу шифрування, що керується даними.

Отримані результати підтверджують надійність та точність роботи програмного засобі для розшифрування повідомлення та підтримують безпеку обміну інформацією.

На рис. 3.5 відображено проміжні значення у вигляді байтів, завдяки чому можна спостерігати за механізмом розшифрування, та порівняти кінцеве значення розшифрованих даних із початковим значенням вхідних даних на рис. 3.3.

Рисунок 3.5 — результат тестування методу розшифрування

Використовуючи переваги програмного середовища для розробки коду Intellij IDEA, що використано для розробки програмного застосунку можна порівняти результати виконання обох процесів, шифрування та розшифрування (рис 3.6).

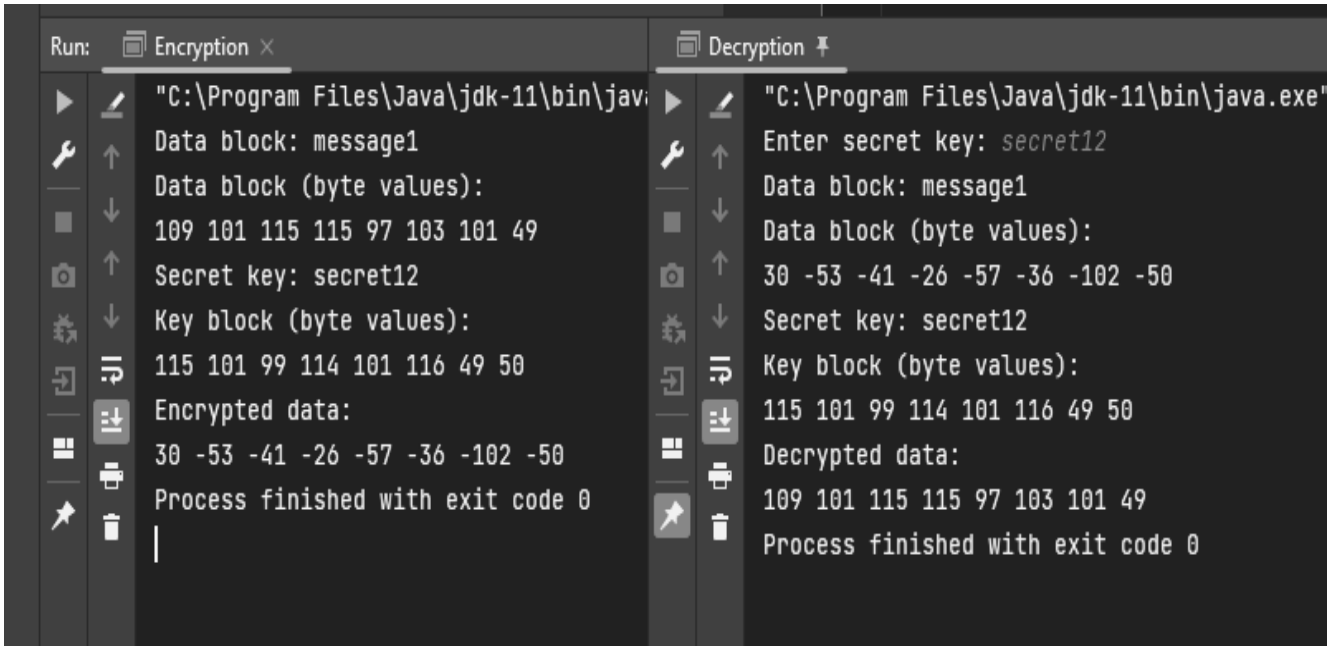


Рисунок 3.6 — порівняння результатів для перевірки на правильність роботи алгоритмів

Перевіривши рядки із вкладинки із результатами виконання шифрування в класі «Encryption» та рядки з вкладинки із результатами виконання розшифрування в класі «Decryption» результати наступні:

Оригінальне повідомлення у вигляді байтів (Data block): 109, 101, 115, 115, 97, 103, 101, 49.

Зашифроване повідомлення у вигляді байт коду (Encrypted data): 30, -53, -41, -26, -57, -36, -102, -50 у знаковому форматі, або 30, 181, 169, 154, 185, 164, 230, 178 у беззнаковому вигляді.

Розшифроване повідомлення у вигляді байтів (Decrypted data): 109, 101, 115, 115, 97, 103, 101, 49.

ВИСНОВКИ

В даній роботі було розроблено метод шифрування, що керується даними, і відповідний програмний засіб на мові Java. Метод шифрування базується на здатності блоків даних впливати на вибір операцій шифрування, що забезпечує гнучкість та адаптивність шифрування до конкретних вимог і контексту.

Розроблений програмний засіб був реалізований з використанням мови програмування Java 11 та інтегрованої середовища розробки IntelliJ IDEA 2023, що забезпечило зручний та ефективний процес розробки, а також порівння результатів, було враховано принципи SOLID, зокрема принцип єдиного обов'язку.

Результати тестування програмного засобу підтверджують, що розроблений метод шифрування працездатний та виконує свої функції належним чином. Програмний засіб пройшов через різноманітні тестові сценарії, включаючи шифрування різних типів вхідних даних, різних ключів та використання різних операцій шифрування.

Розроблений програмний засіб може бути розглянуто як оптимізаційний варіант для пришвидшення процесу блокового шифрування даних, як звичайних повідомлень так і великих обсягів інформації. Метод забезпечує конфіденційність та збереження цілісності даних та є стійким до зламу за допомогою криптоаналізу, так як порядок операцій шифрування базується на значенні характеристичних ознак даних секретного ключа.

У процесі тестування було перевірено правильність шифрування та розшифрування повідомлень з використанням розробленого методу. Результати тестування показали, що програмний засіб забезпечує коректність розшифрованих повідомлень, а також відтворення оригінальних повідомлень після розшифрування.

Отже, результати підтверджують успішну реалізацію поставлених завдань та досягнення мети розробки програмного засобу шифрування. Програмний засіб працездатний, надійний та готовий до використання для забезпечення безпеки обміну інформацією шляхом шифрування даних з використанням розробленого методу.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Thomas Schamberger "Data-Driven Encryption: A Survey" URL: https://eprint.iacr.org/2019/556.pdf (дата звернення: 24.05.2023).
2. Oliynykov R.V. A new encryption standard of Ukraine: The Kalyna block cipher. URL: http://eprint.iacr.org/2015/650 (дата звернення: 22.05.2023).
3. Data-Driven Encryption. URL: https://docs.informatica.com/master-data-management/multidomain-mdm/10-3-hotfix-1building-the-data-model/data-encryption/data-encryption-architecture.html (дата звернення: 24.05.2023).
5. What Is Data Encryption? URL: https://www.trellix.com/en-us/security-awareness/data-protection/what-is-data-encryption.html (дата звернення: 24.05.2023).
6. Data-driven control on encrypted data. URL: https://arxiv.org/abs/2008.12671 (дата звернення: 24.05.2023).
7. A BRIEF HISTORY OF ENCRYPTION AND CRYPTOGRAPHY. URL: https://www.thalesgroup.com/en/markets/digital-identity-and-security/magazine/brief-history-encryption (дата звернення: 24.05.2023).
8. A. Orgun. "Data-Driven Approaches for Cybersecurity" URL: https://www.sciencedirect.com/book/9780128038437/computer-and-information-security-handbook (дата звернення: 24.05.2023).
9. Gustavus J. Simmons. Advances in Cryptology: Proceedings of CRYPTO 84. Berlin: Springer. – 1985. – pp. 411–431 (дата звернення: 18.05.2023).
10. Hong S. Provable Security against Differential and Linear cryptanalysis for SPN Structure. LNCS, pp. 273-2838 (дата звернення: 20.05.2023).
11. Thomas H. Cormen. Introduction to Algorithms. – MIT Press, 2001. – p. 1292 (дата звернення: 20.05.2023)
12. Лисицький К.Є. Удосконалена конструкція початкового перетворення для SPN шифрів / К.Є.Лисицький // XIX Міжнародна науково-практична конференція «Безпека інформації в інформаційно-телекомунікаційних системах». – с. 133-153 (дата звернення: 21.05.2023).
13. Горбенко І.Д. Порівняльний аналіз алгоритмів блокового симетричного шифрування. ХНУРЕ, 2005. – Вып. 141. – С. 25–30.

14. Rahul Awati, Data security and privacy: What is Blowfish? URL: https://www.techtarget.com/searchsecurity/definition/Blowfish (дата звернення: 21.05.2023).
15. Гвоздецька К. П. Порівняння симетричного і асиметричного шифрування / Збірник тез доповідей : випуск 62 (м. Тернопіль, 12 жовтня 2021 р.). – Тернопіль. – 2021.
16. Лободенко Г. Я. Методи виявлення атак шифрувальників на основі аналізу характеристик зашифрованих файлів / М-во освіти і науки України, Харків. нац. ун-т радіоелектроніки. – Харків, 2020. – 75 с.
17. Куліш Д. А. Дослідження алгоритму симетричного шифрування інформації DES для використання в безпровідних системах передачі інформації / М-во освіти і науки України, Харків. нац. ун-т радіоелектроніки. – Харків, 2019. – 75 с.
18. Кохан С. А. Використання штучних нейронних мереж у криптоаналізі блочного симетричного шифру AES / Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління: тези доповідей одинадцятої міжнародної науково-технічно ї конференції, 8–9 квітня 2021р. – ВА ЗС АР; НТУ "ХПІ"; НАУ, ДП "ПДПРОНДІАВІАПРОМ"; УмЖ, 2021. – Т. 2, секції 3-5. – С. 42.
19. Бакаєв С. В. Програмна оптимізація алгоритму шифрування PRESENT : пояснювальна записка до атестаційної роботи здобувача вищої освіти на другому (магістерському) рівні, спеціальність 125 Кібербезпека / М-во освіти і науки України, Харків. нац. ун-т радіоелектроніки. – Харків, 2020. – 72 с.

ДОДАТОК А
ПРОТОКОЛ ПЕРЕВІРКИ БАКАЛАВРСЬКОЇ ДИПЛОМНОЇ РОБОТИ НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Засіб для шифрування, що керується даними

Автор роботи: Ольховий Михайло Вікторович

Тип роботи: бакалаврська дипломна робота .

Підрозділ кафедра захисту інформації ФІТКІ .

Показники звіту подібності Unicheck

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

Аналіз звіту подібності (відмітити потрібне):

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

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

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

Особа, відповідальна за перевірку _____ Каплун В. А.
(підпис) (прізвище, ініціали)

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

Автор роботи _____ Ольховий М.В.
(підпис) (прізвище, ініціали)

Керівник роботи _____ Лужецький В.А.
(підпис) (прізвище, ініціали)

ДОДАТОК Б

JAVA КОД ПРОГРАМНОГО ЗАСОБУ ЗАШИФРУВАННЯ

```
package org.example;

public class Encryption {
    private static final int[] OPERATIONS = {0, 1, 2, 3, 4, 5, 6, 7, 8};

    public static void main(String[] args) {
        String data = "message1";
        String key = "secret12";

        byte[] dataBlock = data.getBytes();
        byte[] keyBlock = key.getBytes();

        System.out.println("Data block: " + data);
        consoleLog("Data block", dataBlock);

        byte[] encryptedDataBlock = encryptData(dataBlock, keyBlock);



        System.out.println("\nSecret key: " + key);
        consoleLog("Key block", keyBlock);

        System.out.println("\nEncrypted data:");
        for (byte b : encryptedDataBlock) {
            System.out.print(b + " ");
        }
    }

    public static byte[] encryptData(byte[] dataBlock, byte[] keyBlock) {

        int keyIndex = 0;
        byte[] encryptedDataBlock = new byte[dataBlock.length];

        for (int i = 0; i < dataBlock.length; i++) {
            int keyByte = keyBlock[keyIndex];
            int dataByte = dataBlock[i];
```

```
            int onesCount = countOnes(keyByte);

            int operationIndex = onesCount % OPERATIONS.length;

            int encryptedByte = performOperation(dataByte, keyByte, operationIndex);

            encryptedDataBlock[i] = (byte) encryptedByte;

            keyIndex++;

            if (keyIndex >= keyBlock.length) {
                keyIndex = 0;
            }
        }

        return encryptedDataBlock;
    }

    public static int countOnes(int number) {
        int count = 0;
        for (int i = 0; i < 8; i++) {
            if (((number >> i) & 1) == 1) {
                count++;
            }
        }

        return count;
    }

    public static int performOperation(int dataByte, int keyByte, int operation) {
        switch (operation) {
            case 0:
                return (dataByte + keyByte) % 256;
            case 1:
                return (dataByte - keyByte + 256) % 256;
            case 2:
                return (keyByte - dataByte + 256) % 256;
            case 3:
                return ~dataByte;
```

```
            case 4:
                return dataByte - ~keyByte;
            case 5:
                return (~keyByte + ~dataByte) & 0xFF;
            case 6:
                return (dataByte ^ keyByte) & 0xFF;
            case 7:
                return (keyByte + ~dataByte) & 0xFF;
            case 8:
                return (~keyByte + dataByte) & 0xFF;
            default:
                return dataByte;
        }
    }

    private static void consoleLog(String message, byte[] bytesBlock) {
        System.out.println(message + " (byte values): ");
        for (byte b : bytesBlock) {
            System.out.print(b + " ");
        }
    }
}
```

ДОДАТОК В

JAVA КОД ПРОГРАМНОГО ЗАСОБУ РОЗШИФРУВАННЯ

```
package org.example;

import java.util.Scanner;

public class Decryption {
    private static final int[] OPERATIONS = {0, 1, 2, 3, 4, 5, 6, 7, 8};

    public static void main(String[] args) {
        String encryptedData = "message1";

        System.out.print("Enter secret key: ");
        Scanner scanner = new Scanner(System.in);

        String key = scanner.next();

        byte[] encryptedDataBlock = {30, -53, -41, -26, -57, -36, -102, -50};
        byte[] keyBlock = readBytes(key);

        System.out.println("Data block: " + encryptedData);
        consoleLog("Data block", encryptedDataBlock);

        final byte[] decryptedData = decryptData(encryptedDataBlock, keyBlock);

        System.out.println("\nSecret key: " + key);
        consoleLog("Key block", keyBlock);

        System.out.println("\nDecrypted data:");
        for (byte b : decryptedData) {
            System.out.print(b + " ");
        }
    }

    public static byte[] decryptData(byte[] encryptedDataBlock, byte[] keyBlock) {

        int keyIndex = 0;
        byte[] decryptedDataBlock = new byte[encryptedDataBlock.length];
```

```
    for (int i = 0; i < encryptedDataBlock.length; i++) {
        int keyByte = keyBlock[keyIndex];
        int encryptedByte = encryptedDataBlock[i];

        int onesCount = countOnes(keyByte);

        int operationIndex = onesCount % OPERATIONS.length;

        int decryptedByte = performInverseOperation(encryptedByte, keyByte, operationIndex);

        decryptedDataBlock[i] = (byte) decryptedByte;

        keyIndex++;
        if (keyIndex >= keyBlock.length) {
            keyIndex = 0;
        }
    }

    return decryptedDataBlock;
}

public static int countOnes(int number) {
    int count = 0;
    for (int i = 0; i < 8; i++) {
        if (((number >> i) & 1) == 1) {
            count++;
        }
    }

    return count;
}

public static byte[] readBytes(String inputData) {
    return inputData.getBytes();
}

public static int performInverseOperation(int encryptedByte, int keyByte, int operation) {
    // оборотні операції для:
```

```
        switch (operation) {
            case 0:
                return (encryptedByte - keyByte + 256) % 256;
            case 1:
                return (encryptedByte + keyByte) % 256;
            case 2:
                return ((encryptedByte + keyByte - 256) % 256 + 256) % 256;
            case 3:
                return (encryptedByte ^ 0xFF);
            case 4:
                return encryptedByte + ~keyByte;
            case 5:
                return ~(encryptedByte + 256 - ~keyByte) & 0xFF;
            case 6:
                return (encryptedByte ^ keyByte) & 0xFF;
            case 7:
                return ~((encryptedByte + 256 - keyByte)) & 0xFF;
            case 8:
                return (encryptedByte + 256 - ~keyByte) & 0xFF;
            default:
                return encryptedByte;
        }
    }

    private static void consoleLog(String message, byte[] bytesBlock) {
        System.out.println(message + " (byte values): ");
        for (byte b : bytesBlock) {
            System.out.print(b + " ");
        }
    }
}
```

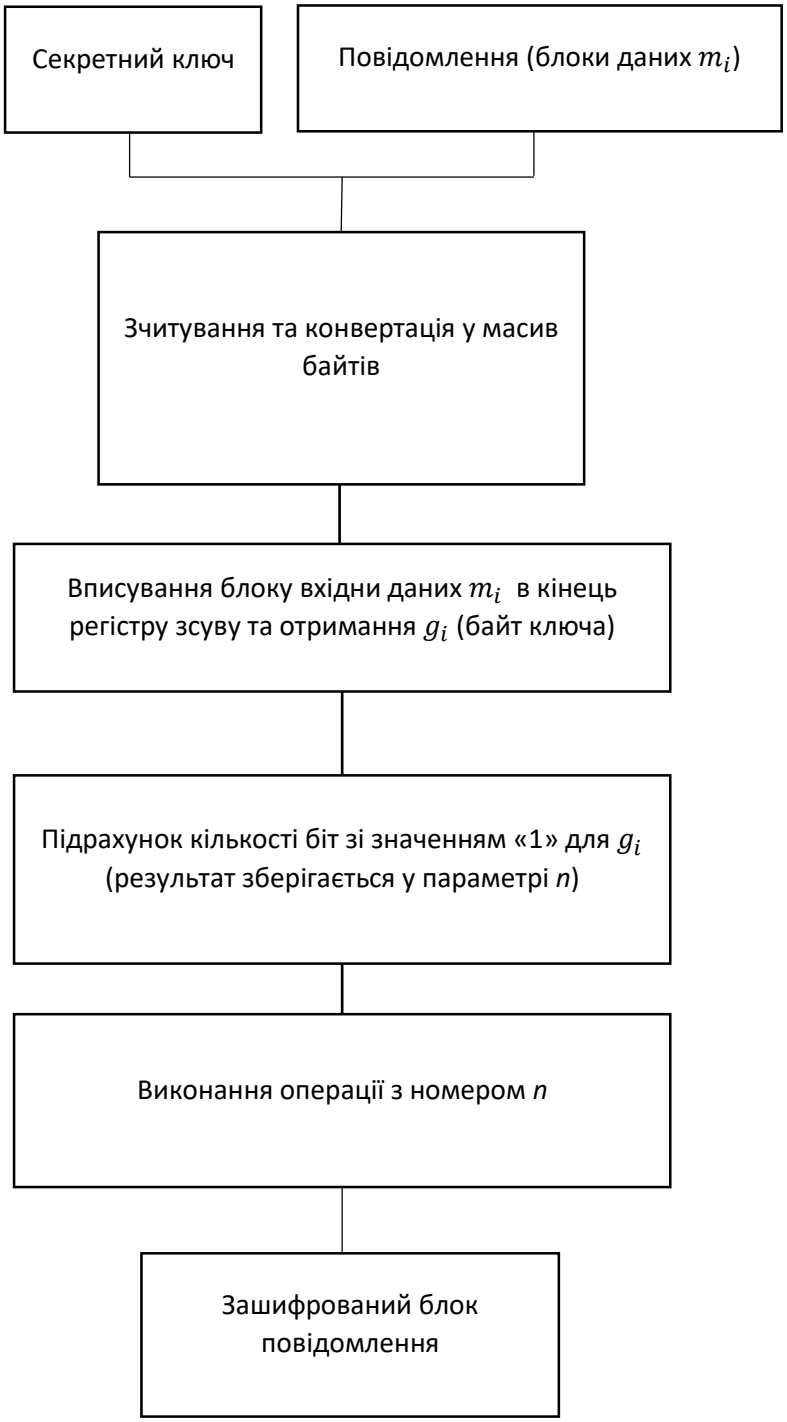
ІЛЮСТРАТИВНА ЧАСТИНА

Засіб для шифрування, що керується даними

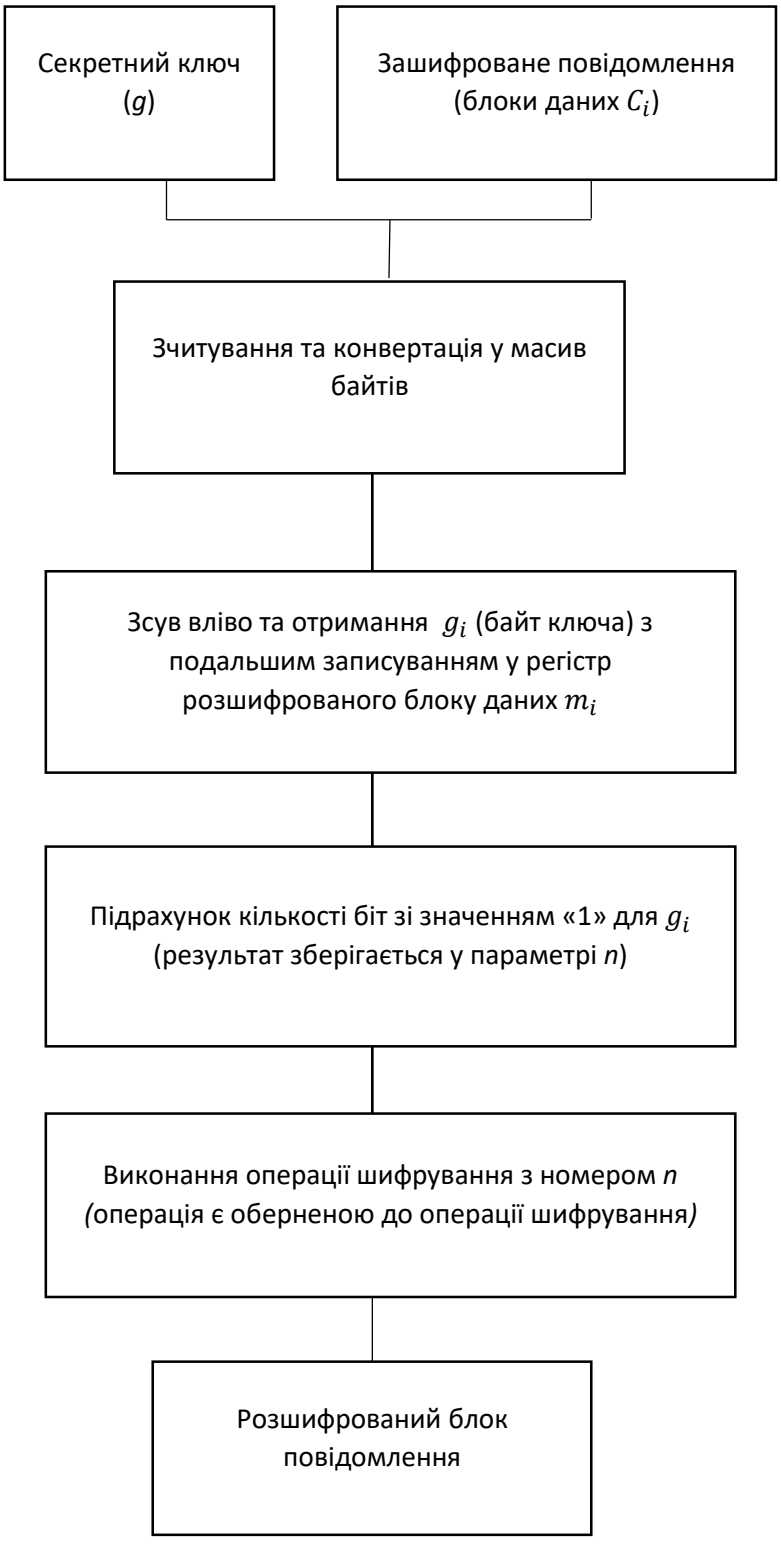
Виконав: студент 2 курсу групи 1БС-21мс
спеціальності 125 Кібербезпека
_____ Ольховий М.В.
19 червня 2023 р.

Керівник: зав. кафедри ЗІ д.т.н., проф
_____ Лужецький В.А.
19 червня 2023 р.

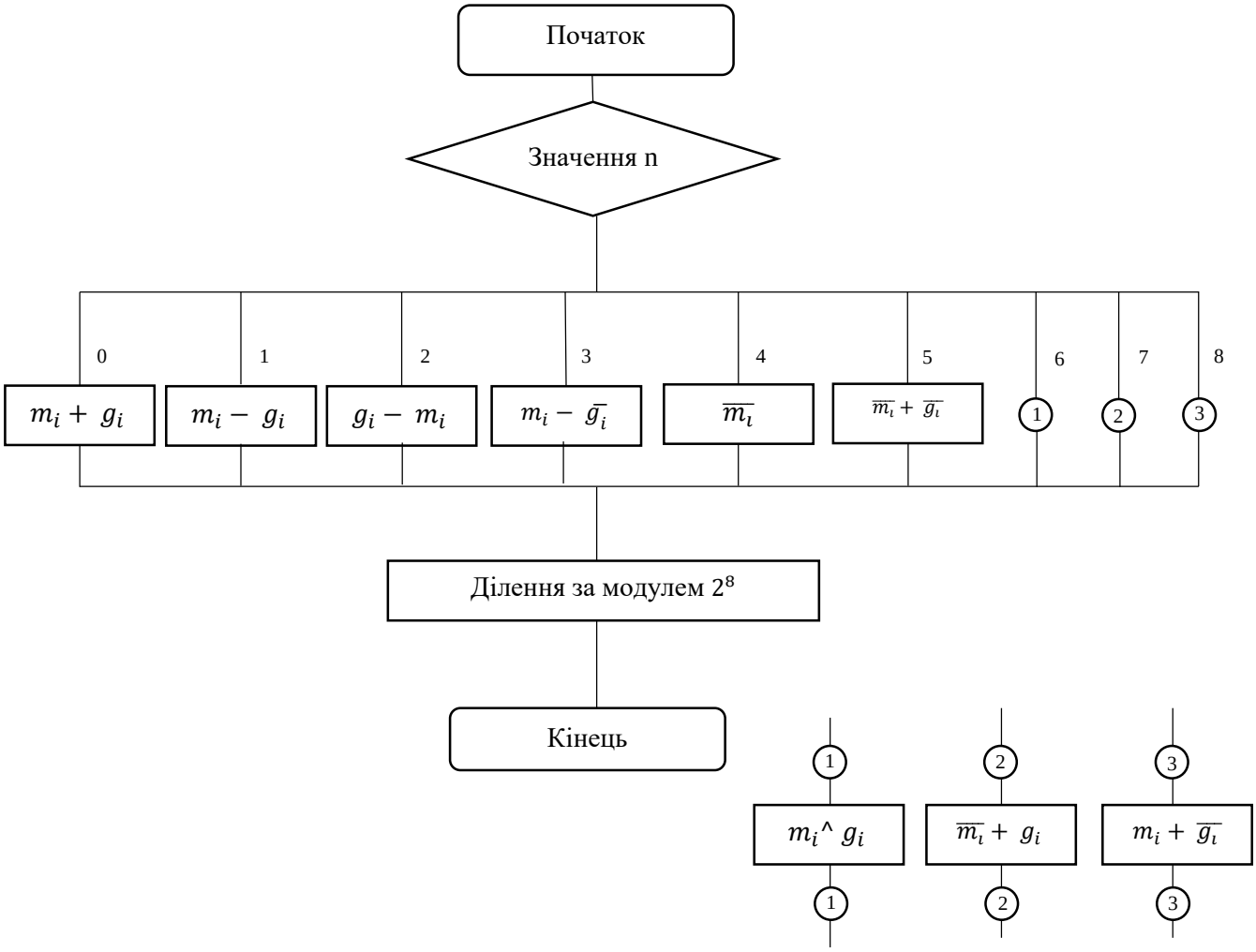
МЕТОД ЗАШИФРУВАННЯ



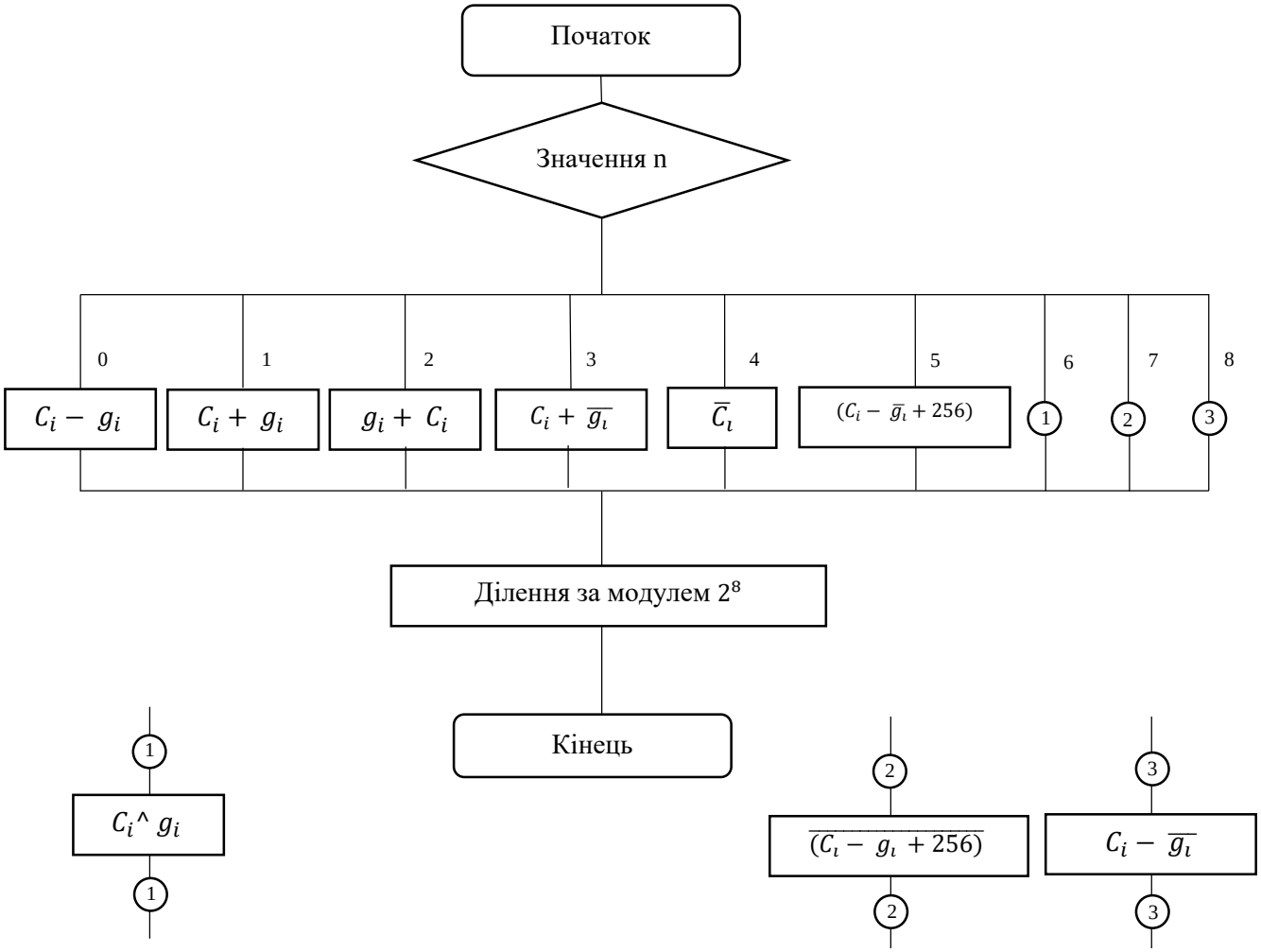
МЕТОД РОЗШИФРУВАННЯ



АЛГОРИТМ ВИЗНАЧЕННЯ ОПЕРАЦІЇ ЗАШИФРУВАННЯ, НА ОСНОВІ ХАРАКТЕРИСТИЧНИХ ОЗНАК ДАНИХ



АЛГОРИТМ ВИЗНАЧЕННЯ ОПЕРАЦІЇ РОЗШИФРУВАННЯ, НА ОСНОВІ ХАРАКТЕРИСТИЧНИХ ОЗНАК ДАНИХ



МОДУЛІ ПРОГРАМНОГО ЗАСОБУ ШИФРУВАННЯ

Модуль зчитування вхідних даних

(відповідає за спосіб зчитування вхідних даних, зчитування даних, зчитування ключа, конвертацію даних у масив байтів та поділ на блоки розміром 8x8 байт, конвертацію ключа у масив байтів)

Модуль регістру зсуву

(відповідає за зсув секретного ключа за допомогою блоку даних вхідного повідомлення та виділення байту ключа з наступною передачею його до модулю керування)

Модуль керування

(відповідає за підрахунок кількості бітів зі значенням «1» та визначення, на основі цього підрахунку, операції шифрування)

Модуль операцій

(відповідає за збереження та виконування 9-ти операцій шифрування із наступним отриманням блоку зашифрованих даних)

РЕЗУЛЬТАТИ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАСОБУ ШИФРУВАННЯ