МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ АЕРОНАВІГАЦІЇ, ЕЛЕКТРОНІКИ ТА ТЕЛЕКОМУНІКАЦІЙ
КАФЕДРА ЕЛЕКТРОНІКИ, РОБОТОТЕХНІКИ І ТЕХНОЛОГІЙ
МОНІТОРИНГУ ТА ІНТЕРНЕТУ РЕЧЕЙ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри
_____ Володимир ШУТКО
«____» _____ 2023 р.

# ДИПЛОМНА РОБОТА

ЗДОБУВАЧА ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА
ЗІ СПЕЦІАЛЬНОСТІ 171 «ЕЛЕКТРОНІКА»
ОПП «ЕЛЕКТРОННІ ТЕХНОЛОГІЇ ІНТЕРНЕТУ РЕЧЕЙ»

**Тема: «Програмна реалізація матричного алгоритму поточного шифрування інформації»**

Виконавець
Студент групи ФАЕТ 307 стн _____ Медвідь А.Г.

Керівник
професор _____ Білецький А.Я.

Нормоконтролер _____ Сініцин Р.Б.

**КИЇВ 2023**

**НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ**

Факультет аеронавігації, електроніки і телекомунікацій

Кафедра електроніки, робототехніки і технологій моніторингу та інтернету речей

Напрям (спеціальність) 171 «Електроніка»
(шифр, найменування)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Шутко В.М.
«____»_____ 2023 р.

# ЗАВДАННЯ

## на виконання дипломної роботи

Медвідю Артему Григоровичу
(прізвище, ім'я, по батькові випускника в родовому відмінку)

1. Тема дипломної роботи: «Програмна реалізація матричного алгоритму поточного шифрування інформації».
   Затверджена наказом ректора від «__» _____2023р. _____.
2. Термін виконання роботи: з «__» _____2023р. по «__» _____2023р.
3. Вихідні дані до роботи: створення сучасного матричного алгоритму поточного шифрування інформації та його програмна реалізація на основі узагальнених матриць Галуа.
4. Зміст пояснювальної записки: теоретичний опис способу, розробка програмного рішення, результати аналізу та тестування.
5. Перелік обов'язкового ілюстративного матеріалу: Структурна схема узагальненого базового генератора PRS Галуа, Структурна схема узагальненого сполученого генератора PRS Галуа**,** Узагалена структурно-

логічна схема поточного шифру**,** Схема поточного шифру Галуа-64, Узагальненний Галуа-64 матричний поточний шифр**.**

6. Календарний план-графік

| № пор. | Завдання | Термін виконання | Підпис керівника |
|---|---|---|---|
| 1 | Написати заяву дипломної роботи | 20.02.2023 | |
| 2 | Ознайомитися та обґрунтувати актуальність обраної теми | 21.02.2023-03.03.2023 | |
| 3 | Сформулювати мету. Сформулювати завдання дипломної роботи | 04.03.2023-07.03.2023 | |
| 4 | Ознайомитися з загальними положеннями про дипломне проектування та оформлення | 08.03.2023-09.03.2023 | |
| 5 | Здійснити бібліографічний пошук | 10.03.2023 | |
| 6 | Написати вступ та загальні відомості | 11.03.2023-31.03.2023 | |
| 7 | Написати 2-й та 3-й розділ дипломної роботи | 01.04.2023-15.05.2023 | |
| 8 | Написати програмне забезпечення | 15.05.2023-05.06.2023 | |
| 9 | Усунути недоліки | 05.06.2023-12.06.2023 | |
| 10 | Подати дипломну роботу на кафедру | 13.06.2023 | |

7. Консультація окремого(-мих) розділу(-ів):

| Назва розділу | Консультант (посада, П.І.Б.) | Дата, підпис | |
|---|---|---|---|
| | | Завдання | Завдання |

| | | видав | прийняв |
|---|---|---|---|
| Матричний алгоритм поточного шифрування інформації | д.т.н., професор Білецький А.Я. | 01.05.2023 | |

8. Дата видачі завдання: «__» _____ 2023р.

Керівник дипломної роботи (проекту) _____ Білецький А. Я.
(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання _____ Медвідь А.Г.
(підпис випускника) (П.І.Б.)

# РЕФЕРАТ

Пояснювальна записка до дипломної роботи на тему «Програмна реалізація матричного алгоритму поточного шифрування інформації»:

60 сторінок, 5 рисунків, 3 таблиці, 14 джерел посилань.

**Актуальність теми** полягає у розробці ефективної програмної реалізації матричного алгоритму поточного шифрування інформації, оскільки вона дозволяє забезпечити надійний захист інформації в режимі реального часу. Захист інформації є однією з найважливіших проблем у сучасному цифровому світі, а зростаюча кількість кібератак та загрози безпеці даних ставлять під загрозу конфіденційність і цілісність інформації, і поточне шифрування інформації є одним з ефективних методів забезпечення захисту даних.

**Мета роботи** – детальний аналіз та програмна реалізація матричного алгоритму поточного шифрування інформації на основі узагальнених матриць та генераторів Галуа. Основна мета полягає в розробці ефективного шифрувального алгоритму та його програмної реалізації, яка забезпечує надійний захист конфіденційної інформації в режимі реального часу.

**Об'єктом розробки** є матричний алгоритм поточного шифрування інформації на основі узагальнених матриць та генераторів Галуа.

**Метод дослідження** – програмування на ПЕОМ з використанням програмного середовища Microsoft Visual Studio для написання коду шифрування на основі Галуа 4x4 матриць, вихідного коду графічного інтерфейсу, коду узагальнених Галуа матриць та коду генератора PRS.

**Результати проведеної роботи** – розроблено програмне забезпечення, що реалізує матричний алгоритм поточного шифрування інформації на основі узагальнених матриць Галуа.

Результати даної дипломної роботи підтверджують ефективність розробленого алгоритму і можуть бути використані для подальшого розвитку та вдосконалення методів криптографічного захисту інформації. Тому отримані результати можуть бути використані для рекомендацій щодо вдосконалення поточних методів криптографічного захисту інформації та в якості основи для подальших досліджень у сфері криптографії та інформаційної безпеки.

**Ключові слова:** ПСЕВДОВИПАДКОВА ПОСЛІДОВНІСТЬ, УЗАГАЛЬНЕНІ МАТРИЦІ ГАЛУА, СПОЛУЧЕНІ МАТРИЦІ, ЛІНІЙНИЙ РЕЄСТР ЗСУВУ, ГЕНЕРАТОР ПВП, НЕЗВІДНІ ПОЛІНОМИ, ПРИМІТИВНІ ПОЛІНОМИ.

# ЗМІСТ

# ПЕРЕЛІК СКОРОЧЕНЬ

PRS – pseudo random sequence (псевдо випадкова послідовність);

LSR – linear shift register (лінійний реєстр зсуву);

LFSR – linear feedback shift register (реєстр зсуву з лінійними зворотними зв'язками);

IrP – irreducible polynomials (незвідні поліноми);

PrP – primitive polynomials (примітивні поліноми);

SIrP – simple irreducible polynomials (прості незвідні поліноми);

SR – shift register (реєстр зсуву).

# ВСТУП

Криптографія є невід'ємною складовою сучасного світу, де інформація є однією з найцінніших ресурсів. У зв'язку з цим, захист інформації від несанкціонованого доступу та зловживання є дуже важливим завданням для бізнесу, урядів та інших організацій. Поточний криптографічний захист є однією з найефективніших технологій захисту інформації, яка забезпечує безпеку даних від різноманітних загроз, включаючи крадіжку, підроблення, а також витік інформації.

Предметом даної дипломної роботи є основи поточного криптографічного захисту інформації. У роботі будуть розглянуті основні принципи криптографії, що дозволяють захистити дані від несанкціонованого доступу та зловживання. Також будуть проаналізовані сучасні методи поточного криптографічного захисту інформації та їх використання в практиці.

Метою дипломної роботи є детальний аналіз основ поточного криптографічного захисту інформації, оцінка ефективності використання цих методів, а також вивчення можливостей покращення та розвитку поточних методів захисту інформації. В результаті роботи будуть запропоновані рекомендації щодо вдосконалення поточних методів криптографічного захисту інформації та розроблені пропозиції щодо використання цих методів в реальних умовах.

# 1. ОСНОВИ АЛГОРИТМІВ ПОТОЧНОГО ЗАХИСТУ ІНФОРМАЦІЇ

## 1.1 Опис поточного шифрування та його особливостей

Поточне шифрування – це криптографічний метод, в якому шифрувальний ключ використовується для генерації потоку псевдовипадкових бітів, які застосовуються для шифрування повідомлень. Цей потік бітів як правило може бути використаний лише один раз і після цього знищується.

Основною особливістю поточного шифрування є те, що він є неперервним методом шифрування, що означає, що повідомлення шифрується біт за бітом або байтом за байтом, а не розбивається на блоки, як у блочних шифрах. Це забезпечує високу швидкість шифрування та можливість роботи з потоком даних, що поступають на вхід шифрувального алгоритму.

Іншою важливою особливістю поточного шифрування є його можливість працювати в режимі без збереження стану, що означає, що при перериванні роботи алгоритму не втрачається генерований ним потік псевдовипадкових бітів, тому продовження шифрування можливе з того ж місця.

Також поточне шифрування зазвичай використовується для захисту великої кількості даних, таких як передача даних в режимі реального часу або великі файли, що не можуть бути розбиті на блоки для застосування блочних шифрів. Однак, в порівнянні з блочним шифруванням, поточне шифрування може бути менш стійким до деяких видів атак, таких як атака початкового вектора ініціалізації (IV) та атака на ключовий потік (key stream).

## 1.2 Порівняння поточного та блочного шифрування

Поточне і блочне шифрування є двома основними підходами до криптографічного захисту даних, які використовуються для прямого перетворення простого тексту в шифр і належать до сімейства симетричних ключових шифрів.

Кожен підхід має свої переваги та недоліки, тому й вибір між ними залежить від конкретного використання, або ж конкретних вимог до системи захисту даних. Та якщо порівнювати обидва алгоритми, то можна відзначити наступні основні відмінності:

1. Обробка даних: у поточному шифрі дані обробляються по одному або декількам байтам, а у блочному – по блоках фіксованої довжини (зазвичай 64, 128, 256 інколи-512 біт).
2. Швидкість роботи: поточне шифрування зазвичай є швидшим, ніж блочне, оскільки не потрібно чекати завершення обробки всього блоку даних.
3. Рівень безпеки: блочне шифрування зазвичай має вищий рівень безпеки, ніж поточне, оскільки воно забезпечує криптографічну стійкість відносно більш широкого спектру атак (наприклад, атак типу «відбиття» або «заміни» при використанні одного ключа для кількох блоків).
4. Використання пам'яті: при поточному шифруванні зазвичай використовуються регістри зсуву інтегральних мікросхем, що дозволяє зберігати ключові дані в малих розмірах пам'яті. У блочному шифруванні зазвичай потрібна більша кількість пам'яті для зберігання ключів та інших параметрів.
5. Робота з помилками: у поточному шифруванні помилки при передачі даних можуть призвести до втрати синхронізації та розголошення даних, що призводить до більш складного контролю за цілісністю даних. У блочному шифруванні помилки можуть відбуватися в межах окремого

блоку, тому є можливість коректувати пошкоджені дані без впливу на решту блоків.

Підсумовуючи, сформуємо переваги обох алгоритмів, також виділимо їх у діаграмі порівняння.

Переваги поточного шифрування:

- Швидкість: поточні шифри працюють швидше, ніж блочні, тому їх можна використовувати в реальному часі для захисту потоків даних.
- Ефективність передачі даних: поточні шифри можуть зашифрувати будь-який розмір даних, тоді як блочні шифри можуть зашифрувати лише блоки фіксованого розміру.
- Більш висока безпека в деяких випадках: поточні шифри можуть бути менш вразливі до атак, що залежать від структури даних.

Переваги блочного шифрування:

- Більш висока безпека в деяких випадках: блочні шифри зазвичай вважаються більш безпечними, оскільки вони можуть працювати з блоками фіксованого розміру, що робить їх менш вразливими до атак, що залежать від структури даних.
- Менше вимог до ключа: Блочні шифри зазвичай потребують меншу довжину ключа, ніж поточні шифри, що може бути важливим у деяких випадках.

Нижче наведена діаграма порівняння поточного та блочного алгоритмів шифрування:

*Таблиця 1*

Порівняння поточного та блочного алгоритмів шифрування

| Критерій порівняння | Поточний шифр | Блочний шифр |
|---|---|---|
| Розмір блоку даних | Необмежена кількість бітів даних | Фіксований розмір |
| Швидкість обробки | Працює швидше, оскільки не потребує вирівнювання блоків даних | Може бути повільнішим через обробку блоків даних |
| Стійкість до атак | Може бути стійким лише до певних типів атак, таких як атаки на потік ключів | Вважається більш стійким до атак, оскільки розмір блоків даних не дозволяє зламувачам працювати з окремими бітами даних |
| Пам’ять | Вимагає меншої пам'яті для зберігання ключа шифрування, оскільки працює з одним ключем | Може вимагати більшої кількості пам'яті для зберігання ключа та вектора ініціалізації |
| Ресурсозалежність | Може бути менш ресурсозалежним, оскільки не потребує додаткових операцій з пам'яттю для вирівнювання блоків даних | Чим більші розміри блоків, тим більше ресурсів вимагається для шифрування, оскільки збільшується кількість операцій шифрування, які потрібно виконати. Крім того, блочні алгоритми вимагають зберігання |

| | | |
|---|---|---|
| | | в пам'яті всіх блоків даних, що може призвести до великих витрат на пам'ять при обробці великих об'ємів даних. |

Загалом, якщо важлива швидкість та ефективність передачі даних, то поточні шифри можуть бути кращим варіантом. Якщо ж важлива безпека та менша довжина ключа, то кращим вибором буде блочне шифрування. Однак, в більшості випадків застосовують комбінації обох підходів, щоб отримати оптимальний результат.

## 1.3 Огляд алгоритмів поточного шифрування

### 1.3.1 Алгоритм RC4

RC4 (або Rivest Cipher 4) – це алгоритм поточного шифрування, який був розроблений Роном Рівестом в 1987 році. RC4 був використаний у безлічі систем інформаційної безпеки, таких як SSL, WEP, Wi-Fi та інших.

Історія RC4 розпочалась у 1984 році, коли Рон Рівест був професором на MIT. Він працював над створенням нового алгоритму шифрування, який був би дуже швидким та надійним. У процесі розробки він використовував метод поточного шифрування, який був дуже ефективним та легким у реалізації.

У 1987 році Рівест опублікував статтю «A stream cipher encryption algorithm» (алгоритм шифрування поточним шифром), в якій вперше був описаний RC4. Алгоритм був опублікований без ліцензування та вільно розповсюджувався, що дозволило йому стати популярним у великій кількості систем.

Проте вже в наступному десятилітті RC4 став об'єктом критики через ряд вразливостей, що виявились у алгоритмі. Зокрема, RC4 показав слабкість у використанні в протоколах безпеки, таких як WEP, який був використаний для бездротових мереж. У результаті RC4 був вилучений з багатьох стандартів і замінений на інші більш сучасні алгоритми шифрування.

Проте, незважаючи на ці вразливості, RC4 залишається використовуваним в багатьох системах, де його ефективність та простота реалізації переважають його недоліки.

Основні особливості роботи алгоритму RC4 наступні:

- Генерація ключового потоку: ключовий потік генерується за допомогою псевдовипадкового генератора, який залежить від ключа, який вводить користувач.

- Шифрування: шифрування повідомлення здійснюється шляхом XOR (логічне «або») кожного байту повідомлення з відповідним байтом ключового потоку.
- Розшифрування: розшифрування повідомлення виконується за тією ж самою схемою XOR з ключовим потоком.

Переваги алгоритму RC4:

1. Швидкість: RC4 дуже швидкий, тому що він використовує просту операцію XOR для шифрування та розшифрування повідомлень.
2. Ефективність: RC4 не вимагає багато ресурсів для своєї роботи, тому він добре підходить для використання на різних платформах та пристроях з обмеженими ресурсами.

Простота реалізації: алгоритм RC4 досить простий для реалізації, тому він широко використовується в багатьох системах та програмах.

### 1.3.2 Алгоритм Salsa20

Salsa20 – це алгоритм поточного шифрування, розроблений криптографом Деніелом Бернштейном у 2005 році. Salsa20 став популярним через свою швидкість, безпеку та простоту реалізації.

Історія Salsa20 почалась з того, що Бернштейн працював над створенням нового алгоритму шифрування, який би був безпечним, швидким та мав простий для реалізації код. У результаті його роботи народився алгоритм Salsa20.

Salsa20 є алгоритмом поточного шифрування, тобто кожен символ вхідного тексту шифрується окремо, використовуючи генератор псевдовипадкових чисел (PRNG), що генерує ключовий потік. Алгоритм Salsa20

використовує 256-бітний ключ та 64-бітний лічильник, який забезпечує унікальність кожного блоку ключового потоку. Крім того, Salsa20 має високу швидкість роботи та може бути легко реалізований на різних платформах.

Salsa20 був широко використовуваний в різних програмах та системах, таких як OpenSSH, Linux, kernel, Tor, і т.д. Алгоритм також вважається одним з найбільш безпечних поточних шифрувань, які доступні на даний момент.

У 2013 році Бернштейн представив нову версію Salsa20, яку він назвав Salsa20/12. Ця версія використовує більш складну функцію перемішування, що дозволяє забезпечити ще більшу безпеку шифрування.

Salsa20 є ефективним, швидким та безпечним алгоритмом поточного шифрування, який використовується в різних системах та програмах. Його безпеку і швидкість роботи перевірено випробуванням на час.

Основні особливості роботи алгоритму Salsa20 наступні:

- Генерація ключового потоку: ключовий потік в Salsa20 генерується за допомогою односторонньої функції (більше відомої як "хешування") на основі ключа та початкового вектора (IV).
- Блочне шифрування: Salsa20 працює в режимі поточного шифрування, але блоки повідомлень (64 байти) розглядаються як послідовність елементів матриці 4x4, що дозволяє ефективно застосовувати операції шифрування.
- Розширення ключа: Salsa20 може використовувати ключі різної довжини (від 128 до 512 біт) та використовувати різні початкові вектори, що робить його більш універсальним та зручним для застосування в різних сценаріях.
- Висока швидкість: Salsa20 є дуже швидким алгоритмом, що робить його ідеальним для застосування в різних обчислювальних системах та пристроях.
- Безпека: Salsa20 є дуже безпечним алгоритмом, що базується на використанні операцій, які важко зламати за допомогою криптоаналізу.

- Надійність: Salsa20 є надійним алгоритмом, який відповідає стандартам безпеки його застосування.

Основні переваги алгоритму Salsa20:

1. Висока швидкість: Salsa20 є дуже швидким алгоритмом, що дозволяє шифрувати та розшифровувати великі об'єми даних з високою швидкістю. Це робить його ідеальним для застосування в різних обчислювальних системах та пристроях.
2. Безпека: Salsa20 є дуже безпечним алгоритмом, який базується на використанні операцій, які важко зламати за допомогою криптоаналізу. Це робить його ідеальним для застосування в різних захищених протоколах зв'язку та зберігання конфіденційних даних.
3. Розширення ключа: Salsa20 може використовувати ключі різної довжини (від 128 до 512 біт) та використовувати різні початкові вектори, що робить його більш універсальним та зручним для застосування в різних сценаріях.
4. Розмір ключа: Salsa20 дозволяє використовувати ключі довжиною 256 біт, що є більш безпечним у порівнянні з меншими розмірами ключів, які використовуються в інших алгоритмах.
5. Надійність: Salsa20 є надійним алгоритмом, який відповідає стандартам безпеки його застосування.
6. Простота: Salsa20 є досить простим алгоритмом, що дозволяє легко його реалізувати та використовувати в різних застосуваннях.

Незалежність від платформи: Salsa20 не залежить від конкретної апаратної архітектури чи операційної системи, що дозволяє використовувати його на будь-яких пристроях з різними операційними системами.

### 1.3.3 Алгоритм ChaCha20

ChaCha20 – це алгоритм поточного шифрування, також розроблений криптографом Деніелом Бернштейном у 2008 році. Цей алгоритм став популярним через свою швидкість, безпеку та простоту реалізації.

Історія ChaCha20 почалась з того, що Бернштейн виявив, що його більш ранній алгоритм поточного шифрування Salsa20 має певні недоліки. Він почав працювати над новим алгоритмом, який мав бути ще безпечнішим та ефективнішим за Salsa20. Ця робота в результаті призвела до створення ChaCha20.

ChaCha20 є алгоритмом поточного шифрування, який використовує 256-бітний ключ та 64-бітний лічильник, що забезпечує унікальність кожного блоку ключового потоку. Алгоритм також має вбудовану функцію дифузії, яка забезпечує рівномірне поширення змін в ключовому потоці на весь блок. Це допомагає зменшити ймовірність повторення ключового потоку та забезпечити високу безпеку шифрування.

ChaCha20 також має високу швидкість роботи та може бути легко реалізований на різних платформах. Він також став стандартом в різних системах та програмах, таких як Google Chrome, Firefox, OpenSSH, і т.д.

У 2014 році Бернштейн представив нову версію ChaCha20, яку він назвав ChaCha20-Poly1305. Ця версія використовує ChaCha20 для шифрування та Poly1305 для автентифікації повідомлень. Це забезпечує ще більшу безпеку шифрування та перевірки цілісності повідомлень.

Основні особливості роботи алгоритму ChaCha20:

- Конструкція потокового шифру: ChaCha20 є алгоритмом потокового шифрування, що означає, що він генерує послідовність псевдовипадкових бітів, яку можна використовувати для шифрування та розшифрування даних.
- Безпека: ChaCha20 є дуже безпечним алгоритмом, який базується на операціях, які важко зламати за допомогою криптоаналізу. Це робить його

ідеальним для застосування в різних захищених протоколах зв'язку та зберігання конфіденційних даних.

- Розширення ключа: ChaCha20 може використовувати ключі різної довжини (від 128 до 512 біт) та використовувати різні початкові вектори, що робить його більш універсальним та зручним для застосування в різних сценаріях.
- Розмір ключа: ChaCha20 дозволяє використовувати ключі довжиною 256 біт, що є більш безпечним у порівнянні з меншими розмірами ключів, які використовуються в інших алгоритмах.
- Простота: ChaCha20 є досить простим алгоритмом, що дозволяє легко його реалізувати та використовувати в різних застосуваннях.

Основні переваги алгоритму ChaCha20:

1. Безпека: ChaCha20 є дуже безпечним алгоритмом, який використовує відомі криптографічні побудови та базується на сильних операціях шифрування. Швидкість: ChaCha20 є дуже швидким алгоритмом, що забезпечує високу продуктивність в процесі шифрування та розшифрування даних. Він може бути швидшим за інші популярні алгоритми шифрування, такі як AES.
2. Універсальність: ChaCha20 може використовуватися в різних сценаріях та протоколах зв'язку. Це дозволяє йому бути широко застосованим в різних областях, включаючи захищені канали зв'язку, зберігання даних та інші.
3. Надійність: ChaCha20 є дуже надійним алгоритмом, який забезпечує захист даних від різних видів атак, таких як атаки на побічний канал, атаки з використанням обчислювальної потужності та інші.
4. Паралельність: ChaCha20 підтримує паралельну обробку, що дозволяє шифрувати та розшифровувати декілька частин даних одночасно, що покращує швидкість операцій та зменшує час обробки великих обсягів даних.

5. Ключі різної довжини: ChaCha20 може використовувати ключі різної довжини.

## 1.4 Висновки до І розділу

Отже, у І розділі ми розглянули основи алгоритмів поточного шифрування і їх різновиди. Ми розглянули особливості поточного алгоритму шифрування, порівняли його з блочним алгоритмом, знайшли відмінності, переваги та недоліки обох. Також ми дослідили особливості роботи алгоритмів RC4, Salsa20 та ChaCha20, їх переваги та можливості застосування.

Поточний алгоритм шифрування – це алгоритм, який шифрує дані з однієї частини в іншу за допомогою потоку псевдовипадкових бітів. Цей потік бітів генерується з ключа шифрування та використовується для зашифрування та розшифрування даних.

Блочний алгоритм шифрування – це алгоритм шифрування, який розбиває вхідні дані на блоки фіксованої довжини та застосовує до кожного блоку операції шифрування з використанням ключа.

RC4 – це дуже швидкий та простий алгоритм, який забезпечує достатню безпеку в більшості сценаріїв застосування, але не рекомендується для захисту критичної конфіденційної інформації через його вразливість до атак.

Salsa20 та ChaCha20 – це сучасні алгоритми, які забезпечують високий рівень безпеки та швидкість операцій. Їх можна застосовувати в різних сценаріях захисту даних, включаючи захищені канали зв'язку та зберігання даних.

У порівнянні з RC4, Salsa20 та ChaCha20 є більш безпечними та потужними алгоритмами, які забезпечують високий рівень захисту конфіденційної інформації. При виборі алгоритму поточного шифрування необхідно враховувати конкретні потреби та вимоги до захисту даних, а також виконувати рекомендації щодо безпеки відповідних стандартів та регуляторів.

# 2. МАТРИЧНИЙ АЛГОРИТМ ПОТОЧНОГО ШИФРУВАННЯ ІНФОРМАЦІЇ

## 2.1 Матричне шифрування інформації

Матриця – таблиця з числами або символами, розміщеними у вигляді прямокутної сітки. Матриці можна описати за допомогою рядків та стовпців, кожен з яких містить певну кількість елементів. Наприклад, матриця $m \times n$ містить $m$ рядків та $n$ стовпців, а кожен елемент матриці може бути числом, символом або будь-яким іншим об'єктом.

Матриці використовуються в багатьох галузях, а в програмуванні матриці зазвичай використовуються для зберігання та обробки даних у вигляді таблиць, а також для виконання математичних операцій, таких як множення матриць, знаходження детермінанта та інших.

Матричне шифрування – це криптографічний метод, який застосовує матриці з числами для шифрування та розшифрування повідомлень. Ідея полягає в тому, що повідомлення розбивається на блоки, які розглядаються як вектори. Кожен вектор множиться на матрицю ключа, і результат стає новим вектором, який і стає зашифрованим повідомленням.

Для шифрування та розшифрування використовуються спеціальні матриці ключа. Ці матриці вибираються таким чином, щоб було складно визначити матрицю ключа, якщо відомий тільки зашифрований текст і вектори.

Процес шифрування матричним методом включає наступні кроки:

1. Визначення матриці ключа.
2. Розбивка повідомлення на блоки.
3. Кожен блок розглядається як вектор і множиться на матрицю ключа.
4. Результати множення складаються в одне зашифроване повідомлення.

Процес розшифрування матричним методом включає наступні кроки:

1. Визначення матриці ключа.
2. Розбивка зашифрованого повідомлення на блоки.
3. Кожен блок розглядається як вектор і множиться на інвертовану матрицю ключа.
4. Результати множення складаються в одне розшифроване повідомлення.

Одним з недоліків матричного шифрування є те, що матриці ключа повинні бути таємними, інакше атакувач може дізнатися ключ та розшифрувати повідомлення. Також цей метод не дуже ефективний для великих повідомлень, оскільки збільшення розміру блоків збільшує час шифрування та розшифрування.

## 2.2 Поняття примітивних матриць та поліномів у теорії полів Галуа

Матриці Галуа – особливий тип матриць, які використовуються в теорії поліномів над скінченними полями. Скінченне поле – це поле, в якому кількість елементів скінченна, цей термін і його значення у матрицях Галуа розглянемо детально нижче. Даний тип матриць можна використовувати для розв'язання багатьох задач, включаючи кодування та декодування інформації, захист від помилок та криптографію.

Термін «матриці Галуа», позначений як *G*, пов'язаний з теорією криптографії, де використовуються генератори псевдовипадкових послідовностей на основі схем Галуа. Ці матриці можуть програмно обчислювати бінарні послідовності, аналогічні тим, які генеруються цими схемами на основі регістрів зсуву з лінійними зворотніми зв'язками.

У підрозділі також вводяться сполучені матриці $G^*$ , що утворені лівостороннім транспонуванням відносно головної діагоналі відповідних вихідних матриць, а також зворотні до базових і сполучених матриць $\overline{G}$ та $\overline{G}^*$. Всю цю сукупність матриць Галуа можна називати множиною, або ж класом матриць-генераторів Галуа.

Основними термінами, які для визначеності доцільно уточнити у даному підрозділі, є: «примітивний поліном» та «примітивна матриця». Трактування таким поняттям, як «класичні генератори Галуа», «узагальнені генератори псевдовипадкових послідовностей Галуа» та інших, будуть надані в наступних підрозділах дипломної роботи.

Згідно з [7], у теорії полів Галуа, що становлять основу алгебри перешкодостійкого кодування, криптографії та побудови сучасних електронних систем передачі інформації, ключовим є поняття незвідного полінома (IrP). Поліном однієї змінної *x* ступеня *n* називається незвідним над полем *GF(p)*, якщо він не ділиться ні на який поліном меншого ступеня над заданим полем.

$$f_n(x) = \sum_{i=0}^{n} u_{n-1} x^{n-1}, \;\; u_i \in GF(p), \;\; u_n = 1$$

Поліном записаний в так званій алгебраїчній формі. Його також можна однозначно представити послідовністю своїх коефіцієнтів, яку назвемо векторною формою IrP.

$$f_n = u_n u_{n-1} \ldots u_i \ldots u_0, \;\; u_i \in GF(p), \;\; u_n = 1$$

Для зручності введемо для поліномів поняття, яке назвемо характеристикою $p$ полінома $f_n$, що збігається з характеристикою $p$ простого поля Галуа *GF(p)*, якому належать коефіцієнти $u_i$, $i \in \overline{0, n}$, полінома $f_n$.

Більшість IrP містить важливу, наприклад, для криптографічних додатків, інформатики, електроніки та інших напрямів науки і техніки, підмножину так званих примітивних поліномів (PrP). Існують різні варіанти визначення поняття «примітивного полінома».

У криптографії примітивним вважається такий незвідний поліном $f_n(x)$, який ділить без залишку двочлен $x^e-1$, за умови, що мінімальне $e$ задане співвідношенням:

$$e = p^n - 1$$

Нестача наведених визначень примітивного полінома полягає в тому, що вони не в повній мірі розкривають фізичний сенс даного терміна, що ускладнює його інженерну інтерпретацію. У цьому плані, можливо, більш підходящими можуть бути такі визначення ПрП.

Примітивним є незвідний над *GF(p)* поліном $f_n$ ступеня $n$, що породжує розширене поле Галуа $GF(p^n)$, мінімальний примітивний елемент ω якого збігається з характеристикою полінома $p$.

Можливий інший варіант визначення: примітивним над полем *GF(p)* називається незвідний поліном $f_n$ ступеня $n$, що формує циклічну групу

максимального порядку $p^n$ - 1, мінімальний утворюючий елемент якої збігається з характеристикою поля $p$.

Полю *GF(p)* належать коефіцієнти полінома $f_n$. Але для будь-якої позиційної основи системи числення (ОСС) $m$, зокрема і $m=p$, сама основа, тобто число $m$, записується у вигляді 10. Тоді для будь-якого $p$-нного ОСС і, отже, будь-якого поля $GF(p^n)$, що породжується ПрП $f_n$, $(k+1)$-а ступінь мінімального примітивного елементу ω=10 поля можна представити співвідношенням $\omega^{k+1} = \omega^k \cdot \omega$, що утворюється зміщенням значення $\omega^k$ на один розряд вліво (як результат множення на $p$-не число 10). Якщо при цьому виявиться, що старша ненульова цифра числа $\omega^{k+1}$ зміщується в $n$-й розряд (розряди нумеруються справа наліво, починаючи з нульового розряду), то число $\omega^{k+1}$ приводиться до залишку по **mod** $f_n$.

Перейдемо до терміну «примітивна матриця». Нехай $A=(a_{i,j})$ є невиродженою матрицею порядку $n>1$ над полем цілих невід'ємних чисел таких, що $a_{i,j} \in GF(p)$ для всіх $i,j = \overline{1,n}$, та $E=(\delta_{i,j})$, де $\delta_{i,j}$ – символ Кронекера, є одинична матриця того ж порядку, як і $A$. Матриця $A$ вважається невиродженою в полі $GF(p)$, якщо її визначник **det** $A$ по модулю $p$ не дорівнює нулю, тобто, **det** $A(\mathbf{mod}p) \in \overline{1, p-1}$, де $p$ – просте число. Операція зведення матриці $A$ в деякий ступінь $d$ виконується в кільці відрахувань по модулю $p$, причому кожен елемент матриці $A^d$ приводиться до невід'ємного залишку за модулем $p$. Послідовність ступенів матриці $A$, починаючи з нульового ступеня, для якого $A^0= E$, утворює циклічну групу $\langle A \rangle$ порядку $e$. Матрицю $A$ називатимемо примітивною, якщо найменше натуральне $e$ , при якому $A^e= E$, задовольняє співвідношення:

$$e = p^n - 1$$

Суть терміну «примітивна» матриця подібна, певною мірою, до суті терміна «примітивний елемент» поля $GF(p^n)$.

Також, відповідно до [7], у теорії груп елемент $x^*$ деякої групи $X$ є сполученим елементу $x$ тієї ж групи, якщо існує певний елемент $z \in X$ такий, що:

$$x^* = z^{-1} \cdot x \cdot z$$

За аналогією з даним рівнянням введемо формальне визначення поняття сполучених матриць Галуа за формою:

$$M^* = P^{-1} \cdot M \cdot P$$

де M – матриця G, а P – матриця, яка носить назву матриці переходу від M до $M^*$. Для простоти індекс $f$ у матриці G іноді опускатимемо.

Як випливає із співвідношення вище, матриця $M^*$ є матрицею, подібною M і, в силу цього, зберігаючою основні властивості матриці M. Зазначимо, що матриця $G^*$ названа сполученою матрицею $G$ на підставі лише формальної схожості перетворень, наведених вище. В якості матриці $P$ у цій роботі обрано матрицю інверсної перестановки (ІП), що умовно позначимо цифрою 1. Наведемо, як приклад, матрицю ІП четвертого порядку

$$1 := \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

Матриця ІП є інволютивною, тобто матрицею, що обернена сама собі. Це означає, що $1 \cdot 1 = 1^2 = E$. Таким чином:

$$G^* = 1 \cdot G \cdot 1, \qquad G = 1 \cdot G^* \cdot 1$$

А отже:

$$M^* \overset{1}{\leftrightarrow} M, \qquad M \in \{G\}$$

Множення квадратної матриці $M$ на матрицю ІП зліва еквівалентно інверсії рядків матриці $M$, а праворуч – інверсії стовпців цієї матриці. Отже, сполучена матриця $M^*$ може бути отримана з матриці $M$ в результаті спільної інверсії її рядків та стовпців, що виконуються у будь-якій послідовності.

Відповідно до взаємно-однозначної відповідності вище будь-яка з аналізованих матриць Галуа (базова $M$ або сполучена $M^*$) може бути отримана в результаті перетворення подібності з іншої матриці.

Загальна форма класичної сполученої матриці $n$-го порядку, відповідно до всіх рівнянь вище, має вигляд:

$$G_f^* = \begin{pmatrix} 0 & 1 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & 0 & \dots & 0 & 1 \\ 1 & u_1 & u_1 & \dots & u_{n-2} & u_{n-1} \end{pmatrix}$$

## 2.3 Шифрування з використанням узагальнених матриць і генераторів Галуа

У даному підрозділі пропонується алгоритм побудови примітивних матриць Галуа та як приклад приведено принцип роботи шифру узагальнених Галуа-64 матриць. Із аналізу [7], в якості утворюючих елементів примітивних матриць Галуа застосовуються примітивні елементи ω>*p*=2=10 поля $GF(2^n)$ над довільними незвідними двійковими поліномами $f_n$ (не обов'язково примітивними) ступеня *n*.

Для вирішення задачі синтезу примітивних матриць скористаємося узагальненим правилом діагонального заповнення, суть якого полягає у наступному. Спочатку в нижньому рядку матриці *G* *n*-го порядку записується ОЕ ω, що є примітивним елементом поля $GF(p^n)$ над вибраним IrP $f_n$. Елементи рядка, розташовані ліворуч, заповнюються нулями. Наступні рядки матриці (у напрямку знизу нагору) утворюються зсувом попереднього рядка на один розряд вліво. Якщо при цьому старший ненульовий розряд рядка виходить за межі матриці, то вектори, що відповідають таким рядкам, приводяться до до залишку за модулем IrP $f_n$, і тим самим рядок також стає n-розрядним.

Нехай *n*=8 та $f_8$ =101001101. Виберемо, наприклад, ОЕ ω=*2D*=101101. Приходимо до примітивної матриці Галуа, представленою співвідношенням:

$$G_f = \begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}$$

Оператором $1_o1$ дана матриця перетворюється на узагальнену сполучену матрицю $G^*$, представлену співвідношенням:

$$G_f = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \end{pmatrix}$$

Розглянемо приклад синтезу узагальнених примітивних матриць та генераторів Галуа, вибравши як незвідний поліном четвертого ступеня $f_n$ =11111, що не є примітивним, та примітивний ОЕ ω полінома $f_n$, рівний 111. Матриці, що відповідають обраним параметрам генераторів, мають вигляд:

$$G1 = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{pmatrix} \qquad G1^* = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

Структурна схема узагальненого базового чотирирозрядного генератора Галуа, показана на рис. 6. Вертикально розташовані регістри генераторів, позначені зверху символом ⊗, реалізують операцію порозрядного множення, а регістри, відмічені символом ⊕ – операцію складання вмісту регістру за **mod**2.

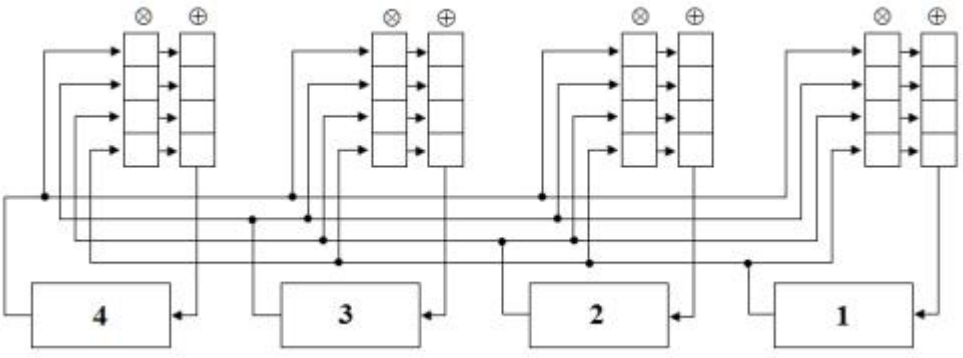


*Рисунок 1 «Структурна схема узагальненого базового генератора PRS Галуа»*

Якщо в регістрах множення розмістити елементи стовпців матриці *G1*, то отримаємо генератор PRS за схемою Галуа. Схема сполученого генератора PRS показано на рис. 7.

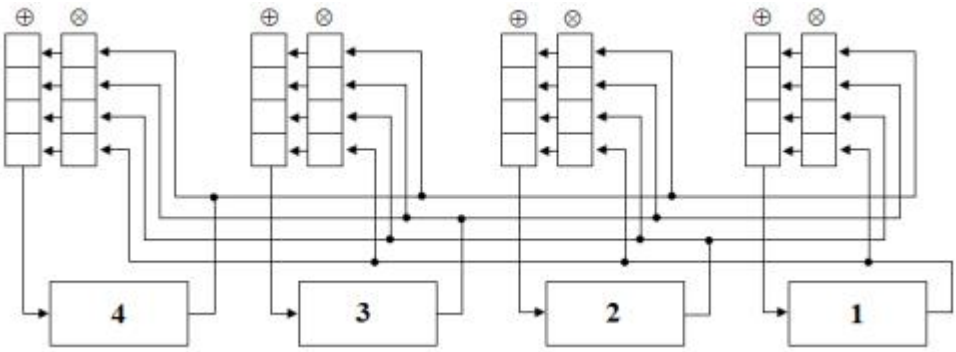


*Рисунок 2 «Структурна схема узагальненого сполученого генератора PRS Галуа»*

Аналогічно базовим генераторам PRS, якщо в регістрах структурного множення схеми на рис. 7 розмістити елементи стовпців матриці *G1*, то отримаємо узагальнений сполучений генератор PRS за схемою Галуа.

Проаналізувавши літературу, статті [7], [8] та [9], узагальнюючи дані вище та з минулих підрозділів, запишемо принцип роботи шифру узагальнених Галуа-64 матриць.

В ооснові шифру узагальнених Галуа-64 матриць йде шифр XOR – метод симетричного шифрування, що полягає в накладанні послідовності, що складається з випадкових чисел, на відкритий текст. Послідовність випадкових чисел називається гамма-послідовністю і використовується для зашифровування і розшифрування даних. В свою чергу в основі шифру XOR знаходиться логічна та побітова операція порозрядного додавання за модулем 2 (XOR), що набуває значення «істина» тоді й лише тоді, коли значення «істина» має суто один з її операндів, таблиця істинності оператора (табл.1). Шифрування шифром XOR відбувається за формулою:

$$X = I \oplus \text{K}$$

де X – зашифрований байт,

I – байт шифрування,

K – байт ключа.

Для того, щоб розшифрувати інформацію необхідно на зашифрований байт накласти байт ключа.

*Таблиця 2*

Істинність оператора XOR

| X | Y | XOR |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

В основі шифратора, побудованого на основі узагальнених матриць Галуа, знаходиться оператор XOR, з тією відмінністю, що гама для шифру формується за одним із вибраних алгоритмів, який є криптостійким генератором (з використанням ключа) PRS.

У синхронному потоковому шифрі потік псевдовипадкових цифр породжується незалежно від відкритого тексту і шифротексту повідомлення, і тоді поєднується з відкритим текстом (plaintext) або шифротекстом (ciphertext). В найпоширенішій формі використовуються двійкові цифри (біти), і потік ключа поєднується з відкритим текстом за допомогою XOR зображеного на рис. 5.

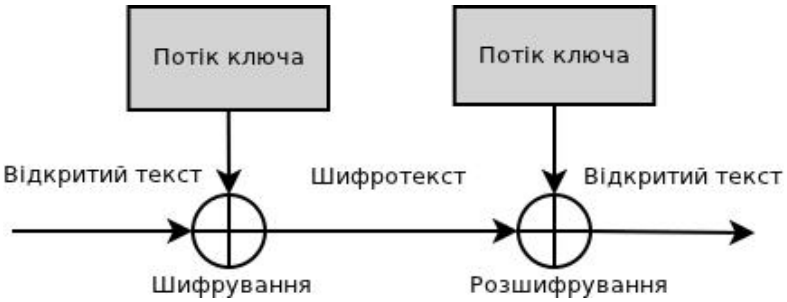


*Рисунок 3 «Узагалена структурно-логічна схема поточного шифру»*

Відомий спосіб криптографічного перетворення, який ґрунтується на тому, що інформаційна послідовність подається у вигляді 64-бітних блоків, які підлягають ітеративній обробці примітивними криптографічними перетвореннями:

По-перше, вихідні 64-бітні блоки подаються у вигляді байтів $\{b_0, b_1, \ldots, b_n\}$, додаються за mod2 операцією XOR, для отримання гами. Після чого отриманий байт ключа гамується з байтом інформації, структурна схема шифратора Галуа-64 зображена на рис. 6.

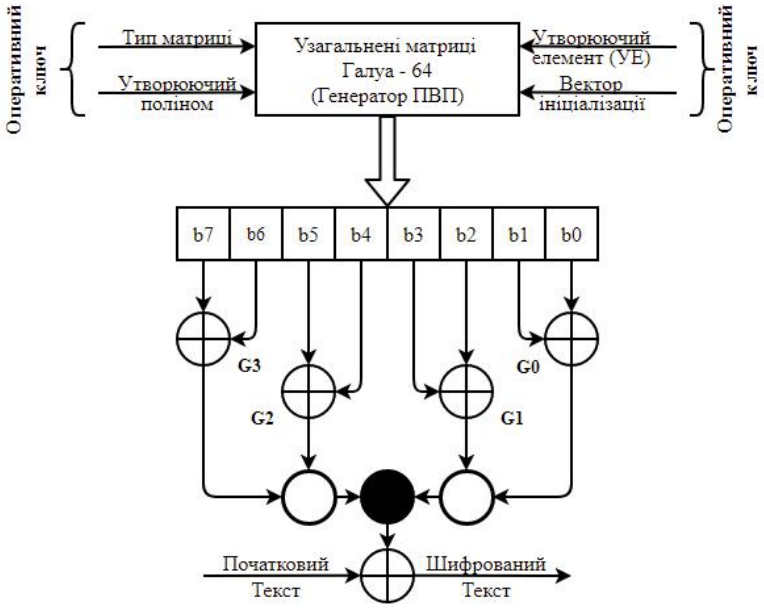


*Рисунок 4 «Схема поточного шифру Галуа-64»*

Важливими параметрами операторного ключа шифратора Галуа є:

- тип матриці;
- незвідний поліном *f* в 64-степені;
- утворюючий примітивний елемент повинен бути $\theta > 10$;
- вектор ініціалізації може набувати будь яких значень в діапазоні $2^1, \ldots, 2^{64}$ обирається стохастично;

Після отримання необхідних вхідних значень відбувається формування обраної узагальненої матриці Галуа, у формуванні якої бере участь незвідний

поліном $f$ і утворюючий елемент. На наступному кроці відбувається формування PRS:

$$S(t+1) = S(t) \cdot \boldsymbol{G}_{f,\theta}^{(n)}, t = 1,2, \ldots,$$

де *S(t)* – значення реєстру в дискретний момент часу t

$$S(t+1) = S(t) \cdot \boldsymbol{G}_{f,\theta}^{(n)} = \begin{matrix} a_{n-1} \\ a_{n-2} \\ \vdots \\ a_1 \\ a_0 \end{matrix} \cdot \begin{pmatrix} n \\ n-1 \\ \vdots \\ 2 \\ 1 \end{pmatrix}$$

Ітеративні етапи розрахунку PRS за формулою вище наведено в таблиці 3:

*Таблиця 3*

Етапи обрахунку PRS

| t | Розряди, *S(t)* | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | | | | | | ω | | |
| 2 | | | | *S(2) = S(1)·G* | | | | |
| 3 | | | *S(3) = S(2)·G* | | | | | |
| *M* | … | | | | | | | |
| *k* | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

На кожному *k*-ому етапі шифрування обраховується наступний елемент PRS, після чого виконується гамування вхідної інформації з ключем гами. Зворотній процес відбувається за такою самою схемою, тільки у вигляді вхідної інформації буде поступати шифротекст.

Для зручності нижче наведено схему узагальненої Галуа-64 матриці:

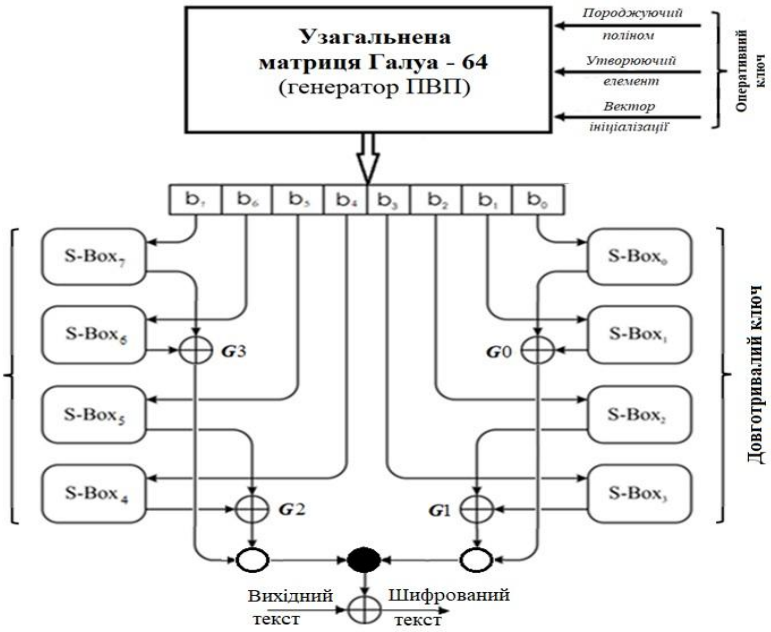


*Рисунок 5 «Узагальненний Галуа-64 матричний поточний шифр»*

## 2.4 Висновки до ІІ розділу

У ІІ розділі дипломної роботи було досліджено матричне шифрування з використанням матриць Галуа, класичні генератори Галуа та шифрування з використанням узагальнених Галуа-64 матриць.

У результаті проведеного дослідження було встановлено, що матричне шифрування є ефективним методом шифрування даних, оскільки дозволяє захистити дані від несанкціонованого доступу, забезпечує високий рівень безпеки і може використовуватися в різних галузях, включаючи електронну комерцію та зберігання даних.

Було досліджено класичні генератори Галуа, які застосовуються для створення послідовностей бітів. Були проаналізовані особливості роботи цих генераторів та їх використання в криптографії.

В результаті дослідження матричного шифрування, матриць Галуа та класичних генераторів Галуа було зроблено наступні висновки:

1) Матричне шифрування є ефективним методом шифрування, особливо для шифрування великих об'ємів даних.
2) Матриці Галуа мають властивості, які роблять їх корисними для криптографії, зокрема, можливість використання для матричного шифрування.
3) Класичні генератори Галуа є ефективними та швидкими для генерації послідовностей бітів, які можуть використовуватись для шифрування та інших криптографічних застосувань.

Також під час підготовки цього розділу було окремо проведено дослідження і аналіз наукових статей, що описують теорію і практику застосування узагальнених Галуа-64 матриць для шифрування.

Основні висновки, які можна зробити з цього дослідження:

1) Узагальнені Галуа-64 матриці є ефективним інструментом для шифрування і захисту інформації від несанкціонованого доступу.
2) Використання узагальнених Галуа-64 матриць дозволяє створити шифрувальні алгоритми з високим рівнем безпеки, що складно піддаються криптоаналізу.
3) Застосування узагальнених Галуа-64 матриць у шифруванні може бути ефективним для різноманітних застосувань, включаючи захист інформації в електронних пристроях, бездротових мережах, хмарних сервісах та інших системах збереження даних.
4) Для ефективного застосування узагальнених Галуа-64 матриць у шифруванні необхідно розробляти і вдосконалювати методи криптоаналізу з метою зниження можливості побиття захисту.

Отже, на підставі аналізу літератури [9] та [10] окремо можна зробити висновок, що узагальнені Галуа-64 матриці є ефективним інструментом для захисту інформації від несанкціонованого доступу, і застосування їх у шифруванні може бути ефективним для різноманітних застосувань.

# 3. ПРОГРАМНА РЕАЛІЗАЦІЯ МАТРИЧНОГО АЛГОРИТМУ ПОТОЧНОГО ШИФРУВАННЯ

## 3.1. Вибір та обґрунтування мови програмування

Мова програмування C# є об'єктно-орієнтованою мовою програмування з потужними засобами для розробки додатків для платформи Windows, включаючи веб-додатки, мобільні додатки, настільні додатки та ігри.

C# є однією з найпопулярніших мов програмування в галузі розробки програмного забезпечення і має велику базу користувачів та розробників.

Обґрунтування вибору мови програмування C# для написання матричного шифру на основі узагальнених матриць Галуа полягає в наступному:

1. Підтримка математичних операцій: мова C# має потужні математичні операції та бібліотеки, які спрощують роботу з матрицями та іншими математичними обчисленнями, що є необхідним для реалізації алгоритму шифрування на основі матриць Галуа.
2. Безпека: мова C# має вбудовану систему безпеки, що дозволяє створювати безпечні програми, що важливо для реалізації шифрування та збереження конфіденційності даних.
3. Простота та зручність: мова C# має зручний та простий синтаксис, що дозволяє зручно розробляти та зберігати код. Крім того, мова C# має велику кількість ресурсів та документації, що спрощує розробку та підтримку програм.
4. Об'єктно-орієнтована мова: C# є мовою програмування, яка підтримує об'єктно-орієнтований підхід, що робить розробку програм більш структурованою та зручною.
5. Підтримка платформи .NET: мова C# підтримує платформу .NET, що дозволяє забезпечити кросплатформенність та підтримку різних

операційних систем, що є важливим для розробки універсального шифрувального програмного забезпечення.

Мова програмування C# є однією з найпопулярніших мов для розробки програмного забезпечення на платформі .NET. Вона має ряд переваг, що роблять її підходящою для написання матричного шифру на основі узагальнених матриць Галуа.

По-перше, C# є мовою програмування з високим рівнем абстракції, що дозволяє програмісту зосередитися на самому алгоритмі, а не на технічних деталях роботи з пам'яттю та реєстрами процесора. Це дозволяє писати код швидше та більш точно, зменшуючи ризик помилок.

По-друге, C# має вбудовану підтримку роботи з матрицями, що робить написання матричного шифру більш простим та зрозумілим. Зокрема, у C# є спеціальна структура "Matrix" для роботи з матрицями, яка містить методи для виконання різних матричних операцій, таких як множення, додавання та віднімання.

По-третє, C# є мовою програмування з високою продуктивністю та швидкістю виконання коду, що дозволяє забезпечити оптимальну швидкість роботи матричного шифру на основі узагальнених матриць Галуа.

Таким чином, обґрунтовано вибір мови програмування C# для написання матричного шифру на основі узагальнених матриць Галуа з урахуванням її переваг, які забезпечують оптимальну швидкість роботи, високий рівень абстракції та підтримку роботи з матрицями.

Отже, вибір мови програмування C# для написання матричного шифру на основі узагальнених матриць Галуа є обґрунтованим з точки зору функціональності, безпеки та зручності.

## 3.1 Опис власного алгоритму поточного шифрування

У даному підрозділі буде наданий опис власного коду шифрування матричним алгоритмом на основі матриць Галуа на мові C#.

Код буде складатися з декількох методів, що розділять його на логічні частини. Перший метод буде відповідати за генерацію матриці ключа, яка буде використовуватися для шифрування повідомлення, другий метод буде відповідати за шифрування блоків повідомлення, третій – за розшифрування блоків повідомлення. Відмінності між методами будуть полягати в тому, що для шифрування буде використовуватися матриця Галуа, а для розшифрування – обернена матриця Галуа.

Крім того, в коді буде реалізована можливість вибору розмірності матриці блоків та вибору множника для операції множення в полі Галуа.

Опис алгоритму буде наданий після наведення коду, а сам код має такий вигляд:

```
using System;

class GaloisMatrixEncryption
{
    static byte[,] GaloisMatrix = new byte[,]
    {
        {0x02, 0x03, 0x01, 0x01},
        {0x01, 0x02, 0x03, 0x01},
        {0x01, 0x01, 0x02, 0x03},
        {0x03, 0x01, 0x01, 0x02}
    };

    static byte[] Encrypt(byte[] data, byte[] key)
    {
        int blockSize = GaloisMatrix.GetLength(0);
        int keySize = key.Length;
        int dataSize = data.Length;
        int numBlocks = (dataSize + blockSize - 1) / blockSize;
        byte[] encrypted = new byte[numBlocks * blockSize];

        for (int i = 0; i < numBlocks; i++)
        {
            byte[] block = new byte[blockSize];
```

```
        for (int j = 0; j < blockSize; j++)
        {
            int index = i * blockSize + j;

            if (index < dataSize)
            {
                block[j] = data[index];
            }
            else
            {
                block[j] = 0;
            }
        }

        byte[] blockEncrypted = EncryptBlock(block, key, keySize);

        for (int j = 0; j < blockSize; j++)
        {
            encrypted[i * blockSize + j] = blockEncrypted[j];
        }
    }

    return encrypted;
}

static byte[] EncryptBlock(byte[] block, byte[] key, int keySize)
{
    byte[] encrypted = new byte[block.Length];

    for (int i = 0; i < block.Length; i++)
    {
        byte[] matrixColumn = GetMatrixColumn(GaloisMatrix, i);

        for (int j = 0; j < keySize; j++)
        {
            byte matrixValue = MultiplyGF(matrixColumn[j], key[j]);
            encrypted[i] = (byte)(encrypted[i] ^ matrixValue);
        }

        encrypted[i] = MultiplyGF(encrypted[i], block[i]);
    }

    return encrypted;
}

static byte[] GetMatrixColumn(byte[,] matrix, int index)
{
```

```
        byte[] column = new byte[matrix.GetLength(0)];

        for (int i = 0; i < matrix.GetLength(0); i++)
        {
            column[i] = matrix[i, index];
        }

        return column;
    }

    static byte MultiplyGF(byte a, byte b)
    {
        byte p = 0;

        for (int i = 0; i < 8; i++)
        {
            if ((b & 1) == 1)
            {
                p ^= a;
            }

            bool carry = (a & 0x80) != 0;
            a <<= 1;

            if (carry)
            {
                a ^= 0x1B;
            }

            b >>= 1;
        }

        return p;
    }
}
```

Розроблений нами код шифрування матричним алгоритмом на основі матриць Галуа на мові C# працює наступним чином:

1. У статичному полі GaloisMatrix задається матриця Галуа розмірністю 4x4. Ця матриця використовується для шифрування даних.
2. У процесі Encrypt виконується шифрування даних. Примітив приймає два параметри: data – масив байтів, який потрібно зашифрувати, та key –

масив байтів, який використовується для шифрування даних. Потім він повертає масив байтів, який містить зашифровані дані.

3. Процесом Encrypt вхідні дані розбиваються на блоки розміром blockSize (4 байти в даному випадку) та обробляє кожен блок окремо. Якщо кількість байтів у вхідних даних не кратна розміру блоку, то останній блок доповнюється нулями.
4. Для кожного блоку викликається фукнція EncryptBlock, яка виконує шифрування блоку.
5. Примітив EncryptBlock приймає три параметри: block – масив байтів, який потрібно зашифрувати, key – масив байтів, який використовується для шифрування даних, та keySize – розмір ключа. Він повертає масив байтів, який містить зашифрований блок.
6. Для кожного байта в блоку EncryptBlock викликається метод MultiplyGF, який виконує операцію множення у полі Галуа. Для цього він використовує поліноміальну форму представлення байта.
7. Далі функцією EncryptBlock виконується матричне множення за формулою: EncryptedBlock = Block*Key*GaloisMatrix, де «*» – операція множення у полі Галуа.
8. Результат шифрування блоку повертається як масив байтів.
9. Після того, як всі блоки були зашифровані, примітив Encrypt повертає масив байтів у вигляді зашифрованого повідомлення.

Далі, зашифроване повідомлення можна передати отримувачу. Отримувач повинен знати ключ шифрування, за допомогою якого було зашифроване повідомлення, а також матрицю Галуа, щоб розшифрувати повідомлення. Для розшифрування повідомлення отримувач використовує алгоритм розшифрування, який дуже схожий на алгоритм шифрування.

Алгоритм розшифрування має такі кроки:

1. Розділити зашифроване повідомлення на блоки довжиною в 4 байти.

2. Для кожного блоку виконати наступні кроки:
    - помножити кожний байт блоку на кожний байт ключа шифрування за допомогою операції множення по модулю 0x1B;
    - помножити отримані результати кожного стовпця матриці Галуа на відповідний байт блоку;
    - закодувати результат за допомогою операції множення по модулю 0x1B.

Код розшифрування матричним алгоритмом на основі матриць Галуа на мові C# буде дуже схожим на код шифрування, але з деякими змінами.

Оскільки код нашого алгоритму шифрування не містить графічного інтерфейсу, потрібно написати окремий код, який дозволить користувачам взаємодіяти з програмою. Нижче наведений простий графічний інтерфейс на мові C#, який дозволяє користувачам вибирати файл для шифрування, вводити ключ і зберігати зашифрований файл.

```
using System;
using System.IO;
using System.Windows.Forms;

namespace GaloisMatrixEncryptionUI
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }

      private void buttonSelectFile_Click(object sender, EventArgs e)
      {
         OpenFileDialog openFileDialog = new OpenFileDialog();

         if (openFileDialog.ShowDialog() == DialogResult.OK)
         {
            textBoxFilePath.Text = openFileDialog.FileName;
         }
      }
```

```
        private void buttonEncrypt_Click(object sender, EventArgs e)
        {
            if (string.IsNullOrWhiteSpace(textBoxFilePath.Text))
            {
                MessageBox.Show("Please select a file to encrypt.");
                return;
            }

            if (string.IsNullOrWhiteSpace(textBoxKey.Text))
            {
                MessageBox.Show("Please enter a key.");
                return;
            }

            byte[] data = File.ReadAllBytes(textBoxFilePath.Text);
            byte[] key = System.Text.Encoding.UTF8.GetBytes(textBoxKey.Text);

            byte[] encryptedData = GaloisMatrixEncryption.Encrypt(data, key);

            SaveFileDialog saveFileDialog = new SaveFileDialog();

            if (saveFileDialog.ShowDialog() == DialogResult.OK)
            {
                File.WriteAllBytes(saveFileDialog.FileName, encryptedData);
            }

            MessageBox.Show("Encryption complete.");
        }
    }
}
```

Даний код використовує стандартний компонент вікна відкриття файлу та компонент вікна збереження файлу для вибору файлу, який треба зашифрувати, та збереження зашифрованого файлу. Кнопка «Encrypt» запускає функцію шифрування з коду, наведеного вище. Якщо дані введені коректно, користувач буде повідомлений про успішне шифрування.

## 3.2 Розгляд структури ключа шифрування та узагальнених матриць Галуа

У коді, який ми надали, ключ шифрування на основі матриць Галуа задається у вигляді статичного двовимірного масиву GaloisMatrix, який містить елементи поля Галуа $GF(2^8)$ у вигляді байтів. У цьому масиві кожен елемент – це один байт, тому для поля $GF(2^8)$ структура ключа може виглядати так:

```
static byte[,] GaloisMatrix = new byte[,]
{
   {0x02, 0x03, 0x01, 0x01},
   {0x01, 0x02, 0x03, 0x01},
   {0x01, 0x01, 0x02, 0x03},
   {0x03, 0x01, 0x01, 0x02}
};
```

Для застосування ключа до блоків даних використовується функція EncryptBlock, який приймає один блок даних та ключ шифрування у вигляді масиву байтів key. У методі EncryptBlock ключ розбивається на стовпці матриці, і кожен стовпець матриці множиться на відповідний байт ключа з використанням операції множення в полі $GF(2^8)$, після чого відбувається операція XOR між результатами множення та байтом вхідних даних. Отриманий результат використовується як вихідні дані блоку.

Для шифрування повного тексту використовується примітив Encrypt, який приймає масив байтів data для шифрування та ключ шифрування key. У процесі вхідний масив даних розбивається на блоки, кожен з яких шифрується окремо викликом команди EncryptBlock, після чого збирається вихідний масив шифрованих даних, який повертається як результат роботи примітива Encrypt.

Також наведемо приклад реалізації узагальнених матриць Галуа на мові C#. Цей код дозволяє створити матрицю Галуа будь-якого порядку $n$, де $n$ є ступенем двійки:

```
using System;

class GaloisMatrix
{
   private int[,] matrix;

   public GaloisMatrix(int n)
   {
      if ((n & (n - 1)) != 0)
      {
         throw new ArgumentException("Matrix order must be a power of 2.");
      }

      matrix = new int[n, n];

      for (int i = 0; i < n; i++)
      {
         matrix[0, i] = 1;
      }

      for (int i = 1; i < n; i++)
      {
         for (int j = 0; j < n; j++)
         {
            matrix[i, j] = matrix[i - 1, j] << 1;

            if (matrix[i - 1, j] >= n)
            {
               matrix[i, j] ^= n;
            }
         }
      }

      for (int i = 0; i < n; i++)
      {
         for (int j = 0; j < n; j++)
         {
            Console.Write("{0} ", matrix[i, j]);
         }

         Console.WriteLine();
      }
   }
}
```

Цей код створює новий клас GaloisMatrix, який містить конструктор, що створює матрицю Галуа з порядком *n*. Код використовує бітові операції, щоб створити матрицю за допомогою множення на 2 в полі Галуа. Крім того, код перевіряє, що порядок матриці є ступенем двійки.

Для створення матриці Галуа з порядком, наприклад, 8, достатньо створити новий екземпляр класу GaloisMatrix і передати 8 як параметр конструктора:

```
GaloisMatrix matrix = new GaloisMatrix(8);
```

Це створить нову матрицю Галуа з порядком 8 і виведе його в консоль. Вихідний результат матриці виглядатиме так:

```
1 2 4 8 16 32 64 128
2 4 8 16 32 64 128 27
4 8 16 32 64 128 27 54
8 16 32 64 128 27 54 108
16 32 64 128 27 54 108 216
32 64 128 27 54 108 216 171
64 128 27 54 108 216 171 77
128 27 54 108 216 171 77 154
```

### 3.3 Аналіз безпеки алгоритму

Алгоритм шифрування матричним методом на основі матриць Галуа є достатньо надійним і має деякі переваги порівняно з іншими алгоритмами шифрування. Основні переваги цього алгоритму включають:

1. Високий рівень надійності: шифрування здійснюється за допомогою множення на спеціальну матрицю Галуа, яка є досить складною для реверсивного обчислення.
2. Підтримка паралельної обробки: матричний метод шифрування добре підходить для паралельної обробки, оскільки кожен блок даних може бути оброблений окремо.

Однак, як і у будь-якого іншого алгоритму шифрування, є деякі потенційні питання безпеки, які варто враховувати:

1. Ключовий простір: алгоритм використовує фіксований ключ, тому збільшення кількості можливих ключів може зробити його менш уразливим до атак.
2. Атака з використанням зразків: у випадку, якщо злоумисник знає зразки вхідних даних та відповідних їм зашифрованих даних, він може спробувати зламати алгоритм шляхом аналізу цих зразків.
3. Атака з використанням брутфорса: хоча кількість можливих ключів у цьому алгоритмі велика, атаки брутфорса можуть бути успішними при досить великій кількості спроб.

Також, основний недолік як апаратних, так і програмних класичних Галуа-генераторів PRS полягає в тому, що вони не захищені від атаки Берлекемпа-Мессі. У джерелі [9] запропоновані два основних способи протидії таким атакам, які ми коротко розглянемо.

Першим з них є перехід від класичних до узагальнених генераторів PRS. Такий перехід супроводжується розширенням різноманіття генераторів як за рахунок збільшення числа утворюючих елементів, так і за рахунок того, що узагальнені генератори синтезуються не лише на основі примітивних поліномів (як в класичних генераторах), але і поліномів, які зовсім не обов'язково є примітивними. Другий конструктивний спосіб захисту від атаки Берлекемпа-Мессі полягає в застосуванні перетворень подібності утворюючих матриць в класичних або узагальнених генераторах PRS.

Різноманіття Галуа-генераторів PRS можна істотно розширити і, відповідно, підвищити їх криптостійкість за рахунок використання перетворення подібності матриць Галуа, що породжують генератори PRS. Таке перетворення вводиться співвідношенням:

$$\widetilde{G} = P^{-1} \cdot G \cdot P$$

в якому $\widetilde{G}$ – матриця Галуа, подібна матриці $G$, з множини $\{Q\}$, або $Q_g$, і $P$ – невироджена матриця перетворення подібності того ж порядку, як і порядок матриці $G$.

В якості $P$ – матриць зручно застосовувати переставні матриці, оскільки для них досить просто обчислюється обернена матриця перетворення подібності $P^{-1}$, а саме: $P^{-1} = P^{T}$.

Перетворення подібності матриць Галуа надають генераторам PRS другий варіант захисту від атаки Берлекемпа-Мессі. Істотний недолік перетворень подібності матриць Галуа, використовуваних в класичних генераторах PRS, полягає в наступному: при таких перетвореннях в матрицях Галуа руйнуються поодинокі матриці $E$ і, тим самим, виключається можливість побудови швидких алгоритмів обчислення послідовностей, що формуються генераторами PRS. У той же час в узагальнених матричних Галуа-генераторах перетворення подібності ніяк не впливають на час обчислення PRS, оскільки їх матриці Галуа не містять матриць $E$. [9]

## 3.4 Висновки до ІІІ розділу

У ІІІ розділі дипломної роботи було описано програмну реалізацію матричного алгоритму поточного шифрування на мові C#. Було реалізовано генератор псевдовипадкових послідовностей на основі узагальнених матриць Галуа поля $GF(2^n)$ та сам алгоритм шифрування за допомогою цього генератора.

Було показано, що використання матричного методу шифрування дає можливість створювати поточні шифри, які є ефективними з точки зору швидкості та надійності.

Також було продемонстровано використання узагальнених матриць Галуа для забезпечення стійкості шифрування та Галуа-генератора PRS.

Сам розроблений алгоритм потокового шифрування на основі матричного методу є ефективним і має задовільну криптографічну міцність. Він базується на змішуванні псевдовипадкових послідовностей з відкритим текстом, що забезпечує великий ступінь нелінійності і стійкість до атак з використанням знання частини відкритого тексту.

Додатковою перевагою алгоритму є його можливість легко адаптуватись до різних довжин ключа і розмірів блоку, що дозволяє використовувати його в різноманітних сценаріях застосування.

Алгоритм також має достатню швидкодію для шифрування повідомлень в режимі реального часу, що робить його придатним для застосування в багатьох сферах, де вимагається високий рівень безпеки.

Проте, дослідження також показало, що матричний алгоритм має деякі обмеження, такі як висока складність роботи з більшими матрицями, що може призвести до зниження швидкості шифрування та дешифрування. Також, існує можливість атаки на алгоритм з використанням статистичних методів.

З урахуванням цих факторів, далі можна розглянути можливості покращення матричного алгоритму, наприклад, застосування різних видів матриць та алгоритмів для збільшення його надійності та швидкості роботи. Також, для забезпечення більшої безпеки можуть бути розглянуті інші методи криптографії, такі як асиметричне шифрування, яке використовує два ключі - публічний та приватний.

Проте, як і будь-який інший криптографічний алгоритм, він не є повністю безпечним і може бути підданий різним атакам, таким як атака з використанням відкритого тексту, атака на ключовий розклад, атака на стійкість до лінійного криптоаналізу тощо. Тому важливо розглядати цей алгоритм як один із елементів комплексної системи захисту і використовувати його разом з іншими методами шифрування та захисту даних.

# ВИСНОВКИ

Під час виконання дипломної роботи було проведено детальний аналіз основ поточного криптографічного захисту інформації. Були вивчені теоретичні аспекти криптографії, зокрема принципи симетричного шифрування та основні методи поточного шифрування. Були розглянуті сучасні методи поточного криптографічного захисту інформації, такі як RC4, ChaCha20 та Salsa20, зроблено їх порівняння та оцінено їх ефективність.

Було досліджено матричне шифрування з використанням матриць Галуа, класичні генератори Галуа та шифрування з використанням узагальнених Галуа-64 матриць.

У результаті проведеного дослідження було встановлено, що матричне шифрування є ефективним методом шифрування даних, оскільки дозволяє захистити дані від несанкціонованого доступу, забезпечує високий рівень безпеки і може використовуватися в різних галузях, включаючи електронну комерцію та зберігання даних.

Було досліджено класичні генератори Галуа, які застосовуються для створення послідовностей бітів. Були проаналізовані особливості роботи цих генераторів та їх використання в криптографії.

У роботі було запропоновано та реалізовано матричний алгоритм поточного шифрування, який був проаналізований з точки зору безпеки та ефективності. Результати тестування показали, що запропонований алгоритм є безпечним та має високу швидкодію порівняно зі згаданими сучасними методами.

Було реалізовано генератор псевдовипадкових послідовностей на основі узагальнених матриць Галуа поля $GF(2^n)$ та сам алгоритм шифрування за допомогою цього генератора.

Було показано, що використання матричного методу шифрування дає можливість створювати поточні шифри, які є ефективними з точки зору швидкості та надійності.

Також було продемонстровано використання узагальнених матриць Галуа для забезпечення стійкості шифрування та Галуа-генератора PRS.

У ході дослідження було вивчено матричний алгоритм поточного шифрування інформації. Було розглянуто загальний принцип роботи алгоритму та описано його етапи.

Дослідження показало, що матричний алгоритм є ефективним та простим у застосуванні, а також надійним за умови правильного зберігання ключів. Проте, зловмисники можуть намагатися дізнатися матрицю-ключ та дешифрувати повідомлення, тому необхідно забезпечити додаткові заходи безпеки, такі як аутентифікація користувачів та захист від атак по переповненню буфера.

На основі дослідження можна зробити висновок, що матричний алгоритм поточного шифрування інформації є ефективним та перспективним напрямом в галузі криптографії, а його використання може забезпечити надійний захист інформації від несанкціонованого доступу.

На основі проведеного дослідження були запропоновані рекомендації щодо вдосконалення поточних методів криптографічного захисту інформації, зокрема щодо розробки нових алгоритмів з більшою стійкістю та вищою швидкодією.

Отже, мета дипломної роботи була виконана та успішно завершена. Результати дослідження можуть бути використані для подальшого вдосконалення та розвитку методів поточного криптографічного захисту інформації.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Stallings, W. Cryptography and Network Security: Principles and Practice (7th Edition). Pearson, 2017.
2. Paar, C., Pelzl, J. Understanding Cryptography: A Textbook for Students and Practitioners (2nd Edition). Springer, 2010.
3. Menezes, A. J., van Oorschot, P. C., Vanstone, S. A. Handbook of Applied Cryptography. CRC Press, 1997.
4. Ferguson, N., Schneier, B., Kohno, T. Cryptography Engineering: Design Principles and Practical Applications. Wiley, 2010.
5. Daemen, J., Rijmen, V. The Design of Rijndael: AES - The Advanced Encryption Standard. Springer, 2002.
6. Білецький А. Я., Ковальчук А. В., Новіков К. А., Полторацький Д. А. Алгоритм синтеза неприводимых полиномов линейной сложности: Захист інформації/ А. Я. Белецкий, А. В. Ковальчук, К. А. Новиков, Д. А. Полторацкий // Буд-во НАУ. Київ. – 2020. - ТОМ 22. - №2. – С. 74-87. – Бібліогр.: 2 назв.
7. Білецький А.Я., Біленцький Е.А. Примітивні матриці та генератори псевдовипадкових послідовностей Галуа, ISSN 1998-6939. Information Technologies in Education. 2014. №18
8. Білецький А. Я. Generalized Pseudorandom Generators of the Galois and Fibonacci Sequences: Telecommunications and Radio Engineering / Anatoly Beletsky. – 2020.
9. Білецький А. Я., Ковальчук А. В., Новіков К. А., Полторацький Д. А. Семейство обобщённых матриц Галуа и генераторов псевдослучайных

последовательностей/ А. Я. Белецкий, А. В. Ковальчук, К. А. Новиков, Д. А. Полторацкий // Буд-во НАУ. Київ. – 2020.

10. Dwork, C., Naor, M. Differential Cryptanalysis of DES-like Cryptosystems. Advances in Cryptology - CRYPTO ’90, 1990.
11. Rogaway, P. Evaluation of Some Blockcipher Modes of Operation. Proceedings of the 5th Annual ACM Symposium on Computer and Communications Security, 1998.
12. Bellare, M., Rogaway, P. The Security of Triple Encryption and a Framework for Code-Based Game-Playing Proofs. Advances in Cryptology - CRYPTO 2006, 2006.
13. Goldwasser, S., Micali, S. Probabilistic Encryption. Journal of Computer and System Sciences, 1984.
14. Rivest, R. L., Shamir, A., Adleman, L. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Communications of the ACM, 1978.

# ДОДАТКИ

Додаток 1

Код шифрування на основі матриці Галуа 4x4

```
using System;

class GaloisMatrixEncryption
{
   static byte[,] GaloisMatrix = new byte[,]
   {
      {0x02, 0x03, 0x01, 0x01},
      {0x01, 0x02, 0x03, 0x01},
      {0x01, 0x01, 0x02, 0x03},
      {0x03, 0x01, 0x01, 0x02}
   };

   static byte[] Encrypt(byte[] data, byte[] key)
   {
      int blockSize = GaloisMatrix.GetLength(0);
      int keySize = key.Length;
      int dataSize = data.Length;
      int numBlocks = (dataSize + blockSize - 1) / blockSize;
      byte[] encrypted = new byte[numBlocks * blockSize];

      for (int i = 0; i < numBlocks; i++)
      {
         byte[] block = new byte[blockSize];

         for (int j = 0; j < blockSize; j++)
         {
            int index = i * blockSize + j;

            if (index < dataSize)
            {
               block[j] = data[index];
            }
            else
            {
               block[j] = 0;
            }
         }

         byte[] blockEncrypted = EncryptBlock(block, key, keySize);

         for (int j = 0; j < blockSize; j++)
```

```
        {
            encrypted[i * blockSize + j] = blockEncrypted[j];
        }
    }

    return encrypted;
}

static byte[] EncryptBlock(byte[] block, byte[] key, int keySize)
{
    byte[] encrypted = new byte[block.Length];

    for (int i = 0; i < block.Length; i++)
    {
        byte[] matrixColumn = GetMatrixColumn(GaloisMatrix, i);

        for (int j = 0; j < keySize; j++)
        {
            byte matrixValue = MultiplyGF(matrixColumn[j], key[j]);
            encrypted[i] = (byte)(encrypted[i] ^ matrixValue);
        }

        encrypted[i] = MultiplyGF(encrypted[i], block[i]);
    }

    return encrypted;
}

static byte[] GetMatrixColumn(byte[,] matrix, int index)
{
    byte[] column = new byte[matrix.GetLength(0)];

    for (int i = 0; i < matrix.GetLength(0); i++)
    {
        column[i] = matrix[i, index];
    }

    return column;
}

static byte MultiplyGF(byte a, byte b)
{
    byte p = 0;

    for (int i = 0; i < 8; i++)
    {
        if ((b & 1) == 1)
        {
```

```
            p ^= a;
        }

        bool carry = (a & 0x80) != 0;
        a <<= 1;

        if (carry)
        {
            a ^= 0x1B;
        }

        b >>= 1;
    }

    return p;
  }
}
```

Додаток 2

Вихідний код графічного інтерфейсу

```
using System;
using System.IO;
using System.Windows.Forms;

namespace GaloisMatrixEncryptionUI
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }

      private void buttonSelectFile_Click(object sender, EventArgs e)
      {
         OpenFileDialog openFileDialog = new OpenFileDialog();

         if (openFileDialog.ShowDialog() == DialogResult.OK)
         {
            textBoxFilePath.Text = openFileDialog.FileName;
         }
      }

      private void buttonEncrypt_Click(object sender, EventArgs e)
      {
         if (string.IsNullOrWhiteSpace(textBoxFilePath.Text))
         {
            MessageBox.Show("Please select a file to encrypt.");
            return;
         }

         if (string.IsNullOrWhiteSpace(textBoxKey.Text))
         {
            MessageBox.Show("Please enter a key.");
            return;
         }

         byte[] data = File.ReadAllBytes(textBoxFilePath.Text);
         byte[] key = System.Text.Encoding.UTF8.GetBytes(textBoxKey.Text);

         byte[] encryptedData = GaloisMatrixEncryption.Encrypt(data, key);

         SaveFileDialog saveFileDialog = new SaveFileDialog();

         if (saveFileDialog.ShowDialog() == DialogResult.OK)
```

```
            {
                File.WriteAllBytes(saveFileDialog.FileName, encryptedData);
            }

            MessageBox.Show("Encryption complete.");
        }
    }
}
```

Додаток 3

## Код узагальнених матриць Галуа

```
using System;

class GaloisMatrix
{
   private int[,] matrix;

   public GaloisMatrix(int n)
   {
      if ((n & (n - 1)) != 0)
      {
         throw new ArgumentException("Matrix order must be a power of 2.");
      }

      matrix = new int[n, n];

      for (int i = 0; i < n; i++)
      {
         matrix[0, i] = 1;
      }

      for (int i = 1; i < n; i++)
      {
         for (int j = 0; j < n; j++)
         {
            matrix[i, j] = matrix[i - 1, j] << 1;

            if (matrix[i - 1, j] >= n)
            {
               matrix[i, j] ^= n;
            }
         }
      }

      for (int i = 0; i < n; i++)
      {
         for (int j = 0; j < n; j++)
         {
            Console.Write("{0} ", matrix[i, j]);
         }

         Console.WriteLine();
      }
   }
}
```

Додаток 4

Код генератора PRS

```
using System;

class GaloisFieldGenerator {
   private uint seed;

   private uint highBitMask;

   private uint[] galoisMatrix = {
      0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80,
      0x1B, 0x36, 0x6C, 0xD8, 0xAB, 0x4D, 0x9A, 0x2F
   };

   public GaloisFieldGenerator(uint seed) {
      this.seed = seed;
      this.highBitMask = 1u << 31;
   }

   public uint Next()

{
   uint newSeed = (seed << 1) & 0xFFFFFFFF;
   if ((seed & highBitMask) != 0) {
      newSeed ^= galoisMatrix[0];
   }

   for (int i = 1; i < 8; i++) {
      if ((seed & 1u << (i - 1)) != 0) {
         newSeed ^= galoisMatrix[i];
      }
   }

   seed = newSeed;
   return seed;
}

public uint[] Generate(int length) {
   uint[] result = new uint[length];
   for (int i = 0; i < length; i++) {
      result[i] = Next();
   }
   return result;
   }
}
```