МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра кібербезпеки

ОЛІЙНИК Назар Петрович

Криптоаналіз симетричного шифру AES / Cryptanalysis of the symmetric AES cipher

спеціальність: 125 – Кібербезпека
освітньо-професійна програма –Кібербезпека
Кваліфікаційна робота

Виконав студент групи КБм -21
Н.П. Олійник

Науковий керівник
к.т.н., доцент С.В. Івасьєв

Кваліфікаційну роботу допущено
до захисту:

«____» _____ ___2022 р.

Завідувач кафедри
_____ В.В.Яцків

ТЕРНОПІЛЬ - 2022

Факультет комп'ютерних інформаційних технологій
Кафедра кібербезпеки
Освітній ступінь «магістр»
спеціальність: 125 - Кібербезпека
освітньо-професійна програма –Кібербезпека

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ В.В.Яцків
____" _____ 2021 року

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

ОЛІЙНИКУ Назару Петровичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи:
Криптоаналіз симетричного шифру AES / Cryptanalysis of the symmetric AES cipher
керівник роботи к.т.н., доцент С.В. Івасьєв

затверджені наказом по університету від 31 грудня 2021 року № 606

2. Строк подання студентом закінченої випускної кваліфікаційної роботи 16 листопада 2022 року.

3. Вихідні дані до кваліфікаційної роботи: завдання на випускну кваліфікаційну роботу студента, наукові статті, технічна література.

4. Основні питання, які потрібно розробити:
- дослідити симетричні алгоритми шифрування;
- дослідити симетричний шифр FIPS-197;
- провести дослідження математичних основ алгоритму AES;
- дослідити раундові перетворення в криптосистемі AES;
- дослідити методи лінійного та диференціального криптоаналізу.
- дослідити показники стікості до атак диференціального та лінійного криптоаналізу.

5. Перелік графічного матеріалу у роботі.
- Схема шифрування алгоритмом AES.
- Бінарна форма ключа AES
- Формування слів в алгоритмі AES.
- Схема проведення раундових перетворень.

- Схема перетворень AES.
- Процес зміни байтів за допомогою таблиці.
- Процес зміни байтів за допомогою таблиці .
- Схема операції ShiftRows.
- Схема операції MixColumns.
- Схема операції AddRoundKey.
- Чотирьохраундовий неможливий диференціал.

6. Консультанти розділів кваліфікаційної роботи

	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 11 жовтня 2021 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строки виконання етапів кваліфікаційної роботи	Примітка
1	Аналіз предметної області	12.2021 р. – 03.2022 р.	
2	Дослідження криптоперетворень AES	03.2022 р. – 05.2022 р.	
3	Криптоаналіз алгоритму AES	05.2022 р. – 11.2022 р.	

Студент _____ Н.П. Олійник.
(підпис)

Керівник роботи _____ к.т.н., доцент С.В. Івасьєв
(підпис)

АНОТАЦІЯ

Кваліфікаційна робота на тему «Криптоаналіз симетричного шифру AES» на здобуття освітнього ступеня «Магістр» зі спеціальності 125 «Кібербезпека» освітньо-професійної програми «Кібербезпека» написана обсягом 70 сторінки і містить 18 ілюстрації, 6 таблиць, 2 додатки та 32 джерела за переліком посилань.

Мета роботи полягає в дослідженні підходів до крипто аналізу асиметричних систем, зокрема AES.

Проведено дослідження симетричних алгоритмів шифрування та аналіз симетричного шифру FIPS-197, котрий був конкурсантом конкурсу AES та побудований основі *S* – блоків.

Проведено дослідження математичних основ алгоритму AES, що дає можливість проаналізувати можливі алгоритми крипто аналізу шифру та досліджено раундові перетворення в криптосистемі AES.

Досліджено методи лінійного та диференціального крипто аналізу та показники стійкості до атак диференціального та лінійного крипто аналізу і розроблено схему криптоаналізу за допомогою неможливих диференціалів для 5 раундового шифру AES.

Систематизовано елементи лінійного та диференціального крипто аналізу та реалізовано програмне забезпечення для дешифрування даних алгоритмом AES на мові C++.

Результати роботи можуть успішно застосовуватися при реалізації систем перевірки криптостійкості асиметричних систем схожих з AES.

Ключові слова: AES, FIPS, АСИМЕТРИЧНІ КРИПТОСИСТЕМИ, КРИПТОАНАЛІЗ, НЕМОЖЛИВІ ДИФЕРЕНЦІАЛИ.

ABSTRACT

The qualification work on the topic "Cryptanalysis of the symmetric AES cipher" for obtaining the Master's degree in the specialty 125 "Cybersecurity" of the educational and professional program "Cybersecurity" is written in the volume of 70 pages and contains 18 illustrations, 6 tables, 2 appendices and 32 sources according to the list of references .

The purpose of the work is to study approaches to cryptoanalysis of asymmetric systems, in particular AES.

A study of symmetric encryption algorithms and an analysis of the FIPS-197 symmetric cipher, which was a competitor of the AES competition and built on the basis of S-blocks, was carried out.

A study of the mathematical foundations of the AES algorithm was carried out, which makes it possible to analyze possible algorithms for cryptographic analysis of the cipher, and round transformations in the AES cryptosystem were investigated.

Methods of linear and differential cryptanalysis and indicators of resistance to attacks of differential and linear cryptanalysis were studied, and a cryptanalysis scheme using impossible differentials for the 5-round AES cipher was developed.

Elements of linear and differential crypto analysis were systematized and software for data decryption using the AES algorithm in C++ was implemented.

The results of the work can be successfully applied in the implementation of systems for checking the cryptoresistance of asymmetric systems similar to AES.

Keywords: AES, FIPS, ASYMMETRIC CRYPTOSYSTEMS, CRYPTOANALYSIS, IMPOSSIBLE DIFFERENTIALS.

ЗМІСТ

ВСТУП

Актуальність роботи. Симетричні криптосистеми незважаючи на поширення симетричних криптосистем мають велике значення та часто є частиною схем обміну ключів та використовуються паралельно з симетричною криптографією. Досить часто їх використовують в задачах де потрібно шифрувати великі обсяги інформації з незначними часовими затримками, наприклад зберігання даних. Одним з однозначних лідерів серед симетричних криптосистем є шифр AES. Дослідження його криптостійкості та засобів взлому є безумовно актуальною задачею зважаючи на широке застосування.

Мета роботи полягає в дослідженні підходів до криптоаналізу симетричних систем, зокрема AES:

Для досягнення даної мети ставились наступні завдання:

- дослідити симетричні алгоритми шифрування;
- дослідити симетричний шифр FIPS-197;
- провести дослідження математичних основ алгоритму AES;
- дослідити раундові перетворення в криптосистемі AES;
- дослідити методи лінійного та диференціального криптоаналізу.
- дослідити показники стікості до атак диференціального та лінійного криптоаналізу.

Об’єкт дослідження - процеси криптографічних перетворень в блокових шифрах.

Предмет досліджень – симетричний алгоритм шифрування AES.

Методи досліджень базуються на алгоритмах шифрування симетричних криптосистем та криптоаналізі.

Наукова новизна одержаних результатів визначається наступним чином:

- Розроблено схему криптоаналізу за допомогою неможливих диференціалів для 5 раундового шифру AES.

Практична цінність одержаних результатів полягає в тому, що:

– Систематизовано елементи лінійного та диференціального криптоаналізу;

– реалізовано програмне забезпечення для дешифрування даних алгоритмом AES на мові C++.

Публікації та апробація до магістерської роботи:

1. Кульчинська Н.З., Олійник Н.П., Дослідження алгоритму шифрування AES. Збірник матеріалів проблемно-наукової міжгалузевої конференції «Автоматизація та комп'ютерно – інтегровані технології» (АКІТ - 2021), Тернопіль, 2021. 145-149 с.

2. Олійник Н.П., Використаання симетричного шифру AES з реалізацією на Javascript. Збірник матеріалів проблемно-наукової міжгалузевої конференції «Автоматизація та комп'ютерно – інтегровані технології» (АКІТ - 2022), Тернопіль, 2022. 73-76 с.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Симетричні алгоритми шифрування

У 2000 році завершено міжнародний проект по створенню алгоритму шифрування AES (Advanced Стандарт шифрування). В результаті інтенсивної спільної роботи провідних криптологів світу було глибоко проаналізовані та досліджені властивості 15 алгоритмів-кандидатів на стандарт симетричного блочного шифра XXI століття. При виконанні проекту створена потужна методологічна і методична база розробка та аналіз симетричних блокових алгоритмів, що дозволила вибрати з претендентів симметричний блоковий шифр Rijndael. Після завершення проекту досліджувався алгоритм Rijndael спеціальними підрозділами Агентства національної безпеки США на відповідність заданому рівню стійкості, можливостей і умов застосування для захисту несекретної інформації в державних і комерційних структурах США. Таким чином, для заміни алгоритму DES і Triple DES (FIPS-46-3) прийнято новий алгоритм симетричного блокового шифрування, який офіційно введено в дію в якості федерального стандарту США – AES (FIPS-197) [1].

Крім того, в 2000-му році був розпочатий проект NESSIE (Нові європейські схеми для підписів, цілісності та Шифрування), метою якого є створення кількох криптографічних примітивів, серед яких є і симетричний блоковий шифр. По суті, до цього часу проект уже законодавчо та прийнято попередні рішення про те, що серед блокових алгоритмів найкраще підходять Rijndael, Camellia і Шакал-2. Вони, очевидно, і рекомендовані в якості стандартів Європейського Союзу.

У 2000 році завершено міжнародний проект створення алгоритму шифрування AES (Advanced Encryption Standard). Внаслідок інтенсивної спільної роботи провідних криптологів світу були глибоко проаналізовано та досліджено властивості 15 алгоритмів-кандидатів на стандарт симетричного блокового шифру XXI століття. За виконання проекту створено потужну методологічну та методичну базу розробки та аналізу симетричних блокових

алгоритмів, що дозволило вибрати з претендентів симетричний блоковий шифр Rijndael. Після закінчення проекту алгоритм Rijndael досліджувався спеціальними підрозділами Агентства національної безпеки США на відповідність заданому рівню стійкості, можливостей та умов застосування для захисту несекретної інформації в державні та комерційні структури США. Таким чином, для заміни алгоритму DES та Triple DES (FIPS-46-3) прийнято новий алгоритм симетричного блочного шифрування, який офіційно вступив у дію як федеральний стандарт США – AES (FIPS-197) [1].

Крім того, у 2000-му році було розпочато проект NESSIE (New European Schemes for Signatures, Integrity and Encryption), метою якого є створення кількох криптографічних примітивів, серед яких є й симетричний блоковий шифр. По суті, до теперішнього часу проект вже закінчено, і прийнято попередні рішення про те, що серед блокових алгоритмів найкращими є Rijndael, Camellia та Shacal-2. Вони, очевидно, і будуть рекомендовані як стандарти Європейського Союзу.

В даний час в Україні офіційно рекомендовано до застосування симетричний блоковий. алгоритм шифрування ДСТУ 28147-89, який, за критеріям стійкості проекту NESSIE забезпечує найнижчий (з допустимих) рівень безпеки – нормальний успадкований. Тому дуже важливим є завдання оцінки та вибору для застосування в Україні одного з симетричних блокових шифрів, що пройшли глибокі дослідження у процесі виконання проектів AES та NESSIE. Проведемо узагальнення результатів, отриманих під час виконання проектів AES та NESSIE, обґрунтування та формулювання рекомендацій для використання в Україні федерального стандарту США FIPS-197

1.2 Симетричний алгоритм FIPS-197

Сучасні симетричні блокові алгоритми шифрування мають широку сферу застосування, що включає, крім традиційного шифрування, побудову кодів

аутентифікації повідомлень, хешфункцій, протоколів аутентифікації і т. д. Реалізація алгоритму може виконуватися апаратно або програмно, захист інформації може забезпечуватися як у локальних системах, так і в телекомунікаційних мережах. У зв'язку з цим до використовуваних і проектованих блокових шифрів, в т.ч. FIPS-197 [1, 2] висуваються жорсткі вимоги, серед яких можна виділити такі.

1. Просте криптографічне ядро із прозорими принципами його проектування.

2. Стійкість, що включає використання стійкого і перевіреного криптографічного ядра, відповідної процедури створення підключів, відсутність слабких ключів та лазівок; стійкість до відомих методів криптоаналізу, включаючи диференціальний та лінійний криптоаналіз, тимчасову атаку тощо, – відсутність аналітичних атак, ефективніших, ніж силові атаки.

3. Простота реалізації, що передбачає можливість ефективного використання шифру на різних процесорах (від 8-бітних до 64-бітних), під різними операційними системами, можливість побудови спеціалізованих апаратних засобів, що реалізують алгоритм.

4. Гнучкість використання: можливість розробки з урахуванням шифру функцій хешування, генераторів псевдовипадкових чисел, кодів аутентифікації повідомлень та потокових шифрів.

5. Висока продуктивність, що дозволяє реалізувати перетворення з мінімальним споживанням обчислювальних ресурсів процесора.

6. Застосовність при реалізації на смарт-картах із забезпеченням прийнятної продуктивності.

7. Прийнятна вартість реалізації.

8. Відсутність патентних обмежень, що юридично дозволяє розробникам вільно використовувати алгоритм у будь-яких додатках.

При проектуванні алгоритму Rijndael розробники використовували максимально прозорі правила побудови та представили докази стійкості до кількох видів криптоаналізу, які можуть бути застосовні до шифру. Алгоритм

має просту структуру, що дозволяє досягти високої продуктивності та виконати ефективну реалізацію на різних платформах, що має досить низьку вартість. Серед усіх кандидатів AES та NESSIE алгоритм Rijndael є одним з найбільш продуктивних та простих у реалізації.

Шифр не має патентних обмежень і може вільно використовуватися розробниками в різних додатках. До цього моменту у відкритих публікаціях не описано жодного з методів, який дозволив би виконати криптоаналіз зі складністю, менш ніж складність силової атаки на шифр.

В даний час алгоритм Rijndael відповідає всім перерахованим вище вимогам. За результатів оцінки конкретних показників безпеки, продуктивності та простоти реалізації Rijndael став фіналістом конкурсів AES і NESSIE, а надалі був прийнятий як федеральний стандарту США (FIPS-197). Алгоритм Rijndael побудований на основі модифікованої SPN-структури (Substitution-Permutation) Network), і є шифром, який використовує підстановку та лінійне перетворення (substitution-linear transformation network) з 10, 12 чи 14 циклами шифрування, залежно від довжини ключа [3]. Алгоритм є байт-орієнтованим та побудований на основі попередньої розробки авторів – шифрі Square.

Основне нелінійне перетворення - підстановка (*S*-box) має математичну структуру і можливо представлена як обчислення зворотного елемента в полі GF(28) та подальше афінне перетворення.

Підключі вводяться з використанням операції додавання по модулю 2 (XOR, EXOR). Розсіювання та поширення досягається за рахунок байтової перестановки (зсуву рядків) та МДР-перетворення, що гарантує максимальну відстань Хеммінга між вхідними та вихідними даними.

Перетворення, що використовуються, є досить відомими і мають хороші криптографічними властивостями. Процедура вироблення підключів має просту структуру, що обумовлює значну нерівномірність розмноження помилок у підключах, що залежить від номера біта у ключі шифрування [7]. Тим не менш, пропозиції деяких дослідників змінити цю процедуру призвели до послаблення модифікованого варіанту шифрування.

В основі проектування алгоритму лежить так звана стратегія wide-trail («широкий слід»), що передбачає максимальну кількість гілок активізації і, відповідно, низьку ймовірність знаходження правильної пари характеристик в диференціальному криптоаналізі та виконання апроксимації у лінійному криптоаналізі.

Для усунення можливості появи шляхів активізації з низьким числом розгалужень авторами шифру пропонуються такі обмеження [8].

1. Здійснювати вибір підстановок (*S*-блоків) з мінімальними диференціальними та кореляційними значеннями.

2. Вибирати циклове перетворення в такий спосіб, щоб перешкоджати можливості побудови шляхів активізації з низьким числом розгалужень.

Розробники алгоритму підкреслюють додаткову перевагу використання стратегії wide-trail рахунок використання ефективного циклового перетворення можна відмовитися від застосування підстановок великої розмірності, що значно заощадить ресурси системи (вимоги до обсягу пам'яті) і дозволить досягти гарного поширення на кількох циклах. При використанні цієї стратегії в AES виконано чергування лінійного відображення та підстановки, що дозволило досягти високого ступеня розсіювання в межах однієї підстановки та поширення в цикловій функції [8].

При виборі підстановки (*S*-блоку) автори керувалися такими критеріями [2]:

- оборотність (бієктивність);

- мінімізація найбільшого значення у таблиці лінійної апроксимації підстановки;

- мінімізація найбільшого значення у таблиці розподілу різниць;

- складність представлення алгебри в полі $GF(2^8)$;

- простота опису.

Відповідно до рекомендацій, викладених у [9], автори обрали мультиплікативне звернення до поля $GF(2^8)$. Для запобігання надзвичайно простому опису алгебри було додано афінне перетворення, що виключає появу

«фіксованих точок» $\left(\left\{x \in GF(2^8) \middle| S(x) = x\right\} = 0\right)$ та «інвертованих фіксованих точок» $\left(\left\{x \in GF(2^8) \middle| S(x) = \bar{x}\right\} = 0\right)$.

Зазначається, що крім варіанту, запропонованого авторами, існують інші підстановки, які мають необхідні властивості. Крім того, завдяки структурі циклової функції, шифр буде стійким до диференційного та лінійного криптоаналізу з більшістю випадкових підстановок, не задовольняючим перерахованим критеріям.

Поширення в ході шифрування є важливою властивістю, що зумовлює стійкість алгоритму і забезпечується перетворенням MixColumn. При виборі автори шифру керувалися такими критеріями [2]:

- оборотність;
- лінійність у GF(2);
- відповідний ступінь поширення;
- можливість швидкої реалізації на 8-бітових процесорах;
- симетрія;
- простота опису.

Відповідно до цих умов авторами було обрано множення на поліном 4-го ступеня в полі $GF(2^8)$.

Байтова перестановка ShiftRow використовується для поширення змін однієї колонки на весь блок та забезпечення захисту від додаткових криптоаналітичних атак. При виборі байтової перестановки застосовувалися такі критерії [2]:

- всі зрушення рядків є різними, причому один із них виконує тривіальне відображення (нульовий зсув);
- має бути забезпечений захист від атаки усічених диференціалів;
- має бути забезпечений захист від Square-атаки;
- перетворення має бути максимально простим.

З можливих варіантів було обрано найпростіший. Процедура розгортання підключів, крім звичайних функцій, додатково необхідна для усунення симетрії

всередині циклової функції та однотипності різних циклів перетворення. При виборі процедури використовувалися такі критерії [2]:

- оборотність перетворення;

- висока продуктивність на різних процесорах;

- наявність спеціальних констант для різних циклів для усунення однотипності циклів шифрування;

- поширення змін ключа шифрування у зміни підключів;

- знання кількох бітів ключа шифрування або підключа не повинно давати можливість обчислити інші біти;

- нелінійність для запобігання визначення різниці підключів по різниці ключа шифрування;

- простота опису.

Розроблена процедура вироблення підключів має низьку складність і, відповідно, високу продуктивність. Однак, завдяки простоті, реалізація критерію поширення змін ключа шифрування має недостатньо добрі властивості [7], що згодом дозволило реалізувати атаку на 9 циклів (з 14) AES-256, засновану на зв'язаних підключеннях [10]. Тим не менш, ця властивість не вдалося використовувати у повному варіанті шифру за будь-якої довжини ключа (128, 192 і 256 бітів).

Крім того, автори алгоритму представили розрахунок складності [2] та математичний доказ стійкості шифру до диференціального та лінійного криптоаналізу [11]. В цілому, при проектуванні автори намагалися дотримуватися наступних критеріїв [12]:

- симетрія циклового перетворення та однотипність різних циклів;
- ортогональність компонентів шифру;
- відсутність арифметичних операцій (за модулем, що перевищує 2).

Симетрія перетворень дозволяє виконати ефективну реалізацію алгоритму на процесорах з паралельною архітектурою (SIMD-інструкції, Single Instruction for Multiple Data), зменшити обсяг коду за рахунок використання дзвінків однієї функції та скоротити обсяг необхідної пам'яті (так як використовується лише одна підстановка «байт-в-байт»).

Згодом, для того, щоб симетрія циклового перетворення та однотипність різних циклів (перший критерій) не призвела до вразливості шифру, був сформульований додатковий критерій процедури виробітку підключів.

Ортогональність компонентів дозволяє значно збільшити стійкість алгоритму рахунок ускладнення алгебраїчного опису перетворень загалом.

Відсутність арифметичних операцій дозволяє значно спростити апаратну реалізацію шифру у вигляді ПЛІС або рекомендований ІМС.

Автори наводять такі аргументи для прийняття алгоритму Rijndael як стандарт AES [12]:

- стійкість: Rijndael має високий рівень стійкості, що відповідає решті кандидатів-фіналістів;

- ефективність: реалізації Rijndael мають значну перевагу у продуктивності по порівняно з реалізаціями інших алгоритмів;

- прозорі принципи проектування та простота, що дозволяє переконатися у відсутності закладок, виконати просту та швидкодіючу апаратну та програмну реалізацію на безлічі платформ тощо;

- розширюваність: Rijndael може бути легко адаптований до інших довжин ключів та блоків.

Виходячи з представлених авторами алгоритму даних та результатів подальших досліджень незалежних фахівців можна зробити висновок про те, що при проектуванні шифру використовувалася добре досліджена у криптографії математична база. У звітах, поданих разом з алгоритмом, досить обґрунтовано та прозоро викладаються принципи проектування та критерії, яким проводився вибір того чи іншого рішення. Спираючись на інтенсивні дослідження шифру в протягом останніх п'яти років, можна очікувати, що заява про відсутність у шифрі прихованих лазівок відповідає дійсності.

Таким чином, спираючись на викладене вище, можна стверджувати, що алгоритм Rijndael має належний рівень математичного обґрунтування, принципи його проектування досить прозорі та повністю опубліковано авторами шифру.

1.3 Криптоаналіз алгоритму шифрування FIPS-197

В даний час у відкритих публікаціях описано велику кількість криптоаналітичних атак проти симетричних блокових алгоритмів шифрування Як найбільш універсальні, слід виділити диференціальний [13] та лінійний криптоаналіз [14]. Крім того, на основі диференціального криптоаналізу розроблена атака з використанням усічених [15] та нездійсненних [16] диференціалів.

Для будь-якого ітеративного шифру, у тому числі Rijndael, потенційно можлива атака на основі зв'язаних ключів [17] та на основі колізій байт-орієнтованих функцій у складі алгоритму [18]. Оскільки при побудові Rijndael за основу був взятий блоковий шифр Square, можливе застосування Square-атаки [19] виду інтегрального криптоаналізу [20]. З перерахованих атак лінійний криптоаналіз відноситься до атак на основі відомих відкритих повідомлень, всі інші методи – до атак на основі вибраних відкритих повідомлень.

Стійкість шифру повинна бути ретельно перевірена по відношенню до всіх відомих методів, та додатково необхідно провести пошук можливих уразливостей, що призводять до потенційно ефективних аналітичних атак.

При виконанні диференціального криптоаналізу проводиться дослідження проходження різниць блоків, що шифруються через цикли алгоритму. Алгоритм вразливий до цього виду криптонападу у разі, якщо після певної кількості циклів шифрування (як правило, $N - 3$, де – кількість циклів у всьому алгоритмі) ймовірність появи заданої різниці на виході (виконання диференціальної характеристики) перевищує значення $2^{-(n-1)}$, де n – розмір блоку, що шифрується (у бітах). Виняток складає бумеранг-атака, ефективна при існуванні характеристик для $\frac{N}{2}$ циклів, але ця атака не застосовується до AES [21].

При проектуванні Rijndael було враховано можливість застосування диференціального криптоаналізу. Більше того, стратегія wide-trail, що використовується, гарантує стійкість шифру до цієї атаки. По оцінкам розробників алгоритму [2], максимальна ймовірність 4-циклової характеристики дорівнює 2^{-150}, а 8-цикловий – 2^{-300}. Враховуючи, що розмір блоку AES становить *n* 128 бітів, а мінімальна кількість циклів шифрування дорівнює 10, можна стверджувати, що алгоритм є стійким до диференціального криптоаналізу, і, більше, у шифрі закладено великий запас стійкості стосовно таких атак.

Пізніше було виконано більш точну оцінку стійкості кандидатів AES стосовно диференціальному криптоаналізу, атаці з використанням усічених та нездійсненних диференціалів [22, 23] і підтверджено стійкість Rijndael до цих видів аналізу.

Лінійний криптоаналіз застосовний до шифрів, в яких існує кореляція між блоками відкритого та зашифрованого тексту. Криптоаналітик, знаходячи суму деяких бітів відкритого та зашифрованих текстів, отримує суму бітів використаних циклових підключів. Лінійний криптоаналіз, запропонований в [16], не застосовується до шифрів, що використовують операцію додавання по модулю $m > 2$ [24]. В алгоритмі Rijndael використовується тільки операція додавання по $m = 2$, тому шифр (принаймні кілька його циклів) потенційно вразливий для цієї атаки.

У зв'язку з цим при проектуванні алгоритму розробники врахували можливість застосування лінійного криптоаналізу. Використана стратегія wide-trail гарантує відсутність лінійних апроксимацій на 4 циклі з ймовірністю, що перевищує 2^{-75}, та на 8 циклів, з ймовірністю, що перевищує 2^{-150}. Цього достатньо, щоб запобігти атаці на весь шифр, і, як і у випадку з диференціальним криптоаналізом, в алгоритмі закладено великий запас стійкості по відношенню до цієї атаки. Це результат підтверджено дослідженнями, виконаними незалежними фахівцями [25].

Таким чином, лінійний криптоаналіз не може бути використаний проти повного варіанту алгоритму шифрування AES.

Атака з використанням усічених диференціалів може бути застосована до шифру, якщо можливо поєднання досить великої кількості диференціальних характеристик із певними вхідними та вихідними різницями. У цьому випадку підсумкова ймовірність того, що обрана різниця залишиться в межах кластера, може бути обчислена незалежно від ймовірностей окремих характеристик [15, 2]. Шифри, що виконують байт-орієнтовані перетворення, потенційно більш уразливі до таких атак, тому при побудові алгоритму Rijndael було враховано можливість використання усічених диференціалів. Дослідження авторів шифру [2] та подальші уточнення незалежних спеціалістів [22] показали, що атака може бути ефективною для 5 циклів шифрування і менше.

Оскільки атака була врахована під час проектування, алгоритм FIPS-197 є захищеним і від цього виду криптонападу.

Аналіз із застосуванням нездійсненних диференціалів передбачає вибір вхідних та вихідних різниць спеціальним чином, що не допускає існування правильної пари характеристик для певної множини ключів шифрування. У разі появи правильної пари при шифруванні ці ключі виключаються з багатьох можливих, що скорочує безліч потенційних варіантів.

Автори алгоритму не проводили дослідження стійкості до аналізу із застосуванням нездійсненних диференціалів. Подальші дослідження, що використовують байт-орієнтовану структуру шифру, показали, що така атака може бути застосована для 5-циклового [26] та 6-циклового [16] варіантів алгоритму.

Відповідно, 7 циклів шифрування і більше, включаючи повний варіант алгоритму Rijndael, є захищеними від цієї атаки.

Square-атака (інтегральний криптоаналіз) є одним із найбільш ефективних методів для алгоритм Rijndael. Атака застосовна до байт-орієнтованих шифрів і вперше була застосована для алгоритму Square [21]. Згідно з дослідженнями авторів шифру, базовий варіант Square-атаки може бути використано проти 4 циклів алгоритму. Використання припущень про значення деяких байтів циклових з'єднань дозволяє збільшити кількість атакованих циклів до 6.

Пізніші дослідження дозволили використовувати незначну вразливість у процедурі вироблення підключень і розробити ефективну Square-атаку для 7 та 8 циклів шифрування [10, 27].

Повний варіант алгоритму шифрування AES при всіх довжинах ключів (128, 192 та 256 бітів) залишається невразливим для Square-атаки.

При виконанні атаки на основі пов'язаних з'єднань при шифруванні використовуються невідомі ключі із заданою криптоаналітиком залежністю. Атака може бути різновидом диференціального криптоаналізу або використовувати особливості процедури розгортання підключів, після чого криптоаналітику стають доступними проміжні значення при шифруванні (подальший розвиток це отримало в слайд-атаці [29]).

На думку авторів шифру [2], процедура розгортання підключів має високий ступінь поширення та розсіювання, що робить атаку пов'язаних ключів неймовірною. Проте дослідження показали значну нерівномірність розмноження помилок у підключах [7]. Надалі цей факт був використаний для реалізації атаки пов'язаних ключів на 9 циклів шифрування алгоритму Rijndael [10].

Тим не менш, достатня кількість циклів шифрування забезпечують стійкість шифру і до цього виду криптонападу.

Атака на основі колізій байт-орієнтованих функцій Rijndael [18] є специфічною, розробленою і в даний час відомою виключно для алгоритму Rijndael. У цій атаці використовується факт, що існує різниця між 4 циклами шифрування AES та випадковою перестановкою $\pi^R : \{0,1\}^{128} \to \{0,1\}^{128}$ що дозволяє реалізувати ефективну атаку на 7 циклів алгоритму.

Ця атака, на відміну від попередніх, не є статистичною, і заснована на використанні так званого феномена днів народження. Успішне завершення гарантовано після виконання порівняно невеликої кількості шифрувань (приблизно 2^{32}).

Повний варіант шифру є невразливим для криптоаналізу на основі колізій. Для багатьох відомих симетричних алгоритмів шифрування існують

так звані слабкі ключі, у яких деякі криптографічні властивості не виконуються. Зокрема такі ключі існують для IDEA [29], DES [30], ДСТУ 28147-89 та інших алгоритмів. Як правило, кількість таких ключів незначно, ймовірність їх появи при генерації мізерна, і додатково можна реалізувати спеціальний фільтр блокування таких ключів шифрування. Однак при використанні алгоритму у складі хеш-функції, генератора псевдовипадкових послідовностей, протоколу аутентифікації тощо. контроль над ключом шифрування не завжди можливий. В цьому випадку можливо поява значної вразливості криптографічного примітиву, створеного з урахуванням шифру.

На даний момент таких ключів для алгоритму шифрування Rijndael не виявлено. Більш того, можна припустити, що завдяки введенню в процедуру вироблення підключей елементів, що реалізують нелінійне перетворення, такі ключі взагалі немає.

Єдиним класом атак, який може бути ефективно використаний зловмисником проти будь-якого криптографічного алгоритму з секретним ключем є клас атак на реалізацію (SideChannel Attacks). У цьому випадку зловмисник має доступ до апаратної реалізації шифратора, що дозволяє йому керувати входом шифратора, проводити вимірювання часу виконання зашифрування/розшифрування, аналіз енергоспоживання, вносити збої в роботу апаратного пристрої/процесора тощо. На підставі побічної інформації відновлюється ключ шифрування.

Математичними методами або структурою алгоритму запобігти таким атакам неможливо, і для них потенційно вразливі всі фіналісти AES та NESSIE, ДСТУ 28147-89, DES, 3DES та інші алгоритми.

З часу представлення алгоритму Rijndael як кандидата AES провідними фахівцями в галузі криптографічних досліджень було опубліковано понад 20 робіт, присвячених вивченню стійкості шифру. Частина результатів досліджень авторів алгоритму, зокрема, присвячених диференційному та лінійному криптоаналізу, а також слабким ключам, було підтверджено під час незалежних досліджень.

Ряд інших криптоаналітичних методів, зокрема, Square-атака, були покращені, що дозволило реалізувати атаку для більшої кількості циклів шифрування. У процедурі вироблення підключів шифру було виявлено незначну вразливість, що дозволило реалізувати атаку, яка, за заявами авторів, неможлива для шифру. При вивченні властивостей та особливостей побудови Rijndael було розроблено новий метод криптоаналізу, який використовує колізії байт-орієнтованих внутрішніх функцій алгоритму.

Перелік методів криптоаналізу, успішно застосованих для алгоритму Rijndael [5] зі зменшеним кількістю циклів, складність проведення атаки (необхідна кількість шифрувань), а також необхідний обсяг даних наведено у табл. 1.

Як випливає з таблиці 1, для AES-128 був виконаний успішний криптоаналіз 6 і 7 циклів шифрування з 10, причому для 7 циклів Square-атаки необхідний обсяг даних складає все безліч відкритих текстів. Мінімальна складність атаки 7 циклів дорівнює 2^{120} операцій шифрування.

Для AES-192 успішно було атаковано 7 циклів шифрування з 12. Мінімальна складність склала 2^{140} операцій шифрування за потреби 2^{32} вибраних пар відкритого та зашифрованого тексту.

При дослідженні AES-256 було знайдено атаки на 7, 8 та 9 циклів із 14. Складність крипто аналізу 7 циклів дорівнює 2^{140} при необхідності 2^{32} вибраних пар відкритого та зашифрованого тексту. Атака на 9 циклів можлива лише за наявності фіксованої залежності між невідомими ключами шифрування і вимагає практично недосяжної множини відкритих текстів (таблиця 1.1).

Таблиця 1.1 – Складність різних методів криптоаналізу AES

Назва атаки	Кількість циклів	Складність	Обсяг даних
Нездійсненні диференціали	5	$\mathbf{2^{31}}$	$2^{29,5}$
Усічені диференціали	7	2^{120}	2^{119}-2^{128}
Square (128-бітовий ключ)	6	2^{44}	2^{32}
Square (128-бітовий ключ)	7	2^{120}	2^{128}
Square (192-бітовий ключ)	7	2^{155}	2^{32}
Square (256-бітовий ключ)	8	2^{172}	2^{119}-2^{128}
Колізії (128-бітовий ключ)	7	2^{128}	2^{32}
Колізії (192, 256-бітовий ключ)	7	2^{140}	2^{32}
Пов'язані ключі (256-бітовий ключ)	9	2^{224}	2^{77}

Таким чином, після 5 років інтенсивних досліджень досі у відкритих джерелах немає жодної атаки, яка може бути ефективною для повноциклового варіанта шифру, а за опублікованими атаками Існує запас стійкості від 3 до 5 циклів. Крім того, наявність прозорих критеріїв та правил проектування, сформульованих розробниками, високий рівень математичного обґрунтування дозволяють стверджувати про відсутність вбудованих розробниками лазівок.

Отже, виходячи з описаного вище, можна зробити висновок про високий рівень захисту, забезпечується алгоритмом шифрування AES.

1.4 Відкриті питання безпеки FIPS-197

Як нелінійне перетворення в алгоритмі шифрування Rijndael використовується одна підстановка (*S*-блок), застосовувана паралельно вісім

байтів опрацьовуваного блоку. В ході шифрування AES-128 ця підстановка використовується 160 разів, і це єдине нелінійне перетворенням при відображенні блоку відкритого тексту на зашифрований.

При побудові підстановки як нелінійну функцію було обрано мультиплікативний обіг елемента в полі GF(28). Ця конструкція досить добре відома в криптографії та гарантує отримання найменших максимальних значень у таблиці розподілу різниць та лінійних апроксимацій [9], забезпечуючи найкращі властивості для захисту від диференціального та лінійного криптоаналізу.

У роботі [31] були досліджені булеві функції, що утворюють підстановку AES, та виявлено, що всі 8 вихідних функцій знаходяться в одному класі щодо деякого афінного перетворення. Це означає, що для двох будь-яких вихідних бітів s_i і s_j підстановки AES існує невироджена матриця A ідеякий вектор B, такі що $s_i(x) = x_j(Ax + B)$. Звідси випливає, що у кожному циклі шифрування AES фактично 128 разів використовується одна нелінійна функція з наступними різними афінними перетвореннями. Існують й інші підтвердження цього результату із узагальненнями афінного перетворення на всі вихідні булеві функції циклової функції AES [32].

Додатково [31] зроблено висновок, що операція обчислення зворотного елемента в полі GF(28) зберігає надмірність, і цією властивістю мають усі бієктивні підстановки «8-8».

Наразі невідомо жодної відкритої публікації, в якій описано аналітичну атаку, котра яким-небудь чином використовує цю властивість AES.

Одним із критеріїв при побудові елементів шифру було використання так званих ортогональних компонентів, що запобігають простому алгебраїчному опису шифруючого перетворення. Загалом це завдання було вирішено за рахунок використання перетворень у полі GF(2) і GF(28).

В [33] пропонується новий алгоритм шифрування BES (Big Encryption System), побудований на базі AES, але кожен біт AES відповідає одному байту BES. При введенні обмеження на множину вхідних байтів B_i та байтів

шифрування ключа K_j, таких що $B_i, K_j \in \{0,1\}$, існує відображення вхідних та вихідних даних та ключів шифрування алгоритму BES у дані AES. Основна відмінність полягає в тому, що всі операції BES виконуються в одному полі GF(28). Більше того, єдине нелінійне перетворення – AES-підстановка – може бути представлена у BES у вигляді лінійного (матричного) перетворення. Це означає можливість представлення циклової функції BES (і, відповідно, AES) у вигляді мультиплікативного обігу елемента в полі GF(28) та наступного афінного перетворення: множення на матрицю M_B та додавання з вектором-ключом $K_i : X_{i+1} = F_r^{AES}(X_i) = M_B(X_i)^{-1} + K_i$, $1 \le r \le 10,12,14$, де всі операції виконуються в полі GF(28).

Таким чином, існує можливість подання всіх операцій AES в одному полі GF(28), та єдиною проблемою є відображення даних BES в дані AES.

В даний час немає жодної відкритої публікації, в якій описано аналітичну атаку, яка якимось чином використовує дану властивість AES.

У роботі [34] пропонується використовувати для опису підстановки (S-блоку) перевизначену систему квадратичну (що містять в одному термі не більше двох змінних) систему рівнянь. Далі автори пропонують побудувати схожу систему рівнянь для алгоритму шифрування. Далі формулюється твердження, що в полі GF(2) складність вирішення такої системи є субекспоненційною та буде залежить від кількості циклів шифрування. Крім того, робиться висновок, що зі збільшенням кількості циклів стійкість AES збільшується не експонентно.

Авторами не було представлено практичної реалізації такої атаки навіть для спрощеного варіанта AES. Деякі фахівці, в т. ч. автори алгоритмів MARS та DES, критично відгукуються про коректності опису та можливості реалізації даного методу [35, 36].

[37] виводиться формула перетворення у вигляді ланцюгових дробів, з властивостями, дуже близькими до властивостям AES. Перетворення, що описує повний варіант шифру, містить 225 термів (у вигляді ланцюгової дроби). Побудувавши систему з 222 таких рівнянь на основі відповідної кількості пар

відкритих та зашифрованих текстів, криптоаналітик має достатньо інформації для вирішення такої системи погляду теоретико-інформаційного підходу). На практиці нині невідомі (за відкритим публікаціям) методи розв'язання систем рівнянь такого типу.

У [38] запроваджується поняття дуального шифру. За допомогою трьох бієктивних перетворень (для відкритого тексту, шифртексту та ключа) виконується перетворення вхідних та вихідних даних алгоритму. Показано [38], що для AES існує безліч еквівалентних уявлень, що виконують те саме шифрує перетворення. У [39] наведено 240 різних еквівалентних уявлень алгоритму Rijndael. Дуальний шифр еквівалентний AES у всіх аспектах. Відповідно, криптоаналітик може виконувати криптоаналіз дуального шифру, так само як і розробник реалізовувати дуальний шифр замість оригінального варіанта AES.

У відкритих публікаціях немає інформації про загрози безпеці у зв'язку з існуванням великого кількість дуальних шифрів AES.

Крім перерахованих атак, була опублікована робота про можливість виконання криптоаналізу AES з використанням кодів, що виправляють помилки, виключно на основі зашифрованих повідомлень при фіксації (обнулення) кількох бітів вхідного блоку [40]. Автор методу заявляє про успішне проведенні експерименту з отримання 2-х біт секретного ключа по 231 зашифрованим блокам з складністю приблизно 231 (імовірність успіху дорівнює 0,68). Спростування цього методу було опубліковано у наступних роботах [41, 42].

1.5 Оцінка продуктивності реалізацій FIPS-197

Як уже було зазначено, одним із критеріїв при проектуванні Rijndael була простота та мінімальна складність реалізації всіх компонентів шифру. Авторам алгоритму вдалося знайти ефективне рішення, та в порівнянні з іншими кандидатами AES і навіть новішими алгоритмами, представленими у конкурсі

NESSIE, Rijndael є одним із найбільш продуктивних шифрів, що значною мірою зумовило перемогу цього алгоритму у другому турі міжнародного конкурсу AES.

В даний час існує досить велика кількість реалізацій шифру, зроблених розробниками різних країн. Кращі з них використовувалися як еталонна модель при тестування продуктивності 128-бітових симетричних блокових алгоритмів у конкурсі NESSIE [43].

Кількість елементарних операцій (арифметичних інструкцій чи звернень до таблиць) передрахувань) та обсяг пам'яті для зберігання констант, необхідних для перетворення одного 128-бітного блоку AES на 32-бітному процесорі при зашифруванні [43] наведено в таблиці 1.2.

Таблиця 1.2 – Кількість елементарних операцій та обсяг пам'яті для зберігання констант, необхідних для перетворення одного 128-бітного блоку AES

Звернення до таблиць передрахувань	Розмір таблиць перерахувань,бітів	Кількість здвигів	Кількість додавань по модулю 2(XOR)	Загальна кількість логічних операцій
160	8x32	30	120	656

Вимірювання продуктивності програмних реалізацій проводилося на кількох апаратних платформах, під управлінням операційних систем Windows та Linux, а також Sparc під керуванням Sun Solaris.

При тестуванні створювалися реалізації з використанням кількох компіляторів при різних комбінаціях параметрів оптимізації. Як підсумкову реалізацію вибиралася найбільш швидкодіюча для даної платформи. Зазначається [43], що використання оптимізації під молодші моделі процесорів (наприклад, під i386 під час роботи на i3 3220) у деяких випадках давало можливість отримати швидшу реалізацію. Перелік платформ та компіляторів наведено в таблиці 1.3.

Таблиця 1.3 – Перелік платформ та компіляторів, використаних для реалізації Rijndael

Назва платформи	Windows 10/i3 3220	Linux/i3 3220	Sparc V9
Використані компілятори	Visual C++ (6.0) Intel C++ (6.0) gcc (3.1.1) Borland	gcc (2.95.2, 3.1.1) egcs (2.91.66) Intel C++ (6.0) Borland	SWC 5.1 gcc (3.0.4) cc

Під час тестування використовувалося кілька методик вимірювання продуктивності, серед яких, на наш погляд, найбільший інтерес представляє методика вимірювання кількості циклів процесора, що витрачаються на один байт блоку відкритого тексту під час шифрування. Перевагою цієї методикою є незалежність від моделі та тактової частоти процесора. У той же час ця інформація достатня для розрахунку продуктивності програмної реалізації (Мбайт/с) на конкретній моделі процесора цього сімейства.

Підсумкові характеристики найбільш швидкодіючих реалізацій під перелічені платформи наведено у таблиці 1.4. Звідси випливає, що кількість тактів процесора, необхідна для шифрування, становить приблизно одну й ту саму величину для всіх платформ. Цей факт можна пояснити тим, що в алгоритмі Rijndael використовуються максимально прості арифметичні операції, які ефективно кодуються компілятором. Звідси можна дійти невтішного висновку, що продуктивність реалізації криптографічних перетворень визначатиметься тактовою частотою процесора.

Якщо використовувати асемблер, можна отримати додаткове збільшення продуктивність. У [43] повідомляється про досягнення продуктивності, що дорівнює 53,3 Мбайт/с.

Таким чином, алгоритм шифрування AES при програмній реалізації є одним з найбільш швидкодіючих, він перевершує за цим показником інші алгоритми, які були протестовані під час конкурсу NESSIE (DES, TripleDES,

Skipjack, IDEA, Camellia, Safer++, Shacal-2 тощо). Результати приведені в таблиці 1.4.

Таблиця 1.4 - Характеристики найбільш швидкодіючих реалізацій Rijndael

Платформа	Компілятор	Довжина ключа	Кількість тактів процесора на 1 байт		Кількість тактів процесора на обчислення підключів
			Шифрування	Дешифрування	
Windows	Visual C++ (6.0)	128	23	23	497
		192	27	27	552
		256	32	32	780
Linux	gcc (3.1.1)	128	26	26	504
		192	31	31	601
		256	35	36	949
Sun 450 МГц	Cc	128	21	22	453
		192	25	28	463
		256	29	32	753

Крім того, досягнутий рівень швидкодії програмної реалізації значно вищий, ніж реальна пропускна спроможність більшості шин вводу/виводу сучасних комп'ютерів, зокрема, мережевих інтерфейсів та каналів підключення жорстких дисків IDE. Це дозволяє реалізувати високопродуктивні програми та системні служби, що виконують шифрування та розшифрування прикладних даних, мережевого трафіку та потоків обміну з жорсткими дисками з мінімальною затримкою та практично у реальному масштабі часу.

У додатках, що вимагають надзвичайно високого рівня захищеності та надійності, необхідно використання апаратних реалізацій криптографічних алгоритмів У конкурсі NESSIE виконувалось дослідження продуктивності та складності реалізації криптографічних примітивів у вигляді однієї IMC під час використання різних варіантів виготовлення модуля. Основні дані щодо алгоритму шифрування AES наведено у таблиці 1.5.

Таблиця 1.5 – Продуктивність апаратних реалізацій AES як однієї ІМС

Тип апаратної реалізації	Швидкодія	Особливості
FPGA XCV 1000 14 МГц	300 Мбайт/с	режим із зворотним зв'язком
FPGA XCV 1000 32 МГц	1940 Мбайт/с	режим без зворотного зв'язку
ASIC 0,5 мкм	524 Мбайт/с	-
ASIC 0,5 мкм	5100 Мбайт/с	конвеєризація
ASIC 0,35 мкм	1950 Мбайт/с	613000 вентилів

Як можна побачити з таблиці 1.5, високою продуктивністю відрізняється конвеєризована апаратна платформа ASIC. Необхідною умовою ефективної роботи у цьому випадку є обробка потоків даних з використанням одного ключа шифрування, що обумовлює застосування таких рішень для захисту даних, що передаються на мережевому і канальному рівні 7-рівневої моделі OSI. Продуктивність рішень, найбільш підходячих для забезпечення безпеки даних користувача (прикладний і представницький рівень у разі мережевої взаємодії) можна порівняти з програмною реалізацією на сучасні процесори.

Досягнутий на даний момент рівень продуктивності програмних реалізацій стандарту шифрування FIPS-197, виконаних у АТ «ІІТ», наведено у таблиці 1.6.

Таблиця 1.6 – Продуктивність програмних реалізацій FIPS-197, виконаних у АТ «ІІТ»

			Шифрування	Дешифрування	
Windows Linux	Borland gcc (3.1.1)	128	18	17	239/348
		192	192	21	249/432
		256	256	25	351/644

Порівнюючи табл. 4 та 6, можна відзначити, що продуктивність програмних реалізацій, виконаних в АТ «ІІТ» є вищою, ніж тестові реалізації, використані в конкурсі NESSIE. Це можна пояснити тим фактом, що українські фахівці проводили оптимізацію не лише за рахунок використання ефективних

методик програмування, але й за рахунок застосування більш простих криптографічно еквівалентних уявлень перетворень циклової функції AES.

Після прийняття Rijndael як федеральний стандарт США FIPS-197, в АТ «ІІТ» було виконано кілька варіантів апаратної реалізації алгоритму Швидкість перетворень при побудові апаратної системи криптографічного захисту інформації на базі ПЛІС складає величину близько 110 Мбайт/с.

Наведені результати попереднього аналізу властивостей та дослідження стійкості алгоритму симетричного блочного шифрування AES дозволяють зробити висновок, що при створенні шифру Rijndael використовувалася надійна математична база, обґрунтовані критерії оцінки та ефективні принципи проектування. Результати досліджень, отримані незалежними фахівцями, дозволяють зробити висновок, що FIPS-197 забезпечує реальну криптографічну стійкість при симетричному блоковому шифрування. Для ухвалення рішення про використання алгоритму в режимі потокового шифру (наприклад, BMGL) та інших удосконалених режимах, на наш погляд, необхідні додаткові дослідження.

Таким чином, на сучасному рівні розвитку засобів та систем криптоаналізу шифр AES (FIPS-197) забезпечує реальну стійкість, є статистично безпечним і, обмежено застосовуватись в Україні. Алгоритм був підданий глибокій експертизі із залученням незалежних фахівців, що володіє прийнятною складністю криптографічних перетворень, може бути реалізовано апаратно та програмно, для процесорів різних класів.

Надалі, завдяки ідеям та принципам, закладеним в алгоритмі Rijndael, рівень безпеки шифру може бути додатково збільшений. Крім того, при використанні подібних принципів та новітніх результатів у галузі криптології, в АТ «ІІТ» розроблено алгоритм «Торнадо», забезпечує ще більший рівень безпеки.

2 ДОСЛІДЖЕННЯ КРИПТОПЕРЕТВОРЕНЬ AES

2.1 Специфікація алгоритму

AES – абревіатура від Advanced Encryption Standard (переклад з англ. удосконалений стандарт шифрування). Після відкриття у 1997 р. Національним Інститутом Стандартів та Технології США (NIST) програми з розробки AES відбулися 3 етапу міжнародного конкурсу Новий стандарт повинен був мати:

- стійкість не меншу, ніж у 3DES;
- швидкість шифрування більша за швидкість 3DES;
- прозору структуру;
- ефективну реалізацію на платформі Pentium Pro;
- Ефективну апаратну реалізацію.

Переможцем конкурсу стали бельгійці Йоан Дамен та Вінсент Реймен з алгоритмом RIJNDAEL (читається Рейн-дал від перших літер авторів). Стандарт почав діяти з 2002р. AES – симетричний ітеративний блоковий алгоритм.

AES – це шифр Фейстеля, що базується на принципах нової мережі підстановок-перестановок. Має нову архітектуру SQUARE (КВАДРАТ), для якої характерно: 1) уявлення, що шифрується в блоці у вигляді двовимірного байтового масиву; 2) шифрування за один раунд всього блоку даних (байт-орієнтована структура); 3) виконання криптографічних перетворень, як над окремими байтами масиву, і над його рядками і стовпцями. Це забезпечує дифузію даних одночасно у двох напрямках - по рядках та по стовпцях. Архітектура SQUARE властива, крім шифру AES(RIJNDAEL), шифрам SQUARE (його назва і дала ім'я всієї архітектури), CRYPTON (один із кандидатів на AES). Друге місце у конкурсі AES посів інший SP-шифр, SERPENT. Очевидно, SP-мережі та, зокрема, архітектура SQUARE, у найближчому майбутньому стануть безроздільно домінувати.

Загальні характеристики AES:

- AES зашифровує та розшифровує 128-бітові блоки даних.

- AES дозволяє використовувати три різні ключі довжиною 128, 192 або 256 біт (залежно від довжини ключа версії шифру позначають AES-128, AES-192 або AES-256).

- Від розміру ключа залежить кількість раундів шифрування:

  - довжина 128 біт – 10 раундів;

  - довжина 192 біта – 12 раундів;

  - довжина 256 біт – 14 раундів.

- Усі раунди, крім останнього, ідентичні.

2.2 Представлення даних у криптоалгоритмі AES

Нагадування: 1 байт=8 бітів, 128 бітів=16x8 бітів=16 байт. Основним елементом, яким оперує алгоритм AES, є байт – послідовність 8 біт, оброблюваних як єдине ціле.

Для формування байтів 128 біт блоку відкритого тексту, вихідного блоку шифротексту та ключа шифру діляться на групи з 8- ми поруч біт, що стоять так, щоб в цілому вийшов масив байт. Нижче на рисунку 2.1 представлена прийнята нумерація біт в межах кожного байта.

№ біта на вході	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	…
№ байта	0								1								2								…
№ біта в байті	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	…

Рисунок 2.1 – Формування байтів 128 бітного блоку

Задавати значення байта зручно у шістнадцятковій системі обчислення. Для цього байт ділиться на дві групи із 4-х біт: група старших біт у байті є першим шістнадцятковим символом, а група молодших біт – другим. Наприклад, для байта 10101100 отримаємо:

$$10101100 = 10101100 = AC$$

Вхідними даними для операцій шифрування є масив із 16 байт $\mathrm{in}_0, \mathrm{in}_1, ..., \mathrm{in}_{15}$.

Позначимо:

$\mathrm{in}_0, \mathrm{in}_1, ..., \mathrm{in}_{15}$ - 16 байт блоку відкритого тексту;

$\mathrm{k}_0, \mathrm{k}_1, ..., \mathrm{k}_{15}$ - 16 байт ключа шифру;

- 16 байт блоку шифротексту.

Перед початком шифрування байти цього масиву розміщуються послідовно в стовпці матриці InputBlock (згори вниз). Усередині алгоритму операції виконуються над матрицею байт, званою матрицею станів State або просто станом. Кінцеве значення матриці стану OutputBlock є виходом алгоритму і перетворюється на послідовність байтів шифротексту $\mathrm{out}_0, \mathrm{out}_1, ..., \mathrm{out}_{15}$. Аналогічно в стовпці матриці InputKey потрапляють і 16 байтів $\mathrm{k}_0, \mathrm{k}_1, ..., \mathrm{k}_{15}$, ключа шифру. Розмірність усіх матриць – 4×4. Схематично таке подання даних виглядає так, як показано на рисунку 2.2.

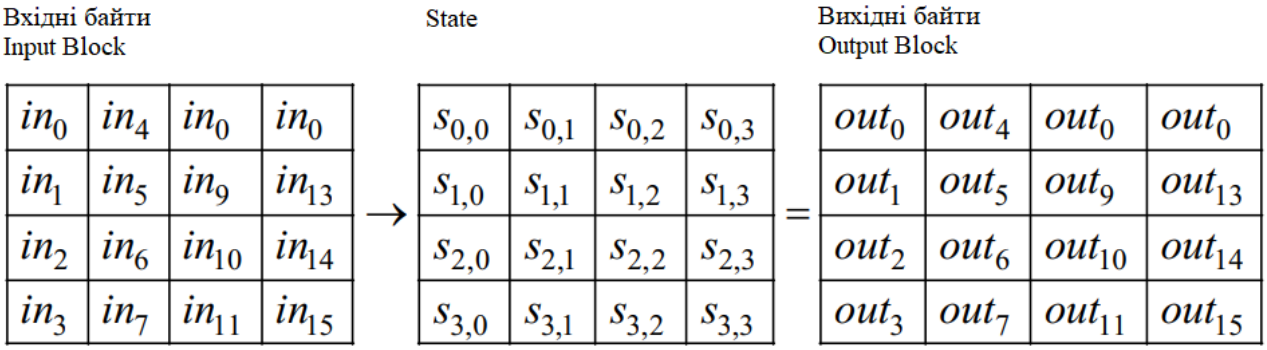

Рисунок 2.2 – Схема шифрування алгоритмом AES

Схмеатично ключ можна представити у вигляді матриці, як це показано на рисунку 2.3.

Байти ключа
Input Key

k_0	k_4	k_0	k_0
k_1	k_5	k_9	k_{13}
k_2	k_6	k_{10}	k_{14}
k_3	k_7	k_{11}	k_{15}

Рисунок 2.3 – Байти ключа AES

Чотири байти в кожному стовпці матриці станів або ключа можна як одне 32-х бітове слово. Тому матриця станів – це масив із 4 слів w_0, w_1, w_2, w_3, де:

$$w_0 = s_{0,0}\ s_{1,0}\ s_{2,0}\ s_{3,0};$$

$$w_0 = s_{0,1}\ s_{1,1}\ s_{2,1}\ s_{3,1};$$

$$w_0 = s_{0,2}\ s_{1,2}\ s_{2,2}\ s_{3,2};$$

$$w_0 = s_{0,3}\ s_{1,3}\ s_{2,3}\ s_{3,0};$$

Матриця, що надходить на вхід кожного раунду називається матрицею InputState, а на виході раунду утворюється матриця OutputState. Очевидно, що на вході першого раунду InputState=InputBlock , а на виході останнього раунду OutputState = OutputBlock.

Наприклад, представимо у вигляді матриці InputBlock текст СКЛАДНІСТЬЗАДАЧІ = $(21\ 14\ 15\ 00\ 05\ 17\ 11\ 21\ 22\ 30\ 09\ 00\ 05\ 00\ 27\ 11)_{10} =$ $(150E\ 0F\ 0005110B15161E\ 09\ 0005001B\ 0B)_{16}$.

$$InputBlock = \begin{pmatrix} 15\ 05\ 16\ 05 \\ 0E\ 11\ 1E\ 00 \\ 0F\ 0B\ 09\ 1B \\ 00\ 15\ 00\ 0B \end{pmatrix}$$

Як зазначалося, у версії алгоритму AES-128, що розглядається ключ шифру складається з 128 бітів, поділених на 16 байтів $k_0, k_1, ..., k_{15},$, та записується в стовпці матриці InputKey. Кожен стовпець матриці InputKey утворює слово, тобто. фактично ключ шифру – це чотири слова $w_0, w_1, w_2, w_3,$ де $w_0 = k_0 k_1 k_2 k_3, w_1 = k_4 k_5 k_6 k_7$ (рисунок 2.4).

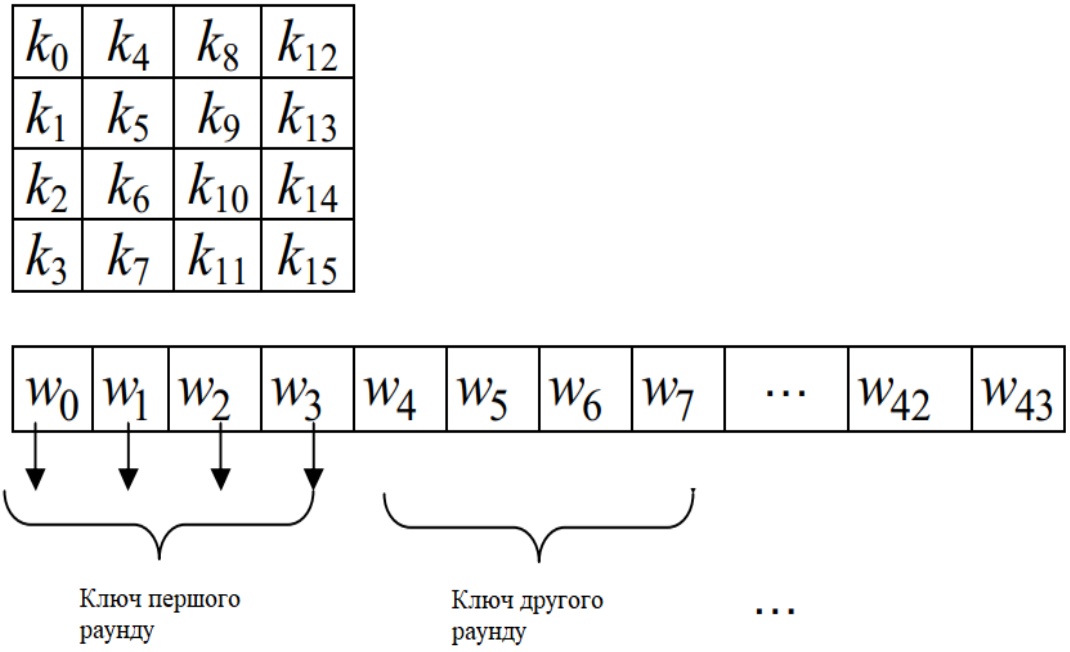

Рисунок 2.4 –Формування слів в алгоритмі AES

З цих слів за допомогою спеціального алгоритму (про нього пізніше) утворюється послідовність з 44 слів: $w_0, w_1, w_2, ..., w_{43},$ (кожне слово по 32 біти). На кожен раунд шифрування подаються чотири слова цієї послідовності. Вони і будуть грати роль раундового ключа. Схема перетворення даних показано малюнку.

Перед першим раундом виконується операція AddRoundKey (Додавання по модулю 2 з початковим ключем шифру). Перетворення, виконані в одному раунді, позначають Round (State, RoundKey), де змінна State - матриця, що описує дані на вході раунду та на його виході після шифрування; змінна RoundKey - матриця, містить раундовий ключ (рисунок 2.5).

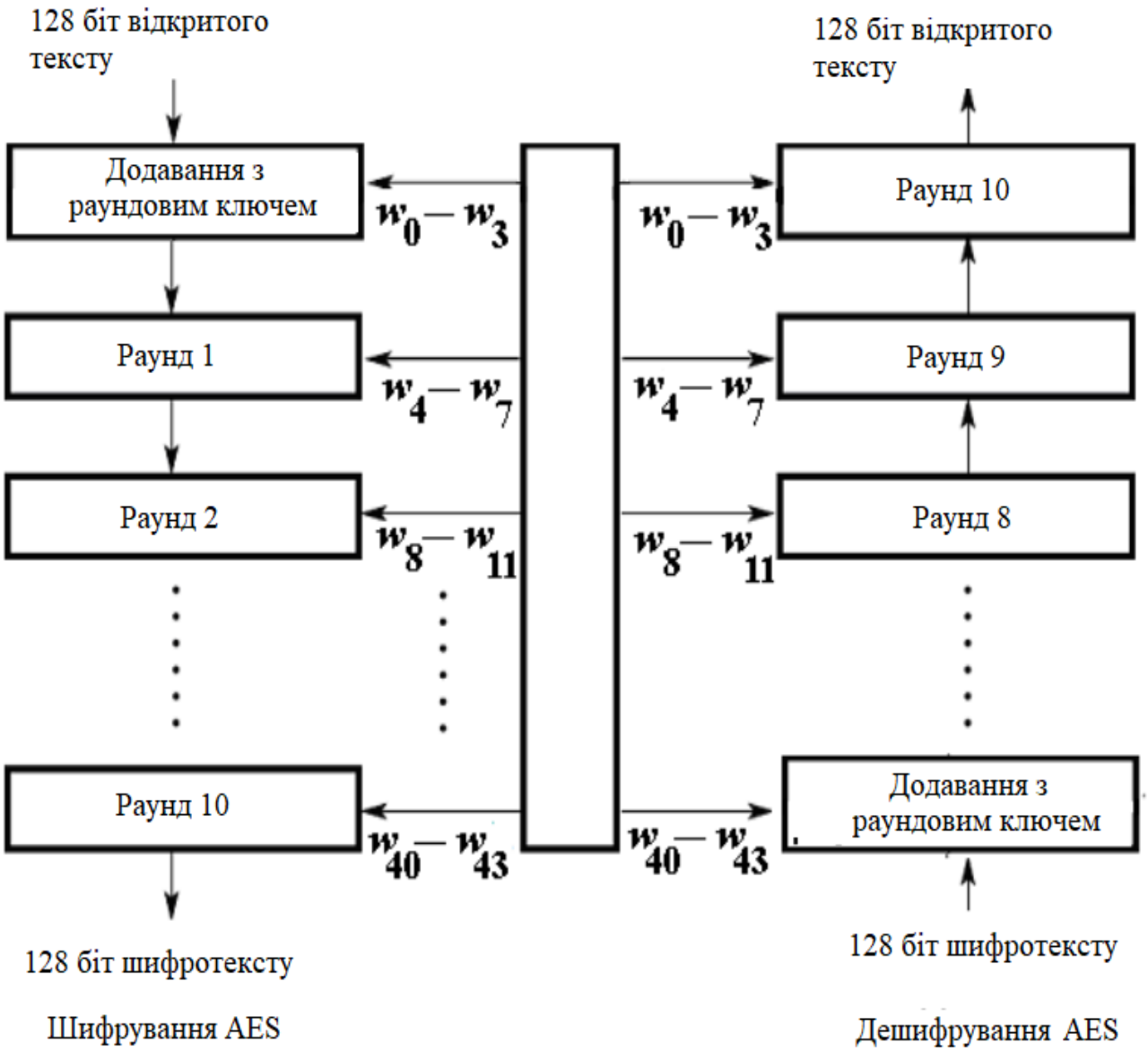

Рисунок 2.5 – Схема проведення раундових перетворень

Раунд складається з 4 різних перетворень:

- SubBytes – побайтова підстановка в S-боксі з фіксованою таблицею замін;

- ShiftRows – побайтове зсув рядків матриці State на різну кількість байт;

- MixColumns – перемішування байт у стовпцях;

- AddRoundKey – додавання з раундовим ключем (операція XOR).

Останній раунд дещо відрізняється від попередніх тим, що не функцію MixColumns. Схематично перетворення можна відобразити на рисунку 2.6.

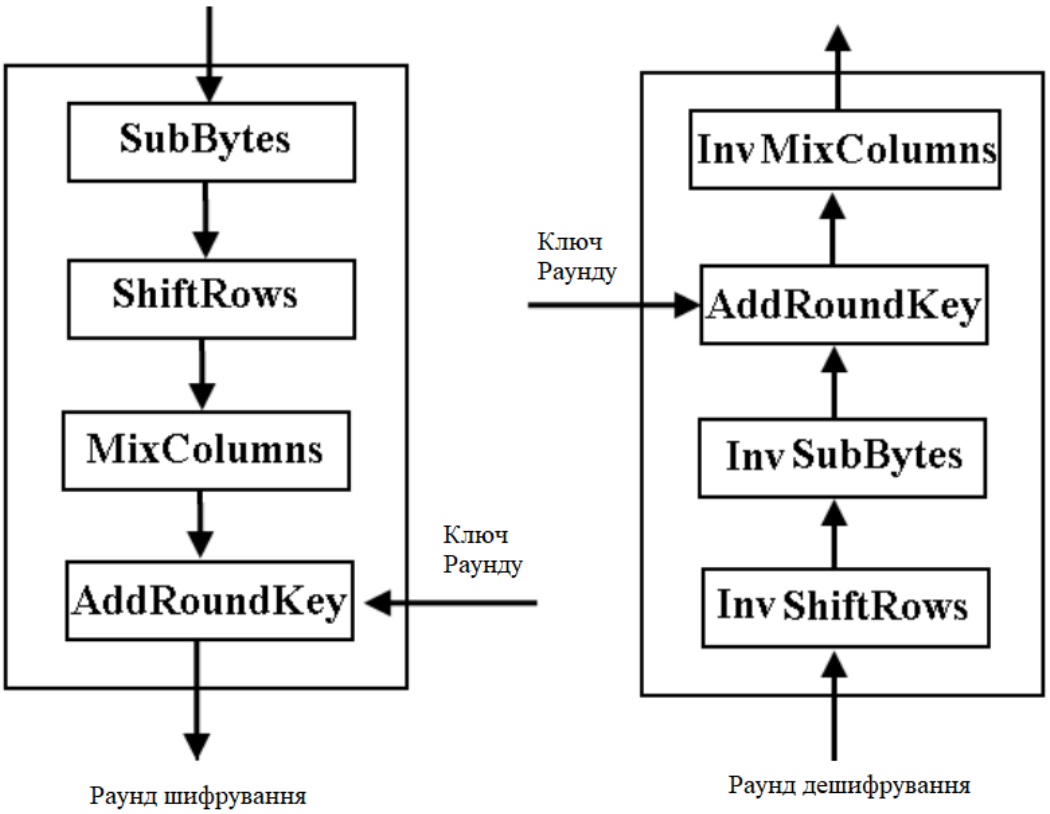

Рисунок 2.6 – Схема перетворень AES

При дешифруванні в кожному раунді виконуються зворотні операції: InvShiftRows, InvSubBytes, AddRoundKey та InvMixColumns (у позначках перед назвою функції з'являється приставка Inv). Порядок виконання операцій при шифруванні та дешифруванні різний, причини чого будуть зрозумілі після детального розгляду кожного перетворення.

2.3 Математичні основи шифру AES

Операції у полі GF(2^8) . Для опису алгоритму використовується кінцеве поле Галуа GF(2^8) побудоване як розширення поля GF(2) = {0,1} за модулем непривідного многочлена $m(x) = x^8 + x^4 + x^3 + x + 1$. Елементами поля GF($2^8$) є багаточлени виду:

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0,$$

ступінь яких менший за 8, а коефіцієнти $b7, b6, \ldots, b0 \in \{0,1\}$. Операції в полі виконуються за модулем m (x). Всього в полі GF(2^8) налічується 2^8= 256 багаточленів.

Подання двійкового числа $b_7b_6b_5b_4b_3b_2b_1b_0$ у вигляді багаточлена з коефіцієнтами $b7, b6, \ldots, b0 \in \{0,1\}$ дозволяє інтерпретувати байт як бітовий багаточлен у кінцевому полі GF(2^8) :

$$b(x) = b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0. \qquad (2.1)$$

Наприклад, байт 63 задає послідовність бітів 01100011 та визначає конкретний елемент поля:

$$01100011 \leftrightarrow x^6 + x^5 + x + 1.$$

Розглянемо основні математичні операції у полі GF(2^8) .

1. Складання байт можна виконати будь-яким із трьох способів:

- подати байти бітовими багаточленами та скласти їх за звичайним правилом підсумовування багаточленів з наступним приведенням коефіцієнтів суми по модулю 2 (операція XOR над коефіцієнтами);

- підсумовувати за модулем 2 відповідні біти в байтах;

- скласти байти у шістнадцятковій системі обчислення.

Наприклад, такі три записи еквівалентні:

- подання у вигляді багаточленів: $(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) = x^7 + x^6 + x^4 + x^2;$

- бітова вистава $\{01010111\} \oplus \{10000011\} = \{11010100\};$

- шістнадцяткове подання: $\{57\} \oplus \{83\} = \{D4\}$.

2. Множення байт виконується за допомогою представлення їх багаточленами та перемноження за звичайними алгебраїчними правилами.

Отриманий твір необхідно навести за модулем багаточлена $m(x) = x^8 + x^4 + x^3 + x + 1$ (Результат приведення дорівнює залишку від поділу твору на m x (x)).

Перемноження багаточленів у полі можна спростити, ввівши операцію множення бітового многочлена (1) на *x*:

$$x(b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0) =$$
$$= b_7x^8 + b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0x.$$

3. Для будь-якого ненульового бітового багаточлена b x ( ) у полі GF(2 ^{8}) 8 існує багаточлен b^{-1} (*x*), зворотний до нього за множення, тобто $b(x)b^{-1}(x) \equiv 1 \bmod m(x)$. Для знаходження зворотного елемента використовують розширений алгоритм Евкліда, за допомогою якого знаходять такі многочлени *a* (*x*) і c (*x*), що $a(x)b(x) + c(x)m(x) = 1$. Отже, $b^{-1}(x) \equiv a(x) \bmod m(x)$.

Багаточлени з коефіцієнтами, що належать полю GF(2^8) . Багаточлени третього ступеня з коефіцієнтами кінцевого поля а $a_i \in GF(2^8)$ 4 мають вигляд:

$$a(x) = a_2x^3 + a_2x^2 + a_1x + a_0 \qquad (2.2)$$

Таким чином, у цих багаточленах у ролі коефіцієнтів при невідомих задіяні байти замість біт. Далі багаточлени (2) представлятимемо у формі слова $[a_0, a_1, a_2, a_3]$. У стандарті AES при множенні багаточленів виду (2.2) використовується приведення за модулем іншого багаточлена $x^4 + 1$.

Для вивчення арифметики аналізованих багаточленів введемо додатково багаточлен $b(x) = b_3x^3 + b_2x^2 + b_1x + b_0$, де $b_i \in GF(2^8)$.

Тоді:

1. Додавання: $a(x) + b(x) = (a_3 \oplus b_3)x^3 + (a_2 \oplus b_2)x^2 + (a_1 \oplus b_1)x + (a_0 \oplus b_0)$.

2. Множення. Це складніша операція. Нехай, ми перемножуємо два багаточлена: $a(x) = a_3x^3 + a_2x^2 + a_1x + a_0$ і $b(x) = b_3x^3 + b_2x^2 + b_1x + b_0$.

Результатом множення буде багаточлен: $c(x) = a(x) \cdot b(x) = c_6x^6 + c_5x^5 + c_4x^4 + c_3x^3 + c_2x^2 + c_1x + c_0$,

Де: $c_0 = a_0b_0; c_1 = a_1b_0 \oplus a_0b_1; c_2 = a_2b_0 \oplus a_1b_1 \oplus a_0b_2;\ c_3 = a_3b_0 \oplus a_2b_1 \oplus a_1b_2 \oplus a_1b_1 \oplus a_0b_3;$ $c_4 = a_3b_1 \oplus a_2b_2 \oplus a_1b_3;\ c_5 = a_3b_2 \oplus a_2b_3;\ c_6 = a_3b_3$.

Щоб результат можна було уявити чотирибайтовим словом, необхідно взяти результат за модулем багаточлена ступеня не більше 4. Автори шифру вибрали для цієї мети багаточлен $x^4 + 1$, для якого справедливо $x^i \bmod (x^4 + 1) \equiv x^{i \bmod 4}$.

Тому в добуткові коефіцієнти при ступенях x^i , i= 0,1,2,3, дорівнюють сумі творів a_jb_k за індексами, для яких $j + k = i \ (\bmod 4)$, j,k= 0,1,2,3. Таким чином, після приведення по модулю $x^4 + 1$ отримаємо:

$$d(x) = a(x) \cdot b(x) = d_3x^3 + d_2x^2 + d_1x + d_0x,$$

де $d_0 = a_0b_0 \oplus a_3b_1(x) \oplus \cdot a_2b_2(x) \oplus a_1b_3;$

$d_1 = a_1b_0 \oplus a_0b_1(x) \oplus \cdot a_3b_2(x) \oplus a_2b_3;$

$d_2 = a_2b_0 \oplus a_1b_1(x) \oplus \cdot a_0b_2(x) \oplus a_3b_3;$

$d_3 = a_3b_0 \oplus a_2b_1(x) \oplus \cdot a_1b_2(x) \oplus a_0b_3;$

Або в матричній формі:

$$\begin{pmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{pmatrix} = \begin{pmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{pmatrix} \cdot \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

2.4 Раундові перетворення AES

Розглянемо докладніше перетворення шифрування раунду.

1. Операція SubBytes. Операція виконує нелінійну заміну байтів, що виконується незалежно з кожним байтом матриці State. Заміна оборотна та побудована шляхом комбінації двох перетворень над вхідним байтом:

- знаходження зворотного (інвертованого) елемента щодо множення в полі GF(2^8) (вважається, що нульовий байт {00} переходить сам у себе);

- виконання деякого афінного перетворення: множення інвертованого байта на багаточлен $a(x) = x^4 + x^3 + x^2 + x + 1$ і сумування з багаточленом $b(x) = x^6 + x^5 + x + 1$ в полі $F_2[x]/x^8 + 1$. Відмітимо, що $a^{-1}(x) = x^6 + x^3 + x$ і $a^{-1}(x)b(x) = x^2 + 1$.

У матричній формі процедура SubBytes записується як:

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}^{-1} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix},$$

де через x позначені вхідні біти, а через y вихідні. Якщо на вхід функції потрапляє нульовий байт, то результатом заміни буде число $y = b$. Зміна байтів показана на рисунку 2.7.

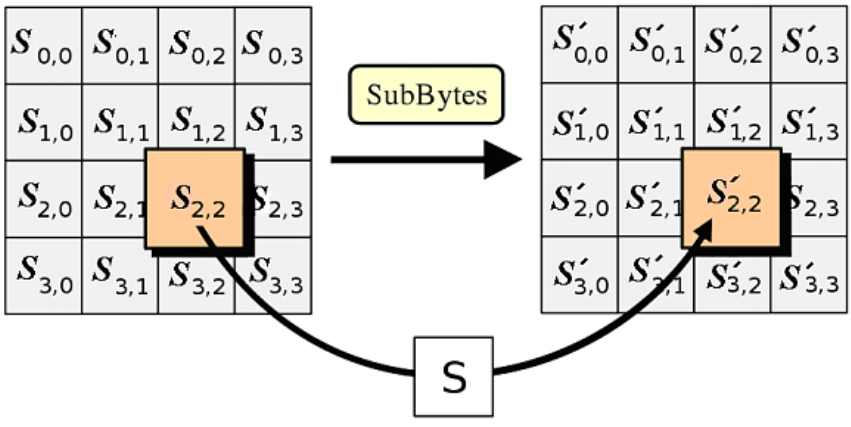

Рисунок 2.7 – Процес зміни байтів за допомогою таблиці

Процес заміни байтів за допомогою таблиці підстановки ілюструє рисунок 2.7. Нелінійність перетворення обумовлена нелінійністю інверсії x^{-1}, а оборотність – оборотністю матриці.

Створену на основі цієї операції спеціальну таблицю замін байтів у шістнадцятковій системі називають *S*-боксом(рисунок 2.8).

	Таблиця перетворення SubBytes															
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	FO	AD	D4	A2	AF	9C	A4	72	CO
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	AO	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	DO	EF	AA	FB	43	4D	33	85	45	F9	02	F7	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DE
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	CB	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	IF	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	OE	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	OD	BF	E6	42	68	41	99	2D	OF	BO	54	BB	16

Рисунок 2.8 – Таблиця перетворень SubBytes

Наприклад, якщо $S_{1,1} = \{8A\}$, то результат заміни цього байта слід шукати на перетині рядка з індексом 8 та стовпця з індексом A, тобто. SubBytes(8*A*)={7*E*}.

2. Операція ShiftRows.

Операція застосовується до рядків матриці State – її перша стрічка нерухома, а елементи нижніх трьох рядків циклічно зсуваються вправо на 1, 2 і 3 байти відповідно.

$$\begin{pmatrix} S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\ S_{1,0} & S_{1,1} & S_{1,2} & S_{1,3} \\ S_{2,0} & S_{2,1} & S_{2,2} & S_{2,3} \\ S_{3,0} & S_{3,1} & S_{3,2} & S_{3,3} \end{pmatrix} \rightarrow \begin{pmatrix} S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\ S_{1,1} & S_{1,2} & S_{1,3} & S_{1,0} \\ S_{2,2} & S_{2,3} & S_{2,0} & S_{2,1} \\ S_{3,3} & S_{3,0} & S_{3,1} & S_{3,2} \end{pmatrix}$$

По суті, це перестановка елементів матриці, в якій беруть участь лише елементи рядків, тому перетворення оборотне.

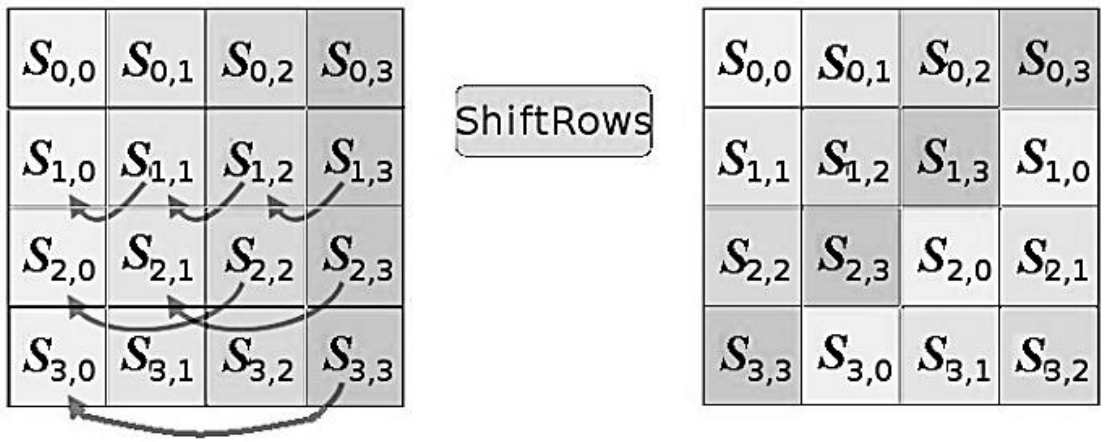

Рисунок 2.9 – Схема операції ShiftRows

3. Операція MixColumns.

За допомогою цієї операції виконується перемішування байтів у стовпці матриці State. Кожен стовпець цієї матриці приймається за многочлен над полем GF(2^8) і множиться на фіксований багаточлен:

$$c(x) = c_3x^3 + c_2x^2 + cx + c_0 = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$$

про модуль багаточлена $x^4 + 1$ (нагадаємо, всі коефіцієнти багаточленів над полем GF(2^8) – байта). Як показано вище, таку операцію можна записати в матричному вигляді як:

$$\begin{pmatrix} c_0 & c_3 & c_2 & c_1 \\ c_1 & c_0 & c_3 & c_2 \\ c_2 & c_1 & c_0 & c_3 \\ c_3 & c_2 & c_1 & c_0 \end{pmatrix} \cdot \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \cdot \begin{pmatrix} S_0 \\ S_1 \\ S_2 \\ S_3 \end{pmatrix} = \begin{pmatrix} s_0' \\ s_1' \\ s_2' \\ s_3' \end{pmatrix}$$

Операцію MixColumns можна представити схематично, як показано на рисунку 2.10.

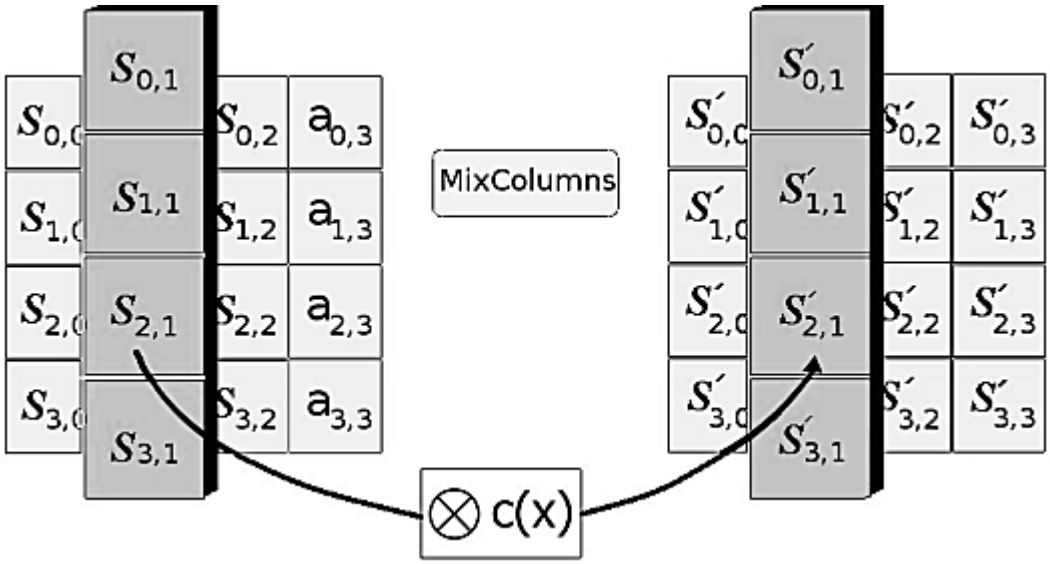

Рисунок 2.10 – Схема операції MixColumns

Багаточлен $c(x)$ – взаємно простий з багаточленом x^4+1 над полем GF(2), тому в полі існує зворотний багаточлен $c^{-1}(x)(\mathrm{mod}\, x^4+1) \Rightarrow$ матриця у цій формулі оборотна.

4. Операція AddRoundKey.

Функція AddRoundKey, State RoundKey побітово складає елементи змінної RoundKey та елементи змінної State по принципу: i -й стовпець даних (i 0,1,2,3) складається з певним 4-байтовим фрагментом розширеного ключа W [4r+1], де r – номер потокового раунду алгоритму. При шифруванні перша додавання ключа раунду відбувається до першого виконання операції SubBytes (рисунок 2.11).

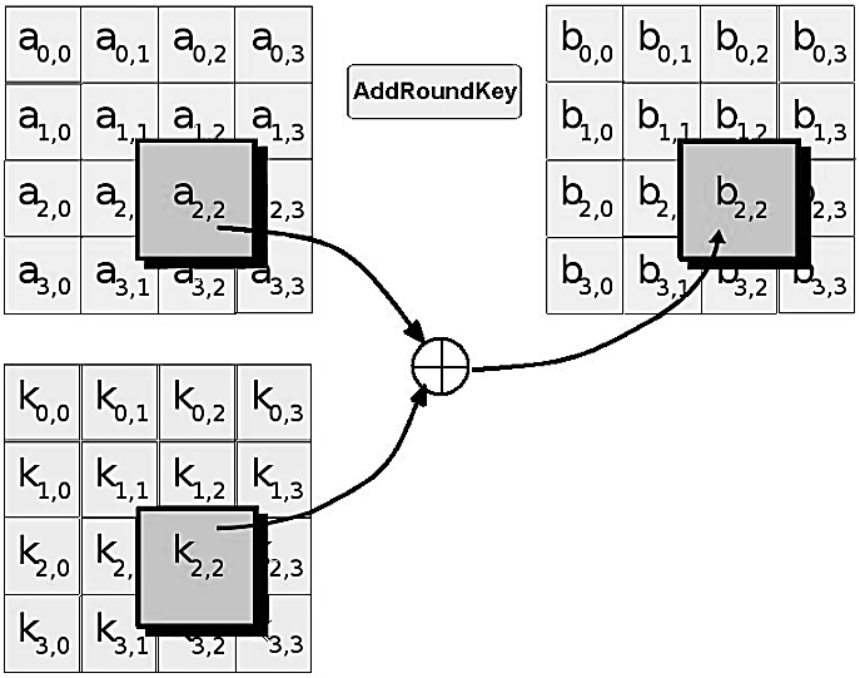

Рисунок 2.11 – Схема операції AddRoundKey

Наступний малюнок демонструє властивості розсіювання та перемішування інформації в ході шифрування алгоритмом AES. Видно, що два раунди забезпечують повне розсіювання та перемішування інформації (рисунок 2.12).

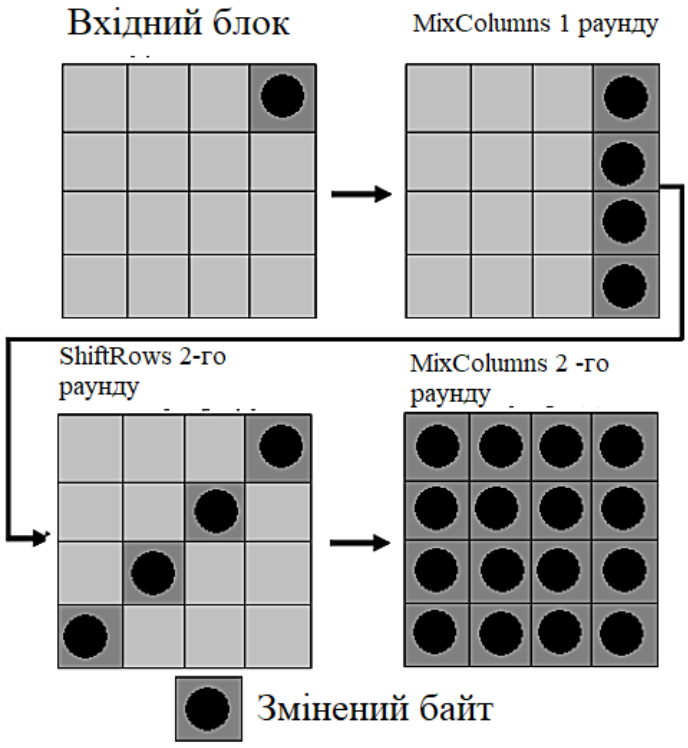

Рисунок 2.12 – Властивості розсіювання

Досягається це за рахунок використання функцій ShiftRows та MixColumns. Операція SubBytes додає шифрування стійкість проти диференціального криптоаналізу, а операція AddRoundKey забезпечує необхідну секретну випадковість.

3 КРИПТОАНАЛІЗ АЛГОРИТМУ AES

3.1 Понятийний апарат лінійного та дифференціального криптоаналізу

Нагадаємо коротко основний понятійний апарат лінійного та диференціального криптоаналізу. Наслідуючи роботу [14], введемо ряд визначень. Диференціальна ймовірність DP^f та лінійна ймовірність LP^f відповідно для ключозалежної функції f з n-бітним входом x і n-бітним виходом y, $x, y \in \mathrm{GF}(2)^n$, відповідно:

$$\mathrm{DP}^{\mathrm{f}}(\Delta x \to \Delta y) = \frac{\#\left\{x \in GF(2)^n \mid f(x) \otimes f(x \otimes \Delta x) = \Delta y\right\}}{2^n} \tag{3.1}$$

$$LP^f(\Gamma\Gamma \to \Gamma x) = \left(\frac{\#\{x \in GF(2)^n | x \cdot \Gamma x = f(x) \cdot \Gamma y\}}{2^{n-1}} - 1\right), \tag{3.2}$$

де Δx та Δy є вхідною та вихідною відмінністю (Різницею), а Γx і Γy вхідний і вихідний масками; $x\Gamma x$ означає результат побітного добутку x та Γx. $(DP^f_{\max}$ *і* $DL^f_{\max})$ - максимальне значення диференціальної та лінійної ймовірності для ключозалежної функції f визначається відповідно як:

$$DP^f_{\max} = \max_{\Delta x \neq 0, \Delta y} DP^f(\Delta x \to \Delta y), \tag{3.3}$$

$$DL^f_{\max} = \max_{\Delta x \neq 0, \Delta y} DL^f(y \to \Delta x), \tag{3.4}$$

У загальному випадку, ключозалежна функція f є сильною, якщо значення $DP^f_{\max}$ х та $DL^f_{\max}$ функції f є досить малими [14]. Нас надалі і цікавитимуть значення $DP^f_{\max}$ і $DL^f_{\max}$ для випадків, коли як функції f виступають циклові перетворення та послідовності циклових перетворень ітеративних шифрів (ключозалежні функції), а також підстановочні перетворення (Незалежні функції).

Нехай π – підставна таблиця з n-бітними входами та n-бітними виходами.

$$\sum_{\Delta y \in Y} DP^{\pi}(\Delta x \to \Delta y) = 1, \tag{3.5}$$

$$\sum_{\Delta x \in X} LP^{\pi}(x \to \Delta y) = 1, \tag{3.6}$$

і, більше того, якщо π – підстановка, то:

$$\sum_{\Delta x \in X} DP^{\pi}(\Delta x \to \Delta y) = 1, \tag{3.7}$$

$$\sum_{y \in Y} LP^{\pi}(x \to y) = 1, \tag{3.8}$$

Ці результати видаються достатньо очевидними, виходячи з визначень (3) та (5), застосованих до підстановок (незалежних перетворень). Вони є відображенням відомих фактів, що полягають у тому, що суми осередків таблиці XOR різниць та суми квадратів осередків таблиць лінійних апроксимацій підстановок по рядках і по стовпцях дорівнюють 2^n і $(2^{n-1})^2$ відповідно, де n – бітовий розмір входу та виходу підстановки порядку 2^n.

Важливим для подальшого поняття випадкової підстановки. Ми на ньому зупинимося окремо.

3.2 Випадкові підстановки

Нагадаємо, що раніше у нашій роботі [15] поняття випадкової підстановки було визначено наступним чином. Під випадковою (квазівипадковою) підстановкою розуміється підстановка, яка задовольняє одночасно трьом критеріям випадковості:

1. Число інверсій η_n у підстановці степеня n приблизно дорівнює числу "антиінверсій", а практично, якщо:

$$\left|\eta - \frac{n(n-1)}{4}\right| \le a\sigma_\eta, \sigma_\eta = \frac{n^{3/2}}{6}.$$

2.Кількість циклів ξ_n у підстановці ступеня n близько до $\ln n$, а практично, знаходиться в межах:

$$\left|\xi_n \ln n\right| \le a\sigma_\xi, \sigma_\xi \sqrt{\ln n}.$$

3. Число зростання θ_n у підстановці степеня n приблизно дорівнює числу спадань, а практично:

$$\left|\Theta_n - \frac{n}{2}\right| \le a\sigma_\Theta, \sigma_\Theta \sqrt{n/12}$$

У цих співвідношеннях а – параметр, що вибирається значною мірою із суб'єктивних міркувань (принаймні з умови, що безліч допустимих підстановок не стане менше деякого практично доцільного числа). У наших пропозиціях використовувалося значення $a = 1$. Залишається помітити, що з повного множини підстановок порядку 2^n у цьому випадку наведені критерії відбору проходять 53% усіх підстановок.

У наступних наших публікаціях [16, 18], присвячених дослідженню диференціальних і лінійних властивостей випадкових підстановок тапідстановочних перетворень, що розвивають результати робіт luka O'Connor-a [17, 19], ми визначили ще дві умови, яким підкоряються випадкові підстановки. Вони ґрунтуються на двох твердженнях. Нагадаємо тут їх, бо вони важливі для подальшого розгляду. В позначеннях роботи [16] нехай: $\Pr(\Lambda_\pi(\Delta X, \Delta Y) = 2k)$ буде ймовірністю того, що значення осередку диференціальної таблиці випадково взятої підстановки π порядку 2^n для переходу вхідної різниці ΔX у відповідну вихідну різницю ΔY дорівнюватиме $2k$. Ця ймовірність визначається теоремою наведеною нижче.

Затвердження 1. Для будь-яких ненульових фіксованих ΔX , $\Delta Y \in Z_{2^n}$ у припущенні, що підстановка π обрана рівноймовірно з множини:

$$\Pr(\Lambda_\pi(\Delta\Delta X,\Delta Y=2k)=\binom{2^{n-1}}{k}^2\cdot\frac{k!\cdot 2^k\cdot\Phi(2^{n-1}-k)}{2^n!}, \tag{3.9}$$

де функція Φ(d) визначається виразом:

$$\Phi(\mathrm{d})=\sum_{i=0}^{d}(-1)^i\cdot\binom{d}{i}^2\cdot 2^i\cdot i!\cdot(2d-2i)!. \tag{3.10}$$

Закон розподілу ймовірностей (3.10) отриманий для повної множини підстановок, однак чудовою його властивістю є те, що він виявляється справедливим і для усіченого (причому, суттєво) безлічі підстановок, формуються симетричними шифрами. Такі перетворення, що здійснюються на різних ключах зашифрування, формують безліч підстановок випадкового типу (це основне властивість, якого прагнуть розробники при побудові шифру). Про це свідчать численні результати експериментів.

Виходить, що для безлічі підстановок, що визначаються шифруючими перетвореннями, виконується властивість, що нагадує ергодичну властивість випадкових процесів (середнє за безліччю реалізацій збігається із середнім за часом для однієї досить довгої реалізації [20]). Ця властивість виявляється в тому, що закон розподілу (3.10), отриманий на основі аналізу всієї множини $2^n!$ рівноймовірних підстановок є справедливим і для безлічі осередків таблиці XOR різниць кожної окремо взятої випадкової підстановки ступеня 2^n.

Підтвердженням цього факту є те, що закону ймовірностей Pr(Λ Δ π(, X Y Δ) = 2k), розглядається стосовно окремої підстановці, з високою точністю виконується умова нормування, характерна для повної групи подій:

$$\sum_{k=0}^{k^*}\Pr(|\Lambda_\pi(\Delta X,\Delta Y)=2k)=1.$$

Тут Λ Δ π(, X Y Δ) − значення XOR таблиці (її комірки) для пари значень різниць входів і виходів ΔX , $\Delta Y \in Z_{2^m}$, $\Delta X = X + X'$, $\Delta Y = \pi(X) + \pi(X')$ підстановки $\pi \in S_{2^m}$. Значення k^* є половиною від максимального числа переходів XOR таблиці випадкової підстановки (фактично співвідношення (3.10) – це узагальнення властивостей (3.8)-(3.11)). Виконані численні перевірки підтверджує це положення.

Цілком аналогічне за змістом твердження, справедливе для ймовірності значень лінійних апроксимаційних таблиць $LAT_\pi^*(\alpha,\beta)$ випадкових підстановок [19, 21].

Нехай $\lambda^*(\alpha, \beta)$ буде випадковим значенням розподілу $LAT_\pi^*(\alpha,\beta) = |LAT_\pi^*(\alpha,\beta)-2^{n-1}|$, коли підстановка π обрана рівноймовірно з множини 2^n і маски α, β не нульові. Тоді $\lambda^*(\alpha, \beta)$ приймає тільки парні значення та:

$$\Pr(\lambda^*(\alpha\alpha,\beta = 2k) = \frac{(2^{n-1}!)^2}{2^n!} \cdot \begin{pmatrix} 2^{n-1} \\ 2^{n-2} + |k| \end{pmatrix} \tag{3.11}$$

для $|k| \le 2^{n-1}$. І для цього розподілу справедливе нормування:

$$\sum_{k=0}^{k^*} \Pr(\lambda^*(\alpha,\beta) = 2k^*) = 1 \tag{3.12}$$

Тут k^* − половинне значення максимального для таблиці $LAT_\pi^*(\alpha,\beta)$ усунення. Більш того, можна переконатися, що для розподілу (3.2) справедливе і нормування (3.9), яке в цьому випадку записується у вигляді:

$$\frac{2^n - 1}{(2^{n-1})^2} \cdot \sum_{k=-2^{n-1}}^{2^{n-1}} \frac{(2^{n-1}!)}{2^n!} \cdot \begin{pmatrix} 2^{n-1} \\ 2^{n-2} + |k| \end{pmatrix} 2 = 1 \tag{3.13}$$

На основі викладених результатів видається логічним на додаток до вже відомих підходи сформувати (сформулювати) нове (чи уточнене) визначення випадкової підстановки, як і зроблено у роботі [23]. Ми тут його нагадаємо

визначення 2. Підстановка є випадковою, якщо разом із виконанням трьох критеріїв випадку, запропонованих у роботі [24], для осередків її XOR таблиці та таблиці лінійних апроксимацій виконуються закони розподілу ймовірностей (3.10) (критерій випадковості 3.7) та (3.12) (критерій випадковості 3.8).

3.3 Шифруючі перетворення

Як випадкові підстановки. Найважливіший висновок робіт [15] і [17] у тому, що наведені вище критерії випадковості підстановок виконуються й у шифруючих перетворень всіх сучасних блокових симетричних шифрів, що розглядаються як підстановочні перетворення.

Саме собою окреме шифруюче перетворення (окремий цикл) перестав бути випадкової підстановкою, оскільки йому виконуються закони розподілу ймовірностей (3.11) та (3.13). Воно не укладається у рамки випадкових підстановок і за інверсіями, і за зростанням, і за циклами (хоча б тому, що є безліч входів у підстановку, які впливають не на всі значення виходів). Однак при реалізації механізмів перемішування (лінійних перетворень), що використовуються в кожному циклі, послідовність шифруючих перетворень набуває властивостей випадкової підстановки (чого саме і прагнуть всі розробники шифрів). Цей, начебто, тривіальний висновок залишився непоміченим розробниками шифрів та криптоаналітиками при формуванні оцінок показників стійкості шифрів до атак диференціального та лінійного криптоаналізу (вони не могли правильно інтерпретувати результати, оскільки були пов'язані повномасштабними версіями шифрів, які не піддаються обчислювальним експериментам). Як уже зазначалося вище, у всіх відомих роботах показники багатоциклових перетворень (стійкість до атак диференціального та лінійного криптоаналізу) безпосередньо зв'язувалися та зв'язуються з відповідними показниками *S*-блокових конструкцій, що використовуються як нелінійні перетворення кожної циклової функції.

Наша позиція полягає в тому, що підсумкові (асимптотичні) показники стійкості (максимуми повних диференціалів таблиць XOR) різниць послідовностей шифруючих перетворень як і максимуми лінійних апроксимаційних таблиць цих же перетворень в ній) залежать тільки від числа циклів шифруючого перетворення та розміру його бітового входу.

Зафіксуємо цей висновок у вигляді твердження. Для кожного блочного симетричного шифру (з числа відомих ітеративних БсШ) існує цілком певна кількість циклів, після якого шифр набуває властивостей випадкової підстановки. Подальше нарощування числа циклів не впливає на підсумкові диференціальні та лінійні властивості шифру. Це є одним і тим самим всім шифруючих перетворень з однаковим бітовим розміром входу.

Можна відзначити, що це твердження в першій частині видається у певному сенсі досить очевидним у тому сенсі, що кожен реальний шифр будується так, щоб набір його циклових перетворень тією чи іншою мірою мав властивості випадкової підстановки. При нашому підході ця властивість визначається як проміжний результат, що переходить в асимптотичне значення, однакове для всіх шифрів (з однаковим бітовим розміром входу), що піддається розрахунку.

Нас надалі буде цікавити саме момент (кількість циклів), починаючи з якого шифруючий перетворення стає випадковою підстановкою. Саме в цьому напрямі ми і будуватимемо доказ (обґрунтування) поданого твердження.

Продемонструємо справедливість цього затвердження з прикладу розгляду диференціальних показників шифру-підстановки. Як один з таких показників у нашому випадку. У разі виступатиме максимальне значення повного диференціалу.

Ми почнемо доказ цього твердження (скоріше не доказ, а пояснення його) правомірності) з кінця, тобто. Припустимо, що БСШ має деяку кількість циклів, після яких шифр стає випадковою підстановкою, тобто. має закон розподілу ймовірностей переходів різниць (3.11).

Покажемо, що подальше нарощування числа циклів не впливає підсумкові диференціальні властивості цього шифру.

Важливо відразу відзначити, що особливістю випадкової підстановки, яка задовольняє критерію 4, є те, що ми маємо справу не з фіксованим розподілом переходів різниць $\Delta x \rightarrow \Delta y$ (закріпленим розподілом значень входів (осередків) таблиці XOR різниць), а з випадковим. Таблиця XOR різниць випадкової підстановки визначається тим, що для неї є фіксованим число осередків кожного типу, визначених за допомогою закону розподілу $\Pr(\Lambda_f(\Delta x, \Delta y)=2k)$ у вигляді [15]:

$$\Lambda_{m,2k} = (2^m - 1)^2 \cdot \Pr(\Lambda_f(\Delta x, \Delta y)) = \frac{(2^m - 1)^2}{2^m!} \cdot k! 2^k \cdot \Phi(2^{m-1} - k). \qquad (3.14)$$

Відповідно до цього співвідношення таблиця XOR різниць випадкової підстановки має λ_0 осередків, що мають значення $\Lambda_{m,0}$, λ_1 осередків, що мають значення $\Lambda_{m,2}$, λ_2 осередків, що мають значення $\Lambda_{m,4}$ і т.д., – $\lambda_{k_f^*}$ осередків, що мають значення $\Lambda_{m,2k^*}$. Усі ці значення разом дають загальне число ненульових входів (осередків) у підматрицю таблиці XOR різниць дорівнює $2^{n-1} \times 2^{n-1}$, причому самі числа $\lambda_0, \lambda_1, \lambda_2, \ldots, \lambda_{k_f^*}$ визначаються однозначно (3.14).

Тому стосовно шифруючих багатоцикловим перетворенням – випадковим підстановкам, – диференціальні ймовірності DP^f повинні тепер інтерпретуватися в позначеннях підстановочних перетворень для ключозалежної функції f не як фіксовані, а як випадкові значення, що приймаються на безлічі ключів зашифрування (на безлічі підстановок):

$$DP^f(\Delta x,\Delta y)=DP^f(\Delta x \rightarrow \Delta y) = \Pr(\Lambda_f(\Delta x,\Delta y) = 2k) \rightarrow DP^f(\Lambda_f(\Delta x, \Delta y)=2k), \qquad (3.15)$$

причому ці ймовірності слід вважати однаковими для всіх осередків таблиці диференціальних різниць (для всіх варіантів фіксованих поєднань вхідних та вихідних різниць). Повернемося до нашого завдання. Отже, нехай r-циклове шифруюче перетворення (послідовність r-циклових перетворень) f_r з n-бітним розміром входу (і виходу) має властивістю 3.8, тобто. закон розподілу DP^{fr}

(Δx,Δy) переходів вхідних різниць Δx у вихідні різниці Δy має вигляд (3.13) з нормуванням:

$$\sum_{k=0}^{k^*} DP^{f_r}(\Lambda_f(\Delta x, \Delta y) = 2k) = 1.$$

Тоді, якщо на входи чергового циклового перетворення (підстановки) надходять деякі поєднання пар виходів попереднього перетворення випадкового типу (попередньої випадкової підстановки) підпорядковуютьсязакону розподілу XOR різниць таблиці повних диференціалів (3.13), то циклове перетворення може здійснити лише перейменування виходів та відповідних їм різниць, залишаючи результуючий закон розподілу різниць незмінним (для операції XOR підстановка разом із наступним або попереднім лінійним цикловим перетворенням є детермінованими перетвореннями та добуток випадкової в обумовленому сенсі підстановки на будь-яку іншу підстановку, є випадковою підстановкою). Наведемо математичне обґрунтування цього факту (який підтверджується численними експериментами із малими шифрами).

Нас цікавить закон розподілу ймовірностей $DP^{f_{r+1}}(\Delta x, \Delta z)$ для r + 1 циклу перетворень (тут зручніше буде перейти до компактної форми запису, введеній раніше), де Δ_z є вихідною різницею r + 1-але циклового перетворення. У нас є ланцюжок $\Delta x \to \Delta y \to \Delta z$ різниць, спільний закон розподілу ймовірностей для якої позначимо:

$$DP^{f_{r+1}}(\Delta x, \Delta y, \Delta z) = DP^{f_{r+1}}(\Delta x \to \Delta y \to \Delta z).$$

Відповідно до формули множення ймовірностей можемо записати подання для цієї ймовірності у вигляді:

$$DP^{f_{r+1}}(\Delta x, \Delta y, \Delta z) = DP^{f_r}(\Delta x, \Delta y) DP^{f_1}(\Delta z / \Delta x, \Delta y)).$$

Тоді диференціальна ймовірність $DP^{f_{r+1}}(\Delta x, \Delta z)$ для r + 1-але циклового перетворення може бути визначена із спільної ймовірності $DP^{f_{r+1}}(\Delta x, \Delta y, \Delta z)$ шляхом її усереднення по множині проміжних значень $\Delta y \in Z_2^n$, тобто.

$$DP^{f_{r+1}}(\Delta x, \Delta z) = \sum_{\Delta y \in Z_{2^n}} DP^{f_r}(\Delta x, \Delta y) DP^{f_1}(\Delta z / \Delta x, \Delta y).$$

Але в нашому випадку закон розподілу $DP^{f_r}(\Delta x, \Delta y) = \Pr(\Lambda_f(\Delta x, \Delta y) = 2k)$ є одним і тим же для кожної вихідної різниці r-циклового перетворення (для кожного осередку таблиці диференціальних різниць випадкової підстановки), а тому:

$$DP^{f_{r+1}}(\Delta x, \Delta z) = DP^{f_r}(\Delta x, \Delta y) \sum_{\Delta y \in Z_{2^n}} DP^{f_r}(\Delta z / \Delta x, \Delta y). \qquad (3.16)$$

Очевидно далі, що за фіксованих значеннях Δy вихідні різниці Δz не залежать від того, які значення набувають вхідних різниць Δx і, отже,

$$\sum_{\Delta y \in Z_{2^n}} DP^{f_r}(\Delta z / \Delta x, \Delta y) = \sum_{\Delta y \in Z_{2^n}} DP^{f_r}(\Delta z / \Delta y) = \sum_{\Delta y \in Z_{2^n}} DP^{f_r}(\Delta y \to \Delta z). \qquad (3.17)$$

Але відповідно до (3.17) для підстановочного одноциклового перетворення f_1:

$$\sum_{\Delta y \in Y} DP^{f_r}(\Delta x, \Delta y) = \sum_{\Delta y \in Y} DP^{f_r}(\Delta x \to \Delta y) = 1,$$

і, в результаті, приходимо до результату:

$$DP^{f_{r+1}}(\Delta x, \Delta z) = DP^{f_r}(\Delta x, \Delta y) \Rightarrow DP^{f_{r+1}}(\Delta x \to \Delta z) = DP^{f_r}(\Delta x \to \Delta y),$$

де: $DP^{f_r}(\Delta x \to \Delta y) = \Pr(\Lambda_f(\Delta x, \Delta y) = 2k)$.

Останнє і означає, що додаткові циклові перетворення не змінюють закону розподілу різниць на виході шифру. Залишається тепер прокоментувати першу частину затвердження. Для цього зауважимо, що ефективність перемішування вхідного тексту при зашифруванні у криптографії оцінюється такими параметрами статистичної безпеки, як лавинний ефект, коефіцієнт стиснення, ряде кореляційних показників [21].

Якщо розглядати тонку структуру циклового перетворення, то на початку процедури зашифрування (у першому циклі) при застосовуваних при побудові більшості шифрів рішень, як правило, не вдається реалізувати зв'язок кожного вихідного біта циклового перетворення з кожним вхідним бітом. Наприклад, біти входу впливають на вхід лише одного S-блоку багатоблочного нелінійного перетворення, а використовуване подальше лінійне перетворення не має повноти в тому сенсі, що воно передає вплив входу не попри всі виходи поточного перетворення. Для характеристики цієї властивості розробники шифру Rijndael запровадили спеціальну характеристику – коефіцієнт розгалуження, а сам механізм поширення активних бітів у послідовних шарах (циклах) перетворень назвали стратегією широкого сліду [22]. Але цю ж стратегію намагалися реалізувати всі розробники відомих шифрів, хоча вона була часто не такою ефективною, як, скажімо, у Rijndael-я (множення виходів кількох S-блоків на матрицю МДР коду). З іншого боку, відомі та більш ефективні конструкції лінійного шару (наприклад, у шифрі Лабіринт, або у шифрі керованої підстановки [25]). Звичайно, якщо є механізм розширення числа активних (задіяних) у ході перетворення бітів блоку даних, то рано або пізно настане момент, коли будь-який біт входу однаково ефективно діятиме на будь-який біт виходу. Цей момент якраз і буде позначати, що перетворення, що шифрує, стало випадковою підстановкою (результуючий закон розподілу переходів різниць пар входів у відповідні їм різниці пар виходів набуває вигляду (3.14)).

Цілком аналогічні міркування можуть бути наведені по відношенню до лінійних показниками багатоциклових ітеративних процедур шифруючих перетворень

3.4. Попередження показників стійкості до атаків диференціального і лінійного кріптоаналізу

Розрахункові співвідношення визначення максимальних значень повних диференціалів та максимальних значень лінійних корпусів можуть бути отримані застосуванням законів (3.14) та (3.15), справедливих для випадкових підстановок, до шифрам, що розглядаються як випадкові підстановки, що і зроблено в наших роботах [16] та [18].

Як показано в роботі [16], середнє значення максимуму таблиці XOR різниць випадкової підстановки порядку 2n знаходиться шляхом визначення максимального значення $k = k_{max}$ при якому виконується співвідношення:

$$\frac{(2^n-1)^2}{(2^n)^2}\cdot\cdot\binom{2^{n-1}}{k}\cdot k!\cdot 2^k\cdot\Phi(2^{n-1}-k)\approx 1. \qquad (3.18)$$

Якщо це співвідношення застосувати до шифру з *n*-бітовим розміром входу, то для цікавить нас максимального значення диференціальної ймовірності (максимальної ймовірності повного диференціалу) $DP^f_{\max}$ можемо записати вираз:

$$DP^f_{\max}=\frac{k_{\max}}{2^n} \qquad (3.19)$$

У роботі [16] також наведено розрахункове співвідношення, що є гарною апроксимацією співвідношень (21) і (22):

$$DP^f_{\max}=\frac{n+4}{2^n}. \qquad (3.20)$$

У роботі [18] показано, що середнє значення максимуму таблиці лінійних апроксимацій для випадкової підстановки визначається аналогічно попереднього випадку шляхом знаходження значення k^* , що є цілим рішенням (округленням у бік найближчого цілого) рівняння:

$$\frac{(2^n-1)^2\cdot(2^{n-1}!)^2}{2^n!}\cdot\begin{pmatrix}2^{n-1}\\2^{n-2}+|k^*|\end{pmatrix}=1. \qquad (3.21)$$

Відповідно для шифру з n-бітовим розміром входу максимальне значення лінійної ймовірності (максимальної ймовірності лінійного корпусу) $DL^f_{\max}$ представляється у вигляді:

$$DP^f_{\max}=\left(\frac{k_{\max}}{2^{n-1}}\right)^2 \qquad (3.22)$$

Наведемо тут також співвідношення, отримане на основі обробки результатів обчислювальних експериментів, що є зручною заміною виконання розрахунків за співвідношенням (3.23):

$$DL^f_{\max}=\left(\frac{\frac{3}{2}}{2^{n-1}}\right)^2 \qquad (3.23)$$

На основі наведених результатів та обґрунтувань можна стверджувати, що:

1. Сучасні блокові симетричні шифри (при повному наборі шифруючих багатоциклових перетворень) мають властивості випадкових підстановок і для них справедливі закони розподілу ймовірностей для повних диференціалів та лінійних корпусів властиві таблицям диференціальних різниць і лінійних апроксимацій підстановок відповідного ступеня (порядку) (3.13) та (3.14).

2. Максимальні значення повних диференціалів та лінійних корпусів для сучасних БСШ, що визначають за сучасними мірками показники стійкості шифрів до атак диференціального та лінійного криптоаналізу, можуть бути отримані розрахунковим шляхом. Вони не залежать (за достатньої кількості циклових перетворень ні від властивостей використовуваних у шифрах підстановочних конструкцій, ні від методів введення в циклові функції циклових підключень, ні від способу побудови розширюючого лінійного перетворення циклової функції, а є функцією тільки розміру бітового входу шифр (порядку підстановки).

3. Для оцінки стійкості блокових симетричних шифрів (з бітовим розміром входу рівним n) до атак диференціального та лінійного криптоаналізу можна користуватися простими співвідношеннями (3.22) та (3.23).

3.5 Криптоаналіз S-AES за допомогою методу неможливих диференціалів

Метод неможливих диференціалів – це новий спосіб атак на симетричні блокові шифри, заснований на побудові неможливої події, тобто такої події, ймовірність якого дорівнює нулю. Тоді ми можемо відкидати всі ключі, які призводять до неможливих подій. Для знаходження таких подій можна використовувати техніку «зустріч посередині» (miss-in-the-middletechnique), суть якої полягає в тому, щоб знайти дві події ймовірність яких дорівнює 1 і, поєднавши їх, отримати неможлива подія.

S-AES - симетричний блоковий шифр, що володіє властивостями звичайного AES. Він має чотири трансформації: додавання з раундовим ключем AddRoundKey, заміна по таблиці Sub-HalfBytes,зсув рядка ShiftRows і перемішування стовпців MixColumns. Алгоритм складається з двох раундів, але в другий немає перетворення MixColumns. Докладний опис алгоритму можна

знайти у роботі [4]. Для застосування методу неможливих диференціалів ми розширимо кількість раундів до п'яти.

Для застосування методу неможливих диференціалів до SAES можна побудувати такий чотирираундовий неможливий диференціал, як показано на рисунку 3.1, де зафарбовані квадрати – це напівбайти з ненульовою різницею для алгоритму S-AES.

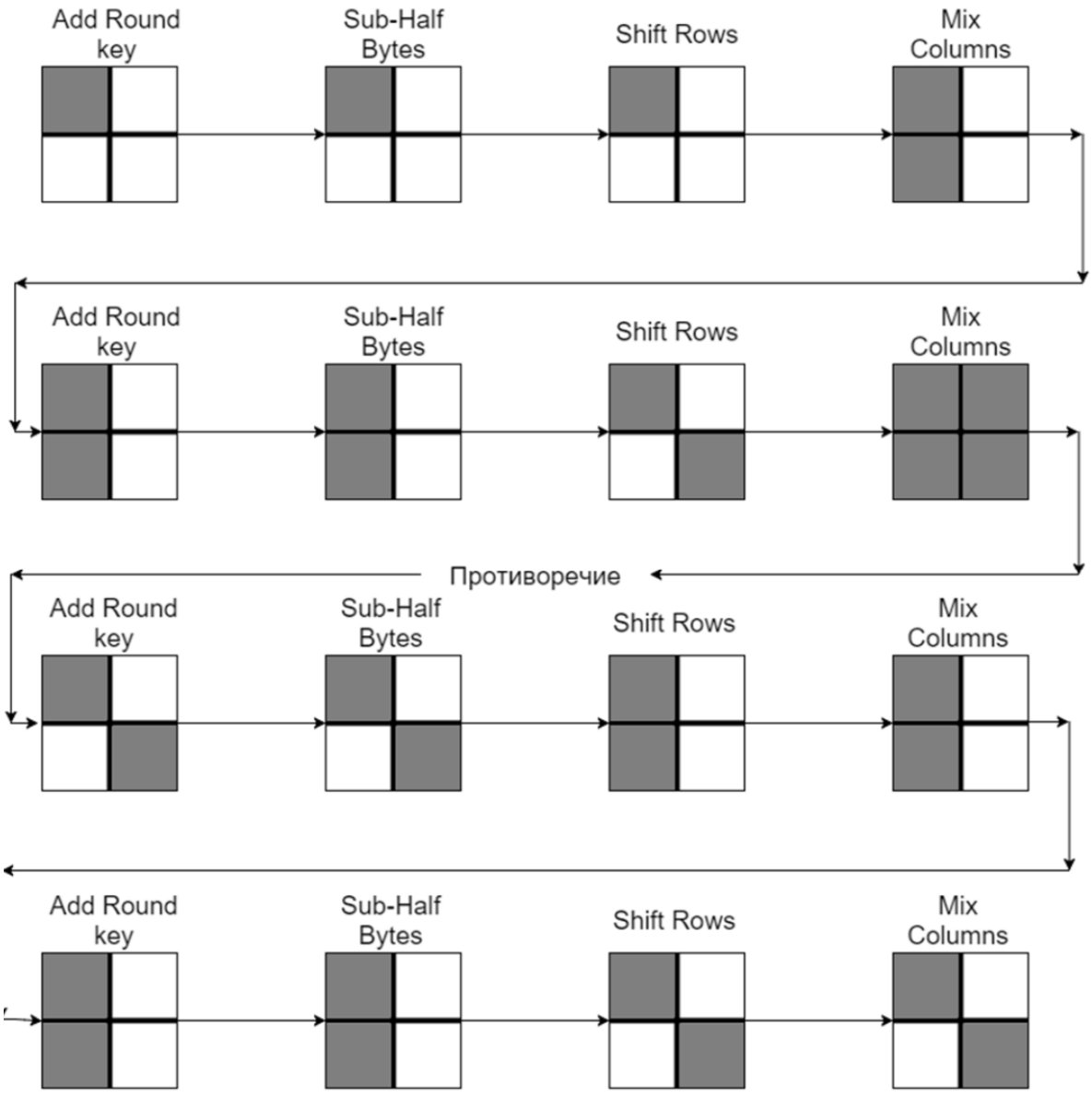

Рисунок 3.1 – Чотирьохраундовий неможливий диференціал

Після чого ми можемо додати ще один раунд на початку та отримати можливість атаки на п'ятираундовий S-AES, що показано на рисунок 3.2.

Для того щоб знайти 1 і 4 напівбайт ключа ми відібрали тексти, які мають відмінності в 1 та 4 полубайте, після чого зашифрували їх, і вибрали тільки такі, які після дешифрування мали не нульові різниці в 1 та 4 напівбайтне. Після цього ми відкидали ключі, які після першого раунду давали різницю між текстами не рівну нулю тільки у першому напівбайтному.

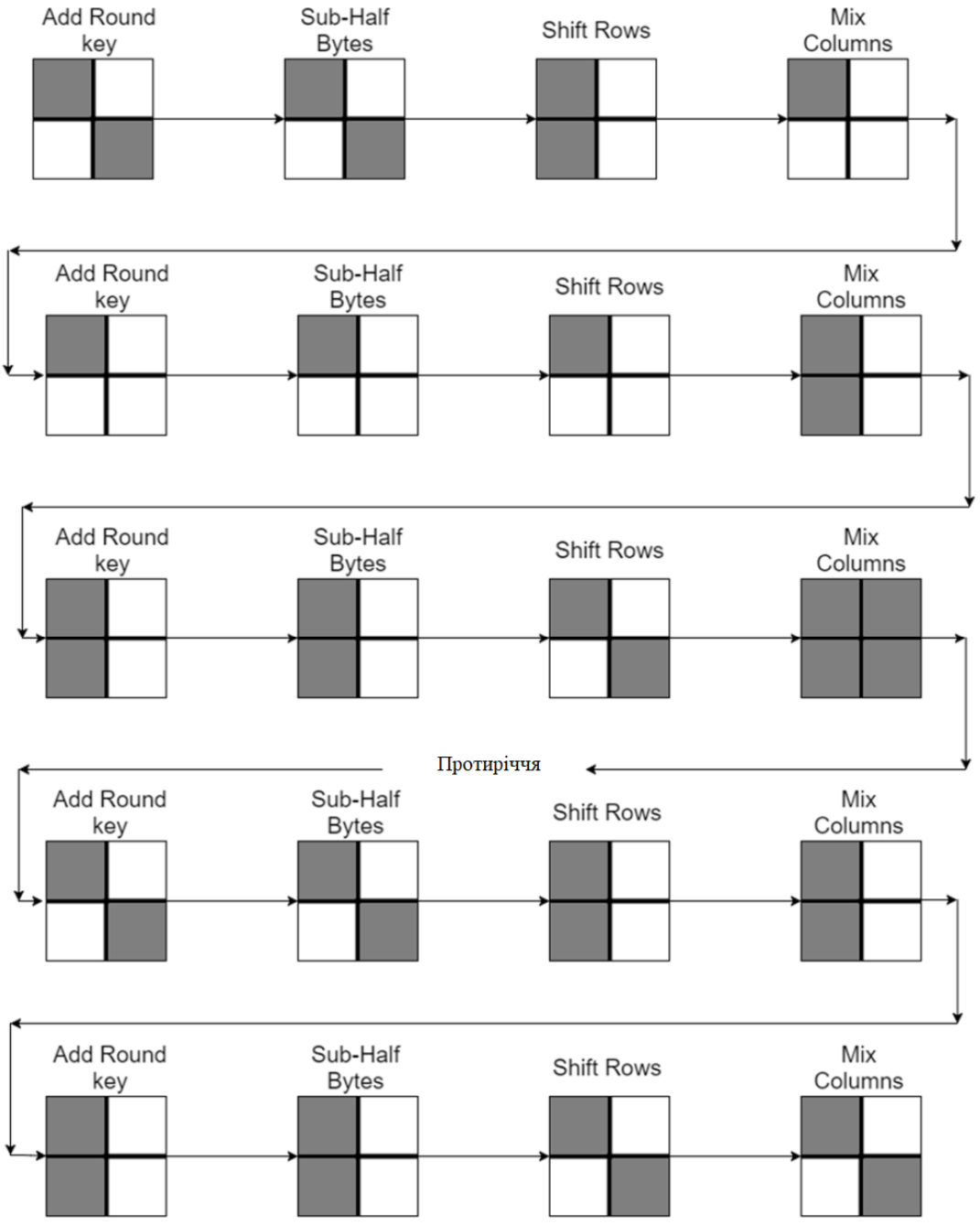

Рисунок 3.2 - П'ятираундовий неможливий диференціал для алгоритму S-AES

Також ми можемо відкидати ключі, які після першого раунду призводять до ненульової різниці у другому напівбайтному, побудувавши трохи інший неможливий диференціал, зображений на рисунку 3.3.

Опишемо детально процес знаходження першого та четвертого біта ключа для п'ятираундового S-AES.

1. Візьмемо 2^{13} пар текстів, які мають відмінність у першому та четвертому полубайте.

2. Зашифруємо ці тексти, і залишимо лише ті з них, які на виході давали різницю в першому та четвертому напівбайтах або у другому та третьому напівбайтах. У нас залишиться 2^6 таких текстів.

3. Виконаємо перетворення одного раунду шифрування для кожної з двох пар текстів з використанням усіх варіантів можливих ключів для напівбайтів з ненульовими значеннями.Таких варіантів 2^8 , після чого всі ключі, які призводили до різниці у першому чи третьому полубайте на вході другого раунду, що відкидалися. Після цих дій залишились тільки правильні біти ключа для першого та четвертого напівбайт (рисунок 3.3).

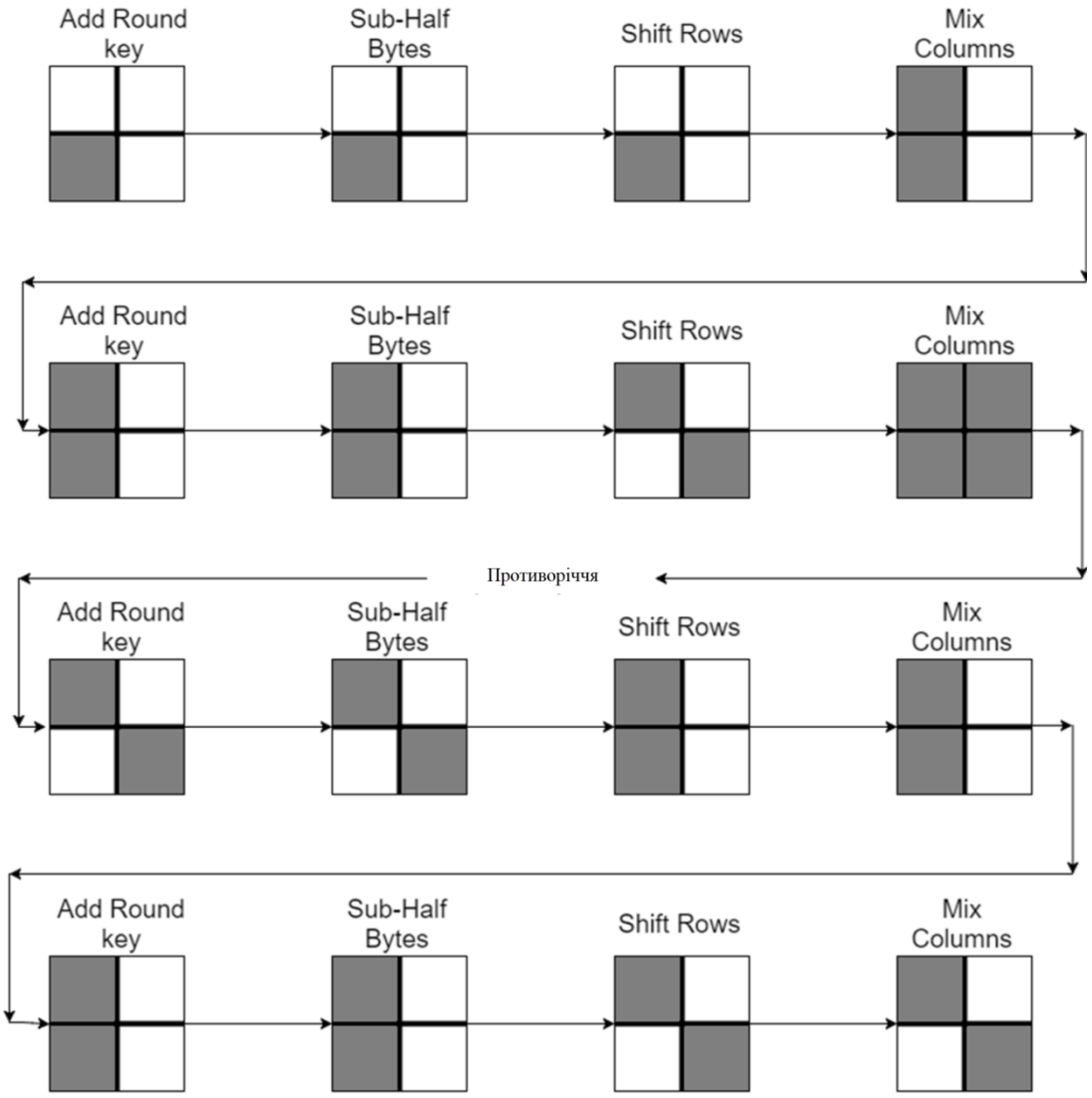

Рисунок 3.3 – Другий варіант побудови неможливих диференціалів для 4 раундів S-AES

За аналогією ми можемо знайти 2 і 3 напівбайти ключа, для цього можна використовувати неможливий диференціал, зображений на рисунок 3.4.

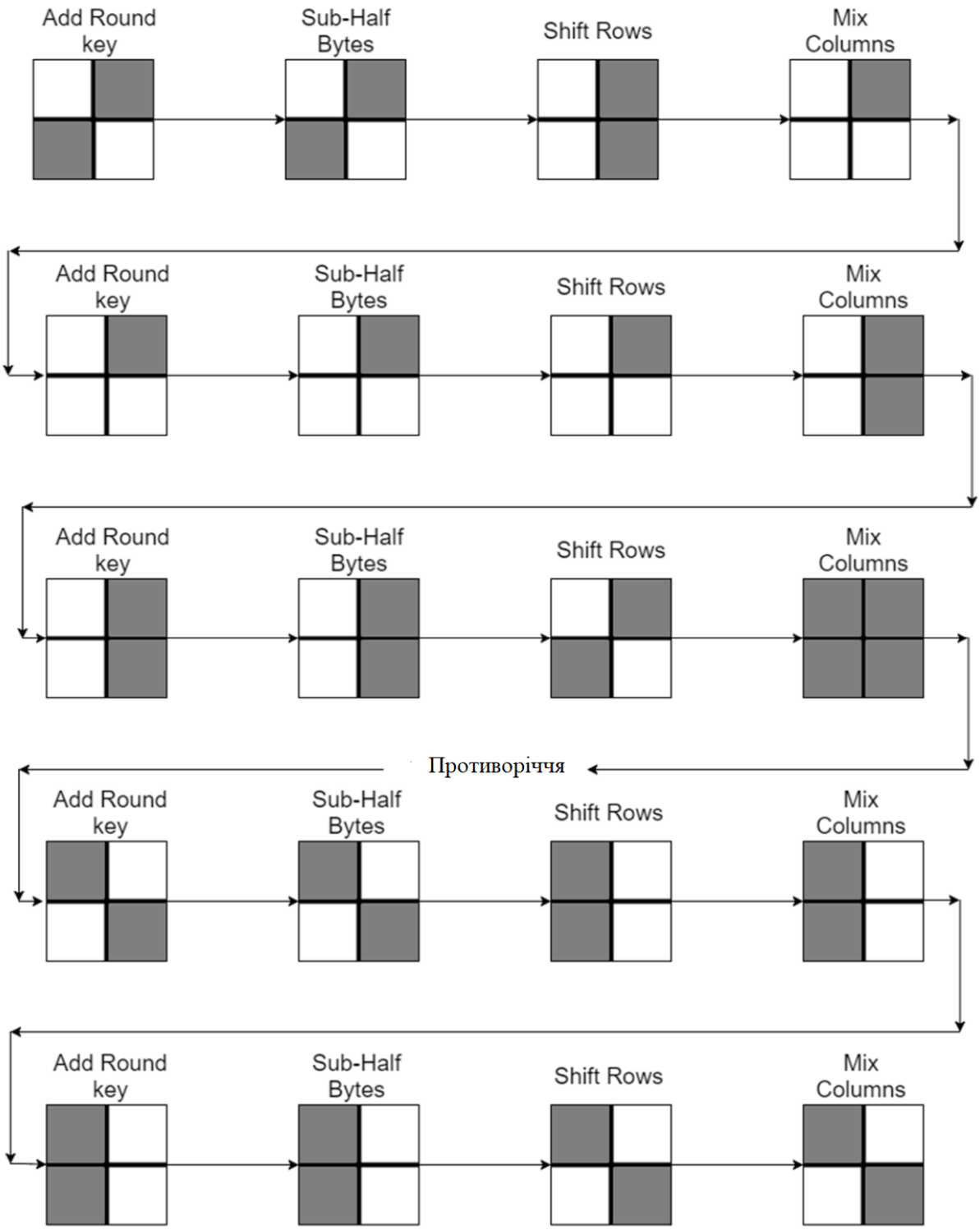

Рисунок 3.4 – Другий варіант побудови неможливих диференціалів для 5 раундів S-AES

Таким чином, ми знайшли перший ключ для п'ятираундового *S*-AES, використовуючи метод неможливих диференціалів. Для реалізації завдання крипто аналізу було створено програмний засіб мовою C++. На рисунку 3.5 приведено головне вікно середовища розробки.

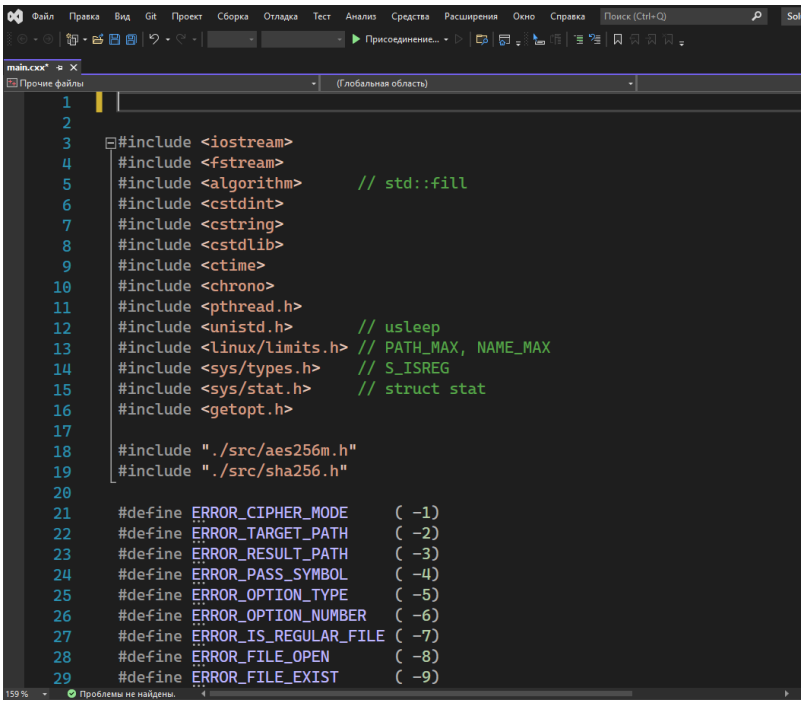

Рисунок 3.5 – Середовище розробки

Також проведено тестування програмного засобу для крипто аналізу 5 раундового шифру AES (рисунок 3.6).

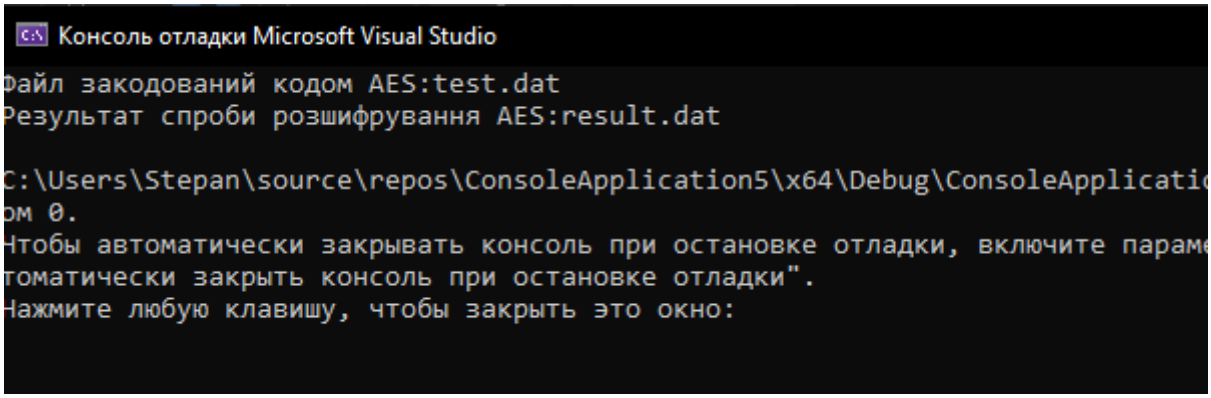

Рисунок 3.6 – Тестування програмного засобу

ВИСНОВКИ

Проведено дослідження симетричних алгоритмів шифрування, що дозволило виявити сильні та слабкі сторони симетричних шифрів.

Проведено аналіз симетричного шифру FIPS-197, котрий був конкурсантом конкурсу AES та побудований основі *S* – блоків.

Проведено дослідження математичних основ алгоритму AES, що дає можливість проаналізувати можливі алгоритми крипто аналізу шифру.

Досліджено раундові перетворення в криптосистемі AES, що дозволяють оцінити як змінюється криптостійкість в залежності від кількості раундів.

Досліджено методи лінійного та диференціального криптоаналізу, що дало можливість побудувати схему криптоаналізу за допомогою неможливих диференціалів.

Досліджено показники стікості до атак диференціального та лінійного крипто аналізу та розроблено схему криптоаналізу за допомогою неможливих диференціалів для 5 раундового шифру AES.

Систематизовано елементи лінійного та диференціального крипто аналізу та реалізовано програмне забезпечення для дешифрування даних алгоритмом AES на мові C++.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Кульчинська Н.З., Олійник Н.П., Дослідження алгоритму шифрування AES. Збірник матеріалів проблемно-наукової міжгалузевої конференції «Автоматизація та комп'ютерно – інтегровані технології» (АКІТ - 2021), Тернопіль, 2021. 145-149 с.

2. Олійник Н.П., Використаання симетричного шифру AES з реалізацією на Javascript. Збірник матеріалів проблемно-наукової міжгалузевої конференції «Автоматизація та комп'ютерно – інтегровані технології» (АКІТ - 2022), Тернопіль, 2022. 73-76 с.

3. Bhushan A. A File Transfer Protocol [Електронний ресурс] / A. Bhushan. – April 1971. – Режим доступу: https://www.rfceditor.org/rfc/pdfrfc/rfc114.txt.pdf.

4. Hauben R. From the ARPANET to the Internet : A Study of the ARPANET TCP/IP Digest and of the Role of Online Communication in the Transition from the ARPANET to the Internet [Електронний ресурс] / Ronda Hauben. – Режим доступу: http://www.columbia.edu/~rh120/other/ tcpdigest_paper.txt.

5. Huckle S. Internet of Things, Blockchain and Shared Economy Applications / S. Huckle, R. Bhattacharya, M. White, N. Beloff // Procedia Comput. Science. – 2016. – Вид. 98. – с. 461–466.

6. Dingledine R. Tor: The second-generation onion router [Електронний ресурс] / Roger Dingledine, Nick Mathewson, Paul Syverson. – Режим доступу: https://www.freehaven.net/anonbib/cache/draft-tor-design-2004.pdf.

7. Cjdns [Електронний ресурс]. – Режим доступу: https://github.com/cjdelisle/cjdns/blob/master/doc/Whitepaper.md.

8. Chris O'Brien. Open Garden uses FireChat in Tahiti to create cell phone network that eliminates need for carriers. (October 20, 2015). [Електронний ресурс]. – Режим доступу: https://venturebeat.com/2015/10/20/open-garden-uses-firechat-in-tahiti-to-createa-cell-phone-network-that-eliminates-need-for-carriers.

9. Foster I. The Grid: Blueprint for a new computing infrastructure /Foster Ian, Kesselman Carl. – San Francisco, CA, USA : MorganKaufmann Publishers Inc., 1999. – 677 p.

10. Касянчук М.М. Якименко І.З., Волинський О.І., Івасьєв С.В. Теоретичні основи аналітики та алгоритми оптимізації обчислень простих чисел. Поступ в науку. Збірник наукових праць Бучацького інституту менеджменту і аудиту. Матеріали Міжнародної проблемно-наукової міжгалузевої конференції «Інформаційні проблеми комп'ютерних систем, юриспруденції, енергетики, економіки, моделювання та управління (ПНМК-2010)». В.6, т.1.- с.33-36.

11. Касянчук М.М. Теорія та математичні закономірності досконалої форми системи залишкових класів. Праці Міжнародного симпозіуму „Питання оптимізації обчислень (ПОО–XXXV)". Т.1. Київ–Кацивелі. 2009.– С. 306–310.

12. Лабунец В.Г. Теоретико-числовые преобразования над полями алгебраических чисел. Применение ортогональных методов при обработке сигналов и анализе систем. Свердловск: УПИ, 1981, с.44-54.

13. Маккелан Дж. Х., Рэйдер Ч.М. Применение теории чисел в цифровой обработке сигналов. М.: Радио и связь, 1983. - 264 с.

14. Николайчук Я.М., Волинський О.І., Кулина С.В. Теоретичні основи побудови спецпроцесорів у базисі Крестенсона. Вісник Хмельницького національного університету. 2007. № 3. Т.І(93). С. 85-90.

15. Николайчук Я.М. Теорія джерел інформації. Тернопіль: ТзОВ„Терно–граф", 2010. – 536 с.

16. Николайчук Я.Н., Божнев В.П., Зевелев С.Я. Применение методов теории чисел для сжатия измерительной информации в системах телеконтроля процессов бурения. Материалы Всесоюзной конференции молодых ученых нефтяных ВУЗов. М.:МИНХиГП, 1975. С. 134-138.

17. Якименко І.З., Касянчук М.М., Тимошенко Л.М., Івасьєв С.В., Николайчук Я.М. Алгоритм знаходження системи модулів модифікованої досконалої форми системи залишкових класів, Матеріали МНПК СІЕТ.- Одеса. 2014. С. 115-117.

18. Ященко В.В., Варновский Ю.В. , Нестеренко Г.А. Кабатянский Введение в криптографию. Питер, 2001. 271 с.

19. Abdallah M., Skavantzos A. On MultiModuli residue number systems with moduli of forms ra, rb-1, rc+1. IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS-I: REGULAR PAPERS, VOL. 52, NO. 7 , 2005. P. 132.

20. Fischer W. Seifert P. Note on fast computation of secret RSA exponents Information Security and Privacy (ACISP 2002), Vol. 2384 of Lecture Notes in Computer Science, Springer-Verlag, 2002, P. 136–143.

21. Gbolagade K.A., Cotofana S. D. Residue Number System Operands to Decimal Conversion for 3-Moduli Sets, Proceedings of 51st IEEE Midwest Symposium on Circuits and Systems (MWSCAS-08). Knoxville, USA. – 2008. P. 791-794.

22. Hosseinzadeh M., Navi K., Timarchi S. Design Residue Number System Circuits in Current mode. 14th Iranian Conference of Electrical Engineering, 2006. P.178–142.

23. Kozaczko D., Kasianchuk M., Yakymenko I., Ivasiev S.Vector Module Exponential in the Remaining Classes System. Proceedings of the 2015 IEEE 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS–2015). Warsaw, Poland. V.1, September, 2015. P.161–163.

24. Lakhani G. Some Fast Residual Arithmetic Adders. International Journal of Electronics. - 1994. - P. 225-240.

25. Lenstra H. W. Divisors in residue classes. Math. Comput. 1984. Vol. 42. N 165. P.331-340.

26. Nykolaychuk Ya. M., Kasianchuk M.M., Yakymenko I.Z., Theoretical Foundations of the Modified Perfect form of Residue Number System. Cybernetics and Systems Analysis, 2016, P. 219-223.

27. Sousa L. Efficient Method for Magnitude Comparison in RNS Based on Two Pairs of Conjugate Moduli. Electrical and Computer Engineering Department, INESC-ID/IST, TU Lisbon. P. 152.

28. Tsmots I., Teslyuk V., Teslyuk T., Ihnatyev I. Basic Components of

Neuronetworks with Parallel Vertical Group Data Real-Time Processing. Advances in Intelligent Systems and Computing II. CSIT 2017. Advances in Intelligent Systems and Computing, vol. 689, Springer, Cham. P. 558 – 576.

29. Yakymenko I., Kasyanchuk M., Nykolajchuk Ya. Matrix Algorithms of Processing of the Information Flow in Computer Systems Based on Theoretical and Numerical Krestenson's Basis. Proceedings of the X–th International Conference "Modern Problems of Radio Engineering, Telecommunications and Computer Science" (TCSET–2010). L'viv–Slavske. 2010. – P.241.

30. Yang L. L., Hanzo L. A residue number system based parallel communication scheme using orthogonal signaling: Part II—Multipath fading channels, IEEE Trans. Veh. Technol. 2002. - Vol. 51. P. 1541-1553.

31. Yang L. L., Hanzo L.Minimum-distance decoding of redundant residue number system codes. Proc. IEEE ICC '2001. – Helsinki (Finland), 2001. P. 2975-2979.

32. Yang L. L., Hanzo L. Performance analysis of coded M-array orthogonal signaling using errors-and-erasures decoding over frequency-selective fading channels. IEEE J. Select. Areas Community. 2001. Vol. 19. P. 211-221.

ДОДАТОК А

Код програмного засобу

```
#include <iostream>
#include <fstream>
#include <algorithm>      // std::fill
#include <cstdint>
#include <cstring>
#include <cstdlib>
#include <ctime>
#include <chrono>
#include <pthread.h>
#include <unistd.h>       // usleep
#include <linux/limits.h> // PATH_MAX, NAME_MAX
#include <sys/types.h>    // S_ISREG
#include <sys/stat.h>     // struct stat
#include <getopt.h>

#include "./src/aes256m.h"
#include "./src/sha256.h"

#define ERROR_CIPHER_MODE     ( -1)
#define ERROR_TARGET_PATH     ( -2)
#define ERROR_RESULT_PATH     ( -3)
#define ERROR_PASS_SYMBOL     ( -4)
#define ERROR_OPTION_TYPE     ( -5)
#define ERROR_OPTION_NUMBER   ( -6)
#define ERROR_IS_REGULAR_FILE ( -7)
#define ERROR_FILE_OPEN       ( -8)
#define ERROR_FILE_EXIST      ( -9)
#define ERROR_THREAD_CREATE   (-10)
#define ERROR_THREAD_JOIN     (-11)
#define ERROR_NOT_AES_CRYPTED (-12)

using std::cout;
using std::endl;
using std::cerr;

int finish = 0;

int checkPassSymbols(const char* str);
int isRegularFile(const char* path);
```

```
void* spinner(void* none);

int main(int argc, char* argv[]) {
	int inv = -1;
	int mode = 0;
	char* input = nullptr;
	char* output = nullptr;
	uint8_t* hash = nullptr;
	pthread_t pid = 0;
	int retval = 0;

	char help[] =
		"REQUIRED OPTIONS\n"
		"	-e | -d\n"
		"		Encryption | decryption operation respectively\n"
		"\n"
		"	--mode (-m) m\n"
		"		Cipher mode, m can be \"ECB\" (\"ecb\") or \"CBC\"
(\"cbc\")\n"
		"\n"
		"	--input (-i) i\n"
		"		Operation input, i can be either a full path to a regular file\n"
		"		or a filename of a regular file (if the file is in current
directory)\n"
		"		with maximum length of PATH_MAX (see value in
<linux/limits.h>)\n"
		"\n"
		"	--output (-o) o\n"
		"		Operation output, o can be either a full path with a filename\n"
		"		or a filename (file will be created in current directory)\n"
		"		with maximum length of PATH_MAX (see value in
<linux/limits.h>)\n"
		"\n"
		"	--pass (-p) p\n"
		"		Password that is used for creating a secret key for
encryption(decryption)\n"
		"		operation, p is a string of chars in (0x20, 07f) ASCII-table
codes with\n"
		"		unlimited length\n";

#ifndef DEBUG
	char help_build[] = "\nBUILD: DEFAULT (-O2 OPTIMIZATION)\n";
#else
	char help_build[] = "\nBUILD: DEBUG (-O0 OPTIMIZATION)\n";
#endif
```

```
	// ------------------------------------------------
	// ----------------- GETOPT_LONG START ------------------
	// ------------------------------------------------

	const char* optstring = "edm:i:o:p:h";

	const struct option longopts[] = {
		{ "mode",   required_argument, NULL, 'm' },
		{ "input",  required_argument, NULL, 'i' },
		{ "output", required_argument, NULL, 'o' },
		{ "pass",   required_argument, NULL, 'p' },
		{ "help",   no_argument,       NULL, 'h' },
		{ NULL, 0, NULL, 0 }
	};

	int opt_curr  =  0;
	int longindex = -1;

	while ((opt_curr = getopt_long(argc, argv, optstring, longopts, &longindex)) !=
-1) {
		switch (opt_curr) {
			case 'e' : {
				inv = 0;
				break;
			}
			case 'd' : {
				inv = 1;
				break;
			}
			case 'm' : {
				char* mode_str = optarg;
				if (!std::strcmp(mode_str, "ECB") || !std::strcmp(mode_str,
"ecb"))
					mode = 1;
				else if (!std::strcmp(mode_str, "CBC") ||
!std::strcmp(mode_str, "cbc"))
					mode = 2;
				else {
					cerr << "main: Invalid cipher mode, rerun with -h for
help" << endl;
					return ERROR_CIPHER_MODE;
				}
				break;
			}
			case 'i' : {
				input = optarg;
```

```
                        if (std::strlen(input) > PATH_MAX) {
                                cerr << "main: Invalid input path, rerun with -h for
help" << endl;
                                return ERROR_TARGET_PATH;
                        }
                        break;
                }
                case 'o' : {
                        output = optarg;
                        if (std::strlen(output) > PATH_MAX) {
                                cerr << "main: Invalid output path, rerun with -h for
help" << endl;
                                return ERROR_RESULT_PATH;
                        }
                        break;
                }
                case 'p' : {
                        if (checkPassSymbols(optarg)) {
                                cerr << "main: Invalid symbol in password, rerun
with -h for help" << endl;
                                return ERROR_PASS_SYMBOL;
                        }
                        SHA256 sha256;
                        std::string hashString = sha256(optarg,
KEY_HASH_SIZE);
                        hash = reinterpret_cast<uint8_t*>(&hashString[0]);
                        std::memset(optarg, 0x00, std::strlen(optarg));
                        std::fill(hashString.begin(), hashString.end(), 0x00);
                        break;
                }
                case 'h' : {
                        cout << help << help_build;
                        return 0;
                }
                case '?' : default : {
                        cerr << "main: Invalid option, rerun with -h for help" <<
endl;
                        return ERROR_OPTION_TYPE;
                }
        }
    }

    // -----------------------------------------------------
    // ------------------ GETOPT_LONG END ------------------
    // -----------------------------------------------------
```

```
	// Checking if input is a regular file
	if (!isRegularFile(input)) {
		cerr << "main: No such input or input is not a regular file, rerun with -h
for help" << endl;
		return ERROR_IS_REGULAR_FILE;
	}

	// Opening infile (input) if it is a regular file
	std::ifstream infile(
		input,
		std::ios::in |
		std::ios::binary |
		std::ios::ate
	);

	if (!infile.is_open()) {
		cerr << "main: " << std::strerror(errno) << endl;
		return ERROR_FILE_OPEN;
	}

	// If infile is opened
	unsigned long long infile_size = infile.tellg();
	infile.unsetf(std::ios::skipws);
	infile.seekg(0, std::ios::beg);

	// Checking if outfile already exists
	/*if (std::ifstream(output)) {
		cout << "main: File already exists" << endl;
		return ERROR_FILE_EXIST;
	}*/

	// Opening outfile if it does not exist
	std::ofstream outfile(
		output,
		std::ios::out |
		std::ios::binary
	);

	if (!outfile.is_open()) {
		cerr << "main: " << std::strerror(errno) << endl;
		return ERROR_FILE_OPEN;
	}

	// If outfile is opened
	if (!inv) {
		if (pthread_create(&pid, NULL, spinner, NULL)) {
```

```
                cout << "main: Bad thread creation" << endl;
                return ERROR_THREAD_CREATE;
            }

            auto start = std::chrono::steady_clock::now();

            if (mode == 1)
                AES_ECB_EncryptFile(infile_size, &infile, &outfile, hash);
            else if (mode == 2) {
                uint8_t iv[BLOCK_SIZE];
                std::srand(std::time(nullptr));
                for (size_t i = 0; i < BLOCK_SIZE; ++i)
                    iv[i] = rand() % 256;
                AES_CBC_EncryptFile(iv, infile_size, &infile, &outfile, hash);
            }

            auto end = std::chrono::steady_clock::now();
            long double diff =
std::chrono::duration_cast<std::chrono::milliseconds>(end - start).count();

            finish = 1;
            if (pthread_join(pid, (void**)&retval)) {
                cout << "main: Bad thread join" << endl;
                return ERROR_THREAD_JOIN;
            }

            cout.precision(4);
            cout << "File size:      " << static_cast<double>(infile_size) / 1048576
<< " Mbyte" << endl;
            cout << "Time taken:     " << diff / 1000 << " seconds" << endl;
            cout << "Encrypted file: " << output << endl;
            cout << std::fixed;
        }
        else {
            if (!(infile_size % BLOCK_SIZE)) {
                if (pthread_create(&pid, NULL, spinner, NULL)) {
                    cout << "main: Bad thread creation" << endl;
                    return ERROR_THREAD_CREATE;
                }

                auto start = std::chrono::steady_clock::now();

                if (mode == 1)
                    AES_ECB_DecryptFile(infile_size, &infile, &outfile,
hash);
                else if (mode == 2)
```

```
                        AES_CBC_DecryptFile(infile_size, &infile, &outfile,
hash);

                        auto end = std::chrono::steady_clock::now();
                        long double diff =
std::chrono::duration_cast<std::chrono::milliseconds>(end - start).count();

                        finish = 1;
                        if (pthread_join(pid, (void**)&retval)) {
                                cout << "main: Bad thread join" << endl;
                                return ERROR_THREAD_JOIN;
                        }

                        cout.precision(4);
                        cout << "File size:      " << static_cast<double>(infile_size) /
1048576 << " Mbyte" << endl;
                        cout << "Time taken:     " << diff / 1000 << " seconds" << endl;
                        cout << "Decrypted file: " << output << endl;
                        cout << std::fixed;
                }
                else {
                        cerr << "main: File is not AES-crypted" << endl;
                        return ERROR_NOT_AES_CRYPTED;
                }
        }

        std::memset(hash, 0x00, 32);

        infile.close();
        outfile.close();

        return 0;
}

int checkPassSymbols(const char* str) {
        int length = std::strlen(str);

        for (int i = 0; i < length; ++i)
                // if (str[i] <= SPACE || str[i] >= DEL)
                if (str[i] <= 0x20 || str[i] >= 0x7f)
                        return 1;

                return 0;
}

int isRegularFile(const char* path) {
```

```
        struct stat s;
        stat(path, &s);
        return S_ISREG(s.st_mode);
}

void* spinner(void* none) {
        static constexpr char spin_chars[] = "/-\\|";
        int i = 0;

        while (!finish) {
                cout << "Processing, please wait... " << spin_chars[i++ %
sizeof(spin_chars)];
                cout.flush();
                usleep(100000);
                cout << "\r";
        }

        cout << "Processing, please wait... Done" << endl;
        return nullptr;
}
```

```
#include <iostream>
#include <fstream>
#include <algorithm>      // std::fill
#include <cstdint>
#include <cstring>
#include <cstdlib>
#include <ctime>
#include <chrono>
#include <pthread.h>
#include <unistd.h>       // usleep
#include <linux/limits.h> // PATH_MAX, NAME_MAX
#include <sys/types.h>    // S_ISREG
#include <sys/stat.h>     // struct stat
#include <getopt.h>

#include "./src/aes256m.h"
#include "./src/sha256.h"

#define ERROR_CIPHER_MODE     ( -1)
#define ERROR_TARGET_PATH     ( -2)
#define ERROR_RESULT_PATH     ( -3)
#define ERROR_PASS_SYMBOL     ( -4)
#define ERROR_OPTION_TYPE     ( -5)
#define ERROR_OPTION_NUMBER   ( -6)
#define ERROR_IS_REGULAR_FILE ( -7)
#define ERROR_FILE_OPEN       ( -8)
#define ERROR_FILE_EXIST      ( -9)
#define ERROR_THREAD_CREATE   (-10)
#define ERROR_THREAD_JOIN     (-11)
#define ERROR_NOT_AES_CRYPTED (-12)

using std::cout;
using std::endl;
using std::cerr;

int finish = 0;

int checkPassSymbols(const char* str);
int isRegularFile(const char* path);
void* spinner(void* none);

int main(int argc, char* argv[]) {
        int inv = -1;
        int mode = 0;
```

```
	char* input = nullptr;
	char* output = nullptr;
	uint8_t* hash = nullptr;
	pthread_t pid = 0;
	int retval = 0;

	char help[] =
		"REQUIRED OPTIONS\n"
		"	-e | -d\n"
		"		Encryption | decryption operation respectively\n"
		"\n"
		"	--mode (-m) m\n"
		"		Cipher mode, m can be \"ECB\" (\"ecb\") or \"CBC\" 
(\"cbc\")\n"
		"\n"
		"	--input (-i) i\n"
		"		Operation input, i can be either a full path to a regular file\n"
		"		or a filename of a regular file (if the file is in current 
directory)\n"
		"		with maximum length of PATH_MAX (see value in 
<linux/limits.h>)\n"
		"\n"
		"	--output (-o) o\n"
		"		Operation output, o can be either a full path with a filename\n"
		"		or a filename (file will be created in current directory)\n"
		"		with maximum length of PATH_MAX (see value in 
<linux/limits.h>)\n"
		"\n"
		"	--pass (-p) p\n"
		"		Password that is used for creating a secret key for 
encryption(decryption)\n"
		"		operation, p is a string of chars in (0x20, 07f) ASCII-table 
codes with\n"
		"		unlimited length\n";

#ifndef DEBUG
	char help_build[] = "\nBUILD: DEFAULT (-O2 OPTIMIZATION)\n";
#else
	char help_build[] = "\nBUILD: DEBUG (-O0 OPTIMIZATION)\n";
#endif

	// ------------------------------------------------------
	// ----------------- GETOPT_LONG START ------------------
	// ------------------------------------------------------

	const char* optstring = "edm:i:o:p:h";
```

```
	const struct option longopts[] = {
		{ "mode",   required_argument, NULL, 'm' },
		{ "input",  required_argument, NULL, 'i' },
		{ "output", required_argument, NULL, 'o' },
		{ "pass",   required_argument, NULL, 'p' },
		{ "help",   no_argument,       NULL, 'h' },
		{ NULL, 0, NULL, 0 }
	};

	int opt_curr  =  0;
	int longindex = -1;

	while ((opt_curr = getopt_long(argc, argv, optstring, longopts, &longindex)) !=
-1) {
		switch (opt_curr) {
			case 'e' : {
				inv = 0;
				break;
			}
			case 'd' : {
				inv = 1;
				break;
			}
			case 'm' : {
				char* mode_str = optarg;
				if (!std::strcmp(mode_str, "ECB") || !std::strcmp(mode_str,
"ecb"))
					mode = 1;
				else if (!std::strcmp(mode_str, "CBC") ||
!std::strcmp(mode_str, "cbc"))
					mode = 2;
				else {
					cerr << "main: Invalid cipher mode, rerun with -h for
help" << endl;
					return ERROR_CIPHER_MODE;
				}
				break;
			}
			case 'i' : {
				input = optarg;
				if (std::strlen(input) > PATH_MAX) {
					cerr << "main: Invalid input path, rerun with -h for
help" << endl;
					return ERROR_TARGET_PATH;
				}
```

```
                        break;
                }
                case 'o' : {
                        output = optarg;
                        if (std::strlen(output) > PATH_MAX) {
                                cerr << "main: Invalid output path, rerun with -h for
help" << endl;
                                return ERROR_RESULT_PATH;
                        }
                        break;
                }
                case 'p' : {
                        if (checkPassSymbols(optarg)) {
                                cerr << "main: Invalid symbol in password, rerun
with -h for help" << endl;
                                return ERROR_PASS_SYMBOL;
                        }
                        SHA256 sha256;
                        std::string hashString = sha256(optarg,
KEY_HASH_SIZE);
                        hash = reinterpret_cast<uint8_t*>(&hashString[0]);
                        std::memset(optarg, 0x00, std::strlen(optarg));
                        std::fill(hashString.begin(), hashString.end(), 0x00);
                        break;
                }
                case 'h' : {
                        cout << help << help_build;
                        return 0;
                }
                case '?' : default : {
                        cerr << "main: Invalid option, rerun with -h for help" <<
endl;
                        return ERROR_OPTION_TYPE;
                }
        }
    }

    // -----------------------------------------------------
    // ------------------ GETOPT_LONG END -------------------
    // -----------------------------------------------------

    // Checking if input is a regular file
    if (!isRegularFile(input)) {
        cerr << "main: No such input or input is not a regular file, rerun with -h
for help" << endl;
        return ERROR_IS_REGULAR_FILE;
```

```
}

// Opening infile (input) if it is a regular file
std::ifstream infile(
        input,
        std::ios::in |
        std::ios::binary |
        std::ios::ate
);

if (!infile.is_open()) {
        cerr << "main: " << std::strerror(errno) << endl;
        return ERROR_FILE_OPEN;
}

// If infile is opened
unsigned long long infile_size = infile.tellg();
infile.unsetf(std::ios::skipws);
infile.seekg(0, std::ios::beg);

// Checking if outfile already exists
/*if (std::ifstream(output)) {
        cout << "main: File already exists" << endl;
        return ERROR_FILE_EXIST;
}*/

// Opening outfile if it does not exist
std::ofstream outfile(
        output,
        std::ios::out |
        std::ios::binary
);

if (!outfile.is_open()) {
        cerr << "main: " << std::strerror(errno) << endl;
        return ERROR_FILE_OPEN;
}

// If outfile is opened
if (!inv) {
        if (pthread_create(&pid, NULL, spinner, NULL)) {
                cout << "main: Bad thread creation" << endl;
                return ERROR_THREAD_CREATE;
        }

        auto start = std::chrono::steady_clock::now();
```

```
            if (mode == 1)
                AES_ECB_EncryptFile(infile_size, &infile, &outfile, hash);
            else if (mode == 2) {
                uint8_t iv[BLOCK_SIZE];
                std::srand(std::time(nullptr));
                for (size_t i = 0; i < BLOCK_SIZE; ++i)
                    iv[i] = rand() % 256;
                AES_CBC_EncryptFile(iv, infile_size, &infile, &outfile, hash);
            }

            auto end = std::chrono::steady_clock::now();
            long double diff =
std::chrono::duration_cast<std::chrono::milliseconds>(end - start).count();

            finish = 1;
            if (pthread_join(pid, (void**)&retval)) {
                cout << "main: Bad thread join" << endl;
                return ERROR_THREAD_JOIN;
            }

            cout.precision(4);
            cout << "File size:      " << static_cast<double>(infile_size) / 1048576
<< " Mbyte" << endl;
            cout << "Time taken:     " << diff / 1000 << " seconds" << endl;
            cout << "Encrypted file: " << output << endl;
            cout << std::fixed;
        }
        else {
            if (!(infile_size % BLOCK_SIZE)) {
                if (pthread_create(&pid, NULL, spinner, NULL)) {
                    cout << "main: Bad thread creation" << endl;
                    return ERROR_THREAD_CREATE;
                }

                auto start = std::chrono::steady_clock::now();

                if (mode == 1)
                    AES_ECB_DecryptFile(infile_size, &infile, &outfile,
hash);
                else if (mode == 2)
                    AES_CBC_DecryptFile(infile_size, &infile, &outfile,
hash);

                auto end = std::chrono::steady_clock::now();
```

```
                        long double diff =
std::chrono::duration_cast<std::chrono::milliseconds>(end - start).count();

                        finish = 1;
                        if (pthread_join(pid, (void**)&retval)) {
                                cout << "main: Bad thread join" << endl;
                                return ERROR_THREAD_JOIN;
                        }

                        cout.precision(4);
                        cout << "File size:      " << static_cast<double>(infile_size) /
1048576 << " Mbyte" << endl;
                        cout << "Time taken:     " << diff / 1000 << " seconds" << endl;
                        cout << "Decrypted file: " << output << endl;
                        cout << std::fixed;
                }
                else {
                        cerr << "main: File is not AES-crypted" << endl;
                        return ERROR_NOT_AES_CRYPTED;
                }
        }

        std::memset(hash, 0x00, 32);

        infile.close();
        outfile.close();

        return 0;
}

int checkPassSymbols(const char* str) {
        int length = std::strlen(str);

        for (int i = 0; i < length; ++i)
                // if (str[i] <= SPACE || str[i] >= DEL)
                if (str[i] <= 0x20 || str[i] >= 0x7f)
                        return 1;

                return 0;
}

int isRegularFile(const char* path) {
        struct stat s;
        stat(path, &s);
        return S_ISREG(s.st_mode);
}
```

```
void* spinner(void* none) {
	static constexpr char spin_chars[] = "/-\\|";
	int i = 0;

	while (!finish) {
		cout << "Processing, please wait... " << spin_chars[i++ %
sizeof(spin_chars)];
		cout.flush();
		usleep(100000);
		cout << "\r";
	}

	cout << "Processing, please wait... Done" << endl;
	return nullptr;
}
```

ДОДАТОК Б

Світокопія публікацій

ЗАХІДНОУКРАЇНСЬКИЙНАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІВАНО-ФРАНКІВСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ НАФТИ І ГАЗУ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ВОДНОГО ГОСПОДАРСТВА ТА ПРИРОДОКОРИСТУВАННЯ
НАЦІОНАЛЬНИЙ ТРАНСПОРТНИЙ УНІВЕРСИТЕТ
НАДВІРНЯНСЬКИЙ КОЛЕДЖ НТУ
ГАЛИЦЬКИЙ КОЛЕДЖ ІМ. В. ЧОРНОВОЛА

Проблемно-наукова міжгалузева конференція

АВТОМАТИЗАЦІЯ ТА КОМП'ЮТЕРНО-ІНТЕГРОВАНІ ТЕХНОЛОГІЇ

(АКІТ – 2021)

20—22 лютого 2021 року

Тернопіль

$$\begin{Vmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{Vmatrix} = \begin{Vmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{Vmatrix} * \begin{Vmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{Vmatrix} + \begin{Vmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{Vmatrix}$$

Рисунок 2 - Побудова *S*-box

ShiftRows працює з рядками State (рисунок 3). При цій трансформації рядки стану циклічно зсуваються на r байт по горизонталі, в залежності від номера рядка. Для нульовою рядки $r = 0$, для першого рядка $r = 1$ і т. д.

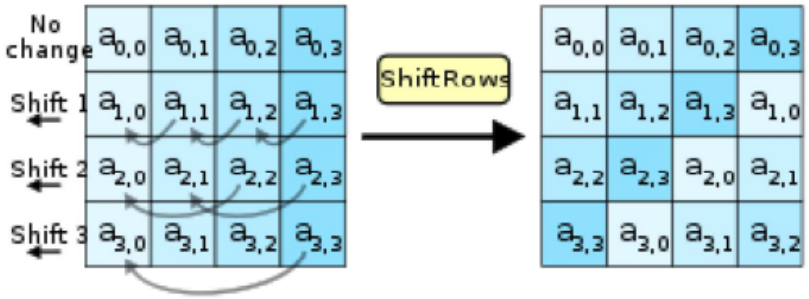

Рисунок 3 - ShiftRows ()

Таким чином кожна колонка вихідного стану після застосування процедури ShiftRows складається з байтів з кожної колонки початкового стану. Для алгоритму Rijndael патерн зсуву рядків для 128 і 192-бітних рядків однаковий. Однак для блоку розміром 256 біт відрізняється від попередніх тим, що 2, 3, і 4-ті рядки зміщуються на 1, 3, і 4 байта, відповідно (рисунок 4).

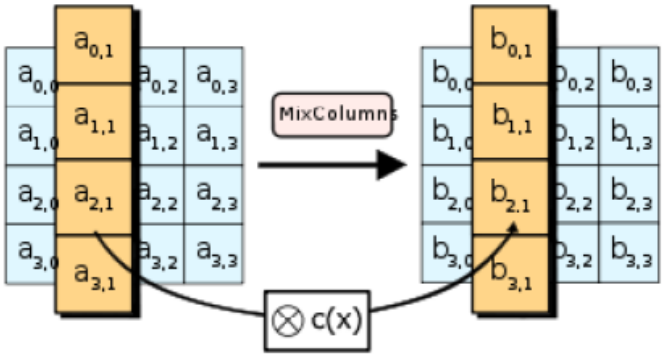

Рисунок 4 - MixColumns ()

У процедурі MixColumns, чотири байти кожної колонки State змішуються, використовуючи для цього оборотну лінійну трансформацію. MixColumns обробляє стану по колонках, трактуючи кожну з них як поліном четвертого ступеня. Над цими поліномами проводиться множення в $GF(2^8)$ по модулю x^4+1 на фіксований многочлен $c(x)=3x^3+x^2+x+2$. Разом з ShiftRows, MixColumns вносить дифузію в шифр.

2. ShiftRows -перестановочний цикл, в якому кожен рядок стану циклічно перемішується за певну кількість кроків.

3. MixColumns -операція перемішування, яка оперує стовпцями стану, складаючи 4 біта в кожному стовпці.

4. AddRoundKey.

4. Final Round (без MixColumns).

1. SubBytes(рисунок 1).

2. ShiftRows(рисунок 3).

3. AddRoundKey.

На початку шифрування input копіюється в масив State за правилом

$$s[r,c]=in[r+4c],$$

для $0 \le r < 4$ і $0 \le c < Nb$.

Після цього до State застосовується процедура AddRoundKey () і потім State проходить через процедуру трансформації (раунд) 10, 12, або 14 (в залежності від довжини ключа), при цьому треба врахувати, що останній раунд дещо відрізняється від попередніх. У підсумку, після завершення останнього раунду трансформації, State копіюється в output за правилом: $out[r+4c]=s[r,c]$, для i.

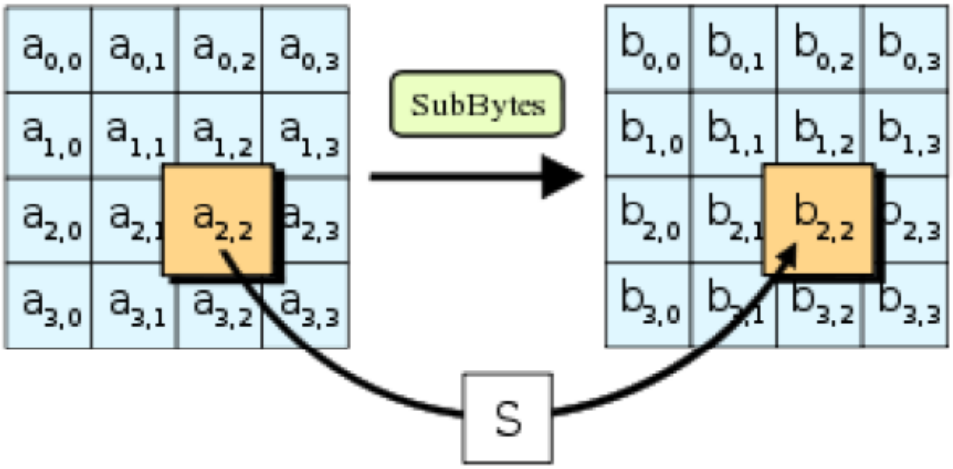

Рисунок 1 - SubBytes ()

Процедура SubBytes () обробляє кожен байт стану, незалежно виробляючи нелінійну заміну байтів використовуючи таблицю замін (S-box). Така операція забезпечує нелінійність алгоритму шифрування. Побудова S-box складається з двох кроків.

По-перше, проводиться взяття зворотного числа в полі Галуа $GF(2^8)$.

По-друге, до кожного байту b з яких складається S-box (рисунок 2) застосовується наступна операція:

$$b_i' = b_i \oplus b_{(i+4) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+6) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus c_i ,$$

де $0 \le i < 8$, і де b_i є і-ий біт b, а c_i - і-ий біт константи $c=63_{16}=99_{10}=01100011_2$.

Таким чином, забезпечується захист від атак, заснованих на простих алгебраїчних властивостях.

У процедурі AddRoundKey, RoundKey (рисунок 5) кожного раунду об'єднується з State. Для кожного раунду Roundkey виходить з CipherKey використовуючи процедуру

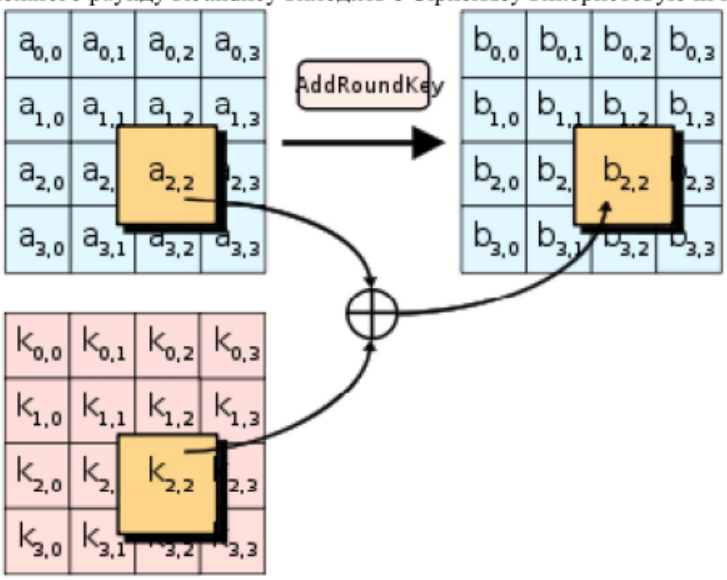

Рисунок 5 - AddRoundKey ()

KeyExpansion; кожен RoundKey такого ж розміру, що і State. Процедура виробляє побітовий XOR кожного байта State з кожним байтом RoundKey.

2. Крипостійкість і мінуси AES

У червні 2003 року Агентство національної безпеки США постановив, що шифр AES є досить надійним, щоб використовувати його для захисту відомостей, що становлять державну таємницю (англ. Classified information). Аж до рівня SECRET було дозволено використовувати ключі довжиною 128 біт, для рівня TOP SECRET були потрібні ключі довжиною 192 і 256 біт.

У порівнянні з DES, AES має більш криптостійкий ключ (128 - 256 біт проти 56 у DES). Є побоювання з приводу злому XSL атаками, але можливість таких атак поки не доведена (але і не спростована). В цілому, поки AES залишається стандартом де-факто і добре зарекомендував себе як алгоритм шифрування. +AES, при всій своїй алгебраїчної простоті, дуже потужний алгоритм шифрування, який поки не вдалося зламати. У порівнянні з DES, AES має всі шанси на довге і тривале життя на вотчині дядечка Сема. А забезпечення конфіденційності даних було і залишається однією з найбільш пріоритетних завдань сучасності.

Висновок. Проведені дослідження дозволяють зробити висновок, що використання алгоритму AES залишається актуальною задачею незважаючи на постійно зростаючі вимоги до криптостійкості сучасних алгоритмів шифрування.

Перелік використаних джерел.

1. Панасенко С. Интересные алгоритмы шифрования, часть 2. // BYTE/Россия. — 2006 — № 5 — с. 74-79.

2. Панасенко С.П., Батура В.П. Основы криптографии для экономистов: учебное пособие. Под ред. Л.Г. Гагариной. — М.: Финансы и статистика, 2005 — 176 с.


АВТОМАТИЗАЦІЯ
ТА КОМП'ЮТЕРНО-ІНТЕГРОВАНІ
ТЕХНОЛОГІЇ
проблемно-наукова міжгалузева
конференція молодих науковців
аспірантів та студентів
м. Тернопіль
2022

ЗАХІДНОУКРАЇНСЬКИЙНАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ПРИКАРПАТСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТЕФАНИКА
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ВОДНОГО ГОСПОДАРСТВА ТА ПРИРОДОКОРИСТУВАННЯ
НАЦІОНАЛЬНИЙ ТРАНСПОРТНИЙ УНІВЕРСИТЕТ
НАДВІРНЯНСЬКИЙ КОЛЕДЖ НТУ
ГАЛИЦЬКИЙ КОЛЕДЖ ІМ. В. ЧОРНОВОЛА

Проблемно-наукова міжгалузева конференція

АВТОМАТИЗАЦІЯ ТА КОМП'ЮТЕРНО-ІНТЕГРОВАНІ ТЕХНОЛОГІЇ

(АКІТ – 2022)

21—23 лютого 2022 року

Тернопіль

БЕЗПЕКА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

УДК 004.56.5(043.2)

Олійник Н.П.[1]

[1]*Західноукраїнський національний економічний університет*

ВИКОРИСТАННЯ СИМЕТРОЧНОГО ШИФРУ AES З РЕАЛІЗАЦІЄЮ НА JAVASCRIPT

Вступ. З розвитком і постійним розповсюдженням комп'ютерних телекомунікацій в різних сферах людської діяльності, все більш гостро постає питання безпечного обміну даними через незахищені канали передачі. Одним із ефективних засобів захисту інформації є криптографічні перетворення. У 2001 році, після кількох років відкритого обговорення, американський Національний інститут стандартів і технологій затвердив новий розширений стандарт шифрування AES (Advanced Encryption Standard), що прийшов на зміну DES (Data Encryption Standard), який більше 20-ти років був дефакто загальносвітовим стандартом шифрування [1]. Мета даного проекту полягає у розробці алгоритму шифрування AES на базі мови програмування JavaScript.

Мета: реалізація алгоритму шифрування AES на базі мови програмування JavaScript.

1. Довжина блоку шифрування AES

Алгоритм AES перетворює блок довжиною 128 бітів на інший блок тієї ж довжини. Для перетворення застосовується розклад ключів w, що отримується з ключа. 128-бітний блок AES представляється у вигляді матриці 4×Nb. Стандарт допускає лише одне значення Nb = 4, тому довжина блоку завжди 128 біт, хоча алгоритм може працювати з будь-яким Nb. Довжина ключа дорівнює 4Nk байт.

Таблиця 1 - Стандарти шифрування AES

	N_k	N_b	N_r
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

Алгоритм шифрування блоку складається з Nr раундів - застосування однієї і тієї ж групи перетворень до 128-бітного блоку даних. Стандарт допускає такі комбінації цих трьох параметрів.

2. Як використовувати AES

Цей алгоритм перетворює один 128-бітний блок в інший, використовуючи секретний ключ, який потрібен для такого перетворення. Для розшифровки отриманого 128-бітного блоку використовують друге перетворення з тим самим секретним ключем.

Виглядає це так, як показано на рисунку 1.

```
export const cipher = encrypt(block, key) // шифруємо block за допомогою key
export const block = decrypt(cipher, key) // розшифровуємо cipher за допомогою key
```

Рисунок 1 – Перетворення використані в JavaScript

Розмір блоку завжди дорівнює 128 біт. Розмір ключа має також фіксований розмір. Щоб зашифрувати довільний текст будь-яким паролем можна зробити так:

1. Отримати хеш від пароля.
2. Перетворити хеш на ключ за правилами описаними в стандарті AES.
3. Розбити текст на блоки по 128 біт.
4. Зашифрувати кожен блок функцією cipher.

Це можна записати так, як показано на рисунку 2.

```
export const hash = md5(password) // MD5 хеш має довжину 128 бит
key = keyexpansion(hash) // перетворюємо хеш в ключ
blocks = split(text, 16) // розбити текст на блоки по 16 байт

for (i = 0; i < blocks.length; i++)
cipher[i] = encrypt(blocks[i], key)
```

Рисунок 2 – Шифрування блоків в JavaScript

Щоб розшифрувати масив блоків cipher, потрібно застосувати до кожного блоку decrypt:

```
hash = md5(password)
key = keyexpansion(hash)

for (i = 0; i < cipher.length; i++)
blocks[i] = decrypt(cipher[i], key)

text = merge(blocks) // з'єднати всі блоки в один рядок

export default text;
```

Рисунок 3 – Розшифрування блоків в JavaScript

Звичайно, довжина тексту може бути не кратна 128 біт. У таких випадках можна доповнити текст нулями до потрібної довжини, а до зашифрованих даних додати кілька байт із зашифрованим розміром оригінального тексту. Функції aes.encrypt та aes.decrypt у файлі aes.js у прикладі використовують цей підхід.

3. Шифрування блоку данних

Для шифрування тексту AES застосовує не пароль або хеш від пароля, а так званий «розклад ключів», що отримується з ключа. Цей розклад можна як Nr + 1 матриць розміру 4×Nb. Алгоритм шифрування робить Nr + 1 кроків і кожному кроці він, крім інших дій, бере одну матрицю 4×Nb з «розкладу» і поелементно додає її до блоку даних.

Алгоритм шифрування отримує на вхід 128-бітний блок даних input і розклад ключів w, що виходить після KeyExpansion. 16-байт input він записує у вигляді матриці s розміру 4×Nb, яка називається станом AES, і потім Nr разів застосовує до цієї матриці 4 перетворення. Наприкінці він записує матрицю у вигляді масиву та подає його на вихід – це зашифрований блок. Кожне із чотирьох перетворень дуже просте.

- AddRoundKey бере з розкладу ключів одну матрицю розміру 4×Nb та поелементно додає її до матриці стану. Якщо двічі застосувати AddRoundKey, нічого не зміниться, тому перетворення зворотне до AddRoundKey це воно саме.
- SubBytes замінює кожен елемент матриці стану відповідним елементом таблиці SBox: sij = SBox [sij]. Перетворення SubBytes оборотне. Назад до нього знаходиться за допомогою таблиці InvSBox.
- ShiftRows зрушує i-ий рядок матриці s на і позицій вліво, рахуючи і з нуля. Зворотне перетворення InvShiftRows переміщує рядки вправо.
- MixColumns множить кожен стовпець матриці s ліворуч на особливу матрицю розміру 4×4.

Код шифрування можна зобразити так, як показано на рисунку 4.

```
4
5   const encryption = () => {
6     AddRoundKey(0)
7
8     for (var i = 1; i <= Nr - 1; i++) {
9       SubBytes()
10      ShiftRows()
11      MixColumns([0x02, 003, 0x01, 0x01])
12      AddRoundKey(i)
13    }
14
15    SubBytes()
16    ShiftRows()
17    AddRoundKey(Nr)
18  }
19
20  export default encryption;
```

Рисунок 4 – Процес шифрування

Для шифрування використовують [a b c d] = [{02} {03} {01} {01}]. Можна перевірити, що перетворення зворотне до MixColumns[{02}{03}{01}{01}] це MixColumns[{0e}{0b}{0d}{09}]. Для розшифрування необхідно застосувати зворотні перетворення у зворотньому порядку.

Висновки. В данній роботі було відображено використання симетричного шифрування AES на базі коду JavaScript

Перелік використаних джерел:

1. Bruce Schneier - Practical Cryptography: Designing and Implementing Secure Cryptographic Systems, 2003, 231 с.

2. S.V. Belim, A.O. Mayorov-Zilbernagel. Algorithm for Searching the Broken Pixels and Eliminating Impulse Noise in Images Using a Method of Association Rules. Science and Education of the Bauman MSTU, 12:716± 737, 2014. URL: http://technomag.bmstu.ru/doc/744983.html.