МАГІСТЕРСЬКА РОБОТА

МР.КІ-25.00.00.000 ПЗ

Група КІм-24-2

Грицан Микола

2025

Міністерство освіти і науки України

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра комп'ютерних систем і мереж

Грицан Микола Іванович
(прізвище, ім'я, по батькові)

УДК 004. 056
(індекс)

МАГІСТЕРСЬКА РОБОТА

Тема: *Покращення стійкості симетричних криптоалгоритмів для реалізації на пристроях з обмеженими обчислювальними ресурсами*
(назва роботи)

Комп'ютерна інженерія
(назва освітньої програми)

123 – комп'ютерна інженерія
(шифр і назва спеціальності)

М. І. Грицан
(підпис, ініціали та прізвище здобувача освітнього ступеня)

Науковий керівник *Заячук Ярослав Іванович, к.т.н., доцент*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Допущено до захисту
Завідувач кафедри

проф. *С. І. Мельничук*
(посада) (підпис) (дата) (ініціали та прізвище)

Рецензент

_____
(посада) (підпис) (дата) (ініціали та прізвище)

Робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

Івано-Франківськ – 2025

Івано-Франківський національний технічний університет нафти і газу

Інститут *Інформаційних технологій*

Кафедра *Комп'ютерних систем та мереж*

Спеціальність *123 – Комп'ютерна інженерія*

Освітній рівень *магістр*

ЗАТВЕРДЖУЮ:

Зав. кафедрою КСМ

проф. С. І. Мельничук

“ ____ ” _____ 2025 р.

ЗАВДАННЯ

НА ВИКОНАННЯ МАГІСТЕРСЬКОЇ РОБОТИ СТУДЕНТОВІ

Грицану Миколі Івановичу

(прізвище, ім’я, по-батькові)

1. Тема магістерської роботи ***“Покращення стійкості симетричних криптоалгоритмів для реалізації на пристроях з обмеженими обчислювальними ресурсами”***

Керівник проекту *Заячук Ярослав Іванович*

затверджена наказом по університету від _____

2. Термін здачі студентом закінченої роботи *10.12.25*

3. Вихідні дані до проекту (роботи) *Матеріали та результати, отримані під час проходження переддипломної практики, методичні вказівки, технічна література*

4. Зміст розрахунково - пояснювальної записки (перелік питань, що їх належить розробити)

Аналіз симетричних криптоалгоритмів

Метод підвищення стійкості симетричних блокових криптоалгоритмів

Реалізація методу шифрування із підвищенням криптостійкості

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

6. Консультанти по дипломній роботі, із зазначенням розділів роботи, що стосуються їх

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
нормоконтроль	***О. В. Мойсеєнко***		

7. Дата видачі завдання *12.03.2025*

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів бакалаврської роботи	Термін виконання етапів роботи	Примітка
1	*Аналіз симетричних криптоалгоритмів*	*12.03.2025-31.05.2025*	*Виконано*
2	*Метод підвищення стійкості симетричних блокових криптоалгоритмів*	*01.06.2025-31.07.2025*	*Виконано*
3	*Побудова алгоритму підвищення стійкості симетричних блокових криптоалгоритмів*	*01.08.2025-30.09.2025*	*Виконано*
4	*Реалізація методу шифрування із підвищенням криптостійкості*	*01.10.2025-15.11.2025*	*Виконано*
5	*Оформлення пояснювальної записки*	*16.11.2025-10.12.2025*	*Виконано*

Студент-магістр _____ ***Грицан М. І.***
(підпис)

Керівник роботи _____ ***Заячук Я. І.***
(підпис)

АНОТАЦІЯ

Було проведено аналіз симетричних криптоалгоритмів, що дозволив виявити недоліки їх застосування. Наведено ключові підходи криптоаналізу, які можуть ослабити захист шифрів і призвести до розголошення початкового повідомлення. Крім того, досліджено методи генерації перестановок і запропоновано новий алгоритм формування унікальної перестановки.

У роботі описано підхід до підвищення криптостійкості, що ґрунтується на створенні унікальних перестановок для кожного блоку даних із застосуванням псевдовипадкового згенерованого числа.

Розроблено програмне забезпечення, яке реалізує запропонований метод.

ШИФРУВАННЯ ДАНИХ, КРИПТОСТІЙКІСТЬ, СИМЕТРИЧНІ АЛГОРИТМИ, ПЕРЕСТАНОВКИ

SUMMARY

An analysis of symmetric encryption algorithms was conducted, which made it possible to identify the weaknesses of their application. The main methods of cryptanalysis that can make ciphers vulnerable and pose a threat of plaintext disclosure are described. In addition, methods for generating permutations were examined, and a new algorithm for creating a unique permutation was proposed.

The study developed an approach to enhancing cryptographic strength based on the generation of unique permutations for each data block using a generated pseudorandom number.

Software was created in which the proposed method was implemented.

DATA ENCRYPTION, CRYPTOSECURITY, SYMMETRIC ALGORITHMS, PERMUTATIONS

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

AES – Advanced Encryption Standard

CBC – Cipher Block Chaining

CLI – Command-Line Interface

DES – Data Encryption Standard

ECB – Electronic Codebook

FEAL – Fast data Encipherment ALgorithm

GCM – Galois/Counter Mode

GUI – Graphical User Interface

HMAC – Hash-based message authentication code

IV – Initialization Vector

TEA – Tiny Encryption Algorithm

ВСТУП

Сьогодні існує значна кількість різних методів криптоаналізу, які дають змогу виявляти слабкі місця та робити вразливими багато сучасних алгоритмів шифрування, особливо симетричних. Тому питання підвищення рівня криптостійкості симетричних систем шифрування залишається одним із найважливіших у галузі криптографії. Це пояснюється тим, що в умовах стрімкого розвитку цифрових технологій, зростання обсягів передавання інформації та розширення комунікаційних каналів особливої ваги набуває забезпечення надійного захисту даних, які пересилаються через різноманітні мережі зв’язку.

Для підтримки високої швидкості обміну інформацією та стабільного функціонування систем зв’язку у більшості пристроїв застосовується симетричне шифрування. Такі алгоритми переважають завдяки своїй продуктивності та меншій обчислювальній складності у порівнянні з асиметричними, однак мають потенційні ризики уразливості.

Актуальність теми дослідження. Небезпечними є ситуації, коли можливе проведення атак із використанням адаптивно підібраного відкритого тексту, адже такий підхід значно підвищує шанси зловмисника отримати доступ до конфіденційної інформації, що передається через канал із симетричним шифруванням. Саме тому проблема посилення криптостійкості таких алгоритмів залишається надзвичайно актуальною. Додатково слід зазначити, що у процесі створення енергоефективних пристроїв часто застосовуються спрощені симетричні шифри, які не завжди забезпечують достатній рівень захисту. Використання таких рішень у системах, де критично важлива цілісність і безпека даних, обґрунтовує необхідність розроблення методів, які дозволяють підвищити стійкість шифрування без істотного збільшення енергоспоживання.

Метою роботи є створення методу, що дозволяє підвищити стійкість симетричних криптоалгоритмів до взлому, та програмна реалізація цього методу.

Об’єктом дослідження є процес шифрування інформації симетричними криптоалгоритмами.

Предметом дослідження є метод підвищення стійкості симетричних криптоалгоритмів до взлому.

Методи дослідження. У процесі виконання роботи застосовувалися такі методи:

- аналітичний розгляд особливостей симетричних шифрів і їхніх потенційних вразливостей;

- пошук шляхів підвищення стійкості до криптоаналізу;

- розроблення та програмна реалізація запропонованого методу.

Наукова новизна: отримав подальший розвиток алгоритм, який підсилює криптостійкість симетричного шифрування за рахунок використання перестановок підблоків, що здійснюються відповідно до чисел, згенерованих індивідуально для кожного блоку даних. Такий підхід дозволяє ускладнити структуру шифру та підвищити рівень захисту інформації.

Практичне значення: розроблений метод та його програмна реалізація забезпечують можливість підвищення стійкості симетричних алгоритмів до атак без потреби у значних обчислювальних ресурсах. Запропонований підхід може ефективно використовуватись для захисту передавання даних у малопотужних пристроях, де важливим є поєднання енергоефективності та безпеки.

Структура і обсяг роботи. Магістерська робота складається зі вступу, 3 розділів, висновків і списку літератури, що включає 38 найменувань. Основна частина роботи викладена на 55 сторінках машинописного тексту. Робота містить 28 рисунків, 3 таблиці та 1 додаток на 5 сторінках.

1 АНАЛІЗ СИМЕТРИЧНИХ КРИПТОАЛГОРИТМІВ

1.1 Симетричні криптоалгоритми

Salsa20 – це потоковий симетричний алгоритм шифрування, орієнтований на програмну реалізацію та високу ефективність передачі даних [2]. Автором і розробником цього методу є Деніел Бернштейн, який представив його на спеціальному конкурсі, організованому з метою створення європейських стандартів у сфері захисту інформації та шифрування повідомлень, що пересилаються електронними поштовими сервісами.

Основою роботи Salsa20 є використання трьох основних операцій (рис. 1.1):

- додавання чисел розміром 32 біти;
- побітове додавання за модулем два;
- циклічні зсуви бітів.

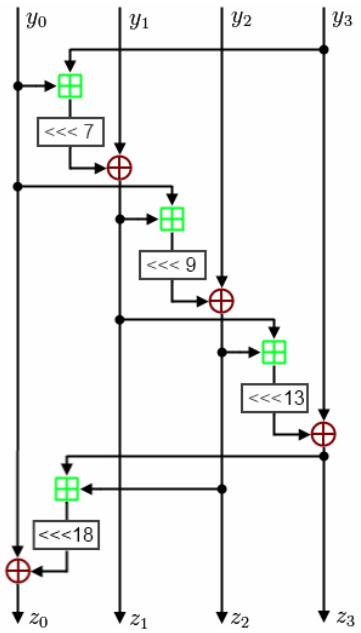

Рисунок 1.1 – Функції алгоритму Salsa20

Базові трансформації, які виконує шифр, мають певну схожість із алгоритмом AES, хоча у випадку Salsa20 застосовується хеш-функція, що проходить двадцять циклів перетворення. Висока швидкість роботи алгоритму пояснюється тим, що процес обчислень може бути ефективно розпаралелений:

обробка кожного стовпця і кожного рядка здійснюється незалежно, тому ці операції не залежать одна від одної.

На відміну від багатьох інших криптосистем, у Salsa20 практично відсутні додаткові підготовчі обчислення перед запуском циклів шифрування. Зміна ключа також не викликає труднощів. У низці інших алгоритмів потрібні проміжні дані, що кешуються у процесорі та залежать від ключа, через що при зміні останнього необхідно проводити нові обчислення. Для Salsa20 достатньо просто завантажити новий ключ у пам’ять, і процес може одразу продовжуватись.

На базі цього алгоритму створено спрощені модифікації – Salsa20/8 і Salsa20/12, які містять відповідно 8 та 12 раундів шифрування, тоді як у початковій версії їх 20 (число після косої риски у назві вказує на кількість раундів шифрування.). Хоча варіант Salsa20/8 забезпечує значно вищу продуктивність, ніж деякі інші популярні шифри (наприклад, AES чи RC4), проте для нього відома криптоатака, хоч і з дуже великою часовою складністю. Повна ж версія Salsa20/20 була розроблена саме для досягнення максимальної стійкості до криптоаналізу.

Пізніше тим самим автором була запропонована модифікація XSalsa20, у якій довжина nonce збільшена з 64 до 192 біт. Це розширення дозволило створити більш надійний криптографічний генератор псевдовипадкових чисел, адже при використанні короткого 64-бітного nonce виникає потреба в додатковому лічильнику, щоб уникнути колізій у процесі шифрування.

Аналіз безпеки цього шифру проводили різні дослідники. Зокрема, у 2005 році Пол Кровлі повідомив про атаку, що базувалася на спрощеному диференціальному криптоаналізі, спрямовану на Salsa20/5 і мала часову складність 2^{165}. Наступного року з’явилася інформація про атаку на Salsa20/6 із складністю 2^{117}. А у 2008 році група експертів – Омасон, Фішер, Казаї, Меєр і Решбергер – оприлюднили результати дослідження, де описано атаку на Salsa20/8, для якої потрібно приблизно 2^{251} операцій.

На сьогодні не зафіксовано жодної відомої ефективної атаки на повні версії Salsa20/12 та Salsa20/20, які залишаються безпечними для практичного застосування у сучасних криптографічних системах.

DES – це симетричний блочний метод шифрування інформації, який під час свого створення характеризувався відносно короткою довжиною ключа [1]. На сьогодні цей алгоритм вважається застарілим і небезпечним для використання через вже згадану обмежену довжину ключа, а також через невеликий розмір блоку даних – лише 64 біти (рис. 1.2).

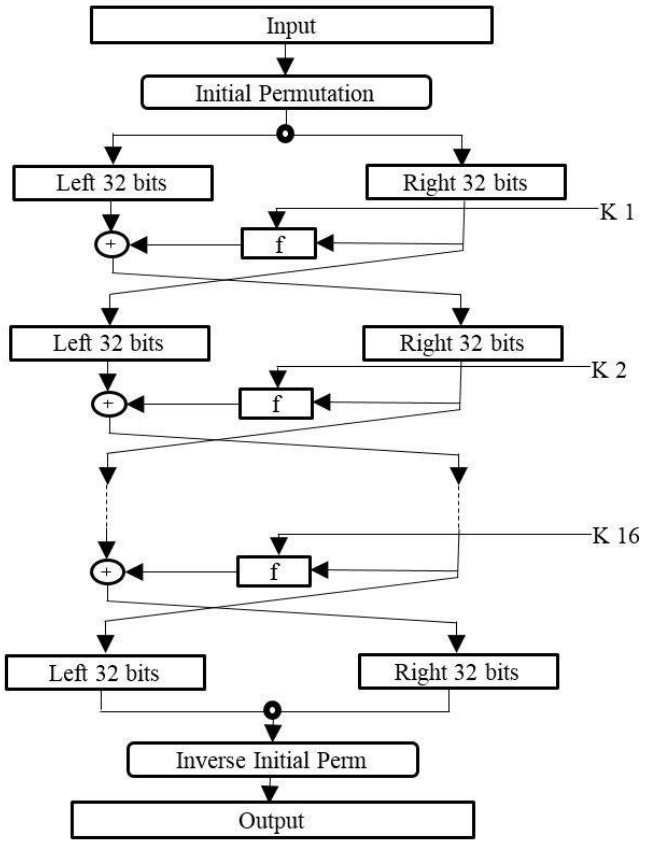

Рисунок 1.2 – Функції алгоритму DES

Критика на його адресу лунала ще з моменту появи у 1976 році, особливо через прихованість деяких принципів функціонування та структурних елементів. Це, у свою чергу, породило занепокоєння серед експертів-криптоаналітиків щодо потенційного контролю з боку Агентства національної безпеки США, адже 56-бітовий ключ не забезпечував належного рівня захисту, і існували підозри, що урядові структури мали технічні можливості для дешифрування повідомлень.

Попри свої недоліки, саме DES та дослідження його стійкості заклали підґрунтя для розвитку теорії блокових шифрів, формування криптоаналітичних підходів та поглибленого вивчення безпеки даних. Розроблення DES розпочалося в 1970-х роках у відповідь на запит Національного бюро стандартів (нині

відомого як NIST). Метою було створення безпечного методу передавання конфіденційної інформації між державними установами. Компанія IBM подала свою пропозицію на другий конкурс у серпні 1974 року, тобто більш ніж через рік після оголошення першого етапу відбору. Представлений алгоритм базувався на шифрі, розробленому Хорстом Фейстелем [3, 4].

Поступово DES був замінений алгоритмом AES (Rijndael), який став новим стандартом симетричного блочного шифрування на початку 2000-х років. Недоліки DES ставали дедалі очевиднішими: потреба в апаратній реалізації для прийнятної швидкодії, зростання потужності процесорів, що зробило можливим його злам методом повного перебору, а також неефективність модифікацій, спрямованих на підвищення криптостійкості [5]. Восени 1997 року Національний інститут стандартів і технологій США оголосив відкритий конкурс на створення наступника DES, висунувши такі критерії:

- шифр має бути блочним;
- розмір блоку – 128 біт;
- довжина ключів – 128, 192 або 256 біт.

Алгоритм AES повністю відповідав цим вимогам, що і забезпечило йому статус нового стандарту (рис.1.3).

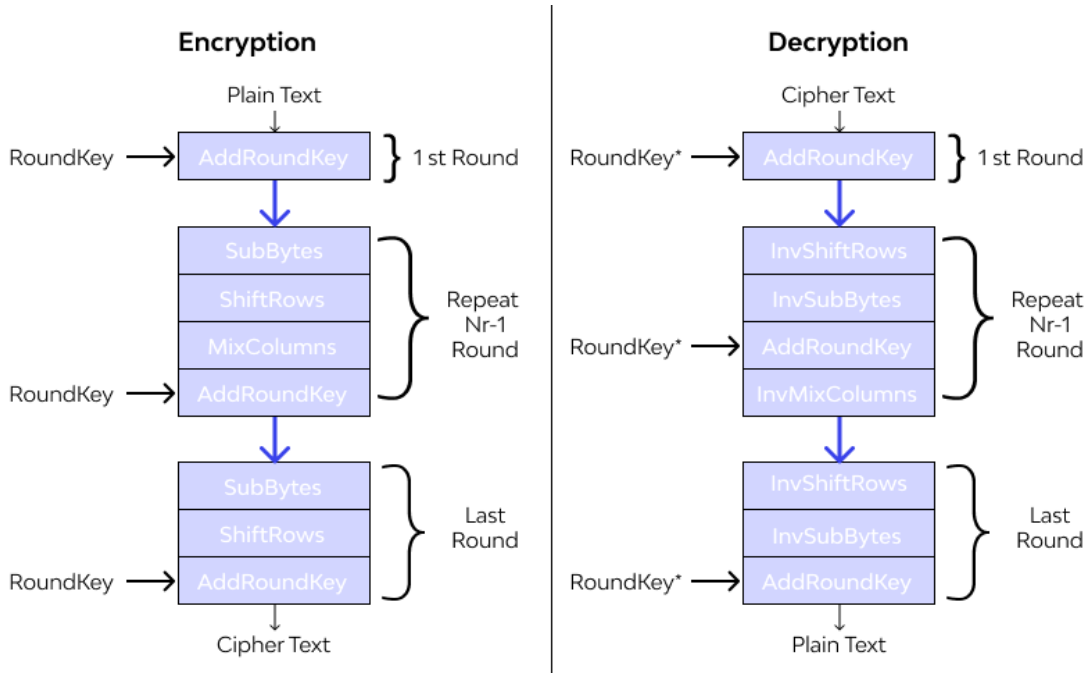

Рисунок 1.3 – Алгоритм AES

Blowfish – ще один представник блокових симетричних алгоритмів шифрування, розроблений криптографом Брюсом Шнаєром. Його робота ґрунтується на простих і швидких операціях, таких як додавання, підстановка та операція XOR (рис. 1.4). Алгоритм побудовано на основі мережі Фейстеля. Blowfish став популярним завдяки своїй відкритості – він не мав патентних обмежень, на відміну від більшості інших ефективних на той час методів, а серед безкоштовних рішень переважали ненадійні варіанти [6]. Отже, цей алгоритм розглядався як гідна альтернатива застарілому DES.

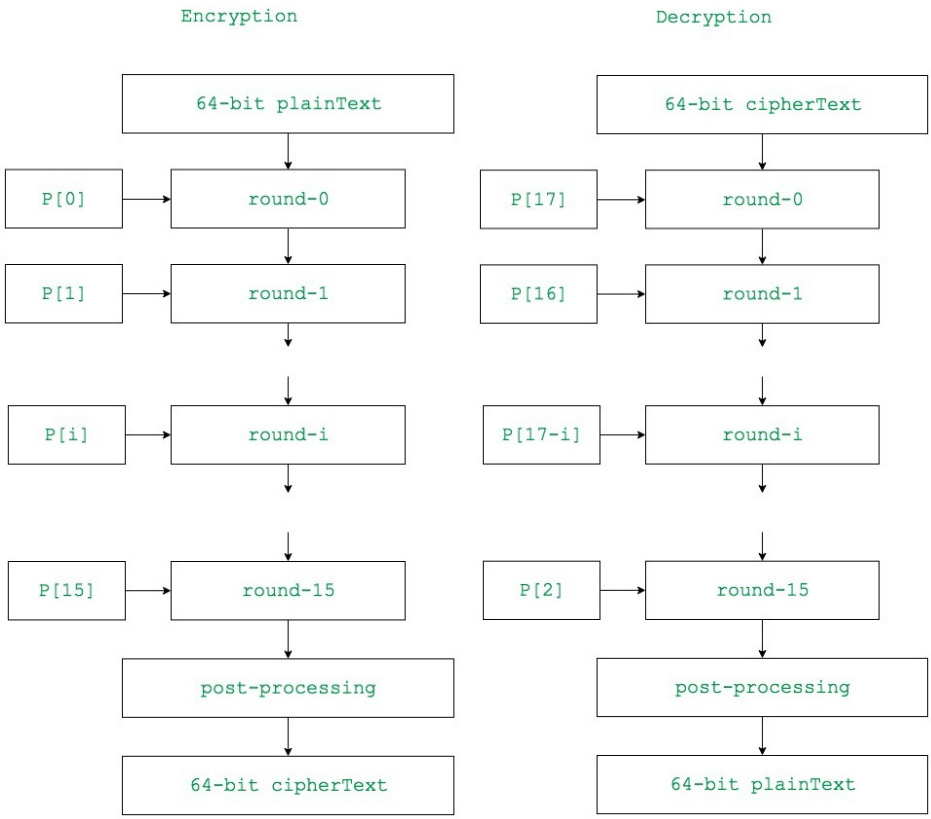

Рисунок 1.4 – Алгоритм Blowfish

Під час створення Blowfish його автор визначив ключові принципи:

- висока швидкість роботи: шифрування на 32-бітних процесорах виконується всього за 26 тактів;

- простота реалізації: використання елементарних операцій зменшує ризик помилок у реалізації;

- компактність: алгоритм не потребує значних ресурсів;

- можливість налаштування рівня стійкості під потреби конкретного застосування.

Також розглянемо алгоритм **TEA** [7]. Через те, що його автори не запатентували даний алгоритм, він отримав широке поширення та використовується у багатьох криптографічних застосунках.

У цьому методі відкритий текст перед шифруванням розбивається на блоки по 64 біти кожен. Якщо обсяг даних не кратний 64 бітам, а шифрування є блоковим, додаткові байти заповнюються значенням 0b00000001 у двійковій системі, щоб забезпечити кратність. Ключ K, який має довжину 128 біт (16 байт), ділиться на чотири підключі по 32 біти кожен і позначається як $K0$ – перший, а решта три – $K1$, $K2$, $K3$. Після цієї підготовки кожен блок у 64 біти обробляється протягом 64 раундів, тобто 32 циклів, відповідно до алгоритму, описаного нижче (рис. 1.5).

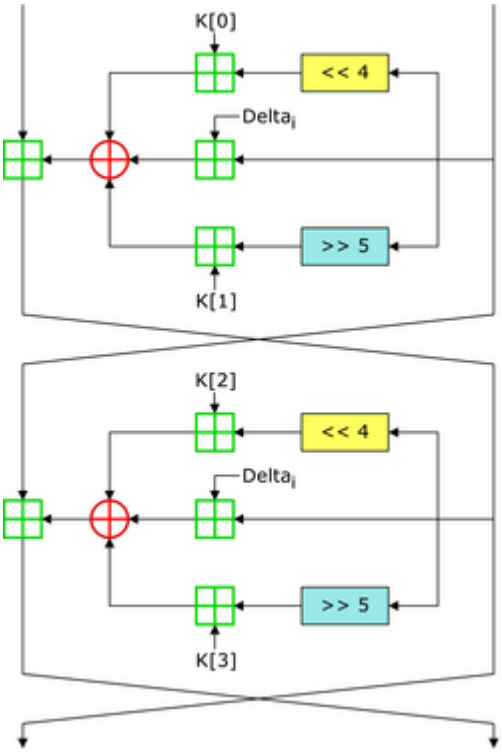

Рисунок 1.5 – Алгоритм TEA

Якщо на вхід кожного раунду $n(1 \le n \le 64)$ подають ліву (L_n) і праву (R_n) частини, то на виході формуються наступні значення лівої (L_{n+1}) та правої (R_{n+1}) половин, які обчислюються за такими правилами:

- якщо n непарне $(n = 2 \cdot i - 1$, де $1 \le i \le 32)$, тоді $R_{n+1} = L_n \boxplus \left(\{ [R_n \ll 4] \boxplus K[0] \} \oplus \{ R_n \boxplus i \cdot \delta \} \oplus \{ [R_n \gg 5] \boxplus K[1] \} \right)$;

- якщо n парне ($n = 2 \cdot i$, де $1 \le i \le 32$), тоді $R_{n+1} = L_n \boxplus \left(\{ [R_n \ll 4] \boxplus K[2] \} \oplus \{ R_n \boxplus i \cdot \delta \} \oplus \{ [R_n \gg 5] \boxplus K[3] \} \right)$;

- ліва половина нового стану визначається просто: $L_{n+1} = R_n$;

Константа $\delta = \left(\sqrt{5} - 1 \right) \cdot 2^{31} = 2654435769 = 9E3779B9_h$ виводиться із золотого перетину і використовується для кожного раунду. Операції $\boxplus$ та $\oplus$ представляють додавання за модулем 2^{32} та побітове виключне АБО (XOR) відповідно, а $\langle \ll, \gg \rangle$ – це побітові зсуви вліво або вправо на задану кількість біт. У кожному циклі константа множиться на номер i, що знижує ефективність атак, заснованих на симетрії раундів.

Перевага TEA полягає в простоті реалізації, хоча відсутність у базовій функції перетворень нелінійності компенсується великою кількістю раундів. Алгоритм був розроблений Девідом Вілерем та Роджером Нідгемом і забезпечує захищеність, порівнянну з шифром IDEA, оскільки застосовує операції із тих самих ортогональних алгебраїчних груп, що теоретично оберігає від лінійного криптоаналізу.

Найбільшу вразливість TEA проявляє при атаках, що використовують пов’язані ключі. Це пов’язано з простотою розкладу ключів: у парних раундах застосовуються підключі $K2$ і $K3$, у непарних – $K0$ і $K1$. Додатково кожен із чотирьох ключів має три еквівалентні варіанти, тому ефективна довжина ключа складає лише 126 біт замість повних 128. У 1997 році Брюс Шнаєр разом із двома іншими криптографами продемонстрував три атаки, засновані на цій особливості.

Простіша атака із ймовірністю 2 може виявити диференціальну характеристику ключа, що потребує 2^{34} відкритих текстів після 32 раундів. Складніша комбінована атака, що об’єднує диференціальний криптоаналіз із методом відкритих ключів, потребує 2^{23} вибраних текстів і має час виконання 2^{32} бітових операцій. При 10 раундах TEA демонструє високу стійкість, а для диференціальної атаки необхідно $2^{52,5}$ вибраних текстів із часовою складністю 2^{84}. При 17 раундах кількість відкритих даних для успішної атаки зростає до 1920, а часова складність приблизно 2^{123}.

Таблиця 1.1 – Порівняння алгоритмів сімейства TEA

Найменування алгоритму	Рік оприлюднення	Довжина блоку, bit	Довжина ключа, bit	Кількість раундів
TEA	1994	64	128	64
XTEA	1997	64	128	64
XTEA-1	1997	64	128	32
XTEA-2	1997	128	128	64
XTEA-3	1997	128	256	64
XXTEA	1998	64M	128	52+12M
RTEA	2007	64	128/256	48/64

Розроблено також модифіковані версії TEA: XTEA, XXTEA, RTEA та Raiden. XTEA має блоки по 64 біти та ключ 128 біт, застосовує 64 раунди мережі Фейштеля і була опублікована в 1997 році. Ця версія усуває вразливість до атак на пов'язаних ключах завдяки новому алгоритму розкладу ключів, хоча трохи знижує стійкість до диференціального криптоаналізу. Існують варіанти з 32 раундами (XTEA-1), блоком 128 біт (XTEA-2) або ключем 256 біт (XTEA-3). Через рік з'явився XXTEA, а в 2007 – модифікація RTEA із 48 раундами мережі Фейштеля. Таблиця 1.1 показує порівняння характеристик усіх алгоритмів, що базуються на TEA.

1.2 Різновиди атак на криптоалгоритми

Робота криптоаналітика полягає у виявленні ключа K та послідовності даних P, або обох цих складових одночасно. Окрім цього, він може використовувати ймовірнісну інформацію про P, наприклад, знати, що це цифрові фотографії, українські тексти, оцифрований звук чи інші види даних. Основна мета криптографії полягає у захисті відкритого тексту та/або ключа від несанкціонованого доступу сторонніх осіб, які намагаються перехопити або підслухати повідомлення. При цьому вважається, що канали зв'язку між відправником і отримувачем повністю під контролем зловмисників. Сутність криптографії полягає у забезпеченні конфіденційності переданої інформації, а також у вирішенні проблем таємності, перевірки автентичності, цілісності даних

та запобігання шахрайству з боку людей. Процес або спробу криптоаналізу зазвичай називають розкриттям.

Брюс Шнаєр виділив чотири основні типи криптоатак.

1 Атака із використанням шифротексту – метод, при якому криптоаналік отримує лише шифротексти декількох повідомлень, зашифрованих одним алгоритмом. Його завдання – знайти відкриті тексти максимальної кількості повідомлень або виявити ключі, що були використані для шифрування, щоб потім дешифрувати інші тексти, зашифровані тими ж ключами. Шифротексти здобуваються шляхом перехоплення повідомлень через відкриті канали зв'язку. Формально це виглядає так: $C_1 = E_k(P_1), C_2 = E_k(P_2), ...C_i = E_k(P_i)$. Завдання криптоаналітика – отримати один із відкритих текстів $P_1, P_2, ...P_i$, ключ K або алгоритм, що дозволяє визначати P_{i+1} за $C_{i+1} = E_k(P_{i+1})$.

Атака, при якій використовуються відкриті тексти – тип атаки, коли спеціаліст має доступ не лише до шифротекстів, а й до частини або всього відкритого повідомлення. Мета – виявити ключ, що шифрує дані, щоб згодом можна було читати інші повідомлення. У загальному вигляді схема атаки така: P_1, $C_1 = E_k(P_1)$, P_2, $C_2 = E_k(P_2), \ldots,$ P_i, $C_i = E_k(P_i)$. Криптоаналік повинен знайти ключ K або метод отримання наступного відкритого тексту P_{i+1} з $C_{i+1} = E_k(P_{i+1})$.

Атака із підібраним відкритим текстом – форма атаки, при якій аналітик може обирати відкриті тексти для шифрування. Це забезпечує більшу гнучкість і потенційно більше інформації про ключ, ніж попередній метод. Завдання полягає у визначенні ключа або алгоритму, що дозволяє дешифрувати повідомлення, зашифровані тим же ключем. Формально: $C_1 = E_k(P_1), C_2 = E_k(P_2), ...C_i = E_k(P_i)$ де відкриті тексти $P_1, P_2, ...P_i$ обираються криптоаналітиком.

Атака з адаптивно підібраним відкритим текстом – це модифікація атаки з підібраним текстом, яка дозволяє спеціалісту коригувати наступні відкриті тексти на основі результатів попереднього шифрування. Цей метод ефективніший за класичні атаки, оскільки експерт може обирати один блок даних, а на підставі його аналізу підбирати інші. Така форма атаки застосовувалася ще у першій

половині XX століття. Наприклад, британські військові знали частину відкритих даних, що передавали німецькі війська, завдяки мінуванню певних зон Північного моря та наявності координат.

Крім того, існують додаткові методи криптоатак:

- атака на основі підібраного шифротексту – коли криптоаналік має доступ до системи (чорного ящика), яка дешифрує повідомлення. Можна обирати шифротексти та отримувати відповідні відкриті тексти: $C_1, P_1 = D_k(C_1), C_2, P_2 = D_k(C_2), ..., C_i, P_i = D_k(C_i)$. Завдання – визначити ключ K. Цей метод зазвичай використовується для алгоритмів з відкритим ключем, але іноді ефективний і для симетричних систем;

- атака з використанням вибраного ключа – тип атаки, що базується на знанні певного зв'язку між ключами. Це не простий перебір, а визначення математичних залежностей, що часто буває непрактичним;

- бандитський криптоаналіз – застосування силових чи шахрайських методів для отримання даних: шантаж, погрози, хабарі та інше. Його використовують, коли криптосистема дуже стійка.

Окремо виділяють інженерний криптоаналіз, який спрямований на атаки на програмні, апаратні або комбіновані реалізації шифрів. Вивчають час виконання криптографічних операцій, споживання електроенергії, вплив електричних, магнітних полів, іонізуючого опромінення або високих температур, щоб отримати інформацію про алгоритм або ключ.

1.3 Різновиди аналізу стійкості криптоалгоритмів

Диференціальний криптоаналіз – це один із підходів до аналізу криптографічних систем, який насамперед застосовується до блокових шифрів, хоча може бути придатним і для потокових алгоритмів чи геш-функцій. Основна ідея методу полягає у вивченні різниць між парами даних, що проходять через окремі раунди шифрування, та дослідженні, як зміни на вході впливають на

результати шифрування. Іншими словами, аналіз зосереджується на тому, як варіювання початкових значень відображається у вихідних даних.

Цей метод був запропонований ізраїльськими дослідниками Елі Біхамом і Аді Шаміром, які вперше використали його для атак на алгоритм DES і подібні системи, хоча найбільш ефективно він показав себе проти шифру FEAL.

Варто підкреслити, що стійкість DES істотно залежить від структури S-блоків та кількості раундів шифрування; за внесення змін у ці параметри безпека системи може зрости. Якщо кількість етапів перевищує 19, то аналіз потребує більше часу, ніж повний перебір ключів, тобто метод грубої сили. Алгоритм вважають захищеним від диференціального аналізу, якщо для проведення атаки потрібно більше відкритих текстів, ніж їх теоретично можливо при певній довжині блоку.

Загалом, диференціальний криптоаналіз переважно має теоретичний характер, оскільки на практиці його ефективність часто обмежується великими витратами часу та обсягом необхідних даних.

Лінійний криптоаналіз – ще один спосіб криптографічного зламу, що використовує лінійні апроксимації для наближеного відтворення роботи алгоритму. Спочатку цей метод розробили для DES і FEAL, а в 1993 році його офіційно представив Міцуро Мацуї, японський криптоаналітик. Надалі підхід почали застосовувати і для дослідження інших шифрів, і нині він вважається одним із найвідоміших поряд із диференціальним. З його допомогою можливі атаки також на потокові шифри.

Для прикладу, при атакуванні DES із 16 раундами шифрування потрібно приблизно 50 днів обчислень на процесорі із частотою близько 1 ГГц, використовуючи близько 2^{43} пар текстів. Якщо ключ належить до категорії слабких ключів, то лінійний криптоаналіз ефективний і проти алгоритму RC5. Цим методом також вдалося зламати NUSH та NOEKEON.

Процес аналізу відбувається у два основні етапи: спершу визначаються залежності між відкритими, зашифрованими даними та ключем, а потім ці співвідношення застосовуються разом із відомими парами (відкритий текст –

шифротекст), щоб встановити певні біти ключа. Сутність підходу полягає у побудові рівнянь виду:

$$P_{i_1} \oplus P_{i_2} \oplus ... \oplus P_{i_a} \oplus C_{j_1} \oplus C_{j2} \oplus ... \oplus C_{j_b} = K_{k_1} \oplus K_{k_2} \oplus ... \oplus K_{k_c},$$

де, P_i – біти відкритого тексту;

C_j – біти зашифрованого повідомлення;

K_k – біти ключа.

Ймовірність правильності отриманих співвідношень для окремих бітів множин P, C і K зазвичай наближається до 0,5, що дозволяє робити статистичні висновки щодо структури ключа.

Інтерполяційний криптоаналіз – це вид аналітичного підходу, де роботу S-блоків представляють через алгебраїчну функцію, наприклад, поліноміальну, квадратичну або раціональну над полем Галуа. Коефіцієнти цієї функції можна знайти методом інтерполяції Лагранжа, використовуючи відомі відкриті тексти як точки даних. Атаку можна вдосконалити, якщо обрати такі тексти, що спрощують отримані рівняння. У спрощеній формі метод зводиться до побудови многочлена за шифротекстом, який при невеликій кількості невідомих коефіцієнтів і наявності достатнього набору пар (шифротекст – відкритий текст) дає можливість відновити функцію шифрування навіть без точних відомостей про ключ.

Цей підхід був запропонований наприкінці 1990-х років Ларсом Кнудсеном у співпраці з Томасом Якобсеном [1].

Частотний криптоаналіз – це метод, що базується на аналізі частоти появи символів у шифротексті. Його основа полягає у припущенні, що певні статистичні закономірності вживання знаків у відкритих текстах частково зберігаються і після шифрування. Простими словами, частотний аналіз спирається на розподіл літер у текстах певної мови – тобто, що деякі символи трапляються з характерною регулярністю у великих обсягах тексту.

Такий тип криптоаналізу сьогодні використовується переважно в освітніх цілях, адже з середини XX століття більшість сучасних криптографічних систем створювалися так, щоб бути стійкими до частотного аналізу [8, 9].

1.4 Постановка завдання

У межах даного розділу було виконано комплексне дослідження симетричних алгоритмів шифрування даних. Зокрема, розглянуто класифікацію симетричних шифрів, проаналізовано їхню криптостійкість до різних типів криптоаналітичних атак, описано найбільш поширені методи криптоаналізу та додаткові способи атак, спрямовані на розкриття ключової інформації або відкритого тексту.

Таким чином метою роботи є створення методу, що дозволяє підвищити стійкість симетричних криптоалгоритмів до взлому, та програмна реалізація цього методу.

Для досягнення зазначеної мети необхідно виконати такі завдання:

- дослідити принципи роботи симетричних алгоритмів шифрування та визначити їхні слабкі місця;

- проаналізувати існуючі методи атак на системи шифрування;

- розробити новий підхід до підвищення криптостійкості симетричних алгоритмів;

- створити програмну реалізацію розробленого методу.

2 МЕТОД ПІДВИЩЕННЯ СТІЙКОСТІ СИМЕТРИЧНИХ БЛОКОВИХ КРИПТОАЛГОРИТМІВ

2.1 Генерація унікальних перестановок

Процес побудови всіх можливих перестановок для певної множини або мультимножини має велике значення не лише для фахівців, які займаються комбінаторними задачами, але й для інших галузей, де використовуються подібні обчислювальні підходи. Існує широкий спектр задач, у яких необхідно застосовувати алгоритми для роботи з перестановками.

Під перестановкою порядку N розуміють розташування N різних об'єктів у визначеному порядку в один ряд. Для прикладу, для трьох елементів a,b та c існує шість можливих варіантів їх розміщення: abc,acb,bac,bca,cab,cba. Таким чином, для множини, що містить N елементів, можливо побудувати $N!$ різних перестановок. Це пояснюється тим, що перша позиція може бути заповнена будь-яким із N елементів, друга – будь-яким із решти $N-1$, оскільки один елемент уже використаний, і так далі. Отже, на останнє місце залишається лише один можливий елемент. У загальному вигляді кількість усіх можливих перестановок обчислюється за формулою:

$$N\cdot(N-1)\cdot(N-2)\cdot\ldots\cdot 1 = N!\ .$$

Описуючи підходи до побудови перестановок, варто почати з алгоритму L, який називають також алгоритмом лексикографічної генерації перестановок.

Цей метод для впорядкованої початкової послідовності $a_1a_2\ldots a_n$, де $a_1 \le a_2 \le \ldots \le a_n$, дозволяє створити всі перестановки множини $\{a_1,a_2,\ldots,a_n\}$ у лексикографічному порядку.

Алгоритм складається з чотирьох основних кроків.

Крок 1. Розглядається поточна перестановка $a_1a_2\ldots a_n$.

Крок 2. Присвоюється $j \leftarrow (n-1)$. Якщо виконується умова $a_j \geq a_{j+1}$, індекс j інкрементується, доки не знайдеться така позиція, для якої $a_j < a_{j+1}$. Якщо ж $j = 0$, алгоритм завершує роботу, адже це означає, що вже були згенеровані всі можливі перестановки, починаючи з $a_1 a_2 \ldots a_j$. У цьому випадку саме елемент із індексом j підлягає збільшенню в наступній перестановці.

Крок 3. Встановлюється $l \leftarrow n$. Якщо $a_j \geq a_l$, індекс l зменшується, доки не стане виконаною умова $a_j < a_l$. Після цього елементи a_j і a_l міняються місцями. Оскільки частина послідовності $a_{j+1} \geq \ldots \geq a_n$ впорядкована за спаданням, то елемент a_l буде найменшим із тих, що більші за a_j. Перед перестановкою виконувалася послідовність $a_{j+1} \geq \ldots \geq a_{l-1} \geq a_l > a_j \geq a_{l+1} \geq \ldots \geq a_n$, а після обміну вона набуває вигляду $a_{j+1} \geq \ldots \geq a_{l-1} \geq a_j > a_l \geq a_{l+1} \geq \ldots \geq a_n$.

Крок 4. Присвоюються $k \leftarrow (j+1)$ і $l \leftarrow n$. Поки виконується умова $k < l$, елементи a_k і a_l міняються місцями ($a_k \leftrightarrow a_l$), після чого k збільшується на 1, а l зменшується на 1. Ці дії повторюються доти, поки $k > l$, після чого алгоритм повертається до першого кроку [1].

У більш узагальненому вигляді ця процедура дозволяє знайти лексикографічного наступника будь-якого комбінаторного об’єкта $a_1 a_2 \ldots a_n$. Для цього потрібно виконати три дії:

- знайти найбільший індекс j, для якого елемент a_j можна збільшити, тобто $a_j < a_{j+1}$;

- збільшити a_j на найменше можливе значення, що забезпечує правильний порядок;

- сформувати найкоротший, у лексикографічному сенсі, шлях, який доповнить нову частину до повного об’єкта $a_1 \ldots a_j$.

Розглянутий алгоритм реалізує цю загальну ідею для задачі генерації перестановок (рис. 2.1). Крім алгоритму L, існують й інші методи та способи побудови перестановок, описані в працях [10].

Рисунок 2.1 – Приклад лексикографічної перестановки

Найпростіша зміна перестановки полягає у тому, щоб поміняти місцями два суміжні елементи. Будь-яку перестановку можливо впорядкувати, якщо правильно обрати послідовність таких обмінів. Тобто, рухаючись цим шляхом, можна отримати будь-яку бажану перестановку, починаючи від початково впорядкованої послідовності елементів [1].

Алгоритм простих змін, який також називають Алгоритмом P, працює наступним чином. Для заданої послідовності $a_1 a_2 \ldots a_n$, що містить n різних елементів, він створює всі можливі перестановки шляхом повторних замін суміжних пар. Для цього використовується допоміжний масив $c_1 c_2 \ldots c_n$, що є таблицею інверсій. Алгоритм проходить усі комбінації цілих чисел, таких що $1 \le c_j \le j$ для кожного $1 \le j \le n$. Додатково застосовується масив $o_1 o_2 \ldots o_n$, який визначає напрямок зміни елементів у масиві c_j [1].

Існує також Алгоритм T, що відповідає за переходи простих змін. Він обчислює таблицю $t[1], t[2], \ldots, t[n!-1]$ так, щоб дії Алгоритму P відповідали послідовним обмінам $a_{t[k]} \leftrightarrow a_{t[k]+1}$ для всіх $1 \le k \le n!$, якщо $n \ge 2$. Робота цього алгоритму ділиться на п’ять етапів [1].

1. Встановлюються значення $N \leftarrow n!, d \leftarrow \frac{N}{2}, t[1] \leftarrow 1$ і $m \leftarrow 2$.

2. Перевіряємо, чи $m = n$; якщо так, роботу завершено, а якщо ні, тоді $m \leftarrow m+1$, $d \leftarrow d \cdot m$ і $k \leftarrow 0$.

3. Збільшуємо $k \leftarrow k+d$ і $j \leftarrow m-1$ до тих пір, поки $j \neq 0$.

4. Встановлюємо $t[k] \leftarrow t[k]+1$ та $k \leftarrow k+d$.

5. Оновлення $t[k] \leftarrow j$ під час циклу $j < m-1$, після чого перевіряється, чи $k < N$; якщо так, повертаємося до третього етапу, інакше – до другого.

Наприклад, для $n = 4$ таблиця виходить так: $(t[1], t[2], \ldots, t[23]) = (3, 2, 1, 3, 1, 2, 3, 1, 3, 2, 1, 3, 1, 2, 3, 1, 3, 2, 1, 3, 1, 2, 3)$.

Крім того, існує загальний генератор перестановок, відомий як Алгоритм G. Він працює для таблиці Сімса $(S_1, S_2, \ldots, S_n)$, де кожна множина S_k має $k+1$ елементів $\sigma(k, j)$. Алгоритм генерує всі перестановки $a_1 a_2 \ldots a_{n-1}$ для $\{0, 1, \ldots, n-1\}$, використовуючи додатковий масив $c_n c_{n-1} \ldots c_1$. Основною перевагою цього методу є те, що він проходить усі перестановки $a_1 a_2 \ldots a_{k-1}$ до моменту досягнення a_k, після чого виконує $k!$ циклів перед зміною a_k і так далі. Це дозволяє швидко пропускати всі перестановки, що закінчуються неважливими суфіксами, якщо такі елементи не впливають на задачу [1].

Варто також згадати метод Кнута для генерації всіх перестановок. Його суть полягає в тому, що на основі всіх перестановок довжини N можна отримати $N+1$ перестановку довжини $N+1$. Наприклад, для перестановки 3241 можна створити п’ять варіантів довжиною 5: 53241, 35241, 32541, 32451, 32415 (рис. 2.2).

```
3 2 4 1 0.5 → 4 3 5 2 1
3 2 4 1 1.5 → 4 3 5 1 2
3 2 4 1 2.5 → 4 2 5 1 3
3 2 4 1 3.5 → 3 2 5 1 4
3 2 4 1 4.5 → 3 2 4 1 5
```

Рисунок 2.2 – Метод Кнута для перестановок 3241

Перший спосіб побудови всіх перестановок цим методом передбачає два кроки [1]:

- додавання в кінець перестановки певних чисел, що визначаються формулою $\frac{2i+1}{2},(0 \le i \le N)$;

- перенумерацію отриманих перестановок у порядку зростання елементів.

Існує також метод рекурентного пошуку перестановок (рис. 2.3), який базується на ідеї, що вихідну задачу слід звести до аналогічної задачі, але на меншому наборі елементів [1].

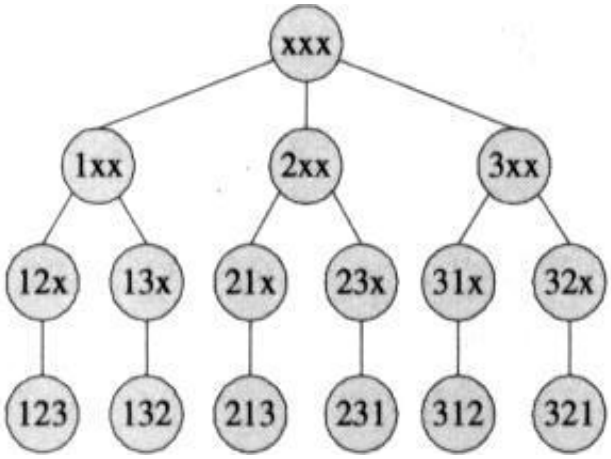

Рисунок 2.3 – Рекурсія всіх перестановок для множини {1, 2, 3}

Алгоритм рекурентної генерації перестановок можна представити у вигляді дерева, і, наприклад, для $n = 3$ воно матиме шість листків із усіма результативними перестановками та три гілки на початковому рівні [1].

Для генерації всіх можливих перестановок у лексикографічному порядку вибрано Factoradic permutation algorithm (різновид Алгоритм Т), який детальніше описаний далі.

2.2 Підвищення стійкості симетричних криптоалгоритмів

Підхід до підвищення криптостійкості можна умовно розділити на три основні категорії операцій, що відображено пунктирними блоками на рисунку 2.4.

Блок-схема алгоритму шифрування представлена на рисунку 2.5.

Перший етап відрізняється від наступних і має свої особливості. На цьому етапі відбувається отримання ключа шифрування та блока даних, які необхідно обробити і передати, що на схемі позначено як вхідні дані.

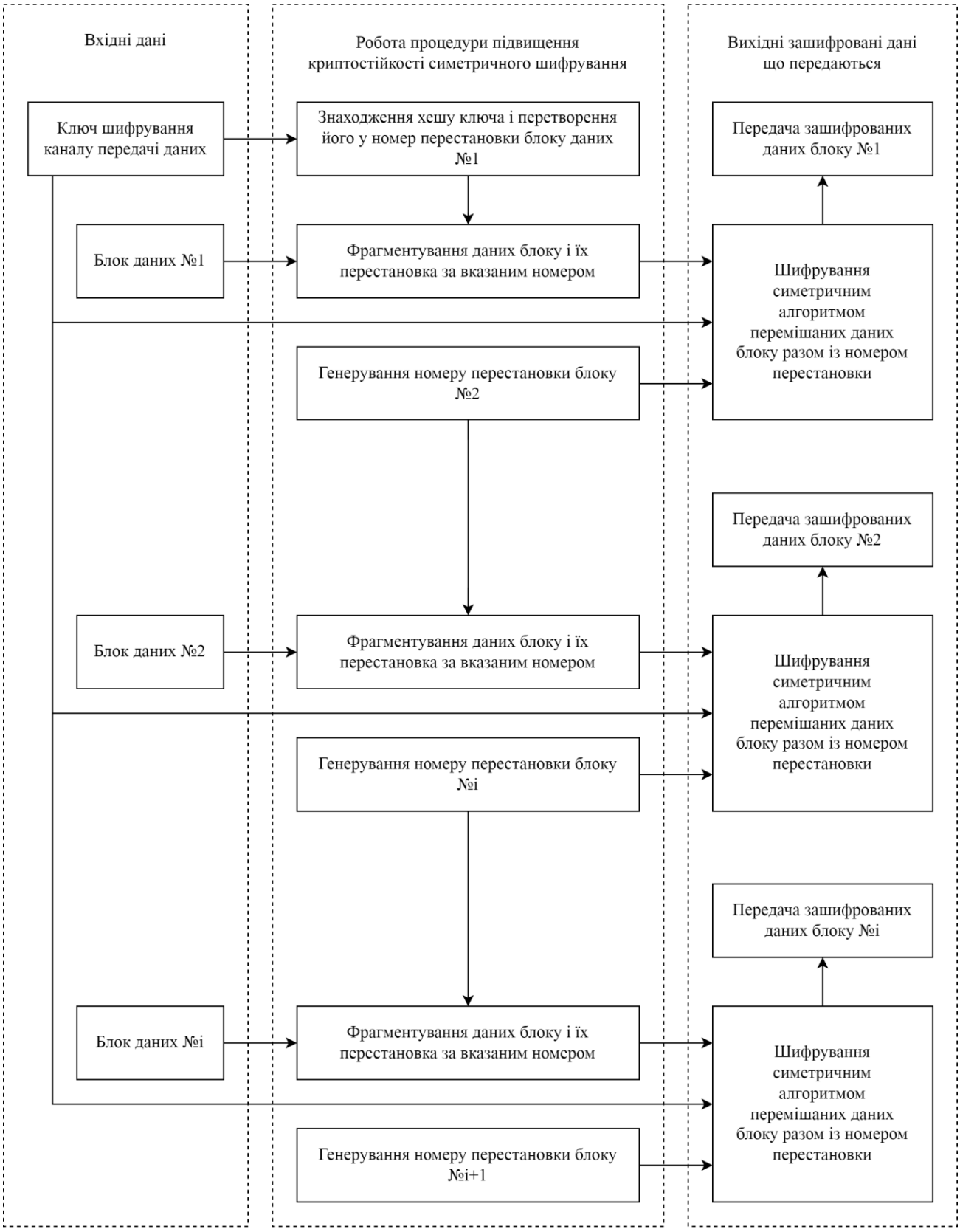

Рисунок 2.4 – Структурна схема шифрування

Для початкового етапу спершу визначається хеш ключа, який далі певним чином перетворюється для узгодження з межами мінімального та максимального значень номера перестановки підблоків даних.

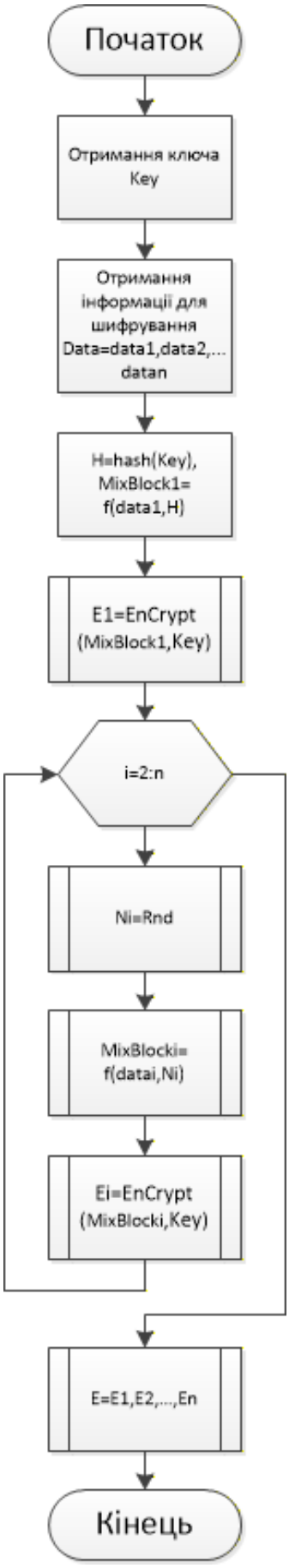

Рисунок 2.5 – Блок-схема алгоритму шифрування

Блок даних розбивається на фрагменти, після чого виконується перестановка за номером, отриманим із хешу ключа шифрування.

Генерується число псевдовипадковим алгоритмом, що використовується для перестановки другого блоку даних.

Пересортовані дані разом із цим числом формують єдиний блок для подальшої обробки симетричним алгоритмом шифрування. Число розміщується в кінці блока, після основних даних.

Шифровані дані передаються каналом зв’язку, і опрацювання переходить до наступного етапу.

Другий блок подається аналогічно першому для перестановки, але номер перестановки визначається згенерованим на попередньому етапі псевдовипадковим числом. Після цього дані доставляються отримувачу, де виконується розшифрування.

Для всіх наступних блоків шифрування здійснюється аналогічно другому блоку: дані шифруються та передаються через інформаційний канал.

Дешифрування повторює процес шифрування у зворотному порядку, із деякими відмінностями, що відображено на рисунку 2.6.

Блок-схема алгоритму шифрування представлена на рисунку 2.7.

Дані, отримані від передавача, подаються разом із ключем у симетричний алгоритм дешифрування через спільний ключ закритого каналу передачі.

Ключ піддаються хешуванню, і отримується номер перестановки для першого блока розшифрованих даних, аналогічно прямому процесу шифрування.

Частина даних, що містить корисну інформацію, передається в алгоритм зворотної перестановки, де підблоки розсортуються у зворотному порядку, відновлюючи початкову структуру даних.

Алгоритм зворотної перестановки обробляє підблоки, починаючи з останнього блока.

Номер блока для перестановки обчислюється діленням числа перестановки на факторіал кількості вже опрацьованих блоків, яка починається з одиниці. На відміну від прямого алгоритму, де число ділилося лише на кількість блоків, тут

застосовується саме ділення на факторіал. Остача від ділення визначає блок, з яким потрібно виконати перестановку, а ціла частина використовується для наступних блоків у зворотному порядку.

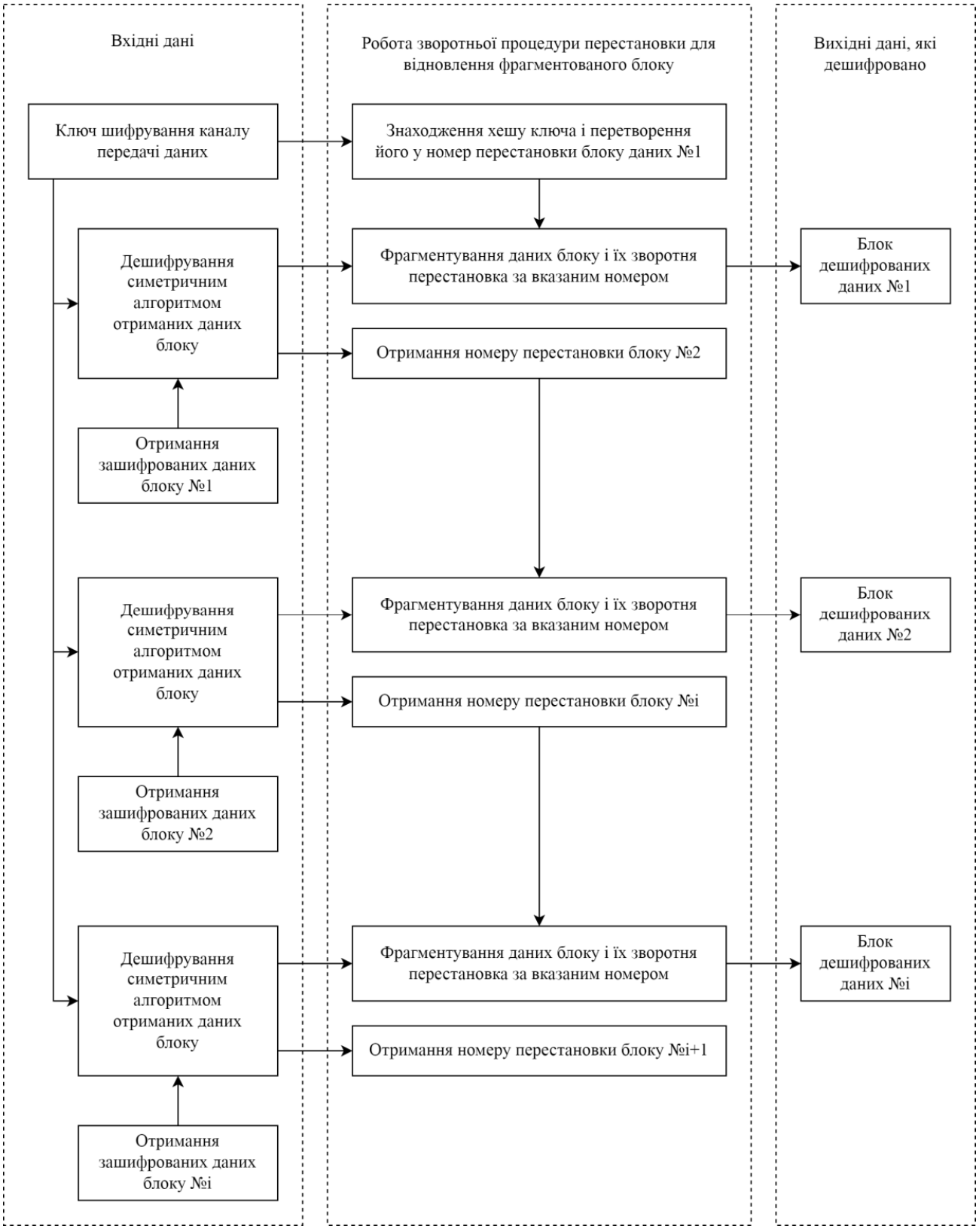

Рисунок 2.6 – Структурна схема дешифрування

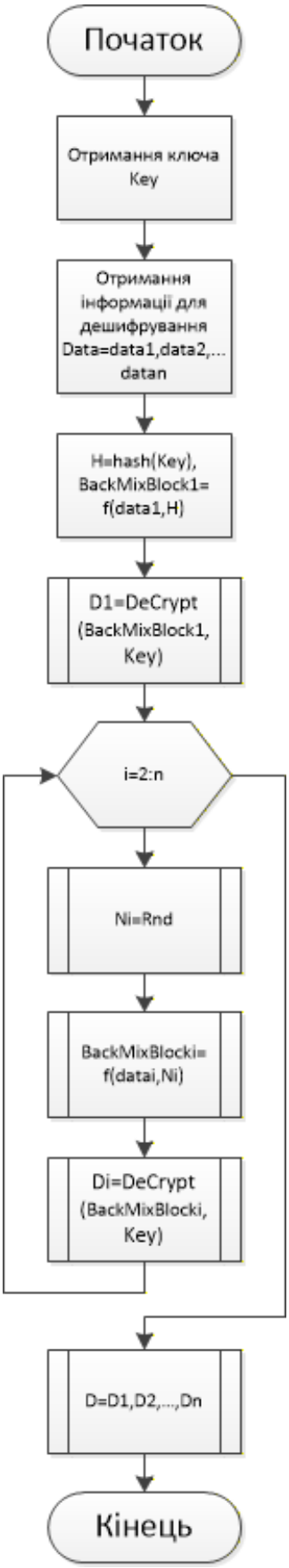

Рисунок 2.7 – Блок-схема алгоритму дешифрування

Після завершення зворотного сортування інформація відновлюється і передається за призначенням.

З цього блока зчитується номер перестановки для наступного блока.

Другий блок дешифрується аналогічно, але перестановка здійснюється за числом, отриманим із попереднього блока.

Для всіх подальших блоків процес дешифрування повторюється так само, як і для другого блока.

Для ілюстрації алгоритму на рисунку 2.8 показано блок даних, розділений на п’ять підблоків. Це лише приклад, оскільки відтворювати усі етапи обробки великої кількості підблоків нераціонально. Така кількість підблоків обрана для демонстрації різних випадків перестановки і відображення усіх операцій на кожному етапі.

На практиці рекомендується використовувати більше ніж 12 субблоків, щоб забезпечити більшу кількість варіантів перестановки, що значно підвищує стійкість до криптоаналізу шифротексту.

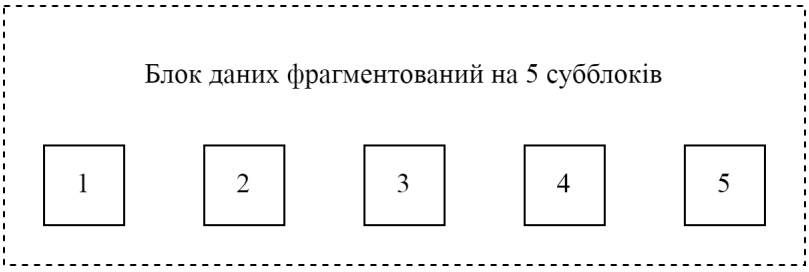

Рисунок 2.8 – розділення блоку на 5 фрагментів меншого розміру

Факторіал числа п’ять дорівнює 120. Це число представляє максимальну кількість можливих перестановок субблоків. Оскільки воно менше 255, його можна зберігати в одному байті. Для алгоритму симетричного шифрування з блоками по 128 біт (16 байт) один байт виділяється під номер перестановки, а решта 15 байт ділиться на п’ять менших блоків по три байти кожен, над якими проводиться процес перестановки. Можна також використати інший варіант, коли блок розділяється на 12 частин. У такому випадку для збереження номера перестановки знадобляться вже 4 байти, і з 16 байт залишаються 12, які складають 12 підблоків по одному байту кожен. Цей варіант забезпечує значну криптостійкість, оскільки кількість перестановок практично досягає пів мільярда.

А для криптоаналізу зазвичай потрібно не один блок, а багато блоків, що істотно підвищує складність зламу з кожним наступним блоком.

Рекомендується робити підблоки якомога меншими. Це зменшує кількість підряд розміщених даних, що можуть бути відомі криптоаналітику, і таким чином знижує відому йому інформацію, навіть якщо канал передачі даних використовується ним для власних цілей.

Блоки на кожному етапі нумеруються зліва направо, починаючи з нульового номера для першого блоку, далі йде блок під номером один і так далі.

Спочатку генерується номер перестановки в діапазоні від 0 до $5!-1$. Наприклад, це число 83. На першому кроці його потрібно поділити на кількість підблоків, тобто: $\frac{83}{5}=16$, залишок 3. Ціла частина, 16, використовується для наступного кроку, а залишок 3 вказує на підблок, який буде обміняний місцями з нульовим блоком (рис. 2.9).

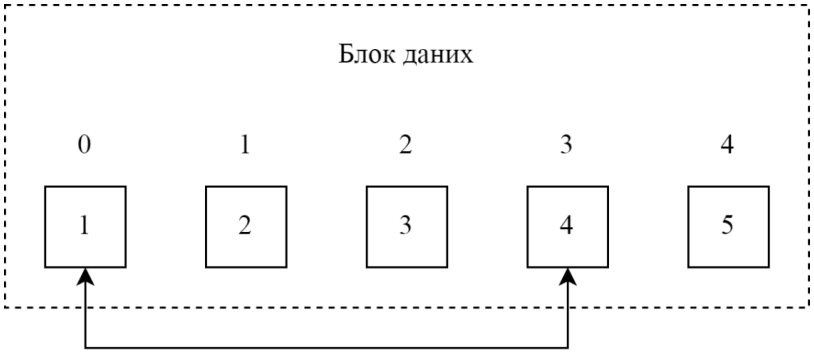

Рисунок 2.9 – Перший етап перестановки

На рис. 2.10 показано, що блоки 1 і 4 помінялись місцями, після чого починається другий етап. Всі блоки перенумеровуються знову зліва направо, починаючи з нуля, і тепер їх залишилося чотири.

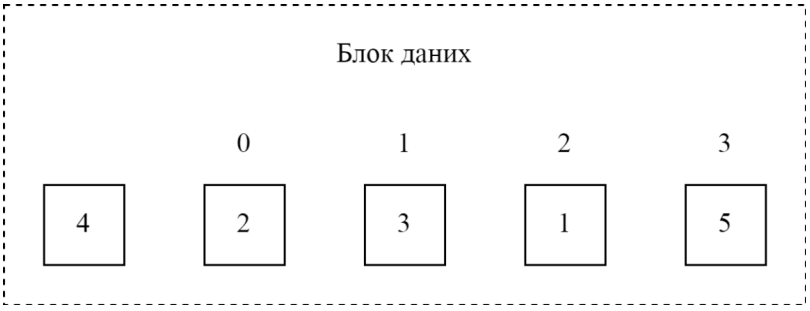

Рисунок 2.10 – Другий етап перестановки

Ціла частина 16 з першого ділення ділиться на 4: $\frac{16}{4}=4$. Ця частина буде використана на наступному кроці. Оскільки залишок від ділення дорівнює нулю, перестановка для цього етапу не проводиться, тобто блок залишається на своєму місці (рис. 2.10), і процес переходить до наступного етапу.

Третій етап починається із перенумерації блоків. Кількість блоків знову зменшилась на одиницю. Тепер можливі варіанти для перестановки нульового блоку обмежуються трьома позиціями. Номер блоку для перестановки обчислюється, наприклад, так: $\frac{43}{3}=14$, залишок 1. Залишок показує, що нульовий блок буде поміняний місцями з блоком під номером один (рис. 2.11).

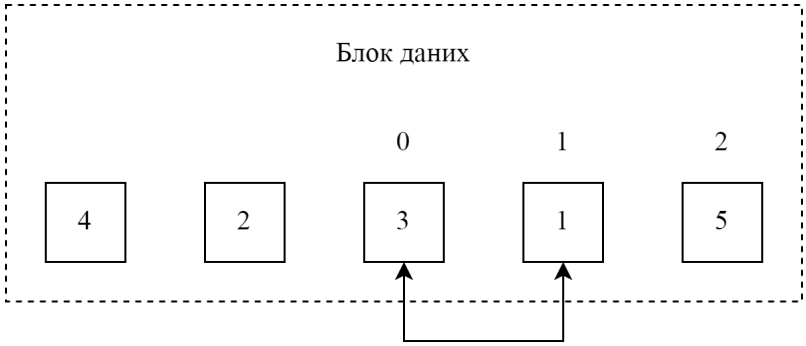

Рисунок 2.11 – Третій етап перестановки

На четвертому етапі залишаються тільки два блоки. Номер блоку для перестановки може бути 0 або 1, і у цьому випадку це число 1. Ділення на 2 дає залишок 1, тому блок із тимчасовим номером 0 обмінюється місцями з блоком номер 1 (рис. 2.12).

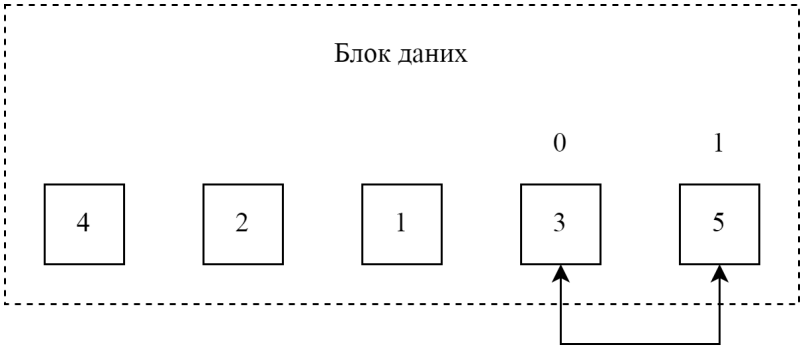

Рисунок 2.12 – Четвертий етап перестановки

Останній елемент залишився на своєму місці. Після сортування отримуємо послідовність: 4, 2, 1, 5, 3 (рис. 2.13).

Рисунок 2.13 – Результат перестановки

Ця послідовність, згенерована за алгоритмом перестановки, унікальна для номера 83 і не повторюється для інших чисел у діапазоні від 0 до 119. Послідовність готова для шифрування і може бути використана в будь-якому симетричному алгоритмі, разом із числом перестановки для наступного блоку даних.

2.3 Діаграма прецедентів

Use Case Diagram відображає взаємодію користувача із програмою для шифрування та дешифрування файлів (рис. 2.14). Вона дозволяє наочно показати функціональні можливості системи та взаємозв’язки між актором і юзкейсами.

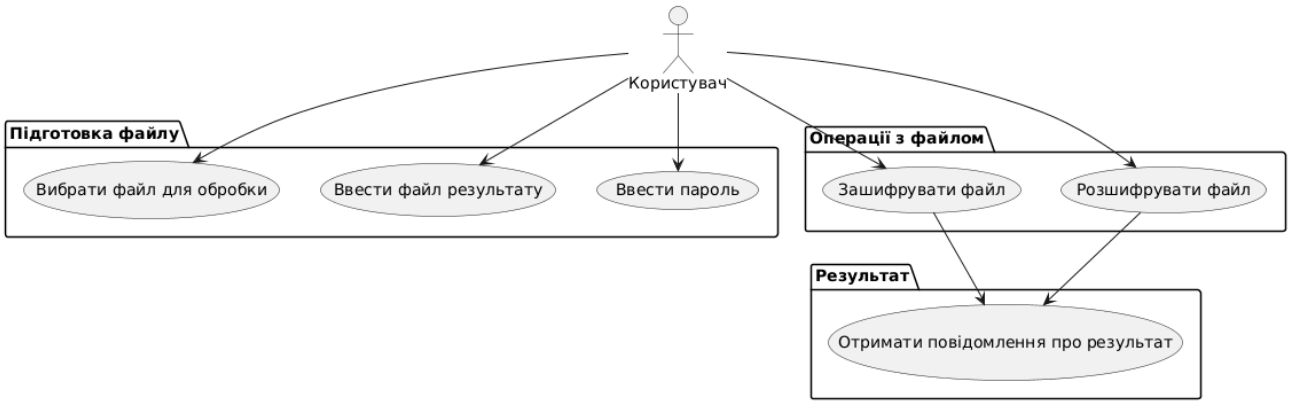

Рисунок 2.14 – Use Case Diagram

Актор: користувач (User) – особа, яка працює з програмою через графічний інтерфейс (GUI). Користувач ініціює всі дії системи, тобто вибір файлу, введення паролю та запуск процесів шифрування або дешифрування.

Діаграма містить наступні юзкейси.

Підготовка файлу:

- вибрати файл для обробки – користувач обирає вихідний файл, який необхідно зашифрувати або розшифрувати;

- ввести файл результату – користувач задає ім’я файлу, куди буде збережено результат обробки;

- ввести пароль – користувач вводить пароль для шифрування або дешифрування файлу.

Операції з файлом :

- зашифрувати файл – система виконує шифрування обраного файлу, використовуючи введений пароль та внутрішні алгоритми перетворення даних;

- розшифрувати файл – система виконує розшифрування файлу, використовуючи пароль користувача та алгоритм відновлення даних.

Результат:

- отримати повідомлення про результат – система інформує користувача про успішне завершення шифрування або розшифрування файлу, або повідомляє про помилку в процесі обробки.

Користувач взаємодіє з усіма юзкейсами через графічний інтерфейс.

Юзкейси шифрування та дешифрування залежать від введених користувачем даних (файлу, файлу результату та паролю).

Юзкейси шифрування та дешифрування об’єднуються з юзкейсом виводу результату, що відображає повідомлення користувачу про успіх або помилку операції.

Діаграма дозволяє:

- візуалізувати функціональні можливості програми.

- визначити основні сценарії взаємодії користувача з системою.

- забезпечити наочність для документації та подальшого аналізу вимог.

2.4 Висновок до розділу

У другому розділі розглянуто методи підвищення стійкості симетричних блокових криптоалгоритмів шляхом використання генерації унікальних перестановок для блоків даних. Було проаналізовано різні алгоритми побудови перестановок – лексикографічний алгоритм (Алгоритм L), алгоритм простих змін (Алгоритм P), Алгоритм T, Алгоритм G, метод Кнута та рекурентний підхід – що дозволяють систематично формувати всі можливі комбінації елементів множини. Кожен із цих методів забезпечує ефективне створення унікальних перестановок, що важливо для підвищення криптостійкості шифрування.

Детально описано процес підвищення стійкості симетричних криптоалгоритмів через розбиття блоків даних на підблоки та їх багатоетапну перестановку, де номери перестановок генеруються на основі хешу ключа та псевдовипадкових чисел. Цей підхід у поєднанні з зменшенням розміру підблоків дозволяє значно ускладнити криптоаналіз, оскільки кількість можливих перестановок швидко зростає зі збільшенням числа підблоків. Наведені блок-схеми шифрування та дешифрування демонструють практичне застосування алгоритму та забезпечують відновлення початкових даних без втрат.

Таким чином, застосування методів генерації унікальних перестановок у поєднанні з процесом шифрування підвищує криптографічну стійкість симетричних блокових алгоритмів і дозволяє створювати надійні системи захисту інформації.

3 РЕАЛІЗАЦІЯ МЕТОДУ ШИФРУВАННЯ ІЗ ПІДВИЩЕННЯМ ЙОГО КРИПТОСТІЙКОСТІ

3.1 Опис інтерфейсу програмного забезпечення

Для реалізації програмного забезпечення використано мову програмування Python.

Програма має графічний інтерфейс користувача (GUI), реалізований за допомогою бібліотеки Tkinter (рис. 3.1). Основна мета інтерфейсу – забезпечити зручний та інтуїтивно зрозумілий доступ до функцій шифрування та розшифрування файлів.

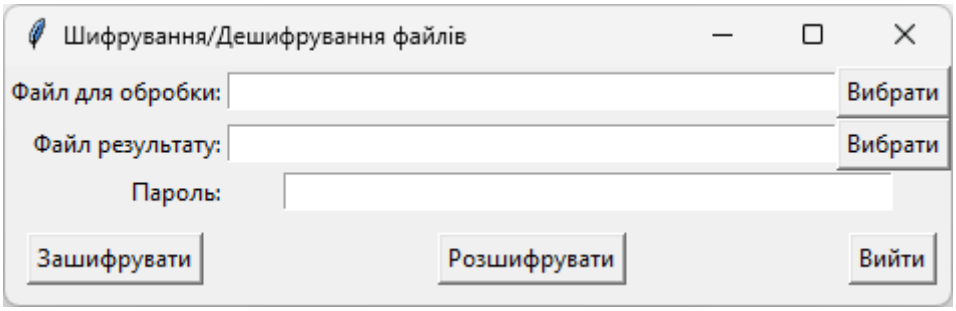

Рисунок 3.1 – Графічний інтерфейс користувача v.1

Основні компоненти інтерфейсу:

- поле для вибору файлу для обробки - елемент: текстове поле (Entry) та кнопка «Вибрати». Призначення: дозволяє користувачу обрати файл, який буде зашифровано або розшифровано;

- поле для вибору файлу результату – елемент: текстове поле (Entry) та кнопка «Вибрати». Призначення: користувач вказує шлях і назву файлу, куди буде збережено результат обробки;

- поле введення пароля – елемент: текстове поле (Entry) із прихованим введенням (show="*"). Призначення: введення користувачем пароля для шифрування або розшифрування файлу;

- кнопки управління:

- «Зашифрувати»: виконує шифрування вибраного файлу із зазначеним паролем;

- «Розшифрувати»: виконує розшифрування файлу із зазначеним паролем.

- «Вийти»: закриває програму.

Взаємодія користувача з інтерфейсом відбувається так. Користувач обирає файл для обробки, натискаючи кнопку «Вибрати» або вводячи шлях вручну.

Користувач обирає файл для збереження результату аналогічним способом.

Користувач вводить пароль для шифрування/розшифрування.

Натискання кнопки «Зашифрувати» або «Розшифрувати» запускає відповідну функцію обробки файлу.

У разі успіху програма відображає повідомлення про успішне завершення операції; у разі помилки – повідомлення про помилку.

Інтерфейс перевіряє, чи всі обов’язкові поля заповнені перед початком обробки. Якщо якесь поле порожнє, програма показує попередження.

Призначення інтерфейсу:

- забезпечує зручну взаємодію користувача з програмою без необхідності введення команд у терміналі;

- виконує функцію контролю правильності введених даних та інформує користувача про стан обробки файлів;

- відокремлює графічну складову від логіки шифрування та розшифрування, що підвищує зручність та безпеку використання.

Можливо також реалізувати програму, яка використовує графічний інтерфейс користувача (GUI) за допомогою бібліотеки PySimpleGUI, для забезпечення зручного доступу до функцій шифрування та розшифрування файлів (рис. 3.2).

Основні компоненти інтерфейсу:

- поле для вибору файлу для обробки – елементи: текстова мітка, поле введення (InputText) та кнопка FileBrowse. Призначення: дозволяє користувачу обрати файл для шифрування або розшифрування;

- поле для вибору файлу результату – елементи: текстова мітка, поле введення (InputText) та кнопка FileSaveAs. Призначення: користувач вказує шлях і назву файлу, куди буде збережено результат обробки;

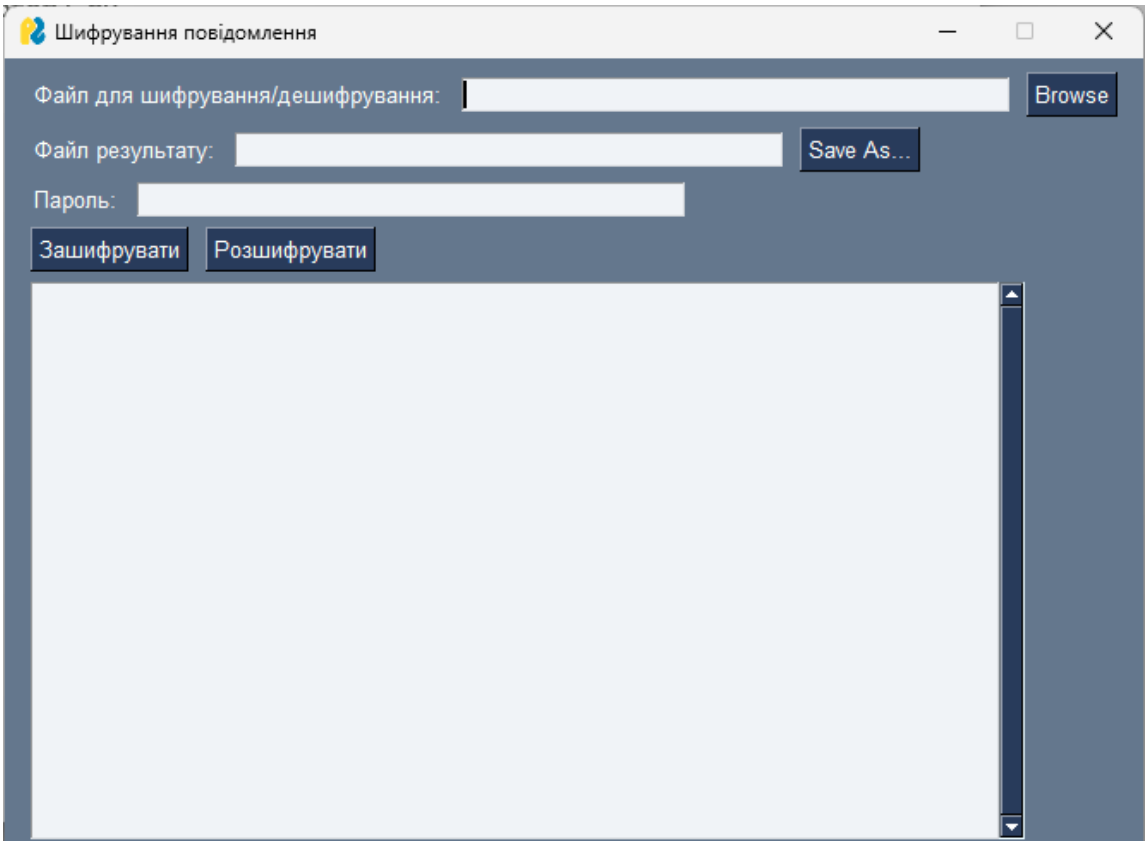

Рисунок 3.2 – Графічний інтерфейс користувача v.2

- поле введення пароля – елемент: текстова мітка та поле введення (InputText) з прихованим введенням (password_char='*'). Призначення: введення пароля для шифрування або розшифрування файлу;

- кнопки дій:

- «Зашифрувати»: виконує шифрування вибраного файлу із зазначеним паролем;

- «Розшифрувати»: виконує розшифрування файлу із зазначеним паролем;

- вивід інформації - елемент: поле Output, де відображаються повідомлення про успіх або помилки обробки файлів.

Взаємодія користувача з інтерфейсом відбувається так. Користувач обирає файл для обробки через кнопку «Browse» або вводить шлях вручну.

Користувач обирає файл для збереження результату через кнопку «SaveAs» або вводить шлях вручну.

Користувач вводить пароль для шифрування/розшифрування.

Натискання кнопки «Зашифрувати» або «Розшифрувати» запускає відповідну функцію обробки файлу.

У разі успіху або помилки повідомлення виводяться у спеціальному полі вікна.

Програма перевіряє наявність всіх обов’язкових полів перед обробкою файлу.

Якщо будь-яке поле порожнє, користувач отримує повідомлення про необхідність заповнення всіх полів.

3.2 Опис коду програмного забезпечення

Описана програма призначена для зашифрування та розшифрування довільних файлів із використанням алгоритму симетричного шифрування AES. Проте незначні модифікації (використання відповідних бібліотек) дають змогу використати будь-який блоковий симетричний криптоалгоритм, наприклад, з описаних в першому розділі, оскільки запропонований алгоритм підвищення криптостійкості виконує додаткові маніпуляції над вхідними даними і не вносить змін у роботу самого криптоалгоритму.

Система забезпечує базовий рівень захисту даних за допомогою пароля, який вводиться користувачем. Для зручності використання може бути реалізовано кілька видів графічного інтерфейсу (п. 3.1), що дозволяє користувачу вибирати файли, вводити пароль і керувати процесом без необхідності використання командного рядка.

Програма реалізує шифрування на рівні блоків, комбінуючи стандартне блочне шифрування із додатковими операціями обробки даних – додаванням метаданих та виконанням XOR-операцій між блоками для підвищення складності розшифрування без правильного пароля.

Основні етапи роботи:

1. Введення даних – користувач обирає вхідний файл, шлях для збереження результату та задає пароль.

2. Генерація ключа – із пароля за допомогою алгоритму SHA-256 формується 256-бітний ключ для AES.

3. Попередня обробка даних – файл ділиться на блоки по 48 байт; кожен блок доповнюється службовими даними та випадковими байтами.

4. Шифрування – відбувається за допомогою алгоритму AES у режимі ECB.

5. Збереження результату – зашифровані байти записуються у новий файл.

6. Розшифрування – виконується у зворотному порядку: зчитування, дешифрування, відновлення блоків та видалення службових байтів.

Функція *md5_permutation_number(password, n_blocks=12)* формує псевдовипадкове число на основі введеного пароля:

```
def md5_permutation_number(password, n_blocks=12):
md5 = hashlib.md5(password.encode('utf-8')).digest()
part1 = struct.unpack('<Q', md5[:8])[0]
part2 = struct.unpack('<Q', md5[8:16])[0]
return (part1 ^ part2) % n_blocks
```

- з пароля генерується MD5-хеш (128 біт);

- хеш розбивається на дві частини по 64 біти, які комбінуються через XOR;

- результат береться за модулем n_blocks (кількість блоків).

Це число (perm_number) використовується для зміни параметрів шифрування в кожному блоці, що створює різні шаблони змішування навіть для одного пароля.

Функція *add_block_metadata(block_data, perm_number)*:

```
def add_block_metadata(block_data, perm_number):
if len(block_data) < 48:
block_data += b'\x00' * (48 - len(block_data))
block = bytearray(block_data)
block += struct.pack('<I', perm_number)
block += generate_random_block_data()
# XOR 3*4 байт
```

```
for i in range(3):
for j in range(4):
block[i*4 + j] ^= block[48 + i*4 + j]
return bytes(block)
```

- приймає частину вихідних даних (до 48 байт);

- доповнює блок нулями до довжини 48 байт;

- додає 4 байти службової інформації (perm_number) і 12 байт випадкових даних;

- виконує XOR між основними 48 байтами і трьома 4-байтовими випадковими послідовностями.

У результаті формується 64-байтовий блок:

| 48 байт даних | 4 байти номера | 12 байт випадкових значень |

Операції XOR забезпечують додаткову непередбачуваність структури блоку.

Функція *encrypt_file(source, dest, password)* виконує:

```
def encrypt_file(source, dest, password):
with open(source, 'rb') as f:
data = f.read()
n_blocks = 12
perm_number = md5_permutation_number(password, n_blocks)
blocks = [add_block_metadata(data[i:i+48], perm_number + i) for i in
range(0, len(data), 48)]
final_data = b''.join(blocks)
key = hashlib.sha256(password.encode('utf-8')).digest()
cipher = AES.new(key, AES.MODE_ECB)
padding_len = 16 - (len(final_data) % 16)
final_data += bytes([padding_len] * padding_len)
encrypted = cipher.encrypt(final_data)
with open(dest, 'wb') as f:
f.write(encrypted)
```

- зчитування початкового файлу у двійковому режимі;

- формування блоків даних із додаванням метаданих;

- генерацію 256-бітного ключа з пароля через SHA-256;

- додавання padding (вирівнювання довжини до кратності 16 байтам, як того вимагає AES);

- шифрування даних у режимі AES.MODE_ECB;

- запис результату у вихідний файл.

Режим ECB робить реалізацію простою, однак у реальних системах рекомендується застосовувати більш стійкі режими, наприклад CBC або GCM.

Функція *decrypt_file(source, dest, password)* виконує:

```
def decrypt_file(source, dest, password):
with open(source, 'rb') as f:
encrypted = f.read()
key = hashlib.sha256(password.encode('utf-8')).digest()
cipher = AES.new(key, AES.MODE_ECB)
decrypted = cipher.decrypt(encrypted)
padding_len = decrypted[-1]
if padding_len < 1 or padding_len > 16:
padding_len = 0
decrypted = decrypted[:-padding_len] if padding_len else decrypted
n_blocks = 12
perm_number = md5_permutation_number(password, n_blocks)
recovered = bytearray()
for i in range(0, len(decrypted), 64):
block = decrypted[i:i+64]
if len(block) < 64:
block += b'\x00' * (64 - len(block))
block = bytearray(block)
block[:48]=permute_block_inverse(block[:48], n_blocks, perm_number)
for j in range(3):
for k in range(4):
block[j*4 + k] ^= block[48 + j*4 + k]
recovered += block[:48]
perm_number = (perm_number + 1) & 0xFFFFFFFF
recovered = recovered.rstrip(b'\x00')
with open(dest, 'wb') as f:
f.write(recovered)
```

- зчитування зашифрованого файлу;

- дешифрування за допомогою AES-ключа, створеного з пароля;

- видалення доданого під час шифрування padding'у;

- відновлення 48-байтових блоків з 64-байтових шляхом виконання XOR у зворотному порядку;

- видалення службових нульових байтів у кінці та запис у вихідний файл.

Графічний інтерфейс користувача створено за допомогою стандартної бібліотеки Tkinter:

```
class EncryptDecryptGUI:
def __init__(self, master):
self.master = master
master.title("Шифрування/Дешифрування файлів")
# Source file
tk.Label(master, text="Файл для обробки:").grid(row=0, column=0,
sticky="e")
self.src_entry = tk.Entry(master, width=50)
self.src_entry.grid(row=0, column=1)
tk.Button(master,text="Вибрати",
command=self.browse_src).grid(row=0, column=2)
# Destination file
tk.Label(master, text="Файл результату:").grid(row=1, column=0,
sticky="e")
...
try:
decrypt_file(src, dest, password)
messagebox.showinfo("Успіх", f"Файл розшифровано: {dest}")
except Exception as e:
messagebox.showerror("Помилка", str(e))
if __name__ == "__main__":
root = tk.Tk()
gui = EncryptDecryptGUI(root)
root.mainloop()
```

Програма відкриває вікно з такими елементами:

- поле для вибору вхідного файлу (через *filedialog.askopenfilename*);

- поле для вибору вихідного файлу (через *asksaveasfilename*);

- поле для введення пароля;

- кнопки: "Зашифрувати" – викликає функцію *encrypt_file*; "Розшифрувати" – викликає функцію *decrypt_file;* "Вийти" – завершує роботу програми.

Програма перевіряє коректність введення даних (наявність файлів і пароля).

Про результати виконання повідомляє користувача через вікна повідомлень (messagebox):

- showinfo – успішна операція;

- showwarning – відсутні необхідні дані;

- showerror – помилка виконання.

Структура програми у вигляді переліку основних функції представлена у табл. 3.1.

Таблиця 3.1 – Основні функції у програмі

Назва	Призначення
md5_permutation_number()	Генерує псевдовипадковий номер блоку з пароля
generate_random_block_data()	Створює випадкові дані для змішування
add_block_metadata()	Додає метадані та виконує XOR-шифрування
encrypt_file()	Повний цикл шифрування файлу
decrypt_file()	Повний цикл розшифрування файлу
permute_block_inverse()	Заглушка для майбутнього відновлення перестановки
EncryptDecryptGUI	Клас графічного інтерфейсу користувача

Використані бібліотеки та модулі демонструє таблиця 3.2.

Таблиця 3.2 – Використані бібліотеки

Модуль	Призначення
tkinter	створення графічного інтерфейсу користувача
filedialog, messagebox	робота з діалоговими вікнами
hashlib	генерація MD5 і SHA-256 хешів
struct	робота з бінарними структурами даних
Crypto.Cipher.AES	реалізація алгоритму AES
Crypto.Random	генерація випадкових байтів
os	робота з файловою системою

Щодо безпеки та особливостей реалізації можна відмітити:

- пароль не зберігається у файлах, а використовується лише для генерації ключа шифрування;

- у програмі використано SHA-256, що забезпечує достатню криптографічну стійкість ключа;

- додаткове змішування блоків через XOR підвищує ентропію даних перед шифруванням;

- реалізована система не призначена для промислового використання, оскільки режим ECB може мати вразливості при шифруванні схожих блоків даних;

- для повноцінного захисту рекомендовано застосувати AES у режимі CBC або GCM із вектором ініціалізації (IV).

3.3 Результати тестування

Тестування виконувалось з метою перевірки працездатності та ефективності запропонованого методу, який спрямований на підвищення рівня криптостійкості симетричних алгоритмів шифрування даних. Реалізація виконана у спрощеній формі, достатній для проведення тестування.

Для перевірки роботи програми необхідно підготувати файл, що міститиме інформацію, яка підлягає шифруванню – Text.txt. Саме цей файл у подальшому підлягає шифруванню описаним способом (рис. 3.3).

Під час роботи програма створює число перестановки для першого блоку, використовуючи геш від ключа шифрування. На основі цього значення здійснюється перестановка окремих частин даних, розмір яких становить чотири байти (рис. 3.4).

З ілюстрації (рис. 3.4) можна побачити, що після перестановки фрагментів даних текст повністю втрачає свою читабельність, хоча теоретично все ще може бути відновлений, проте лише складними аналітичними методами.

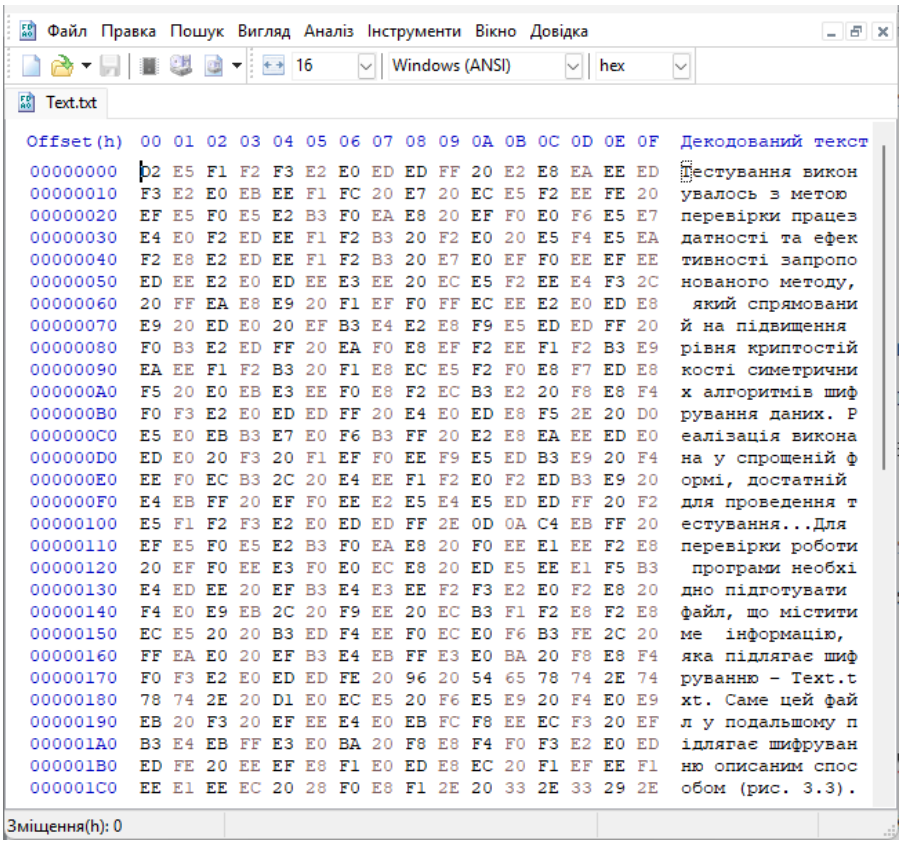

Рисунок 3.3 – Файл з даними для шифрування

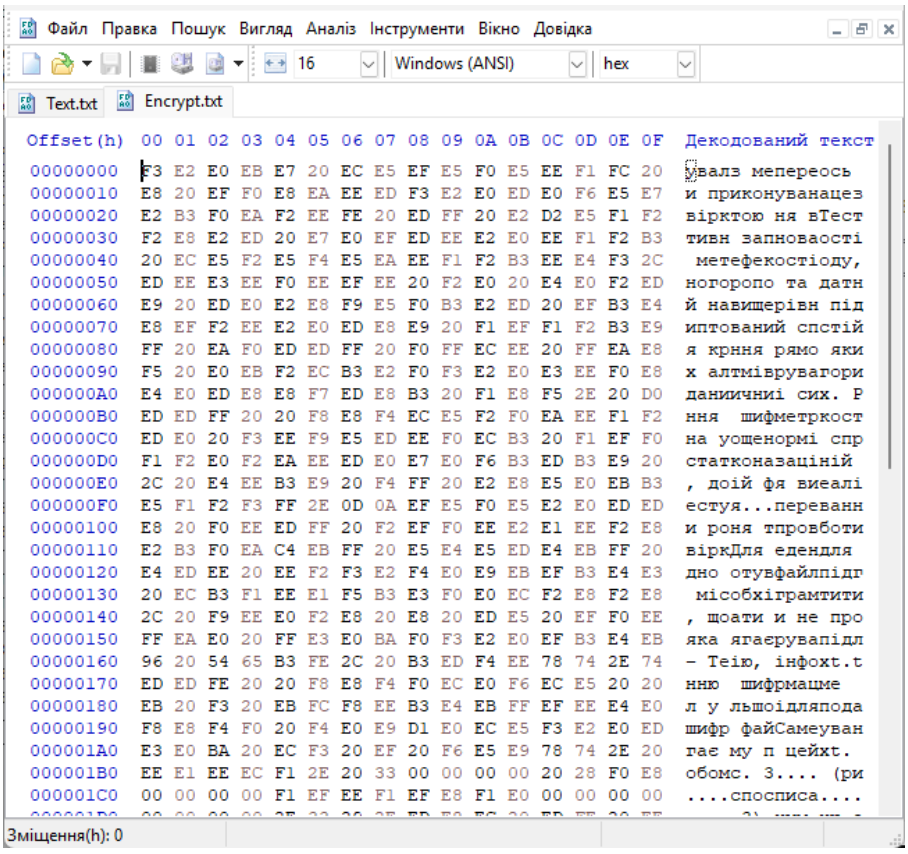

Рисунок 3.4 – Файл з перемішаними фрагментами даних

Після етапу перемішування до кінця блоку додається порядковий номер, який використовують для генерації перестановки наступної частини даних. Далі отримана інформація передається на вхід алгоритму симетричного шифрування. Результат цього процесу – зашифрований файл показано на рис. 3.5.

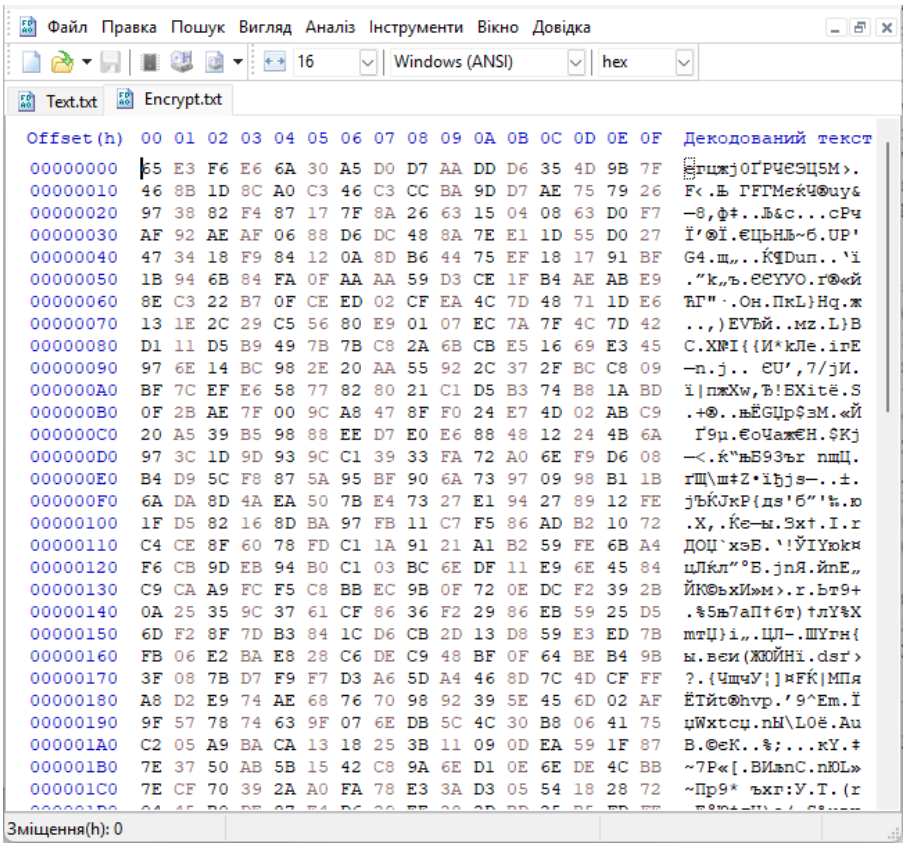

Рисунок 3.5 – Файл із зашифрованими даними

Коли процедура шифрування завершена, відновлення даних без відповідного ключа стає практично нереальним завданням.

Наступним кроком є виконання дешифрування з метою демонстрації коректної роботи запропонованого методу (рис. 3.6).

Для підтвердження універсальності алгоритму наведено ще один приклад із іншим текстом. Вихідні дані для цього експерименту подано на рисунку 3.7.

Подібно до попереднього випадку, дані підлягають перестановці їхніх фрагментів, після чого разом із ключем передаються до алгоритму шифрування. Отриманий шифротекст показано на рисунку 3.8.

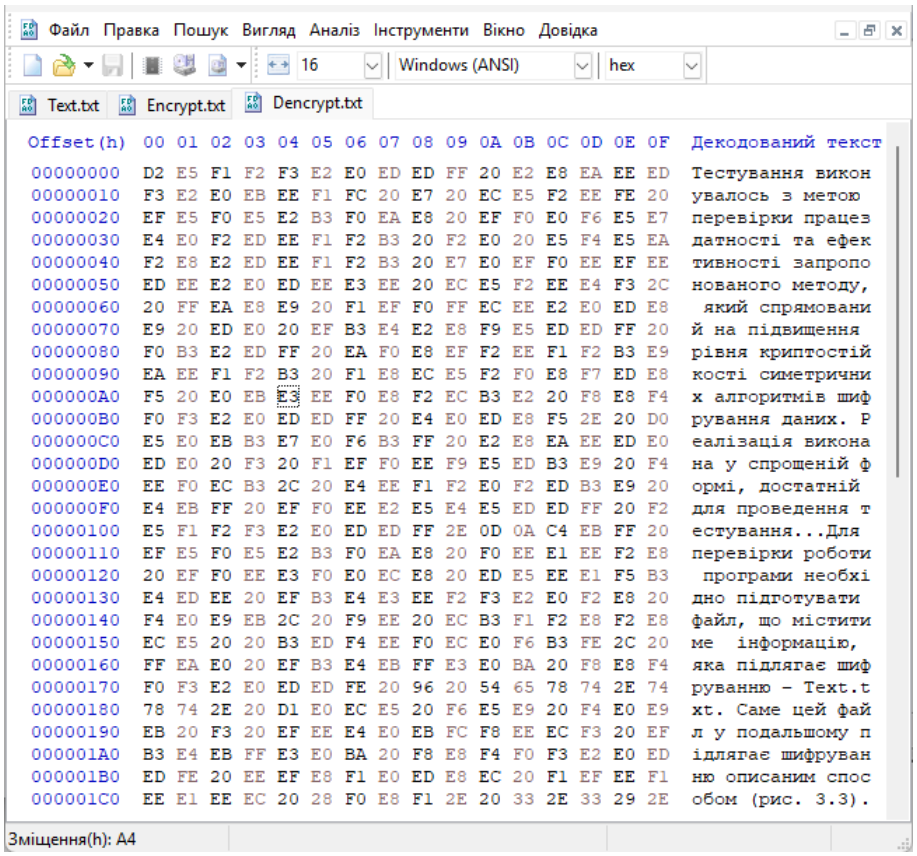

Рисунок 3.6 – Дешифрований файл

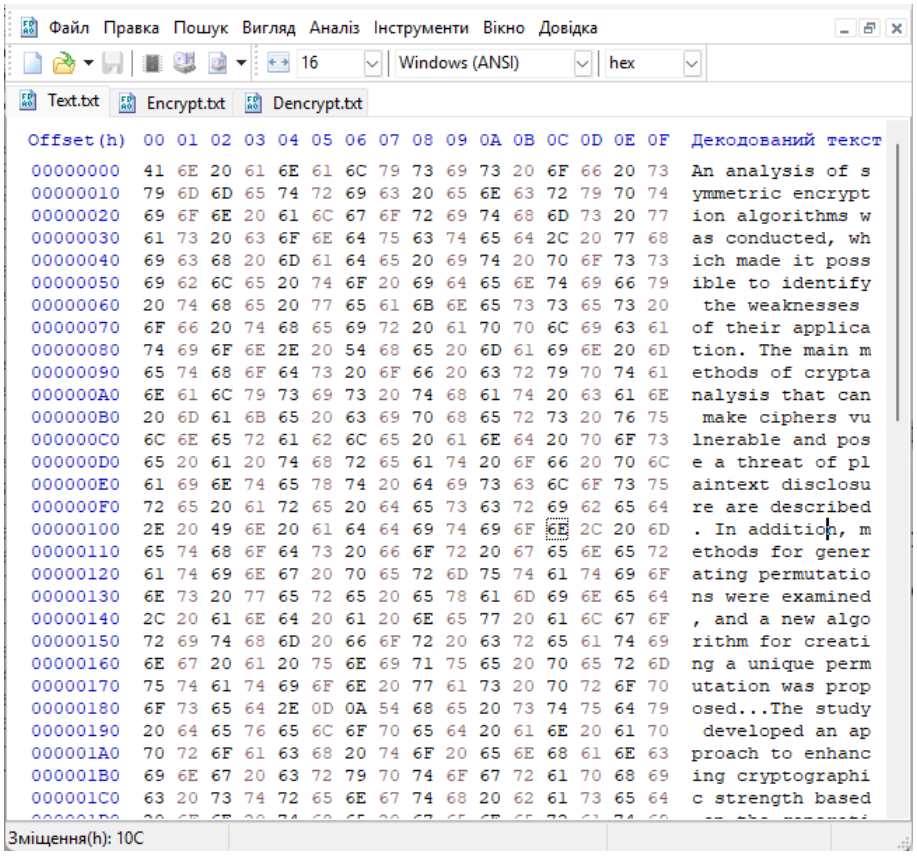

Рисунок 3.7 – Вхідне повідомлення для шифрування

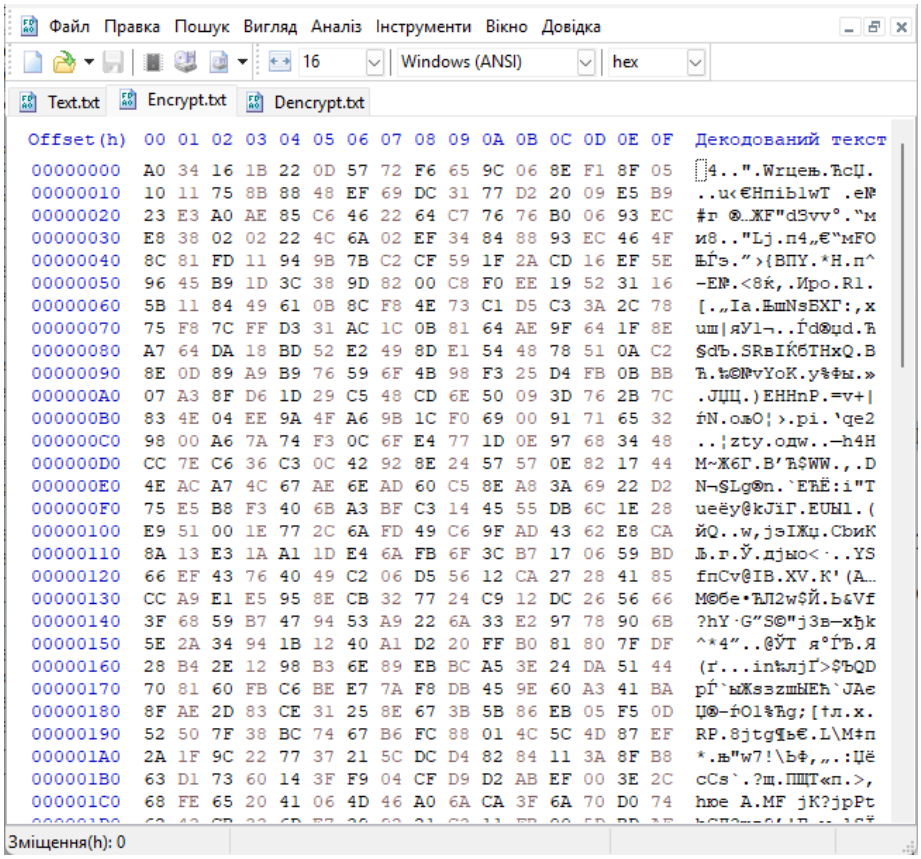

Рисунок 3.8 – Криптограма

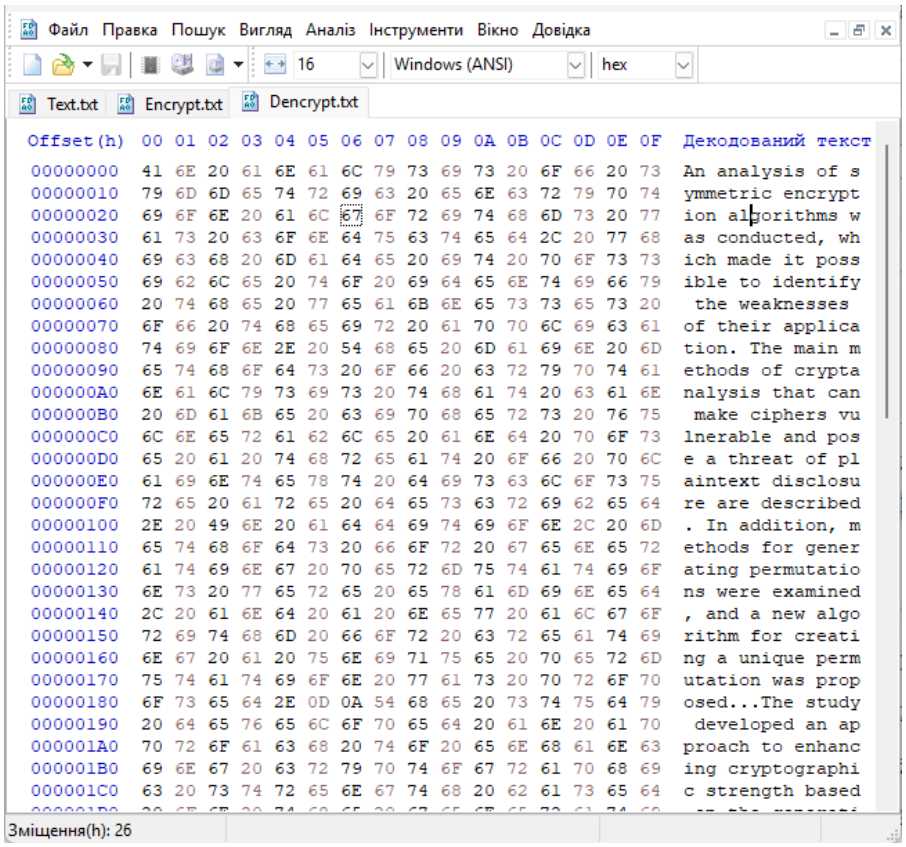

Рисунок 3.9 – Дешифроване повідомлення

Останнім етапом виступає дешифрування інформації, у результаті якого формується відновлений файл. Приклад результату дешифрування наведено на рисунку 3.9.

На основі наведених прикладів можна дійти висновку, що алгоритм функціонує стабільно, забезпечує правильне шифрування та успішне відновлення вихідних даних. Інформацію, зашифровану цим методом, можна безпечно передавати відкритими каналами зв’язку, адже спроби відновлення вмісту за допомогою криптоаналізу є надзвичайно складними для сучасних методів і технічних засобів, на яких здійснюється обчислення задач цього типу.

3.4 Висновок до розділу

Розроблена програма успішно реалізує базову систему шифрування та дешифрування файлів з використанням сучасних криптографічних принципів.

Система демонструє роботу алгоритму AES, застосування хешування для формування ключів, а також роботу з графічним інтерфейсом у Python.

Завдяки зручному GUI користувач може легко виконувати шифрування файлів без спеціальних знань у галузі криптографії.

Розробка може бути використана як навчальний приклад для ознайомлення з методами симетричного шифрування та практичною реалізацією криптографічних алгоритмів у мові Python.

Можливі напрямки вдосконалення:

- реалізувати реальні перестановки блоків у функції permute_block_inverse();
- змінити режим шифрування AES на CBC або GCM із використанням вектора ініціалізації (IV);
- додати індикатор прогресу для великих файлів;
- реалізувати перевірку цілісності даних (через HMAC);
- додати підтримку декількох файлів або цілих папок.

ВИСНОВКИ

У процесі підготовки та написання магістерської роботи було створено алгоритм, призначений для підвищення рівня криптостійкості симетричних систем шифрування. Розроблене рішення також забезпечує додатковий захист від атак, що базуються на використанні відомих або відкритих текстів. Окрім теоретичного опису, було реалізовано програмну версію цього алгоритму.

У ході дослідження досягнуто такі основні результати:

- опрацьовано інформацію щодо класифікації симетричних алгоритмів шифрування, розглянуто їх принципи роботи та детально описано найпоширеніші представники цього класу методів;

- проведено аналіз різних типів атак, спрямованих на злам алгоритмів шифрування, а також вивчено основні підходи криптоаналізу, які застосовуються для пошуку або відновлення ключів шифрування даних;

- відібрано приклад алгоритму зі зниженою стійкістю, який було використано для демонстрації вразливостей слабких шифрів, та наведено опис;

- досліджено наукові джерела, присвячені методам створення унікальних перестановок, систематизовано відомі підходи й розроблено власний спосіб генерації перестановок, який став основою для створення нового алгоритму підсилення криптостійкості;

- сформовано структурну схему алгоритму підвищення стійкості симетричних методів шифрування даних, докладно описано його етапи роботи та логіку функціонування;

- реалізовано програмне забезпечення, яке демонструє роботу алгоритму шифрування, побудованого за запропонованим методом;

- виконано тестування розробленої програми, наведено пояснення щодо її використання та особливостей взаємодії користувача з інтерфейсом;

- наведено приклади результатів шифрування, продемонстровано роботу програмного рішення на практиці.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1 Семеген Б. В. Метод підвищення криптостійкості симетричних алгоритмів шифрування : кваліфікаційна робота за освітнім рівнем «магістр» : 123 Комп’ютерна інженерія / Семеген Б. В. – Тернопіль, 2021. – 86 с.

2 Snuffle 2005: the Salsa20 encryption function. URL: http://cr.yp.to/snuffle.html (дата звернення 01.10.2025).

3 Christof Paar, Jan Pelzl, "Stream Ciphers", Chapter 2 of "Understanding Cryptography, A Textbook for Students and Practitioners", Springer, 2009. URL : https://archive.ph/20121208212741/http://wiki.crypto.rub.de/Buch/movies.php (дата звернення: 02.10.2025).

4 Diffie, Whitfield; Hellman, Martin E. (June 1977). "Exhaustive Cryptanalysis of the NBS Data Encryption Standard". URL: https://web.archive.org/web/20140226205104/http://origin-www.computer.org/csdl/mags/co/1977/06/01646525.pdf (дата звернення: 02.10.2025).

5 Jeff Moser. A Stick Figure Guide to the Advanced Encryption Standard (AES). URL: https://www.webcitation.org/65Yj0G6mK?url=http://www.moserware.com/2009/09/stick-figure-guide-to-advanced.html (дата звернення: 02.10.2025).

6 Bruce Schneier (1993). "Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish)". Fast Software Encryption, Cambridge Security Workshop. URL: https://link.springer.com/chapter/10.1007/3-540-58108-1_24 (дата звернення: 2.11.2025).

7 Алгоритм TEA. URL: https://intellect.icu/algoritmy-blochnogo-simmetrichnogo-shifrovaniya-algoritm-tea-tiny-encryption-algorithm-5741 (дата звернення: 02.10.2025).

8 Menezes A. J., Oorschot P. v., Vanstone S. A. Handbook of Applied Cryptography (англ.) – CRC Press, 1996. – 816 p.

9 Hal Tipton and Micki Krause. Handbook of Information Security Management – CRC Press LLC, 1998.

10 Knuth, D. E. The Art of Computer Programming. – Addison-Wesley, 2005. - Vol. 4.12. URL: https://www-cs-faculty.stanford.edu/~knuth/taocp.html (дата звернення: 2.11.2025).

11 Beth, Thomas; Piper, Fred (1985). The Stop and Go Generator (PDF). EUROCRYPT '84. pp. 88–92. URL: https://link.springer.com/content/pdf/10.1007/3-540-39757-4_9.pdf (дата звернення 10.11.2025).

12 Matt J. B. Robshaw, Stream Ciphers Technical Report TR-701, version 2.0, RSA Laboratories, 1995. URL: http://www.networkdls.com/Articles/tr-701.pdf (дата звернення: 14.11.2025).

13 SVG анімація простого потокового шифру. URL: https://web.archive.org/web/20120326211358/http://l-system.net.pl/crypto/simple_stream_cipher.svg (дата звернення: 15.11.2025).

14 "Principles and Performance of Cryptographic Algorithms" by Bart Preneel, Vincent Rijmen, and Antoon Bosselaers. URL: https://www.drdobbs.com/algorithm-alley/184410756 (дата звернення: 20.11.2025).

15 Zhang, X.; Seo, S.-H.; Wang, C. A Lightweight encryption method for privacy protection in surveillance videos. IEEE Access 2018, 6, 18074–18087.

16 Zhang, K.; Ni, J.; Yang, K.; Liang, X.; Ren, J.; Shen, X.S. Security and privacy in smart city applications: Challenges and solutions. IEEE Commun. Mag. 2017, 55, 122–129.

17 Kim, J.; Lee, D.; Park, N. CCTV-RFID enabled multifactor authentication model for secure differential level video access control. Multimed. Tools Appl. 2020, 79, 23461–23481.

18 Aamir Nizam Ansari, Mohamed Sedkyl, Neelam Sharm. „An Internet of Things Approach for Motion Detection using Raspberry Pi“ in 2015 International Conference on Intelligent Computing and Internet of Things, pp. 131-134

19 Ways to Protect IP Video Surveillance Systems. Електронний ресурс]. – URL : https://www.alliedtelesis.com/ua/en/blog/5-ways-protect-ip-video-surveillance-systems (дата звернення: 25.11.2025).

20 The Fun and User-Friendly Guide to the Secure Real-time Transport Protocol [Infographic]. Електронний ресурс]. URL : https://medium.com/callstatsio/the-fun-and-user-friendly-guide-to-the-secure-real-time-transport-protocol-infographic-b5ddf7da3f3e (дата звернення: 25.11.2025).

21 Nazare, A.C., Jr.; Schwartz, W.R. A scalable and flexible framework for smart video surveillance. Comput. Vis. Image Underst. 2016, 144, 258–275.

22 Sultana, T.; Wahid, K.A. IoT-guard: Event-driven fog-based video surveillance system for real-time security management. IEEE Access 2019, 7, 134881–134894.

23 Guo, J.; Zheng, P.; Huang, J. An efficient motion detection and tracking scheme for encrypted surveillance videos. ACM Trans. Multimed. Comput. Commun. Appl. 2017, 13, 1–23.

24 Obermaier, J.; Hutle, M. Analyzing the security and privacy of cloud-based video surveillance systems. In Proceedings of the 2nd ACM International

25 Workshop on IoT Privacy, Trust, and Security; Association for Computing Machinery: New York, NY, USA, 2016; pp. 22–28.

26 K. M. Hosny, A. Magdi, A. Salah, O. El-Komy and N. A. Lashin, “Internet of things applications using Raspberry-Pi: A survey,” International Journal of Electrical & Computer Engineering, vol. 13, pp. 2088– 8708, 2023.

27 V. A. Daisy, C. V. Joe and S. S. S. Sugi, “An image-based authentication technique using visual cryptography scheme,” in 2017 Int. Conf. on Inventive Systems and Control (ICISC), Piscataway, IEEE, pp. 1–6, 2017. https://doi.org/10.1109/ICISC.2017.8068666

28 M. A. Razzaq, R. A. Sheikh, A. Baig and A. Ahmad, “Digital image security: Fusion of encryption, steganography, and watermarking,” International Journal of Advanced Computer Science and Applications (IJACSA), vol. 8, no. 5, pp. 224–228, 2017.

29 P. Chinnasamy, S. Padmavathi, R. Swathy and S. Rakesh, “Efficient data Security using hybrid cryp-tography on cloud computing,” in Inventive Communication and Computational Technologies, Singapore: Springer, pp. 537–547, 2021.

30 M. Begum and M. S. Uddin, “Digital image watermarking techniques: A review,” Information, vol. 11, no. 2, pp. 110, 2020.

31 S. Gupta, U. Raikar, B. M. P. Patil and R. Molavade, “Image processing based intelligent traffic control system by using Raspberry Pi,” International Journal for Research in Applied Science & Engineering Technology (IJRASET), vol. 6, pp. 66–70, 2018.

32 K. S. Shilpashree, H. Lokesha and H. Shivkumar, “Implementation of image processing on Raspberry Pi,” International Journal of Advanced Research in Computer and Communication Engineering, vol. 4, no. 5, pp. 199–202, 2015.

33 A. S. Alanazi, N. Munir, M. Khan, M. Asif and I. Hussain, “Cryptanalysis of novel image encryption scheme based on multiple chaotic substitution boxes,” IEEE Access, vol. 9, pp. 93795–93802, 2021.

34 Li N. Symmetric Boolean functions depending on an odd number of variables with maximum algebraic immunity / N.Li and W.Qi.. // IEEE Transaction on Information Theory,-2018,- Vol.63, № 5, - PP. 2271-2273.

35 Li N. On the construction of Boolean functions with optimal algebraic immunity / N. Li, L. Qu, W.-F. Qi, G. Feng, C. Li and D. Xie. // IEEE Transaction on Information Theory,-2008,- Vol.53, № 3, - PP. 1330-1334.

36 Lacharme P. On the nonlinearity of power functions./ P. Langevin and P. V´eron. // Designs, Codes and Cryptography. Vol. 47, 2015. - PP. 31 – 43.

37 Meng Q. On the degree of homogeneous bent functions / Q. Meng, H. Zhang, M. Yang and J. Cui. //. Discrete Applied Mathematics Volume 165, № 5. - 2017.- PP. 665-669.

38 Qu L. Constructing symmetric Boolean functions with maximum algebraic immunity / L. Qu, K. Feng, L. Feng and L. Wang // IEEE Trans. on Inf. Theory, vol. 55- № 6.- 2009.- PP. 2406-2412.

ДОДАТКИ

Додаток А

Код програми

```
import tkinter as tk
from tkinter import filedialog, messagebox
import hashlib
import struct
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes
import os
# --- Допоміжні функції ---
def md5_permutation_number(password, n_blocks=12):
    md5 = hashlib.md5(password.encode('utf-8')).digest()
    part1 = struct.unpack('<Q', md5[:8])[0]
    part2 = struct.unpack('<Q', md5[8:16])[0]
    return (part1 ^ part2) % n_blocks
def generate_random_block_data():
    return get_random_bytes(12)  # 3 * 4 байти
def add_block_metadata(block_data, perm_number):
    if len(block_data) < 48:
        block_data += b'\x00' * (48 - len(block_data))
    block = bytearray(block_data)
    block += struct.pack('<I', perm_number)
    block += generate_random_block_data()
    # XOR 3*4 байт
    for i in range(3):
        for j in range(4):
            block[i*4 + j] ^= block[48 + i*4 + j]
    return bytes(block)
def permute_block_inverse(block, n, perm_number):
    return block  # поки що passthrough
def encrypt_file(source, dest, password):
    with open(source, 'rb') as f:
        data = f.read()
    n_blocks = 12
    perm_number = md5_permutation_number(password, n_blocks)
```

```
    blocks = [add_block_metadata(data[i:i+48], perm_number + i)
for i in range(0, len(data), 48)]
    final_data = b''.join(blocks)
    key = hashlib.sha256(password.encode('utf-8')).digest()
    cipher = AES.new(key, AES.MODE_ECB)
    padding_len = 16 - (len(final_data) % 16)
    final_data += bytes([padding_len] * padding_len)
    encrypted = cipher.encrypt(final_data)
    with open(dest, 'wb') as f:
        f.write(encrypted)
def decrypt_file(source, dest, password):
    with open(source, 'rb') as f:
        encrypted = f.read()
    key = hashlib.sha256(password.encode('utf-8')).digest()
    cipher = AES.new(key, AES.MODE_ECB)
    decrypted = cipher.decrypt(encrypted)
    padding_len = decrypted[-1]
    if padding_len < 1 or padding_len > 16:
        padding_len = 0
    decrypted = decrypted[:-padding_len] if padding_len else
decrypted
    n_blocks = 12
    perm_number = md5_permutation_number(password, n_blocks)
    recovered = bytearray()
    for i in range(0, len(decrypted), 64):
        block = decrypted[i:i+64]
        if len(block) < 64:
            block += b'\x00' * (64 - len(block))
        block = bytearray(block)
        block[:48] = permute_block_inverse(block[:48], n_blocks,
perm_number)
        for j in range(3):
            for k in range(4):
                block[j*4 + k] ^= block[48 + j*4 + k]
        recovered += block[:48]
```

```
            perm_number = (perm_number + 1) & 0xFFFFFFFF
        recovered = recovered.rstrip(b'\x00')
        with open(dest, 'wb') as f:
            f.write(recovered)
# --- GUI ---
class EncryptDecryptGUI:
    def __init__(self, master):
        self.master = master
        master.title("Шифрування/Дешифрування файлів")
        # Source file
        tk.Label(master, text="Файл для обробки:").grid(row=0,
column=0, sticky="e")
        self.src_entry = tk.Entry(master, width=50)
        self.src_entry.grid(row=0, column=1)
        tk.Button(master, text="Вибрати",
command=self.browse_src).grid(row=0, column=2)
        # Destination file
        tk.Label(master, text="Файл результату:").grid(row=1,
column=0, sticky="e")
        self.dest_entry = tk.Entry(master, width=50)
        self.dest_entry.grid(row=1, column=1)
        tk.Button(master, text="Вибрати",
command=self.browse_dest).grid(row=1, column=2)
        # Password
        tk.Label(master, text="Пароль:").grid(row=2, column=0,
sticky="e")
        self.pass_entry = tk.Entry(master, width=50, show="*")
        self.pass_entry.grid(row=2, column=1, columnspan=2)
        # Buttons
        tk.Button(master, text="Зашифрувати",
command=self.encrypt).grid(row=3, column=0, pady=10)
        tk.Button(master, text="Розшифрувати",
command=self.decrypt).grid(row=3, column=1, pady=10)
        tk.Button(master, text="Вийти",
command=master.quit).grid(row=3, column=2, pady=10)
```

```
    def browse_src(self):
        filename = filedialog.askopenfilename(title="Виберіть
файл")
        if filename:
            self.src_entry.delete(0, tk.END)
            self.src_entry.insert(0, filename)
    def browse_dest(self):
        filename = filedialog.asksaveasfilename(title="Зберегти
файл як")
        if filename:
            self.dest_entry.delete(0, tk.END)
            self.dest_entry.insert(0, filename)
    def encrypt(self):
        src = self.src_entry.get()
        dest = self.dest_entry.get()
        password = self.pass_entry.get()
        if not all([src, dest, password]):
            messagebox.showwarning("Помилка", "Заповніть всі
поля!")
            return
        try:
            encrypt_file(src, dest, password)
            messagebox.showinfo("Успіх", f"Файл зашифровано:
{dest}")
        except Exception as e:
            messagebox.showerror("Помилка", str(e))

    def decrypt(self):
        src = self.src_entry.get()
        dest = self.dest_entry.get()
        password = self.pass_entry.get()
        if not all([src, dest, password]):
            messagebox.showwarning("Помилка", "Заповніть всі
поля!")
            return
```

```
        try:
            decrypt_file(src, dest, password)
            messagebox.showinfo("Успіх",   f"Файл   розшифровано:
{dest}")
        except Exception as e:
            messagebox.showerror("Помилка", str(e))
if __name__ == "__main__":
    root = tk.Tk()
    gui = EncryptDecryptGUI(root)
    root.mainloop()
```

БІБЛІОГРАФІЧНА ДОВІДКА

Тема дипломної роботи: "Покращення стійкості симетричних криптоалгоритмів для реалізації на пристроях з обмеженими обчислювальними ресурсами"

Обсяг пояснювальної записки 55 аркушів.

28 рисунки;

3 таблиці;

1 додатків.

Дата завершення роботи: *10 грудня 2025 р.*

Підпис студента-дипломника _____ *Грицан М. І.*