Міністерство освіти і науки України

Харківський національний університет імені В.Н. Каразіна

Навчально-наукового інституту комп'ютерних наук та штучного інтелекту

Спеціальність 125 «Кібербезпека»

Освітня програма «Безпека інформаційних та комунікаційних систем»

«Допущено до захисту»

В.о. зав. кафедрою КІСМ

Марина ЄСІНА

_____

« » 2024 р.

Пояснювальна записка

до кваліфікаційної роботи магістра

на тему: «Розробка програмного забезпечення для порівняння ефективності популярних криптографічних алгоритмів шифрування даних за швидкістю, стійкістю до атак та використанням ресурсів»

оцінка «_____»

Голова ЕК

Олександр ЛЕМЕШКО

Керівник: к. т. н., доцент

Нарєжній Олексій Павлович

Рецензент: к. т. н., ст. викл.

Лисицький К. Є

Виконавець: студент групи КБ-61

Тендіт Максим Євгенович

Харків – 2024

РЕФЕРАТ

Пояснювальна записка містить 76 сторінок, 29 рисунків, 3 таблиці, 2 додатки, 14 джерел.

Актуальність теми. У дипломній роботі розроблена модель веб-сервіс, що дозволяє зрозуміти як працюють одні з найпопулярніших алгоритмів шифрування даних, порівняти швидкість їх роботу, необхідні ресурси та захищеність від атак. Для цього було розроблено веб-додаток, котрий має в собі реалізовані функції. Розроблена та впроваджена система зберігання та порівняння даних результатів роботи алгоритмів, механізм експериментальної спроби атаки на алгоритми та алгоритми. Крім того, створено інформаційні елементи котрі допоможуть зрозуміти, навіть не пізнаному в технологіях та методах кібербезпеки користувачу, як працюють алгоритми та де їх використовують.

Об'єктом розробки є веб-сервіс шифрування та дешифрування повідомлень за допомогою симетричних алгоритмів блочного шифрування, з функціями зберігання та порівняння отриманих даних та можливість спроби атаки на зашифрований текст.

Предметом розробки є сервіс для симетричного блочного шифрування тексту з функціями зберігання та аналізу даних, а також можливістю проведення атак на зашифрований текст

Основною проблемою, що вирішується, є забезпечення обізнаності користувачів в алгоритмах шифрування даних та їх особливостях. Шифрування даних має важливе значення для безпеки користувачів всесвітньої мережі інтернет та інших мереж, і їх використання повинно бути забезпечено високим рівнем обізнаності для забезпечення більшої безпеки населення.

Метою проекту є створення ефективного та надійного програмного забезпечення для порівняння ефективності популярних криптографічних алгоритмів шифрування даних за швидкістю, стійкістю до атак та використанням ресурсів, що дозволить користувачам зрозуміти більше про шифрування даних або обрати оптимальний алгоритм для забезпечення безпеки та продуктивності систем обробки даних.

Основними завданнями, на вирішення яких спрямовано проект, є підвищення інформованості користувачів щодо алгоритмів шифрування даних, їх продуктивності та захищеності від атак, що може покращити кібербезпеку в суспільстві та надасть розуміння про складність та різницю в, на перший погляд схожиї, алгоритмах.

Значимість проекту полягає в тому, що він може допомогти у підвищенні рівня кібербезпеки населення, які використовують мережі зв’язку, такі як всесвітня мережа інтернет та в їх обізнанності, що є важливою складовою в безпеці багатьох систем, зокрема фінансових та військових. Проект також може мати значення для розвитку науки, зокрема кібербезпеки, та навчання студентів у цій галузі.

Методи дослідження. Під час дослідження були використані такі методи: аналіз сучасних алгоритмів, аналіз існуючих загроз, порівняння методів захисту та потужності алгоритмів в умовах використання за звичайному персональному комп’ютері, аналіз методологій розробки web-ресурсів.

Практична цінність. Дослідження може бути використане під час ознайомлення студентів та інших людей з шифруванням даних, в першу чергу веб-сервіс направлений на малообізнані групи населення такі, як студенти початкових курсів, школярі та люди без технічної освіти. Запропоновано та реалізовано оптимізований список алгоритмів шифрування одного виду, щоб мати базу для порівняння їх між собою. Дані можуть використовуватися в навчальному процесі під час вивчення подібних тем.

Апробація роботи. Основні положення на результати роботи були представлені у статті на порталі ResearchGate, та мають УДК 004.056.55.

А також представлено на XLIX International scientific and practical conference, New Areas of Scientific Research: Exploring New Frontiers.

Ключові слова: КРИПТОГРАФІЯ, АЛГОРИТМИ ШИФРУВАННЯ, ВЕБ-ДОДАТОК, БЛОКОВІ ШИФРИ, СУЧАСНА КІБЕРБЕЗПЕКА, НАВЧАЛЬНИЙ ІНСТРУМЕНТ, ШВИДКОДІЯ АЛГОРИТМІВ, ЗАХИСТ ДАНИХ, ІНФОРМАЦІЙНА БЕЗПЕКА.

ABSTRACT

The explanatory note contains 76 pages, 29 figures, 3 tables, 2 appendices, 14 sources.

Relevance of the topic. In this thesis, we have developed a web service model that allows you to understand how some of the most popular data encryption algorithms work, compare their speed, required resources, and protection against attacks. For this purpose, we developed a web application that has the implemented functions. A system for storing and comparing data on the results of algorithms, a mechanism for an experimental attempt to attack algorithms and algorithms was developed and implemented. In addition, information elements have been created to help a user who is not familiar with cybersecurity technologies and methods understand how algorithms work and where they are used.

The object of development is a web service for encrypting and decrypting messages using symmetric block encryption algorithms, with the functions of storing and comparing the received data and the possibility of attempting to attack the encrypted text.

The subject of development is a service for symmetric block encryption of text with data storage and analysis functions, as well as the ability to conduct attacks on encrypted text.

The main problem to be solved is to ensure that users are aware of data encryption algorithms and their features. Data encryption is essential to the security of users of the World Wide Web and other networks, and its use must be supported by a high level of awareness to ensure greater public safety.The goal of the work is to create an effective and reliable method of protecting the quantum generator web service from cyber threats, which will ensure the security and confidentiality of random numbers generated by the service.

The goal of the project is to create efficient and reliable software for comparing the effectiveness of popular cryptographic data encryption algorithms in terms of speed, attack resistance, and resource usage, which will allow users to understand more about data encryption or choose the best algorithm to ensure the security and performance of data processing systems.

The main objectives of the project are to raise awareness of data encryption algorithms, their performance and protection against attacks, which can improve cybersecurity in society and provide an understanding of the complexity and difference in seemingly similar algorithms.

The significance of the project lies in the fact that it can help to increase the level of cybersecurity of the population using communication networks such as the World Wide Web and their awareness, which is an important component in the security of many systems, including financial and military ones. The project may also be important for the development of science, in particular cybersecurity, and the education of students in this field.

Research methods. The following methods were used in the study: analysis of modern algorithms, analysis of existing threats, comparison of protection methods and algorithm power in terms of use on a regular personal computer, analysis of web resource development methodologies.Approbation of work.

Practical value. The study can be used to familiarize students and other people with data encryption, primarily the web service is aimed at low-education groups such as primary school students, schoolchildren, and people without technical education. An optimized list of encryption algorithms of the same type has been proposed and implemented to provide a basis for comparing them with each other. The data can be used in the educational process when studying similar topics.

Keywords: CRYPTOGRAPHY, ENCRYPTION ALGORITHMS, WEB APPLICATION, BLOCK CIPHERS, MODERN CYBERSECURITY,

EDUCATIONAL TOOL, ALGORITHM PERFORMANCE, DATA PROTECTION, INFORMATION SECURITY.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

AES	Advanced Encryption Standard
3DES	Triple Data Encryption Standard
RC6	Rivest Cipher 6
IDEA	International Data Encryption Algorithm
CPU	Central Processing Unit
RAM	Random Access Memory
E2EE	End-to-End Encryption
UI	User Interface
API	Application Programming Interface
IDE	Integrated Development Environment
AES-NI	AES New Instructions
NIST	National Institute of Standards and Technology
SHA	Secure Hash Algorithm
HTTPS	HyperText Transfer Protocol Secure
IoT	Internet of Things
MC	Мілісекунди

ВСТУП

У сучасному цифровому світі, де обсяг та цінність інформації постійно зростають, питання захисту даних стають критично важливими для всіх сфер суспільства. Сьогодні кожна державна установа, комерційна організація та навіть приватний користувач потребують надійних інструментів для захисту своїх даних від несанкціонованого доступу, кібератак та порушення конфіденційності. Важливу роль у цьому процесі відіграють криптографічні алгоритми шифрування, які забезпечують конфіденційність, цілісність та доступність інформації, перешкоджаючи її витоку та несанкціонованому використанню. Водночас ефективність алгоритмів шифрування залежить не лише від надійності захисту, але й від продуктивності, стійкості до атак та оптимального використання ресурсів. Тому порівняння ефективності різних алгоритмів є важливим завданням у сфері інформаційної безпеки.

Зокрема, популярні симетричні блочні алгоритми шифрування, які використовуються для захисту даних в Інтернеті, банківських системах та захищених мережах, стають предметом численних досліджень щодо їх стійкості до атак, швидкості виконання та вимог до ресурсів. Оскільки ці алгоритми активно застосовуються в реальних умовах, їх глибоке вивчення дозволяє зробити обґрунтований вибір найбільш відповідного алгоритму для конкретних задач, що має велике значення для підвищення загальної кібербезпеки. Однак користувачам та студентам без технічної підготовки часто важко зрозуміти, як функціонують ці алгоритми, та в чому полягають особливості різних підходів до шифрування. Саме тому виникає потреба у створенні зручних інструментів, які сприятимуть не лише захисту даних, але й підвищенню обізнаності користувачів у сфері криптографії.

У цій дипломній роботі розроблено та впроваджено веб-сервіс, який дозволяє користувачам шифрувати та дешифрувати повідомлення з використанням популярних симетричних алгоритмів блочного шифрування, зберігати результати їх роботи, порівнювати ефективність алгоритмів та навіть здійснювати спроби атак на зашифрований текст. Крім того, сервіс включає інформаційні матеріали, що пояснюють основні принципи роботи цих алгоритмів та їх значення для забезпечення безпеки інформації. Важливою метою роботи є надання користувачам, особливо тим, які не мають технічної освіти, зрозумілої інформації про шифрування, що дозволить їм розуміти важливість захисту своїх даних та обирати оптимальні методи шифрування для власних потреб.

У процесі роботи використовувались сучасні методи дослідження алгоритмів шифрування, аналізу загроз та порівняння ефективності шифрування. Впроваджена модель веб-сервісу також враховує методи кіберзахисту та дозволяє здійснити експериментальну спробу атаки, що може бути корисною як в освітніх цілях, так і для підвищення практичної обізнаності користувачів у сфері кібербезпеки. Крім того, такий сервіс має потенціал стати важливим інструментом у навчальному процесі з кібербезпеки, особливо для студентів початкових курсів та інших категорій користувачів, які лише починають своє ознайомлення з криптографією.

1. ТЕОРЕТИЧНІ АСПЕКТИ КРИПТОГРАФІЧНИХ АЛГОРИТМІВ

Захист інформації це один із ключових аспектів безпеки у сучасному світі. Криптографія відіграє роль у забезпеченні конфіденційності, цілісності та автентичності даних. Завдяки принципам математики та складним алгоритмам вона дає можливість перетворювати інформацію в вид, що є незрозумілим для сторонніх осіб і може бути відновлений лише за допомогою спеціальних секретних ключів.

Сучасні криптографічні алгоритми підрозділяються на два основні класи: симетричний і асиметричний. Симетричні алгоритми, що включають AES і DES, використовують один ключ для шифрування і дешифрування даних, що забезпечує високу швидкість кодування. Асиметричні алгоритми, такі як RSA і ECC, використовують пару ключів (відкритий і закритий) для захисту інформації, забезпечуючи при цьому більшу безпеку, але менш ефективну швидкість у порівнянні з симетричними методами.

Криптографічні алгоритми - це основні існуючі технології, що використовуються в різних поняттях - від безпечного дзвінка в Інтернеті до збереження банківських транзакцій і систем електронного голосування. Тут розглянуто основні теоретичні аспекти криптографічних алгоритмів, що включають їхню історію, основні принципи, види алгоритмів і моделі завантажень.

1.1 Історія розвитку криптографії

Історія криптографії починається тисячоліття тому з простих методів шифрування, що використовувалися для передачі секретних повідомлень, наприклад одним із найвідоміших прикладів є шифр Цезаря, де кожна буква тексту зміщувалася на певну кількість позицій, хоча цей метод був ефективним в

той час, з часом технологічного прогресу він легко піддається дешифруванню сучасними методами.

Під час розвитку комп'ютерів криптографія отримала значний поштовх, і з'явилися алгоритми засновані на складних математичних обчисленнях що роками покращує та покращує рівень безпеки даних. У сімдесятих роках двадцятого століття було розроблено алгоритми DES та RSA, які поклали фундамент сучасної криптографії. На сьогодні алгоритми стали важливим напрямком наукових досліджень, а кожен рік з'являються нові алгоритми, що забезпечують ще вищий рівень захисту інформації, серед котрих на окрему увагу претендують симетричні алгоритми блочного шифрування.

1.2 Основи симетричних алгоритмів шифрування

Симетричні алгоритми можна вважати основним методом захисту інформації в сучасній криптографії так, як їх сутність полягає в тому, що для шифрування та дешифрування використовується один і той самий ключ, який має бути відомий відправнику та одержувачу, тобто обом сторонам, котрі обмінюються даними. Цей тип шифрування характеризується своєю високою швидкістю та ефективністю, включно при роботі з великими обсягами даних, що робить його популярним вибором для захисту інформації в реальному часі, наприклад, в мережах для передачі даних, системах зберігання та накопичення інформації тощо.

Всі симетричні алгоритми шифрування можна поділити на два основних типи: потокові шифри та блочні. Основна різниця між ними полягає в тому, що потокові шифри обробляють дані по одному біту або байту за раз, що дає змогу мати високу швидкість та низьку затримку. В той же час, блочні шифри, навпаки, працюють з фіксованими блоками даних, наприклад це може бути 64, 128 або інша кількість біт, що залежить від алгоритму, і забезпечує вищу стійкість до атак через те, що дані обробляються великими ділянками.

Симетричні алгоритми шифрування мають значні переваги в порівнянні з асиметричними алгоритмами, в першу чергу це швидкість шифрування, завдяки не складній архітектурі ці алгоритми зазвичай працюють швидше, ніж асиметричні, що робить їх ідеальними для обробки великої кількості даних. Також до переваг можна віднести низьке споживання ресурсів, що є важливим для пристроїв з обмеженими ресурсами або великим потоком даних, наприклад мобільних телефонів, або серверів онлайн сервісів. Також значною перевагою є простота реалізації, це дозволяє відносно легко впроваджувати алгоритми в програмне або апаратне забезпечення, а також полегшує інтеграцію в мікропроцесорних системах.

Проте потрібно пам’ятати й про недоліки симетричного шифрування, одна з найбільших проблем це необхідність безпечно передачі ключів між сторонами. Тому що якщо ключ буде перехоплений зловмисником, він зможе отримати доступ до зашифрованої інформації, що призведе до критичної ситуації безпеки переданих даних. З метою захисту процесу передачі ключів також постійно знаходяться рішення, наприклад одне з них це протоколи управління ключами, які забезпечують надійну передачу ключів.

До алгоритмів симетричного шифрування відносять такі алгоритми, як AES, що розшифровується як Advanced Encryption Standard, 3DES - Triple Data Encryption Standard, Blowfish, Twofish, IDEA(International Data Encryption Algorithm), Serpent, Camellia, RC6 і ChaCha20. Перечислені вище алгоритми відносяться до блочних алгоритмів симетричного шифрування, кожен з цих алгоритмів має свої унікальні характеристики, які впливають на продуктивність, стійкість до атак та ефективність використання ресурсів. Саме ці алгоритми ми будемо розглядати в моїй дипломній роботі.

Виходячи з наведеної інформації, симетричні алгоритми шифрування залишаються ключовими компонентами сучасної криптографії та є основою багатьох систем, включаючи банківську, захист комунікації та зберігання даних.

1.3 Огляд криптографічних алгоритмів симетричного блочного шифрування

Сучасний світ використовує великий список алгоритмів для захисту даних. Кожен алгоритм має особливості, що впливають на ефективність, стійкість до атак та їх призначення. У цьому огляді розглянемо основні алгоритми симетричного блочного шифрування, які я використовую в роботі, а саме: AES, 3DES, Blowfish, Twofish, IDEA, Serpent, Camellia, RC6 та ChaCha20. Кожен з наведених алгоритмів має унікальну структуру та призначення, це дозволяє застосовувати їх в різних сферах.

Наприклад, AES є стандартом симетричного шифрування блочного шифрування, що затверджений Національним інститутом стандартів і технологій США(NIST). Цей стандарт підтримує довжини ключів 128, 192 та 256 бітів та забезпечує високий рівень безпеки при високій швидкості шифрування. Ефективність та надійність робить AES одним із найпопулярніших алгоритмів захисту даних по цей день.

Говорячи про алгоримт 3DES, він є розширенням алгоритму DES, яке підвищує його безпеку шляхом послідовного застосування трьох ключів для шифрування даних. Хоча 3DES й є більш стійким до атак, ніж оригінальний DES, він поступається у швидкості та ефективності, що обмежує його використання.

Blowfish є блочним алгоритмом шифрування, що має довжину блоку 64 біт та змінний ключ від 32 до 448 бітів. Алгоритм має високу швидкість та ефективність, особливо це стосується апаратних реалізацій. Проте довжина блоку в 64 біти є проблемою для його ефективності, бо перевага надається алгоритмам що містять блоки від 128 бітів.

Для рятування сімейства алгоритмів fish був винайдений його наступник з назвою Twofish, що має довжину блоку в 128 бітів та можливість використання ключів розміром до 256 бітів. Цей алгоритм також відомий своєю високою швидкість та ефективністю, та був фіналістом конкурсу на новий стандарт AES,

що свідчить про його високі результати. Також цей алгоритм часто використовують в застосунках з відкритим кодом.

International Data Encryption Algorithm, або скорочено IDEA також відноситься до сімейства блочних алгоритмів симетричного шифрування та має фіксовану довжину блоку 64 біт та ключем розміром 128біт. Цей шифр спеціально розроблений для забезпечення високого рівня безпеки та має високу стійкість до широкого спектру атак.

Serpent є блочним шифром із довжиною блоку 128 бітів та ключем, який може бути до 256 бітів. Алгоритм також був фіналістом конкурсу AES і забезпечує дуже високий рівень безпеки. Основною перевагою є його стійкість до різних видів атак, але все ж він поступається AES за швидкістю шифрування.

Camellia є схожим за характеристиками на AES, з довжинами ключів 128, 192 та 256 бітів. Його часто використовують у банківському секторі та в різних комерційних додатках, оскільки він відповідає сучасним стандартам безпеки та ефективності.

RC6 також є блочним шифром, розробленим на основі алгоритму RC5, із довжиною блоку 128 бітів та змінною довжиною ключа до 2048 бітів. RC6 був кандидатом на стандарт AES і є досить швидким та ефективним у програмних реалізаціях, хоча зрештою не був обраний як стандарт.

ChaCha20 є потоковим шифром, який забезпечує високий рівень безпеки та стійкість до різних видів атак. Його відмінною рисою є висока швидкість, особливо на мобільних пристроях, і це робить його популярним у сучасних протоколах безпеки, таких як Transport Layer Security, скорочено TLS.

Всі ці алгоритми використовуються в дуже різних цілях, але їх об’єднує те, що всі вони симетричні алгоритми шифрування блочного типу.

1.4 Критерії ефективності та стійкості криптографічних алгоритмів

Одними з ключових факторів при виборі криптографічного алгоритму є ефективність та швидкість шифрування даних. Ефективність визначає

швидкодію та використання ресурсів алгоритмом, а стійкість відповідає за рівень захисту від атак. Зазвичай ці показники вимірюються за допомогою певних показників, до котрих відносяться швидкість шифрування та дешифрування даних, використання ресурсів, довжина ключа, стійкість до атак, прозорість та простота реалізації алгоритму, гнучкість, сумісність та стандартизація, а також майбутня стійкість до квантових обчислень.

Швидкість алгоритму важлива для обробки великих обсягів даних, особливо в системах з високою пропускною здатністю. Чим швидше алгоритм може виконувати операції шифрування і дешифрування, тим вище його ефективність в реальному часі. Ресурсоємність відображає потребу алгоритму в обчислювальних ресурсах, таких як оперативна пам'ять і центральний процесор. Алгоритми з меншим споживанням ресурсів більше підходять для пристроїв з обмеженими ресурсами, таких як мобільні або вбудовані системи.

Довжина ключа - важливий фактор, який впливає на стійкість алгоритму до атак грубої сили. Чим більша довжина ключа, тим більше обчислювальних ресурсів потрібно для його розкриття. У той же час алгоритм повинен бути стійким до криптоаналітичних атак, включаючи диференціальний і лінійний криптоаналіз. Це гарантує, що алгоритм здатен захистити дані навіть від складних атак.

Прозорість алгоритму має вирішальне значення для забезпечення високої ентропії зашифрованого тексту, щоб зменшити ймовірність виявлення шаблонів у зашифрованих даних. Також важливо, щоб алгоритм був гнучким і масштабованим для використання в різних середовищах, що дозволяє адаптувати його до конкретних вимог безпеки. Простота реалізації знижує ймовірність помилок під час реалізації алгоритму, що важливо для захисту від потенційних вразливостей.

Сумісність алгоритму зі стандартами криптографії дозволяє широко використовувати його в комерційній та державній сферах, а також забезпечує

його інтеграцію з іншими системами. Нарешті, з огляду на розвиток квантових обчислень, варто задуматися про майбутню стійкість алгоритму до квантових атак. Симетричні алгоритми, як правило, більш стійкі до таких загроз, хоча для забезпечення їх безпеки в умовах квантових обчислень може знадобитися збільшення довжини ключа.

Таким чином, оцінка криптографічних алгоритмів за цими критеріями дозволяє вибрати найбільш підходящий алгоритм для конкретних завдань з урахуванням вимог безпеки, продуктивності та адаптивності.

2. АНАЛІЗ АЛГОРИТМІВ ШИФРУВАННЯ

Аналіз алгоритмів шифрування є дуже важливим етапом для визначення оптимального рішення для забезпечення високої безпеки даних та ефективності в певній ситуації та системі. Кожен із описаних в даній роботі алгоритм має унікальні характеристики, що роблять його придатним для первих сфер застосування в залежності від вимог поставлених перед ним.

2.1 Аналіз алгоритму AES (Advanced Encryption Standard)

Як було сказано в попередніх пунктах дипломної роботи, AES або Advanced Encryption Standard, що українською перекладається як «Розширений алгоритм шифрування» є симетричним блоковим шифром, затвердженим Національним інститутом стандартів і технологій США, що скорочено має назву NIST у 2001 році. Цей алгоритм розроблений для забезпечення високого рівня безпеки при шифруванні даних і використовується в багатьох державних та комерційних сервісах та додатках.

AES працює з блоками даних довжиною 128 біт що є 16 байтами і підтримує ключи 3 довжин, а саме 128, 192, 256 біт. Алгоритм заснований на структурі, котра відома як SPN, що має розшифровку substitution-permutation network та складається з серії раундів обробки, кількість котрих визначається довжиною ключа:

- 10 раундів для 128 бітного ключа
- 12 раундів для 192 бітного ключа
- 14 раундів для 256 бітного ключа

Шифрування AES складається з кількох ключових етапів, які можна формалізувати наступним чином:

Підготовка ключа: Генерація раундових ключів із первинного ключа K за допомогою алгоритму розширення ключа. Цей етап забезпечує унікальність ключів для кожного раунду, що істотно підвищує безпеку шифрування.

$$K_i = KeyExpansion(K, i)$$

де K_i — ключ раунду i, K — первинний ключ, а i — індекс раунду.

Ініціалізація: Блоки даних, що підлягають шифруванню, розбиваються на матрицю S розміром 4×4:

$$S = \begin{matrix} S[0.0] & S[0,1] & S[0,2] & S[0,3] \\ S[1.0] & S[1,2] & S[1,2] & S[1,3] \\ S[2,0[& S[2,1] & S[2,2] & S[2,3] \\ S[3,0] & S[3,1] & S[3,2] & S[3,3] \end{matrix}$$

де S[i,j] — елемент матриці в рядку i та стовпці j) розміром 4×4:

Раунди: На кожному раунді виконуються чотири основні операції:

SubBytes: Цей етап включає заміну кожного байта блоку на відповідний байт з таблиці підстановки (S-Box). Таблиця підстановки була спеціально розроблена для забезпечення високої стійкості до атак. Для кожного байта S[i,j]S[i, j]S[i,j] застосовується:

$$S[i, j] = S_s(S[i, j])$$

де S_s — таблиця підстановки.

ShiftRows: На цьому етапі кожен рядок матриці зсувається вліво на певну кількість позицій. Точніше кажучи, перший рядок залишається без змін, другий зсувається на одну позицію вліво, третій — на дві, а четвертий — на три. Це допомагає розподілу байтів та стійкості алгоритму:

$$S[i, j] \leftarrow S[i, (j + i) mod 4]$$

MixColumns: Цей етап виконує змішування стовпців, яке здійснюється за формулою, де нові байти обчислюються як комбінація існуючих байтів:

$$\text{NewColums[i]} = \sum_{j=0}^{3} a_{ij} \cdot S[j]$$

де a_{ij} - це елементи матриці коефіцієнтів, що використовується для змішування, а S[j] — байти з стовпця j.

AddRoundKey: На цьому етапі ключ раунду стає істиною з даними за допомогою виключної диз'юнкції. Це забезпечує інтеграцію ключа в дані:

$$S[i,j] \leftarrow S[i,j] \oplus K_r$$

де K_r – це ключ раунду.

НА цьому моменті настає фінальний раунд, що завершується операціями SubButes, ShaftRows та AddRoundKey, але без MixColums. Цей етап є останнім, де використовується фінальний ключ шифрування та завершує процес шифрування.

Як вже було сказано раніше, AES забезпечує високу стійкість до різноманітних ата, в першу чергу від атак методом підбору, що також називають Брут-Форс та диференційних атак. НА сьогоднішній день не було жодного практично здійсненної компрометації AES і це робить його одним з найбільш надійних.

Цей алгоритм широко використовують в багатьох сферах, наприклад захист у комунікаційних системах, шифрування жорстких дисків, а також у програмних та апаратних продуктах. Популярність алгоритму AES пояснюється не лише безпекою, а також і швидкістю що робить його гарним кандидатом в умовах обмежених ресурсів.

2.2 Аналіз алгоритму 3DES (Triple Data Encryption Standard)

3DES (Triple Data Encryption Standard), як було сказано в попередніх матеріалах дипломної роботи, також є симетричним блоковим шифром та був розроблений як посилення оригінального алгоритму DES. Цей алгоритм був створений щоб забезпечити більшу безпеку шляхом збільшення довжини ключа та використання послідовного шифрування аж з трьома різними ключами. Основною метою розробки даного алгоритму вважається подолання вразливостей класичного алгоритму DES до атак методом перебору, іншими словами брут-форсу, які були можливі через проблему з короткою довжиною ключа, яка була всього 56 біт. Перевага 3DES вже зашифровані в його назву, а саме цифра три, що означає використання трьох DES операцій з ефективною довжиною ключа до 168 біт та це робить алгоритм набагато стійкішим до атак.

Так як цей алгоритм базується на структурі DES, котрий використовує блоки даних розміром всього 64 біта. Під час шифрування застосовуються три послідовні DES-операції, і дуже важливо, що кожна з них використовує окремий ключ. Цю багатократну операцію можна назвати шифрування-дешифрування-шифрування і вона дозволяє значно підвищити рівень захищеності даних.

В залежності від режиму, довжина ключа в 3DES може бути 112 або 168 біт. Режими алгоритму поділяються на три види, перший це режим з трьома ключами (K_1,K_2,K_3) – цей режим є найбільш захищеним та використовує 3 унікальних ключі одночасно. Наступним режимом є режим з двома ключами (K_1 = K_2, K_3), що є середнім варіантом за рівнем захисту та ефективності, тому що використовується на один ключ менше, а також є третій режим, що є алгоритмом з одним ключем, та фактично повертає нас до DES (K_1 = K_2 = K_3).

Говорячи про процес шифрування в 3DES, головне потрібно пам’ятати що він використовує 3 послідовні операції DES, де початковий блок даних P або plaintext, або відкритий текст піддається обробці по наступній системі:

Перше шифрування, це коли початковий текст P шифрується за допомогою використання першого ключа K_1:

$$C_1 = DES_{K1}(P)$$

де, C_1 – це результат першого шифрування, в DES_{K1} – позначення операції DES з використанням ключа K_1

Після того, як ми отримали результат першого шифрування, ми дешифруємо його з використанням ключа K_2:

$$C_2 = DES_{K2}^{-1}(C_1)$$

де C_2 – результат дешифрування, в результат операції дешифрування з використанням ключа K_2.

Після дешифрування результатів першого шуфрування ми повинні знову провести шифрування використовуючи третій ключ K_3:

$$C_3 = DES_{K3}(C_2)$$

де C_2 – кінцевий зашифрований текст, що часто ще має назву chipertext, отриманий в результаті операції шифрування на дешифрований текст першого шифрування за допомогою ключа K_3.

Таким чином, кінцевий результат дій алгоритма є зашифрований текст, зашифрованим блоком даних, що був отриманий в результаті дій застосування трьох послідовних операцій класичного алгоритму DES та використанням трьох ключів.

Говорячи про дешифрування в алгоритмі 3DES ми маємо дуже цікавий та логічний механізм, що є повною протилежністю алгоритму шифрування, тобто спочатку ми виконуємо дешифрування ключем K_3, потім шифрування ключем K_2, а після цього дешифрування ключем K_1. Якщо представити це в вигляді формули, то ми будемо мати наступне:

$$P = DES_{K2}^{-1}(DES_{K2}(DES_{K3}^{-1}(C)))$$

де P – це початковий відкритий текст, а C – зашифрований текст.

Як було сказано вище алгоритм 3DES є значно більш стійким до атак ніж його попередник DES завдяки переробленому механізму роботи з ключами. Але незважаючи на це, є певні теорії що такі атаки як атака зустрічних обробки, може значно знизити його ефективність, Тому через те, що технології швидко розвиваються та з'являються більш швидкі та потужні обчислювальні машини, 3DES поступово посувається більш ефективним алгоритмам, наприклад таким як AES.

Незважаючи на всі ці обмеження протокол довгий час був актуальний для банківських, фінансових систем, де є велика кількість чутливої інформації. Також він застосовується в багатьох протоколах таких, як TLS, IPsec та багатьох інших протоколах передачі даних.

Виходячи з наведеного аналізу алгоритму, незважаючи на покращення, поступово витісняється більш швидкими та безпечними алгоритмами. Але незважаючи на це його роль в історії та науці залишається надзвичайно важливим, оскільки він забезпечив перехідний рівень безпеки та став ґрунтом для розробки більш надійних алгоритмів.

2.3 Аналіз алгоритму Blowfish

Blowfish це алгоритм симетричного блочного шифрування, що використовує блоки довжиною в 64 біти і підтримає ключи змінної довжини, що можуть бути від 32 до 448 біт. Це забезпечує гнучкість алгоритму та можливість налаштування рівня безпеки. Ключовою перевагою даного алгоритму є його відносно висока швидкість обробки інформації, це дозволяє йому бути ефективним в середовищах, таких як вбудовані системи.

Основа алгоритму побудована на структурі Фейстеля, в котрій кожен блок даних розділяється на дві половини, після чого обробляються аж в 16 раундах. Кожен раунд спрямований на обчислення шифрування або дешифрування даних.

Якщо говорити про шифрування в Blowfish, то воно складається з певних XOR операцій, що є операціями виключної диз'юнкції та заміни даних за допомогою P-блоків та S-блоків. Цей процес складається з наступних етапів:

Розбиття даних на половини: 64 бітний блок початкових даних X розділяється на дві половини по 32 біта, такі як L – ліва полована, та R – права половина. Уявимо що L_0 та R_0 – початкові значення лівої та правої половини відповідно.

На початку шифрування ліва половина L_0 XOR-иться з першим елементом P-масиву P_1 , а права половина R_0 XOR-иться з другим елементом P_2.

Основний цикл шифрування в Blowfish, як було сказано вище, виконує 16 раундів над лівою та правою половинами даних, на кожному раунді виконується операція, котру можна представити наступною формулою:

$$f(L_i) = ((S_1[a] + S_2[b] mod\ 2^{32} \oplus S_3[c]) + S_4[d] mod 2^{32})$$

де S_1,S_2,S_3,S_4 – таблиці замін (S-блоки), а a,b,c,d – 8 бітні підблоки значення L_i.

Результат функції $f(L_i)$ XOR-иться з правою половиною R_i, після чого відбувається обмін $L \leftrightarrow R$.

Завершальним раун після завершення 16 раундів 16 раундів, де ліва та права частина знову XOR-иться з останніми елементами P-масиву, точніше права половіна з P_{17}, а ліва з P_{18}.

Після того, як всі раунди були виконані, права та ліва частина даних об'єднуються, утворюючи кінцевий зашифрований блок довжиною 64 біти, що називається C.

Якщо говорити про дешуфрування даних в алгоритмі Blowfish, то структура процесу є аналогічна процедурі шифрування, але P-блоки застосовуються в зворотному порядку. Такий метод дешифрування через те, що алгоритм базується на структурі Фейстеля та це можна вважати гарним плюсом,

бо дозволяє уникнути додаткових обчислень, що підвищує ефективність даного алгоритму.

Зазвичай цей алгоритм застосовують в комерційних, а також програмах з відкритим кодом, де не потрібно частих змін ключів. Хоча цей алгоритм і є швидким, але поступово він витісняється іншими, такими як AES.

2.4 Аналіз алгоритму Twofish

Twofish - це симетричний алгоритм із довжиною блоку 128 біт і підтримкою ключів довжиною 128, 192 або 256 біт. Завдяки своїй структурі Twofish здатний швидко працювати на сучасних процесорах і у вбудованих системах, де важлива ефективність шифрування. Алгоритм поєднує в собі низку криптографічних технік, зокрема шедулери ключів, великі таблиці підстановок (S-блоки) і модульні операції, що робить його стійким до криптоаналітичних атак.

Алгоритм, як і Blowfish , відноситься до сімейства алгоримів Фейстеля, однак має ускладнену структуру за допомогою використання додаткових операцій для підвищення безпеки.

Як і в Blowfish, алгоритм Twofish складається з 16 раундів, де кожен раунд використовує такі елементи: динамічно генеровані S-блоки, матрицю M, котра забезпечує лінійну дифузію та розширення ключа за допомогою псевдовипадкових чисел та функціях максимального заплутування, що часто називають whitening.

Процес шифрування можна поділити на декілька етапів, перший з них це попередня обробка блоку відкритого тексту розміром 128 біт , котрий розбивається на 4 блоки по 32 біт, запишемо їх як X_1,X_2,X_3,X_4. Після чого, кожна частина XOR-иться з частиною відкритого ключа, створюючи так званий «зашумлений» блок, формулу можна записати наступним чином:

$$X_i = X_i \oplus K_i, \text{ для i} = 1,2,3,4$$

Після чого відбувається основний цикл шифрування, де алгоритм виконує 16 раундів певних обчислень і кожний складається з такий дій:

У кожному раунді обчислюється значення функції F для X_0 та X_1, після чтого результат обчислення XOR-иться з X_2 та X_3:

$$Y = F(x_0, x_1) = M \cdot S(x_0) \oplus S(x_1)$$

де S(x) – це операція заміни за допомогою S-блоків для певних підблоків X_0 та X_1, а M – фіксована матриця лінійної дифузії.

Після обчислення Y отримане значення XOR-иться з половинами блоку даних:

$$X_2 = X_2 \oplus Y$$

$$X_3 = X_3 \oplus Y$$

Результати цієї перестановки надаються високий рівень складності та дифузії.

Після того, як 16 раундів операцій були завершені, відбувається завершальний раунд, отримані 4 частини блоку знову XOR-яться з частинами ключа:

$$X_i = X_1 \oplus K_{i+4}$$

В результаті цього, ми отримуємо фінальний зашифрований блок даних.

До головних та ключових елементів алгоритму потрібно віднести функцію F, ця функція використовує динамічно згенеровані блоки котрі залежать напряму від ключа та значно ускладнюють можливість криптоаналізу. Ключі обробляються за допомогою таблиці змін, після чого проводять процедуру розширення з використанням функції псевдовипадкових чисел та додаткових обчислень в полі Галуа.

Цей алгоритм відомий захистом від атак широкого спектру, а також до атак диференціального та лінійного криптоаналізу, а ще від атак зустріч посередині.

До перевах потрібно віднести можливість зміни ключа на 128, 192 або256 бітів, використання XOR операцій, а також полів Галуа.

Застосування цього шифру як правило можливе в усіх моделях програмного забезпечення та системах, але все ж таки часто поступається іншим алгоритмам.

2.5 Аналіз алгоритму IDEA

IDEA або International Data Encryption Algorithm працює з блоками даних розміром 64 біти та ключем розміру 128 біт. Основною особливістю цього алгоритму є поєднання трьох операцій, таких як додавання за модулем 2^{16}, побітового XOR та множенням за модулем $2^{16} + 1$. Це допомагає алгоритму створювати надзвичайно високу дифузію та заплутаність.

Алгоритм шифрування створений з 8 раундів та вихідного перетворення. Замість одного ключа довжиною 128 біт, використовується шість ключів, отриманих за допомогою розширення головного.

Спочатку алгоритм розбиває головний блок даних на 4 частини по 16 біт, після чого початковий 128 бітний ключ розширюється шляхом циклічних зсувів, а потім розбивається на частини по 16 біт. Це й утворює ключі для кожного раунду.

Далі переходимо до раундів шифрування, першим ділом відбувається множення за модулем $2^{16}+1$, де перший та четвертий під блок множаться на субключі, що можна представити наступним чином:

$$X_1 = (X_1 \cdot K_{i1}) mod (2^{16} + 1)$$

$$X_4 = (X_3 \cdot K_{i4}) mod (2^{16} + 1)$$

Наступною операцією маємо побітові операції XOR між під блоками, що допомагає заплутати:

$$X_2 = X_2 \oplus X_3$$

$$X_2 = X_2 \oplus X_4$$

Після того, як всі 8 раундів операцій були закінченні, під блоки X_1 та X_4 знову множаться на останні два субключі:

$$X_1 = (X_1 \cdot K_{17,1}) mod (2^{16} + 1)$$

$$X_4 = (X_1 \cdot K_{17,4}) mod (2^{16} + 1)$$

Далі під блоки X_2 та X_4 з відповідними субключами додаються по модулю 2^{16}:

$$X_2 = \left(X_1 \cdot K_{17,2}\right) mod\ 2^{16}$$

$$X_3 = \left(X_1 \cdot K_{17,3}\right) mod\ 2^{16}$$

В підсумку блок складений з нових під блоків є зашифрованим текстом.

В підсумку аналізу алгоритму IDEA з’являється висновок, що через наявність трьох операцій таких, як додавання, множення та XOR цей алгоритм може складати значну конкуренцію іншим алгоритмам, завдяки цьому цей алгоритм і став міжнароднім та використовується в багатьох різних системах.

2.6 Аналіз алгоритму Serpent

Алгоритм Serpent був розроблений в далекому 1998 році, в той час його творці хотіли зробити його безпечним за допомогою великої кількості раундів та винайшли алгоритм з 32 раундами. Алгоритм підтримує блок 128 біт та ключи розмірів 128, 192 та 256 бітів.

Процес шифрування складається з 23 раундів та вихідного перетворення. Кожен раунд складається з S-блоків, лінійних перетворень та раундових ключів, що отримуються в результаті розширення стандартного ключа.

Вхідний блок даних має розмір 128 біт та буде позначено як B = X_0, де X_0 – це початковий стан. Саме цей блок буде оброблятися протягом 32 раундів наступним чином:

Спочатку відбувається додавання раундового ключа $X_i = X_i \oplus K_i$, після чого, результат отриманий після додавання ключа проходить обробку за

допомогою S-блоків, котрі обираються залежно від раунду. Загалом використовується 8 різних блоків, котрі повторюються кожні 8 раундів.

$$Y_i = S_i(X_i)$$

Кожен S-блок реалізує нелінійність за допомогою дії на 4 біти вхідних даних. Після обробки S-блоками дані проходять через лінійне перетворення і забезпечує дифузію. За допомогою цього змішуються біти в середині блоку та мінімізують залежність між вхідними та вихідними даними.

$$X_{i+1} = LinearTransform(Y_i)$$

Це лінійне перетворення є серією побітових зсувів і циклічних перестановок бітів – це забезпечує рівномірне поширення інформації по всьому блоку.

Коли алгоритм підходить до останнього, 32 раунду, відбувається лінійне перетворення і результат X_{32} XOR-иться з кінцевим раундовим ключем K_{32}.

$$C = X_{32} \oplus K_{32}$$

Як видно з опису алгоритму вище, Serpent був створений з максимальним запасом стійкості. Вважається що за допомогою 32 раундів цей алгоритм є одним із найбільш захищених. Цей алгоритм використовується в системах всіх видів, а також є частиною деяких стандартів шифрування, незважаючи на його вік.

2.7 Аналіз алгоритму Camellia

Camellia був створений компаніями Mitsubishi Electric та NTT як альтернатива AES. Цей алгоритм підтримує блоки даних розміром 128 біт і може працювати в парі з ключами 128, 192 та 256 біт, що також робить його конкурентом AES. Його структура включає використання нелінійних S-блоків, обертання бітів, а також функцій нелінійного змішування для досягнення сильної дифузії і стійкості до різних криптоаналітичних атак.

Процес шифрування починається з розширення початкового ключа. Використовуються операції обертання та застосування S-блоків для генерування

18 або 24 раундів ключів. Для ключа довжиною 128 біт – використовується 18 раундів, а для довших 24 раунди.

На кожному етапі шифрування до поточного стану блоку даних додається раундовий ключ K_i за допомогою операції XOR:

$$X_i = X_i \oplus K_i$$

де X_i — поточний стан блоку після обробки, а K_i — відповідний раундовий ключ.

Основне перетворення в Camellia здійснюється через функцію FFF, яка є комбінацією нелінійних S-блоків і лінійних операцій. Вхідний блок розбивається на чотири 32-бітні частини X_1,X_2,X_3,X_4. Кожна частина проходить через S-блоки для забезпечення нелінійності:

$$Y = S(X_i)$$

де $S(X_i)$ - це S-блок для кожного з частин вхідного блоку. Потім ці частини змішуються через лінійне перетворення для підвищення рівня дифузії:

$$Y = F(X) = S(X_1) \oplus S(X_2) \oplus S(X_3) \oplus S(X_4)$$

Нелінійні функції FL та FL^{-1} застосовуються після кожного другого раунду для додаткового змішування бітів. Функції функції FL та FL^{-1} включають побітові зсуви та XOR для рівномірного розподілу даних, що сприяє підвищенню стійкості до атак:

$$Y = FL(X)$$

$$Y = Fl^{-1}(X)$$

Після завершення всіх раундів зашифрований блок даних обробляється фінальним XOR з останнім раундовим ключем K_f що дає результат шифрування:

$$C = X_{final} \oplus K_f$$

де C - зашифрований текст, а X_{final} - останній результат після всіх перетворень.

Цей шифр має достатньо високу криптологічну стійкість завдяки поєднанню нелінійних та лінійних перетворювань. Він часто використовується в міжнародних стандартах безпеки, наприклад в ISO/IEC, і в багатьох протоколах VPN та комунікаційних системах.

2.8 Аналіз алгоритму RC6

Шифрування в алгоритмі RC6 відбувається за допомогою розширення ключа та кількох раундів перетворень з підсумковою обробкою в кінці. Головною особливість є використання 4 операндів на кожному раунді, а також застосування операцій додавання, XOR, множення і зсуву. Це дало алгоритму право брати участь в конкурсі AES.

Алгоритм RC6 використовує розширення ключа для генерації множини раундових ключів, які потім застосовуються до даних. На кожному етапі шифрування виконуються численні операції з чотирма операндами.

Спочатку з початкового ключа K генерується масив S розміром 2r+4, де r - кількість раундів. Ключ розширюється таким чином, що кожен раундовий ключ Ki отримується через застосування функцій зсуву та додавання до попередніх елементів масиву S.

Для довжини ключа 128 біт генерується 44 елементи масиву S, для 192 біт - 52 елементи, і для 256 біт - 60 елементів.

Основна ідея RC6 полягає в обробці 128-бітного блоку даних через кілька раундів, де кожен раунд включає виконання нелінійної функції. Спочатку блок даних розбивається на чотири 32-бітні частини A,B,C,D, після чого на кожному раунді застосовуються операції XOR, множення і зсуву.

У кожному раунді обчислюється новий стан для операндів за допомогою функції перетворення F:

$$F(A, B, C, D) = (A + Ki) \oplus (B \times C) \oplus (D \ll 5),$$

де $\oplus$ - це операція XOR, x — множення, а $\ll 5$ - це зсув бітів на 5 позицій. Після цього операнди змінюються таким чином:

$$A = F(A, B, C, D), B = F(B, C, D, A), C = F(C, D, A, B), D = F(D, A, B, C)$$

Після завершення всіх раундів фінальний блок зашифрованих даних обчислюється через додавання фінальних раундових ключів і виконання останніх операцій XOR:

$$C = (A + K_f) \oplus (B \times C) \oplus (D \ll 5),$$

де K_f — останній раундовий ключ.

RC6 активно використовується в багатьох криптографічних додатках, зокрема в захищених протоколах і системах. Однак його використання не є таким поширеним, як AES, оскільки він не став стандартом, а деякі з його операцій можуть бути менш ефективними в середовищах з обмеженими ресурсами.

2.9 Аналіз алгоритму ChaCha20

ChaCha20 базується на числових операціях в 32-бітному просторі і використовує алгоритм побудови псевдовипадкових чисел, що включає багато раундів нелінійних перетворень. Основна ідея полягає в тому, щоб створити висококалерійний потік ключів, який використовує велику кількість обчислень і повторюваних перетворень, що забезпечують стійкість шифра до криптоаналітичних атак.

Процес шифрування в ChaCha20 ґрунтується на генерації псевдовипадкових чисел за допомогою циклічного процесу, що включає застосування кількох раундів трансформацій до блоку даних. Основні етапи виглядають наступним чином:

В основному процесі ChaCha20 використовується операція обміну та перетворення для кожного з 16 елементів блоку S_i, і це повторюється протягом 20 раундів. Процес кожного раунду включає: перестановки та змішування, де кожен елемент блоку змінюється через нелінійні перетворення, а саме через операції XOR, додавання, зсуви і множення, а також псевдовипадкове

перетворення, де кожен етап сприяє створенню псевдовипадкового потоку, який потім використовуватиметься для шифрування.

Однією з основних операцій є обчислення для кожного елемента S:

$$S_{i+1} = S_i + S_{i-1} \oplus (S_{i-2} \ll 16) \oplus (S_{i-3} \ll 8)$$

Після 20 раундів виводиться результат у вигляді нових значень блоків. Після цього, кожен блок шифрується шляхом XOR з відкритим текстом для отримання зашифрованого повідомлення:

$$C_i = P_i \oplus K_i,$$

де P_i — це блок відкритого тексту, а K_i — це згенерований потік ключів після 20 раундів.

Після одного повного раунду шифрування, лічильник збільшується на одиницю, і цей процес повторюється для кожного нового блоку даних, що шифрується.

ChaCha20 також є швидким і ефективним у програмному забезпеченні, що робить його ідеальним для використання в середовищах з обмеженими ресурсами, де апаратне прискорення може бути недоступним. Завдяки своїй стійкості до диференціального та лінійного криптоаналізу, а також високій швидкості, ChaCha20 став популярним вибором для використання в багатьох протоколах безпеки, таких як TLS (Transport Layer Security), VPN, а також у протоколах для мобільних пристроїв.

2.10 Порівняння алгоритмів

У цьому підрозділі мною буде наведено основну інформацію порівняння даних алгоритмів, а саме AES, 3DES, Blowfish, Twofish, IDEA, Serpent, Camellia, RC6, та ChaCha20 виходячи з їх характеристики та конфігурацій. Порівняння обґрунтовується на характеристиках таких, як тип шифрування, довжина ключа, розмірність блоку, число раундів, криптостійкість за даними з відкритих джерел, порівняльна швидкість за даними з відкритих джерел, а також придатність до використання в апаратному та програмному забезпеченні. Це порівняння

дозволяє глибше зрозуміти ефективність наведених алгоритмів, а також показати їх суттєві відмінності.

Як відомо, розмір блоку визначає скільки біт початкового тексту може бути оброблено за одну операцію. У більшості сучасних алгоритмів, наприклад AES, Twofish, Camellia, Serpent і RC6, стандартом є 128 біт інформації. Це дуже добре захищає від колізій при повторному використанні однакових ключів. В той самий час, алгоритми 3DES, Blowfish і IDEA використовують 64 біти, а це означає що вони є менш ефективними для обробки великої кількості вхідних даних.

Довжина ключа впливає на стійкість алгоритму до брутфорсу, тобто грубого перебору. Алгоритми з ключем в 256 мають не аби яку перевагу над іншими, до цих алгоритмів відносяться AES, Twofish, Camellia, RC6 і ChaCha20, такі алгоритми відповідають сучасним стандартам та сертифікаціям. Алгоритм Blowfish є одним з найгнучкіших та самим гнучким з нашого списку, бо довжина його ключа може бути від 32 до 448 біт що дає можливість дуже гнучко налаштовувати його розмір.

Кількість раундів також впливає на складність крипто аналітичних атак, але з іншої сторони дуже впливає на продуктивність, а саме швидкість обробки даних. В цій категорії не можна знайти кращий або гірший алгоритм, бо це залежить суто від необхідностей, або система має більшу захищеність та меншу продуктивність, або меншу захищеність та більшу продуктивність. Хоча навіть в цьому варіанті можна знайти золоту середину, наприклад алгоритм ChaCha20 має лише 20 раундів, що також не мало, це допомагає забезпечити гарну захищеність та продуктивність. Але такі алгоритми як AES, Twofish, RC6 та Camellia мають адаптивну кількість раундів, що напряму залежить від довжини ключа, наприклад, AES виконує 10 раундів для 128-бітного ключа, 12 для 192-бітного та 14 для 256-бітного. Таки гнучкість також може бути неабияким плюсом в порівнянні з іншими алгоритмами, бо використовуючи один й той самий

алгоритм, розробник буде мату змогу отримати більшу захищеність чи більшу продуктивність без зміни алгоритму шифрування.

Швидкість також є неабиякою важливою, особливо в наш час, коли кожна система намагається бути якомога швидшою від конкурентів. В цьому виграє з відривом алгоритм ChaCha20, можливо це через те, що цей алгоритм зазвичай використовують в мобільних пристроях, в котрих потужність значно менша ніж в інших пристроях, наприклад стаціонарних комп’ютерах, серверах. ChaCha20 показує найкращу продуктивність у програмному середовищі, завдяки використанню оптимізованих числових операцій у 32-бітному просторі. AES, завдяки апаратному прискоренню (наприклад, за допомогою AES-NI інструкцій), є надзвичайно швидким у сучасних процесорах. Blowfish і Twofish демонструють середню швидкість у програмному застосуванні, але не мають широкої підтримки апаратного застосування, це сильно обмежує їх використання у високопродуктивних системах. Серпент, через складність своєї структури, є повільнішим, ніж AES або ChaCha20, але забезпечує надзвичайно високий рівень криптографічної стійкості.

Крипостійкість, головна характеристика, саме для цього й були створені криптоалгоритми, щоби захищати інформацію від отримання її третіми сторонами. До основних видів атак, котрим алгоритми повинні протистояти, можна віднести наступні, наприклад диференціальний аналіз, атаки "грубої сили" та лінійний криптоаналіз. Serpent і Camellia вважаються дуже стабільними до захищеності від атак, тому що вони мають велику кількість раундів та складні матриці для перетворення, ChaCha20 відома своєю псевдовипадковою генерацією ключів, що також добре захищає від атак. 3DES є застарілим алгоритмом, оскільки він вразливий до атак типу «зустріч посередині», які ефективно знижують складність пошуку. Blowfish також поступається сучасним алгоритмам через обмежений розмір блоку, що впливає на довгострокову безпеку.

Також важливим пунктом порівняння є методи використання, наприклад AES є універсальним алгоритмом і використовується в усіх сучасних протоколах безпеки, наприклад як TLS, IPsec та WPA3. ChaCha20, як було написано раніше, набув популярності у мобільних додатках і VPN через високу швидкість і стійкість у програмному середовищі.А алгоритм 3DES все ще залишається у деяких застарілих системах, але поступово витісняється новими.

Більш структурована головна інформація, порівняльні дані з відкритих джерел або офіційної документації, представлена нижче та можна знайти в таблиці 2.1.

Таблиця 2.1 – Основні порівняльні характеристики алгоритмів шифрування використаних в дипломній роботі

Алгоритм	AES	3DES	Blowfish	Twofish	IDEA
Тип шифрування	Блочний	Блочний	Блочний	Блочний	Блочний
Розмір блоку (біт)	128	64	64	128	64
Довжина ключа (біт)	128, 192, 256	112,168	32-448	128,192,256	128
Кількість раундів	10, 12, 14	48	До 16	16	8
Швидкість виконання	Дуже висока	Низька	Середня	Середня	Середня
Криптостійкість	Висока	Середня	Середня	Висока	Висока
Застосування	Універсальне	Застарілі системи	Файлове шифрування	Програмне або апаратне	Локальні системи

Продовження таблиці 2.1

Алгоритм	Serpent	Camellia	RC6	ChaCha20
Тип шифрування	Блочний	Блочний	Блочний	Блочно-потоковий
Розмір блоку (біт)	128	128	128	Потокові дані
Довжина ключа (біт)	128,192,256	128,192,256	128,192,256	256
Кількість раундів	32	18,24	20	20
Швидкість виконання	Низька	Висока	Середня	Дуже висока
Криптостійкість	Дуже висока	Висока	Висока	Висока
Застосування	Високобезпечні середовища	Альтернатива AES	Загальне	Мобільні додатки та VPN

Ця таблиця, а також описана інформація вище показує що спектр використання алгоритмів, навіть одного типу кодування, дуже відрізняється в залежності від типу алгоритми та поставленої задачі. Вище було піднято тільки самі базові та голосні особливості алгоритмів, та навіть з цієї інформації зрозуміло, що вибір алгоритму може бути до волі складна задача.

3. ПОКАЗНИКИ ПРОДУКТИВНОСТІ ТА СТІЙКОСТІ АЛГОРИТМІВ

Показники продуктивності та стійкості є одними з найбільш важливих та ключових характеристик не тільки криптографічних алгоритмів, а й всіх алгоритмів та функцій що використовуються в системах програмного або апаратного забезпечення. Ці ключові показники визначають придатність алгоритмів до певних сфер або під сфер застосування. Продуктивність систем зазвичай оцінюється за швидкість шифрування та дешифрування даних, ефективність використання ресурсів здатністю масштабуватися до сучасних вимог. Стійкість як правило визначає наскільки алгоритм захищений від атак, до котрим можна віднести диференційний та лінійний криптоаналіз, атаки на основі відомої частини даних, наприклад відомої частини відкритого тексту, а також стійкості до брутфорс атак, що дуже залежить від довжини ключа. Як було сказано в порівнянні алгоритмів, баланс між продуктивністю та стійкістю є дуже важливим для коректної роботи систем і визначає придатність до використання в певних умовах.

3.1 Швидкість шифрування та дешифрування

Швидкість шифрування та дешифрування визначає здатність забезпечення передачі даних без великих затримок та є фундаментальними характеристиками продуктивності криптографічних алгоритмів. У сучасному світі велика кількість інформації передається по мережам за мілісекунди, а повільна робота алгоритму може суттєво збільшити цей час, що може вплинути на задоволення користувачів продуктом, безпеку, законність, а в деяких системах, наприклад медичних, навіть на життя. Параметр швидкості особливо важливий в таких системах, як фінансові послуги, а особливо в брокерських агенціях або біржах, відеоконференціях, онлайн іграх, мобільних додатках та системах обміном інформацією.

Нижче наведені декілька прикладів чому саме швидкість може мати ключове значення в системі.

Розглянемо приклад високонавантажених систем на прикладі банківських систем, платіжних шлюзів а також систем захисту баз даних. В цих системах є велика кількість інформації, що є майже потоковою, великі обсяги транзакцій потребують алгоритмів що можуть шифрувати та дешифрувати інформацію з мінімальними затримками. В таких системах затримка в обробці даних може призвести до уповільнення роботи транзакцій платіжних систем, що буде проблемою створення черг в усіх пунктах з оплатою через онлайн транзакції і може призвести до повного колапсу починаючи від транспортного, закінчуючи споживацькою.

В системах, що використовуються в реальному часі, наприклад VPN, сервіси стримінгу та потокового відео, сигналізації, в таких системах кожна мілісекунда є важлива і затримка часу вплине на саме важливе в цьому типі систем – якість зв’язку. Повільна робота алгоритму шифрування та дешифрування даних може створити проблему так званих лагів в відеопотоці, що призводить до розсинхронізації відео та аудіо, або переривання трансляції.

Енергоспоживання також за часту залежить від швидкості алгоритмів, до цього типу систем та девайсів можна віднести бездротові відеокамери, мобільні телефони, мобільні системи GPS та інше обладнання, що має на меті довгу автономну роботу при обмеженій кількості накопиченої електроенергії. Повільні алгоритми можуть використовувати надлишкові обчислення та впливати на акумуляторні батареї. Вплив шифрування на енергоспоживання на жаль є дуже актуальним для України на сьогодні так, як в результаті збройної агресії російської федерації на енергосистему України постало велике питання в довготривалому забезпеченні зв’язку на мобільних станціях для підтримки комунікації населення України, а також державних та військовий систем.

Під час вибору криптографічного алгоритму для певної системинеобхідно врахувати дуже важливий баланс між щвидкістю роботи даного алгоритму та його рівнем захисту. До прикладу, такі алгоритму як Twofish або Serpent як правило обирають для систем що використовують акцент на максимальній стійкості від атак, де знехтувати швидкістю алгоритму буде доцільніше та дешевше, ніж захищенністю даних. Такий алгоритм як ChaCha20 оптимально працює в програмному середовищі та має гарні показники як захисту інформації, так і швидкості, що зробило його гарним кандидатом для роботи в мобільних пристроях. AES зазвичай вибирають для систем із високим рівнем вимог до швидкості, бо він має підтримку апаратного прискорення, що має назву AES-NI.

Також потрібно пам’ятати що не завжди швидкість роботи залежить від самого алгоритму, але й від апаратної та програмної платформи, заліза на котрому використовується певний алгоритм. До прикладу, сучасні процессори лінійок intel та amd підтримують апаратні інструкціх для AES, що забезпечує високу швидкість, а такі алгоритми як Blowfish або IDEA не мають цієї підтримки, а отже будуть працювати повільніше. Також слід пам’ятати що криптоалгоритми та наука як криптоаналіз також є частиною бізнесу великих технологічний корпорацій, що об’єднуючись можуть створити кращі умови для певного алгоритму в певній апаратній системі, та гірші для іншого щоб отримати більший спільний прибуток в результаті цих дій.

Також важливо що швидкість напряму конвертується в економію грошей в сервісах з великою кількість даних, наприклад соціальні мережі, або сервіси публікації фото та відео повинні обробляти терабайти даних в секунду, а більша швидкість кодування зменшує витрати на кількість серверів, їх потужність, охолодження та енергетику.

У цій роботі мною буде розроблено функціонал визначення швидкості шифрування та дешифрування даних, але також дуже важливо використовувати дослідження відомих науковців та організацій для ознайомлення з швидкість

роботи алгоритмів, для цього мною був проведений аналіз для використаних в моєму алгоритмі, інформація була взята з International Journal of Advanced Computer Science and Applications декількох видань за 2023 та 2024 роки, а також з International Journal of Computer Applications.

Таблиця 3.1, котра розташована нижче містить середній час шифрування та дешифрування для кожного, використаного в цій дипломній роботі, алгоритму. Вона містить дані в співвідношенні у мілісекундах на 1 кілобайт початкових даних, а також орієнтується на ключи в 128 біт:

Таблиця 3.1 – Дані швидкості шифрування та дешифрування алгоритмів з достовірних джерел

Алгоритм	Середній час шифрування (мс)	Середній час дешифрування (мс)	Примітки щодо продуктивності
AES	0.03	0.03	Висока швидкість, особливо для невеликих даних
ChaCha20	0.02	0.02	Оптимізований для швидкості на мобільних пристроях
RC6	0.05	0.05	Трохи повільніший через складні перетворення

Продовження таблиці 3.1

Camellia	0.04	0.04	Подібний до AES, але оптимізований для апаратного виконання
Twofish	0.08	0.08	Повільний через складний розклад ключів
Serpent	0.09	0.09	Пропонує більший рівень захисту від атак на каналах побічної інформації
IDEA	0.06	0.06	Помірна швидкість через використання модульної арифметики
Blowfish	0.04	0.04	Швидкий, але менш безпечний для великих обсягів даних

Продовження таблиці 3.1

3DES	0.12	0.12	Дуже повільний через потрійне застосування DES

3.2 Стійкість до криптоаналітичних атак

Стійкість до атак є таким самим важливим як і швидкість та є одним із ключових критерієв вибору алгоритму шифрування. Стійкість можна описати як здатність алгоритму протистояти спробам зловмисників відновити зашифроване повідомлення для незаконного отримання початкових даниз за допомогою аналізу зашифрованого тексту чи його певних властивостей. Атаки мають достатньо багато типів, котрі визначають випоги до стійкості алгоритмів:

Почати варто з брутфорсу або атакою грубої сили, це найпростіший метод атаки, що має в собі алгоритм перебору всіх можливх ключів для знаходження потрібного. Протидіяти такому типу атаки досить просто, потрібно лише використовувати ключ великої довжини. До прикладу алгоритм AES може мати ключ довжиною до 256 біт, що робить атаку майже неможливою для досягнення позитивного результату.

Наступний пунктом атак можна визначити криптоаналітичні атаки, до котрих відносяться диференціальний аналіз, лінійний криптоаналіз, а також атаки на основі слабких ключів.

Диференційний криптоаналіз зосереджується на вивченні між парами відкритого тексту та відповідних зашифрованих текстів. Для протидії таким типам атак використовуються складні типи методів змішування та дифузії, до таких алгоритмів ми можемо віднести AES, ChaCha20 і Serpent.

Лінійний криптоаналіз спрямований на знаходження статичних залежностей у відкритого тексту та зашифрованого тексту, та ключа. Велика

кількість нелінійних перетворювань зазвичай запобігає цим атакам, до таких алгоритмів відносяться Serpent і Camellia.

Атаки на основі слабких ключів полягає в тому, що в деяких алгоритмах можуть використовуватися специфічні ключі, ця вразливість є в Blowfish та 3DES. Більш нові алгоритми позбавлені цієї вразливості завдяки суворому підготуванню та аналізу проектування, тому цей вид атак пільш актуальний для старих алгоритмів.

Третім пунктом атак доцільно розглянути атаки методом побічних каналів, бо side-channel attacks, що використовують інформацію вимірювання часу виконання алгоритму, електроспоживання або електромагнітних випромінювань. Сучасні алгоритми шифрування як правило розроблені з урахуванням методів захисту від цього типу атак, це відзначається особливо в апаратних реалізаціях.

І останнім, дуже популярним в наукових колах, важливим методом атаки є квантові атаки. Після появи квантових обчислень класичні алгоритми шифрування даних можуть втратити свою ефективність. Однак симетричні алгоритми не дуже втратять свою актуальність через квантові атаки, все тому що квантові обчислення зменшать лише складність атаки грубої сили в половину ступенів, тому наприклад ключ 256 біт або ще довші будуть досі надійними.

У сучасному цифровому світі, з розвитком технологій шифрування також розвиваються і технології атак на ці самі методи шифрування, а їх захист є пріоритетним, бо має дуже велике значення.

Наприклад в банківських та фінансових системах передаються великі обсяги чутливих даних клієнтів, а так, як майже кожна людина в цивілізованих частинах світу має банківські карти, то злом таких систем може затронути все населення нашої планети. У разі успішної атаки будуть завдані надзвичайно великі збитки, а також втрачена довіра клієнта, що є одним з найважливіших факторів для роботи фінансової установи.

Злом мобільних пристроїв або IoT може призвести до витоку особистої чутливої інформації, що може бути фактором для сантажу людини, котрій належить ця інформація, або загроза її життя.

В державних, військових та розвідувальних системах стійкість до атак також є критично необхідною, витік інформації з таких джерел може вплинути на роботу державного сектора, спровокувати протести або втрату державності.

Висока захищеність даних в медичних сервісах також забезпечує те, що дані клієнтів не будуть скомпрометовані або замінені, що може врятувати людське життя.

Наслідки недостатньої стійкості можуть призвести до втрати коштів, репутації, можуть загрожувати життю людей або національній безпеці.

3.3 Використання ресурсів

Алгоритми шифрування відрізняються вимогами до таких ресурсів, як навантаження на процесор, або процесорний час та оперативна пам’ять. Наприклад деякі алгоритми можуть бути ефективнішими через оптимізацію для певних реалізацій, а деякі через складнощі структури ключів, методів алгоритми та матриць вимагаються більше потужності процесора, а особливо пам’яті.

В великих масштабованих системах використання ресурсів грає дуже велику роль, бо чим більша кількість паралельних операцій шифрування відбувається, тим більших обсяг оперативної пам’яті та потужності процесору потрібен, а через низьку оптимізацію компанії можуть втрачати велику кількість коштів на додаткові апаратні ресурси. А нестабільна оптимізація може призвести до проблем продуктивності в системі, наприклад тимчасове збільшення запитів може призвести до перенавантаження та падіння системи, а проблема використання великої кількості ресурсів не дасть їх можливості піднятися для продовження стабільної роботи.

Тому використання ресурсів, особливо в умовах обмеженої кількості ресурсів або великої кількості запитів дуже важливе для забезпечення стабільності захищеності даних.

4. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ПОРІВНЯННЯ КРИПТОГРАФІЧНИХ АЛГОРИТМІВ

Розроблена програма для порівнянна криптографічних алгоритмів створена з використанням мови програмування C# за допомогою фреймворку та бібліотеки Blazor Bootstrap що дозволило побудувати адаптивний інтерфейс користувача в WEB. Основною метою є створення ефективного та надійного програмного забезпечення для порівняння ефективності популярних криптографічних алгоритмів шифрування даних за швидкістю, з додатковими функціями представлення стійкісті до атак та використанням ресурсів, що дозволить користувачам зрозуміти більше про шифрування даних або обрати оптимальний алгоритм для забезпечення безпеки та продуктивності систем обробки даних. В більшості робота направлена на користувачів що не розуміються або слабо розуміються в понятті шифрування даних. В цій роботі реалізована певна кількість алгоритмів з функціями шифрування та дешифрування даних, а саме таких як AES, 3DES, Blowfish, Twofish, IDEA, Serpent, Camellia, RC6, та ChaCha20. Програма пропонує зручний візуальний інтерфейс, що дозволяє легко взаємодіяти з алгоритмами шифрування та порівнювати результати роботи певного алгоритму в залежності від даних та налаштувань алгоритму.

4.1 Архітектура та функціональні можливості програми

Архітектура програмного забезпечення побудована на основі клієнт-серверної моделі, Для реалізації була обрана мова програмування C#, а також з використанням безкоштовної веб платформи з відкритим кодом Blazor, що дозволяє створювати веб-додатки з використанням C# та HTML.

Blazor був обраний як потужний технологічний стек через його дуже інноваційний підхід до розробки програмного забезпечення, а також він має

низку переваг котрі роблять його зручним для реалізації швидких та сучасних проектів.

Він є частиною екосистеми .NET що дає йому перевагу над іншими в вигляді інтеграцій з різноманітними інструментами компанії Microsoft. Також особливістю є можливість використання мови програмування C# замість класичного JavaScript. Через це ми маємо змогу розробляти спільний код для клієнта і серверної частини застосунку, що неабияк пришвидшує написання функціоналу. Це усуває необхідність в використанні додаткових мов або фреймворків. Завдяки компонентному підходу ми маємо неабияку можливість для динамічної розробки, що сприяє збереженню чіткої структури. Завдяки забезпеченню високої швидкості розробки та зручності мною й був обраний цей фреймворк.

Для реалізації функціоналу шифрування та дешифрування даних у проєкті було обрано бібліотеку Bouncy Castle. Ця бібліотека була розроблена групою австралійських вчених та розробників, що відомі під назвою The Legion of Bouncy Castle Inc. Ця організаціє являється неприбутковою і займаються підтримкою та розробкою криптографічних бібліотек з відкритим кодом. Команда поставила перед собою за мету широкий доступ та надійність стандартизованих криптографічних інструментів і підтримують чималу кількість мов програмування. Основною причиною обрання цієї бібліотеку стало те, що розробники активно підтримують її відповідно до актуальних змін у стандартах та рекомендаціях, наприклад таких як NIST, ISO та інші. Також великою перевагою є те, що бібліотека має велику спільноту та добре документована, що робить її надзвичайно доступною для написання інтеграцій навіть для розробників з початковим рівнем знань. Відкритий код також є перевагою, бо він дозволяє використовувати бібліотеку як в комерційних, так і не в комерційних цілях, що робить її зручною для законодавчих вимог дипломної роботи та не може вважатися плагіатом або несанкціонованим використанням.

Ця бібліотека є одним із найпопулярніших і найпотужніших криптографічних інструментів, що широко використовується в розробці безпечних програмних рішень. Її функціонал та підтримка різноманітних алгоритмів шифрування роблять її ідеальним вибором для проєктів, пов'язаних із криптографією.

Як було написано раніше, архітектура застосунку складається з використання Blazor для клієнтської сторони, та з C# .NET для серверної. Простий варіант бази даних був створений на основі файлів формату csv та функцій зчитування та запису інформації в них.

Проект складається в більшості з файлів формату razor, котрі містять в собі як клієнтський код, так і серверний. До кожного алгоритму відносяться 3 файли для шифрування, дешифрування та запису даних, другий містить в собі таблицю результатів зібраних даних за весь час, а третій порівняння роботи алгоритму з результатами використання різних конфігурацій алгоритму. Також користувач може отримати доступ до загальної WEB сторінки, що містить загальну інформацію стосовно проекту, а також має можливість симуляції атаки на зашифрований текст в спеціальній сторінці. Взаємодію користувача з функціональними сторінка веб-додатку можна представити за допомогою UML діаграми котра зображена на рисунку 4.1.

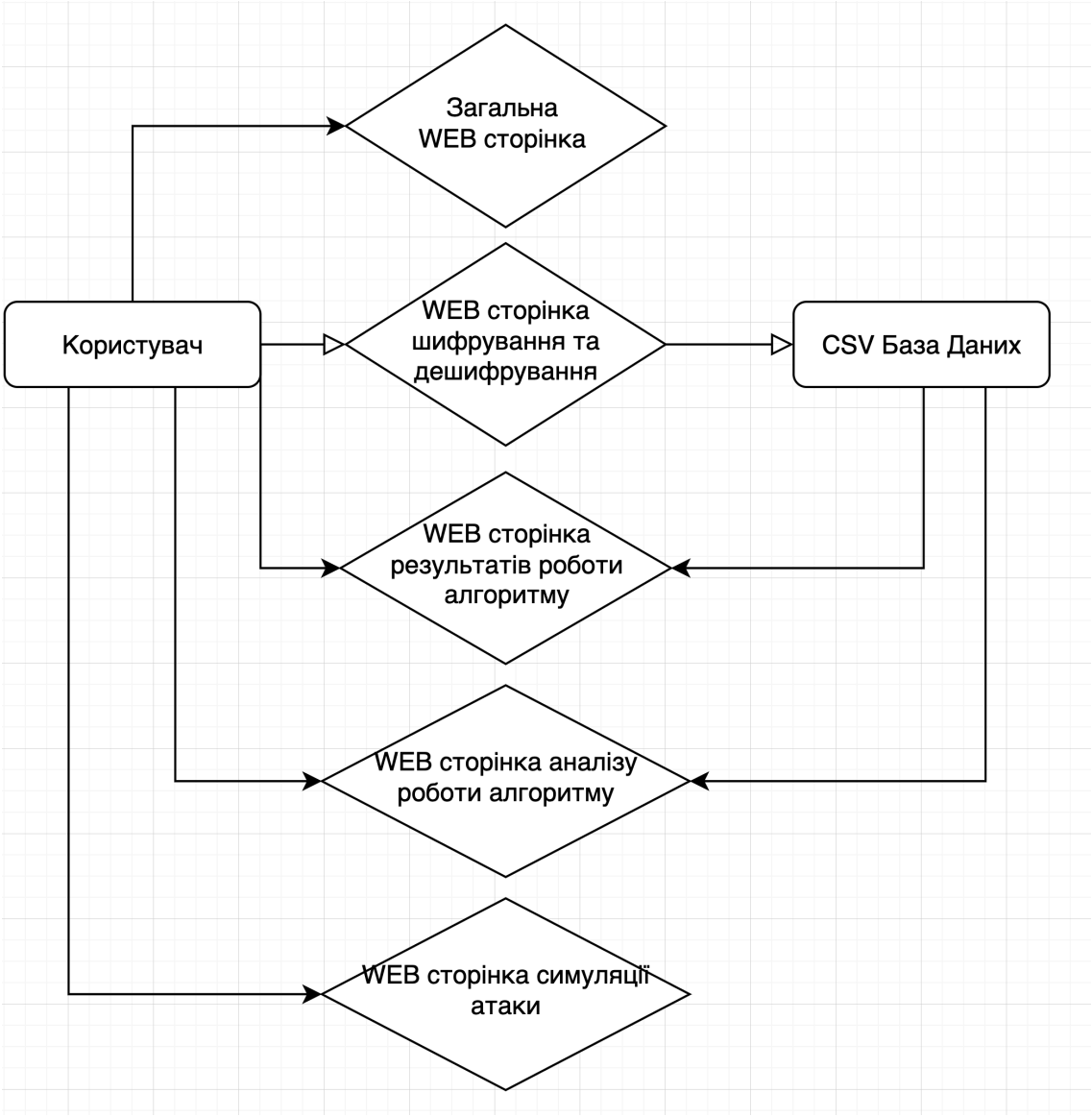


Рисунок 4.1 – UML діаграма з взаємодією користувача та сторінок веб-додатку

Загальна WEB сторінка має назву «Про цей проект» та містить в собі загальну інформацію про проект, навіщо він був створений та ким.

WEB сторінки для шифрування та дешифрування даних містять в собі 9 сторінок, тобто кожен алгоритм має свою сторінку. Кожна сторінка містить загальну інформацію про алгоритм, функцію шифрування та дешифрування даних, вивід результатів шифрування та збереження їх в базу даних. Приклад

вигляду користувацького інтерфейсу даного типу сторінок можна побачити на рисунку 4.2, що містить інтерфейс шифрування та дешифрування для алгоритму AES.

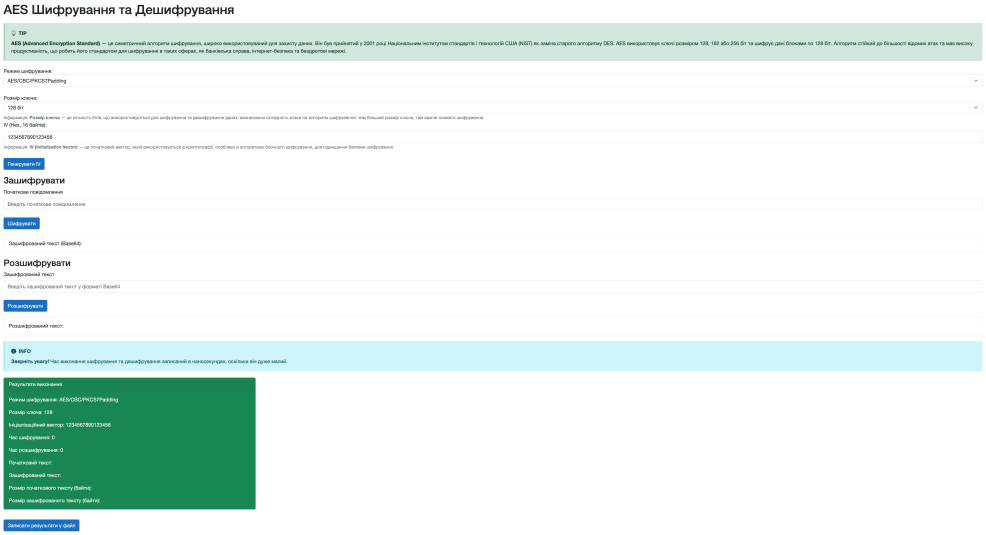


Рисунок 4.2 – Зовнішній вигляд користувацького інтерфейсу шифрування та дешифрування даних алгоритму AES

WEB сторінки результатів роботи алгоритмів також складаються з 9 файлів та 9 сторінок кожна для певного алгоритму, ця сторінка моє таблицю результатів роботи алгоритму, результати зчитаються з бази даних у вигляді CSV файлу. Приклад вигляду користувацького інтерфейсу даного типу сторінок можна побачити на рисунку 4.3, що містить інтерфейс результатів роботи алгоритму AES.

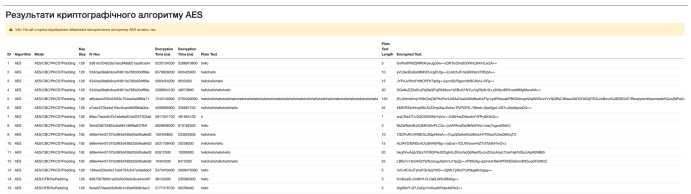


Рисунок 4.3 – Зовнішній вигляд користувацького інтерфейсу результатів роботи алгоритмів на прикладі сторінки для AES

WEB сторінки для аналізу роботи алгоритму також складаються з 9 різних сторінок, кожна для окремого алгоритму та мають в собі побудовані графіки за допомогою Blazor, що дозволяє аналізувати роботу алгоритму порівнюючі різні конфігурації та вхідні дані. Приклад вигляду користувацького інтерфейсу даного типу сторінок можна побачити на рисунку 4.4, що містить інтерфейс для аналізу результатів роботи алгоритмів, на прикладі алгоритму AES.

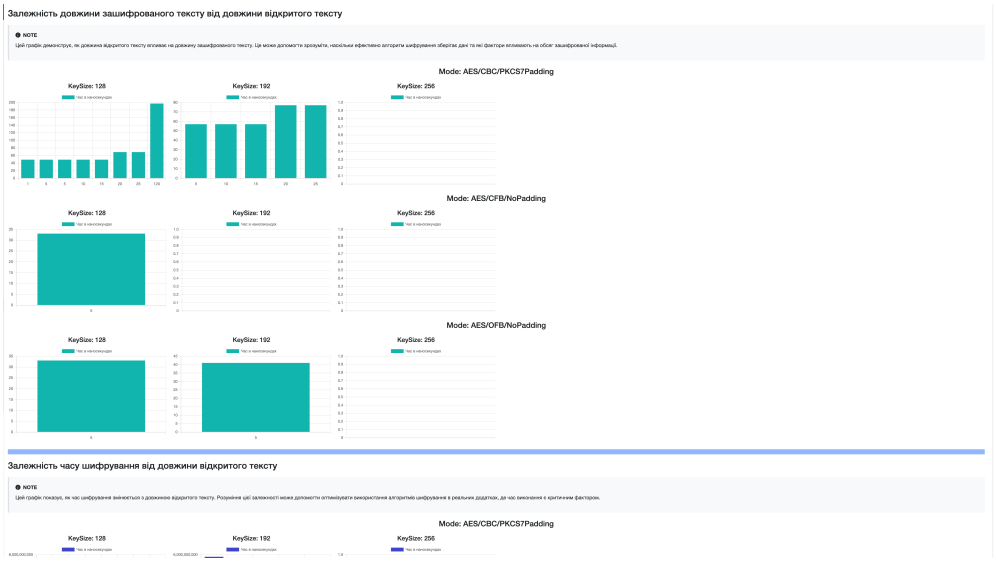


Рисунок 4.4 – Зовнішній вигляд користувацького інтерфейсу для аналізу результатів роботи алгоритмів на прикладі сторінки для AES

CSV база даних також складається з 9 файлів щоб записувати результати кожного алгоритму в окремий файл, це дозволить використовувати менше функцій фільтрування при відображенні даних та знизити використання ресурсів серверу, якщо веб-застосунок буде розміщено віддалено, або ресурсів користувача, якщо застосунок буде розміщений локально на апаратному забезпеченні користувача.

4.2 Опис функціоналу та імплементації сторінки шифрування та дешифрування даних

Так, як реалізація для кожного алгоритму має однаковий фунціонал та імплементацію, то більш детально технічні характеристики буде розглянуто на основі одного алгоритму, а саме AES, а з кодом інших можна ознайомитися в додатку.

Програмне забезпечення підтримує алгоритми котрі були розглянуті в попередніх пунктах дипломної роботи, а саме такі алгоритми як AES, 3DES, Blowfish, Twofish, IDEA, Serpent, Camellia, RC6, та ChaCha20. Незважаючи на різницю в внутрішньому механізмі кожного з цих алгоритмів, структура імплементації даного функціоналу є уніфікованою, це допомагає нам легко підтримувати та розширювати застосунок.

Завдяки цьому функціоналу користувач може вводити текст для подальшого шифрування, обирати параметри такі, як режим роботи та розмір ключа, а також отримувати результат шифрування у форматі BASE64.

Для дешифрування тексту вимагається зашифрований текст разом з ключем, це допомагає відновити початкові дані. Користувач може обрати типи шифрування, котрі підтримує певний алгоритм для більшості це CBC, CFB, OFB, GCM, це забезпечує варіативність та можливість порівняти типи в майбутньому.

Також є можливість зберегти результати шифрування за власної потреби в CSV файл.

У функціоналі даних сторінок реалізовано кілька ключових функцій, котрі відповідають за криптографічні операції. Генерацію ключів, валідацію даних та збереження інформації.

Одною з головних функцій є Encryption котра виконує операцію шифрування заданого алгоритму на основі розміру ключа, режиму шифрування та вектора ініціалізації. Вхідні параметри алгоритму це:

- plainText – початковий текст для шифрування
- mode – режим шифрування
- keySize – розмір ключа в байтах

- ivHex – IV у форматі Hex.

Вихідними даними даної функції є текст у форматі Base64, котрий повертається разом з ключем для дешифрування.

Основні кроки можна описати наступним чином, спочатку відбувається генерація випадкового ключа відповідного розміру, після цього відбувається конертація IV з формату Hex у байтовий масив. Наступним кроком є ініціалізація алгоритму шифрування що відбувається за допомогою бібліотеки Bouncy Castle, після чого відбувається шифрування результату та конвертація в формат Base64.

Наступною важливою функцією є функція дешифрування котра має назву Decryption та виконує дешифрування тексту з використанням ключа, IV та режиму шіфрування. До вхідних параметрів цієї функції відносяться:

- cipherText — текст у форматі Base64, що містить зашифровані дані та ключ.
- mode — режим шифрування.
- keySize — розмір ключа.
- ivHex — IV у форматі Hex.

Вихідними даними даної функції є розшифрований текст.

Нижче, в рисунку 4.5, представлено приклад функції Encryption в одному з алгоритмів:

```
public string Encryption(string plainText, string mode, int keySize, string ivHex)
{
    try
    {
        byte[] inputBytes = Encoding.UTF8.GetBytes(plainText);

        byte[] key = GenerateRandomKey(keySize);

        byte[] iv = ConvertHexStringToByteArray(ivHex);

        var cipher = CipherUtilities.GetCipher(mode);
        cipher.Init(true, new ParametersWithIV(new KeyParameter(key), iv));

        byte[] encryptedBytes = cipher.DoFinal(inputBytes);

        return Convert.ToBase64String(encryptedBytes) + "|" + Convert.ToBase64String(key);
    }
    catch (Exception ex)
    {
        return $"Error: {ex.Message}";
    }
}
```

Рисунок 4.5 – Фрагмент коду функції Encryption

Кроки механізму починаються з розділення cipherText на зашифрований текст та ключ, конвертація вектору ініціалізації у байтовий масив, ініціалізація алгоритму дешифрування побудованого за допомогою бібліотеки Bouncy Castle, після чого відбувається дешифрування та отримання тексту.

Також важливими функціями є ConvertHexStringToByteArray котра конвертує рядок у форматі Hex у байтовий масив. Фрагмент коду цієї функції представлений на рисунку 4.6:

```
public static byte[] ConvertHexStringToByteArray(string hex)
{
    int length = hex.Length;
    byte[] bytes = new byte[length / 2];
    for (int i = 0; i < length; i += 2)
    {
        bytes[i / 2] = Convert.ToByte(hex.Substring(i, 2), 16);
    }
    return bytes;
}
```

Рисунок 4.6 – Фрагмент коду функції ConvertHexStringToByteArray

GenerateRandomKey що генерує випадковий криптографічний ключ заданого розміру та використовує RNGCryptoServiceProvider для генерації криптографічно стійких випадкових чисел представлено на рисунку 4.7:

```
private static byte[] GenerateRandomKey(int keySize)
{
    using (var rng = new RNGCryptoServiceProvider())
    {
        byte[] key = new byte[keySize / 8];
        rng.GetBytes(key);
        return key;
    }
}
```

Рисунок 4.7 – Фрагмент коду функції GenerateRandomKey

Функція GenerateIV генерує випадковий вектор ініціалізації розмірністю 16 байтів, цю функцію ви маєте змогу побачити на рисунку 4.8:

```
private void GenerateIV()
{
    int ivSize = 16;
    byte[] iv = new byte[ivSize];
    using (var rng = new RNGCryptoServiceProvider())
    {
        rng.GetBytes(iv);
    }
    ivHex = BitConverter.ToString(iv).Replace("-", "").ToLower();
}
```

Рисунок 4.8 – Фрагмент коду функції GenerateIV

SaveResultsToFile зберігає результати шифрування та дешифрування в CSV файл використовуючи функцію GetNextId, яка генерує унікальний ідентифікатор для кожного запису, аналізуючи вміст існуючого файлу. Всі ці елементи утворюють структуровану систему, що забезпечує зручність використання та високу якість реалізації криптографічних операцій для різних алгоритмів. Функції SaveResultsToFile та GetNextId ви можете побачити в рисунках 4.9 та 4.10 відповідно:

```
private async Task SaveResultsToFile()
{
    string filePath = "resultsAES.csv";

    // Отримуємо останній ID
    int newId = await GetNextId(filePath);

    // Формуємо рядок CSV
    var csvLine = $"{newId}, AES, {selectedMode}, {selectedKeySize}, {ivHex}, {encryptionTime}, {decryptionTime}, {plainText}, {plainTextLength}, {encryptedText}, {encryptedTextLength}\n";

    // Записуємо у файл
    await File.AppendAllTextAsync(filePath, csvLine);
}
```

Рисунок 4.9 – Фрагмент коду функції SaveResultsToFile

```
private async Task<int> GetNextId(string filePath)
{
    if (!File.Exists(filePath))
    {
        return 1;
    }

    var lines = await File.ReadAllLinesAsync(filePath);
    if (lines.Length == 0)
    {
        return 1;
    }

    var lastLine = lines.Last();
    var values = lastLine.Split(',');

    if (int.TryParse(values[0], out int lastId))
    {
        return lastId + 1;
    }

    return 1;
}
```

Рисунок 4.10 – Фрагмент коду функції GetNextId

4.3 Опис функціоналу та імплементації сторінки результатів роботи алгоритму

Сторінки результатів роботи алгоритмів забезпечують перегляд даних, отриманих в результаті збереження результатів роботі алгиротмів на сторінці шифрування та дешифрування. Головна функція цієї сторінки полягає в відображенні даних з CSV файлу в выгляди таблиці користувацького інтерфейсу. Ця таблиця включає наступні параметри: ідентифікатор запису, тип алгоритму, режим шифрування, розмір ключа, ініціалізаційний вектор (IV), час шифрування та дешифрування, початковий текст і його довжину, а також зашифрований текст і його розмір.

Для представлення даних у зручному форматі на сторінці використовується табличний компонент. Дані завантажуються асинхронно за допомогою методу

OnInitializedAsync, який імітує затримку для оптимізації взаємодії та завантаження записів з файлу, нижче представлено відповідний шматок коду на рисунку 4.11:

```
protected override async Task OnInitializedAsync()
{
    await Task.Delay(500);
    cryptoResults = await LoadCryptoResultsFromCsv("resultsAES.csv");
}
```

Рисунок 4.11 – Фрагмент коду методу OnInitializedAsync

Для обробки CSV-файлу використовується метод LoadCryptoResultsFromCsv, який зчитує файл по рядках, розділяє значення роздільниками і створює об'єкти класу CryptoResult. Цей клас містить всі поля, необхідні для зберігання параметрів криптографічного алгоритму, а з кодом можна ознайомитися на рисунку 4.12 та 4.13:

```
private async Task<List<CryptoResult>> LoadCryptoResultsFromCsv(string filePath)
{
    var resultList = new List<CryptoResult>();

    var csvLines = await File.ReadAllLinesAsync(filePath);

    foreach (var line in csvLines.Skip(1))
    {
        var values = line.Split(',');

        var cryptoResult = new CryptoResult
        {
            Id = values[0],
            Algorithm = values[1].Trim(),
            Mode = values[2].Trim(),
            KeySize = values[3].Trim(),
            IvHex = values[4].Trim(),
            EncryptionTime = values[5].Trim(),
            DecryptionTime = values[6].Trim(),
            PlainText = values[7].Trim(),
            PlainTextLength = values[8].Trim(),
            EncryptedText = values[9].Trim(),
            EncryptedTextLength = values[10].Trim()
        };

        resultList.Add(cryptoResult);
    }

    return resultList;
}
```

Рисунок 4.12 – Фрагмент коду методу OnInitializedAsync

```
private class CryptoResult
{
    public string Id { get; set; }
    public string Algorithm { get; set; }
    public string Mode { get; set; }
    public string KeySize { get; set; }
    public string IvHex { get; set; }
    public string EncryptionTime { get; set; }
    public string DecryptionTime { get; set; }
    public string PlainText { get; set; }
    public string PlainTextLength { get; set; }
    public string EncryptedText { get; set; }
    public string EncryptedTextLength { get; set; }
}
}
```

Рисунок 4.13 – Фрагмент коду об'єкту CryptoResult

Сторінка також містить механізм обробки випадків. Якщо дані все ще завантажуються, виводиться повідомлення «Loading...». Коли дані стають доступними, вони відображаються в таблиці, де кожен рядок являє собою один запис з CSV-файлу. Це дозволяє користувачам аналізувати роботу алгоритму та порівнювати результати. Код відображення даних у вигляді таблиці представлений на рисунку 4.14:

```
<p></p>
<Alert Color="AlertColor.Warning">
    <Icon Name="IconName.ExclamationTriangleFill" class="me-2"></Icon>
    Info: На цій сторінці відображені збережені використання алгоритму AES за весь час
</Alert>

@if (cryptoResults == null)
{
    <p><em>Завантаження...</em></p>
}
else
{
<div class="table-responsive">
    <table class="table">
        <thead>
            <tr>
                <th>ID</th>
                <th>Algorithm</th>
                <th>Mode</th>
                <th>Key Size</th>
                <th>IV Hex</th>
                <th>Encryption Time (ns)</th>
                <th>Decryption Time (ns)</th>
                <th>Plain Text</th>
                <th>Plain Text Length</th>
                <th>Encrypted Text</th>
                <th>Encrypted Text Length</th>
            </tr>
        </thead>
        <tbody>
            @foreach (var result in cryptoResults)
            {
                <tr>
                    <td>@result.Id</td>
                    <td>@result.Algorithm</td>
                    <td>@result.Mode</td>
                    <td>@result.KeySize</td>
                    <td>@result.IvHex</td>
                    <td>@result.EncryptionTime</td>
                    <td>@result.DecryptionTime</td>
                    <td>@result.PlainText</td>
                    <td>@result.PlainTextLength</td>
                    <td>@result.EncryptedText</td>
                    <td>@result.EncryptedTextLength</td>
                </tr>
            }
        </tbody>
    </table>
</div>
```

Рисунок 4.14 – Фрагмент коду відображення таблиці з результатами

Ключовим елементом є використання асинхронних методів для оптимізації продуктивності, а також перевірка та форматування даних перед їх відображенням. Такий підхід забезпечує гнучкість, масштабованість та інтерактивність у роботі з результатами операцій шифрування.

4.4 Опис функціоналу та імплементації сторінки аналізу результатів роботи алгоритмів

Сторінка аналізу доступна для всіх реалізованих криптографічних алгоритмів, таких як AES, Blowfish, Twofish, IDEA, ChaCha20 та інших, сторінка надає користувачам можливість досліджувати різні бенчмарки продуктивності. Сторінки побудовані на основі стандартизованої архітектури, що дозволяє легко інтегрувати нові алгоритми або змінювати методи візуалізації, зберігаючи при цьому узгодженість у дизайні та функціональності.

Кожна сторінка має в собі 6 типів графіків, графік залежності довжини зашифрованого тексту від довжини відкритого тексту демонструє, як довжина відкритого тексту впливає на довжину зашифрованого тексту. Це може допомогти зрозуміти, наскільки ефективно алгоритм шифрування зберігає дані та які фактори впливають на обсяг зашифрованої інформації.

Залежність часу шифрування від довжини відкритого тексту показує, як час шифрування змінюється з довжиною відкритого тексту. Розуміння цієї залежності може допомогти оптимізувати використання алгоритмів шифрування в реальних додатках, де час виконання є критичним фактором.

Залежність часу дешифрування від довжини зашифрованого тексту демонструє, як довжина зашифрованого тексту впливає на час, необхідний для його дешифрування. Це може дати цінні дані для розробників щодо ефективності шифрування при використанні великих обсягів даних.

Залежність часу шифрування від довжини зашифрованого тексту ілюструє, як час шифрування варіюється в залежності від довжини зашифрованого тексту. Це дозволяє зрозуміти, як обробка даних впливає на швидкість шифрування, що може бути критично важливим для додатків з високими вимогами до продуктивності.

Порівняння часу шифрування та дешифрування за довжиною відкритого тексту показує співвідношення часу шифрування та дешифрування. Розуміння цього співвідношення допомагає оцінити ефективність алгоритму в різних

сценаріях, що може бути корисним для вибору правильного методу шифрування в залежності від потреб проекту.

Середній час шифрування та дешифрування за довжиною відкритого тексту ілюструє середні часи шифрування та дешифрування в залежності від довжини відкритого тексту. Це може допомогти в плануванні ресурсів та оцінці швидкості обробки в системах, де шифрування є критично важливим аспектом.

Код організовано таким чином, що дозволяє повторно використовувати компоненти для завантаження та обробки даних, графіки та динамічної взаємодії. Дані для аналізу отримуються з CSV-файлів, що містять тестові параметри, такі як розмір ключа, режим шифрування, час шифрування/розшифрування та довжина відкритого тексту. Після завантаження дані фільтруються за певними критеріями, сортуються і конвертуються у формат, придатний для візуалізації. Код цього механізму ви можете побачити в рисунку 4.15 що розташований нижче:

```
protected override void OnInitialized()
{
    string csvFilePath = "resultsAES.csv";
    var lines = File.ReadAllLines(csvFilePath);

    foreach (var keySize in keySizes)
    {
        chartDataEncrypted[keySize] = new Dictionary<string, ChartData>();
        chartDataEncryptionTime[keySize] = new Dictionary<string, ChartData>();
        chartDataDecryptionTime[keySize] = new Dictionary<string, ChartData>();
        chartDataEncryptionTimeVsEncryptedLength[keySize] = new Dictionary<string, ChartData>();
        chartDataEncryptionDecryptionComparison[keySize] = new Dictionary<string, ChartData>();
        chartDataAverageTime[keySize] = new Dictionary<string, ChartData>();

        barChartsEncrypted[keySize] = new Dictionary<string, BarChart>();
        barChartsEncryptionTime[keySize] = new Dictionary<string, BarChart>();
        barChartsDecryptionTime[keySize] = new Dictionary<string, BarChart>();
        barChartsEncryptionTimeVsEncryptedLength[keySize] = new Dictionary<string, BarChart>();
        barChartsEncryptionDecryptionComparison[keySize] = new Dictionary<string, BarChart>();
        barChartsAverageTime[keySize] = new Dictionary<string, BarChart>();

        foreach (var mode in modes)
        {
            var currentPlainTextLengths = new List<string>();
            var currentEncryptedTextLengths = new List<double?>();
            var currentEncryptionTimes = new List<double?>();
            var currentDecryptionTimes = new List<double?>();

            foreach (var line in lines.Skip(1))
            {
                var columns = line.Split(',');

                if (columns[3].Trim() == keySize.ToString() && columns[2].Trim() == mode)
                {
                    currentPlainTextLengths.Add(columns[8].Trim());
                    currentEncryptedTextLengths.Add(double.Parse(columns[10].Trim(), CultureInfo.InvariantCulture));
                    currentEncryptionTimes.Add(double.Parse(columns[5].Trim(), CultureInfo.InvariantCulture));
                    currentDecryptionTimes.Add(double.Parse(columns[6].Trim(), CultureInfo.InvariantCulture));
                }
            }

            var sortedData = currentPlainTextLengths
                .Select((length, index) => new
                {
                    Length = length,
                    EncryptedLength = currentEncryptedTextLengths[index],
                    EncryptionTime = currentEncryptionTimes[index],
                    DecryptionTime = currentDecryptionTimes[index]
                })
                .OrderBy(item => int.Parse(item.Length))
                .ToList();

            currentPlainTextLengths = sortedData.Select(item => item.Length).ToList();
            currentEncryptedTextLengths = sortedData.Select(item => item.EncryptedLength).ToList();
            currentEncryptionTimes = sortedData.Select(item => item.EncryptionTime).ToList();
            currentDecryptionTimes = sortedData.Select(item => item.DecryptionTime).ToList();

            chartDataEncrypted[keySize][mode] = CreateChartData(currentPlainTextLengths, currentEncryptedTextLengths, 0);
            chartDataEncryptionTime[keySize][mode] = CreateChartData(currentPlainTextLengths, currentEncryptionTimes, 1);
            chartDataDecryptionTime[keySize][mode] = CreateChartData(currentEncryptedTextLengths.Select(e => e.ToString()).ToList(), currentDecryptionTimes, 2);
            chartDataEncryptionTimeVsEncryptedLength[keySize][mode] = CreateChartData(currentEncryptedTextLengths.Select(e => e.ToString()).ToList(), currentEncryptionTimes, 3);
            chartDataEncryptionDecryptionComparison[keySize][mode] = CreateComparisonChartData(currentPlainTextLengths, currentEncryptionTimes, currentDecryptionTimes);
            chartDataAverageTime[keySize][mode] = CreateAverageChartData(currentPlainTextLengths, currentEncryptionTimes, currentDecryptionTimes);

            barChartsEncrypted[keySize][mode] = new BarChart();
            barChartsEncryptionTime[keySize][mode] = new BarChart();
            barChartsDecryptionTime[keySize][mode] = new BarChart();
            barChartsEncryptionTimeVsEncryptedLength[keySize][mode] = new BarChart();
            barChartsEncryptionDecryptionComparison[keySize][mode] = new BarChart();
            barChartsAverageTime[keySize][mode] = new BarChart();
        }
    }
}
```

Рисунок 4.15 – Фрагмент коду завантаження та фільтрування даних

На основі цих даних генеруються графіки для кожного набору параметрів, таких як розмір ключа і режим роботи, з використанням бібліотеки Blazor Bootstrap для створення інтерактивних гістограм. Всі елементи, включаючи графіки та елементи керування, є інтерактивними, що дозволяє користувачам змінювати параметри візуалізації або оновлювати дані без необхідності перезавантаження сторінки.

Стандартизована структура коду використовує шаблони і загальні функцій, таких як CreateChartData, CreateComparisonChartData і CreateAverageChartData, які забезпечують єдиний стиль діаграм для всіх алгоритмів. Такий підхід забезпечує гнучкість і масштабованість системи, що робить її корисною як для користувачів, які аналізують продуктивність окремих алгоритмів, так і для розробників, які бажають розширити функціональність або додати нові криптографічні методи.

4.5 Опис функціоналу та імплементації сторінки симуляції атак

Наступна розроблена сторінка в ході дипломної роботи служить демонстраційним інструментом для вивчення практичного застосування та аналізу стійкості криптографічних алгоритмів. Основна ідея покладена на реалізацію брутфорс алгоритму атаки, що є найпростішим та найбільш універсальним методом злому шифрованих повідомлень.

Основний функціонал даний сторінки передбачає наступні можливості, перш за все це інтерфейс для налаштування параметрів атаки, таких як зашифрований текст, вектор ініціалізації та ключ шифрування, а також відображення процесу атаки і знайденого ключа.

Далі є доцільним описати функції представлені на цій сторінці, по перше це функція StartBruteforce, котра запускає процес перебору ключів, очищує логи, ініціалізує параметри запуску, запускає таймер та викликає метод BruteforceAES. Фрагмент коду ви можете побачити на рисунку 4.16:

```
private async Task StartBruteforce()
{
    logs.Clear();
    StateHasChanged();

    totalKeys = (int)Math.Pow(256, 16);
    checkedKeys = 0;
    progressPercentage = 0;
    remainingKeys = totalKeys;
    estimatedTime = "Очікування...";

    var stopwatch = Stopwatch.StartNew();

    result = await Task.Run(() => BruteforceAES(cipherText, "AES/CBC/PKCS7Padding", ivHex, knownPlainText, stopwatch));

    stopwatch.Stop();
    logs.Add(result);
    StateHasChanged();
}
```

Рисунок 4.16 – Фрагмент коду функції StartBruteforce

Наступною функцією буде доцільно описати BruteforceAES, котра реалізує логіку для спроб розшифрування тексту за допомогою AES, підбирає ключі на основі числової ітерпції, перевіряє чи міститься частина відомого тексту в результаті розшифрування, а також оновлює процес, оцінюючи час що залишився та записує логи. З кодом цієї функції можна ознайомитися на рисунку 4.17 нижче:

```
public async Task<string> BruteforceAES(string cipherText, string mode, string ivHex, string knownPlainText, Stopwatch stopwatch)
{
    int keySize = 128;
    int keyLength = keySize / 8;
    byte[] iv = ConvertHexStringToByteArray(ivHex);

    ulong maxKey = (ulong)Math.Pow(256, keyLength);
    for (ulong i = 0; i < maxKey; i++)
    {

        byte[] key = BitConverter.GetBytes(i).Concat(new byte[keyLength - sizeof(ulong)]).ToArray();


        if (key.Length < keyLength)
        {
            Array.Resize(ref key, keyLength);
        }


        string decryptedText = Decryption(cipherText, key, mode, keySize, ivHex);


        if (!string.IsNullOrEmpty(decryptedText) && decryptedText.Contains(knownPlainText))
        {
            string successLog = $"Успіх! Розшифрований текст містить відомий текст: {decryptedText} з ключем: {Convert.ToBase64String(key)}";
            logs.Add(successLog);
            await InvokeAsync(StateHasChanged);
            return successLog;
        }
        else
        {

            logs.Add($"Спроба з ключем: {Convert.ToBase64String(key)} не вдалася.");
        }

        checkedKeys++;
        remainingKeys = totalKeys - checkedKeys;

        // Оновлюємо прогрес
        progressPercentage = (double)checkedKeys / totalKeys * 100;
        estimatedTime = CalculateEstimatedTime(stopwatch.ElapsedMilliseconds, checkedKeys, totalKeys);

        await InvokeAsync(StateHasChanged);
    }

    return "Брутфорс не вдалося: підходящий ключ не знайдено.";
}
```

Рисунок 4.17 – Фрагмент коду функції BruteforceAES

Наступною функцією є Decryption, що виконує спроби розшифрування, використовуючи для цього бібліотуке BouncyCastle. Використовує метод CipherUtilities.GetCipher для отримання екземпляра алгоритму, виконує розшифрування та повертає відкритий текст, а у разі помилки повідомляє про неї. Нижче ви можете знайти код цієї функції на рисунку 4.18:

```
public string Decryption(string cipherText, byte[] keypossible, string mode, int keySize, string ivHex)
{
    try
    {


        byte[] cipherBytes = Convert.FromBase64String(cipherText);


        byte[] iv = ConvertHexStringToByteArray(ivHex);


        var cipher = CipherUtilities.GetCipher(mode);
        cipher.Init(false, new ParametersWithIV(new KeyParameter(keypossible), iv));


        byte[] decryptedBytes = cipher.DoFinal(cipherBytes);


        return Encoding.UTF8.GetString(decryptedBytes).TrimEnd('\0');
    }
    catch (Exception ex)
    {
        return $"Error: {ex.Message}";
    }
}
```

Рисунок 4.18 – Фрагмент коду функції Decryption

Функція CalculateEstimatedTime допомагає обчислити очікуваний час виконання до завершення підбору ключів, що базується на середньому часі обробки одного ключа та базується на пройденому часі та кількості перевірених ключів, код представлений на рисунку 4.19:

```
private string CalculateEstimatedTime(long elapsedMilliseconds, int checkedKeys, int totalKeys)
{
    if (checkedKeys == 0)
        return "Невідомо";

    double averageTimePerKey = (double)elapsedMilliseconds / checkedKeys;
    double remainingTime = averageTimePerKey * (totalKeys - checkedKeys);
    TimeSpan timeSpan = TimeSpan.FromMilliseconds(remainingTime);

    return $"{(int)timeSpan.TotalMinutes} хв {(int)timeSpan.Seconds} сек";
}
```

Рисунок 4.19 – Фрагмент коду функції CalculateEstimatedTime

І остання функція це ConvertHexStringToByteArray що перетворює шістнадцятирічний рядок у масив байтів, код котрої представлений на рисунку 4.20:

```
private byte[] ConvertHexStringToByteArray(string hex)
{

    int numberChars = hex.Length;
    byte[] bytes = new byte[numberChars / 2];
    for (int i = 0; i < numberChars; i += 2)
    {
        bytes[i / 2] = Convert.ToByte(hex.Substring(i, 2), 16);
    }
    return bytes;
}
```

Рисунок 4.20 – фрагмент коду функції ConvertHexStringToByteArray

5. ОЦІНКА ТА ПОРІВНЯННЯ РЕЗУЛЬТАТІВ ЗА ДОПОМОГОЮ WEB ДОДАТКУ

5.1 Створення тестових даних та пояснення вибраного варіанту тестування web додатку

За допомогою розробленого веб-додатку були виконані експерименти з різними алгоритмами шифрування для оцінки їхньої ефективності та порівняння отриманих результатів. Для забезпечення об'єктивності аналізу було використано однакові вхідні тексти різної довжини, які генерувалися за допомогою інструментів штучного інтелекту. Це дозволило створити стандартизований набір даних, що гарантує однакові умови для кожного алгоритму. Вхідні тексти охоплюють широкий спектр довжин, від кількох речень до значно більших текстів, щоб оцінити, як кожен алгоритм справляється із різним обсягом даних.

Нижче наведено таблицю 5.1 із прикладами текстів, які використовувалися для тестування та були створені за допомогою штучного інтелекту:

Таблиця 5.1 – Приклади текстів що використовувалися для тестування

Довжина тексту	Текст
7 символів	Привіт!
56 символів	Шифрування даних — важливий аспект сучасної кібербезпеки
94 символів	Криптографія — це мистецтво та наука захисту інформації яка є основою довіри в цифровому світі

Продовження таблиці 5.1

243 символів	Симетричні алгоритми шифрування використовуються для забезпечення конфіденційності даних у багатьох сценаріях Популярні алгоритми такі як AES або ChaCha20 забезпечують високий рівень безпеки але різняться за продуктивністю та стійкістю до атак
409 символів	Забезпечення безпеки інформації є одним із ключових викликів сучасного цифрового світу Існує безліч способів захисту даних серед яких шифрування є одним із найефективніших Розвиток криптографії дозволив створити алгоритми які здатні протистояти складним атакам Але важливо розуміти що вибір алгоритму має враховувати не лише його стійкість а й швидкодію енергоефективність та сумісність із різними платформами

Продовження таблиці 5.1

355 символів	У сучасному світі шифрування даних стало невід'ємною складовою безпеки цифрової інформації Його застосування охоплює майже всі аспекти нашого життя від захисту особистих повідомлень до безпечного проведення фінансових операцій через Інтернет Впровадження сучасних алгоритмів дозволяє забезпечити захист від злому водночас забезпечуючи оптимальну швидкодію
388 символів	Одним із найбільш розповсюджених алгоритмів шифрування є AES Завдяки своїй ефективності та високому рівню безпеки він став стандартом для багатьох систем Однак у світі інформаційної безпеки постійно виникають нові виклики що вимагають удосконалення існуючих підходів Наприклад методи квантового шифрування вже сьогодні змінюють правила гри пропонуючи радикально нові способи захисту даних

Вище представлені тексти є лише частиною тестових даних, додатково я буду використовувати тексти довжиною 509, 622, 477, 604, 4292 та 4603 символів. Але не обійшлося без труднощів, на жаль функціонал запису в базу даних погано обробляє не тільки кому, котрі є розділовим знаком для стовбців даних, а й інші знаки, тому на жаль тестування прийдеться робити за допомогою текстів без розділових знаків та інших знаків. Так, як за мету проекту було не саме порівняння роботи алгоритмів, а саме розробка програмного забезпечення, нижче ми будемо розглядати приклад результатів роботи лише одного алгоритму, а саме

AES. А також було використано однаковий вектор ініціалізації для більш валідних тестових результатів.

5.2 Аналіз результатів тестування обраного алгоритму за обраними характеристиками

На перших графіках ми маємо представлення залежності довжини зашифрованого тексту від довжини відкритого тексту для алгоритму AES. Цей графік демонструє, як довжина відкритого тексту впливає на довжину зашифрованого тексту. Це може допомогти зрозуміти, наскільки ефективно алгоритм шифрування зберігає дані та які фактори впливають на обсяг зашифрованої інформації. Також можна помітити що дані пройшли певне сортування бо розташовані від самого малого розміру початкового тексту, до самого великого, а також відбулося сортування по розміру ключа та режиму шифрування. З результатів наведених графіків, представлених на рисунку 5.1, видно читку залежність довжини зашифрованого тексту від довжини початкового для всіх видів поєднань характеристик.

Рисунок 5.1 – Залежність довжини зашифрованого тексту від довжини відкритого тексту для алгоритму AES

На наступній серії графіків ми маємо презентацію залежності часу шифрування від довжини відкритого тексту. Цей графік показує, як час шифрування змінюється з довжиною відкритого тексту. Розуміння цієї залежності може допомогти оптимізувати використання алгоритмів шифрування в реальних додатках, де час виконання є критичним фактором. На жаль як ми бачимо результати досить хаотичні, аде враховуючи що значення є в наносекундах, то різниці в часі роботи має невеликі відхилення одного від одною, якщо вважати час, котрий може помітити людина без додаткових приладів та вимірювань. Також можна припустити що такі результати пов’язані з навантаженням на комп’ютер від інших процесів, що може впливати на результат. Незважаючи на це, на рисунку 5.2, ми бачимо бачимо що шифрування не займало багато часу, це й не дивно, бо багатьма дослідженнями підтверджено що алгоритм AES до волі швидкий алгоритм.

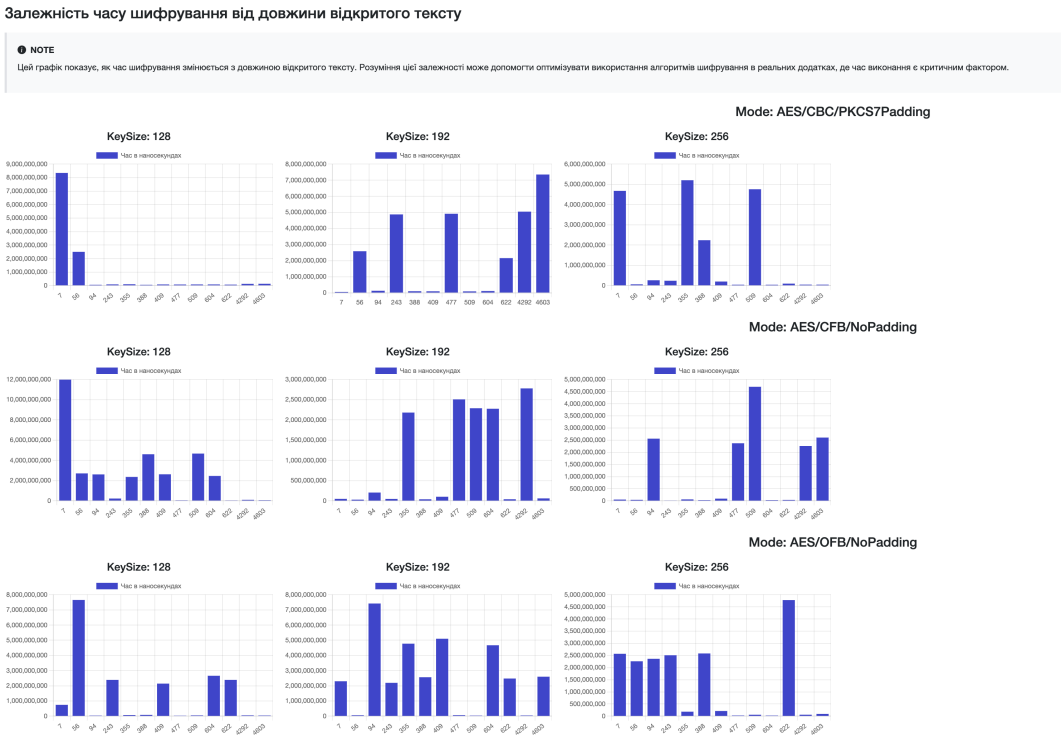


Рисунок 5.2 – Залежність часу шифрування від довжини відкритого тексту

Залежність часу дешифрування від довжини зашифрованого тексту, саме ці графіки відображені на наступному скріншоті, тобто рисунку 5.3. Цей графік демонструє, як довжина зашифрованого тексту впливає на час, необхідний для його дешифрування. Це може дати цінні дані для розробників щодо ефективності шифрування при використанні великих обсягів даних. Так само як і в графіках залежності часу шифрування від довжини відкритого тексту, ми бачимо результати досить хаотичні, але враховуючи що значення є в наносекундах, то різниці в часі роботи також має невеликі відхилення один від одною.

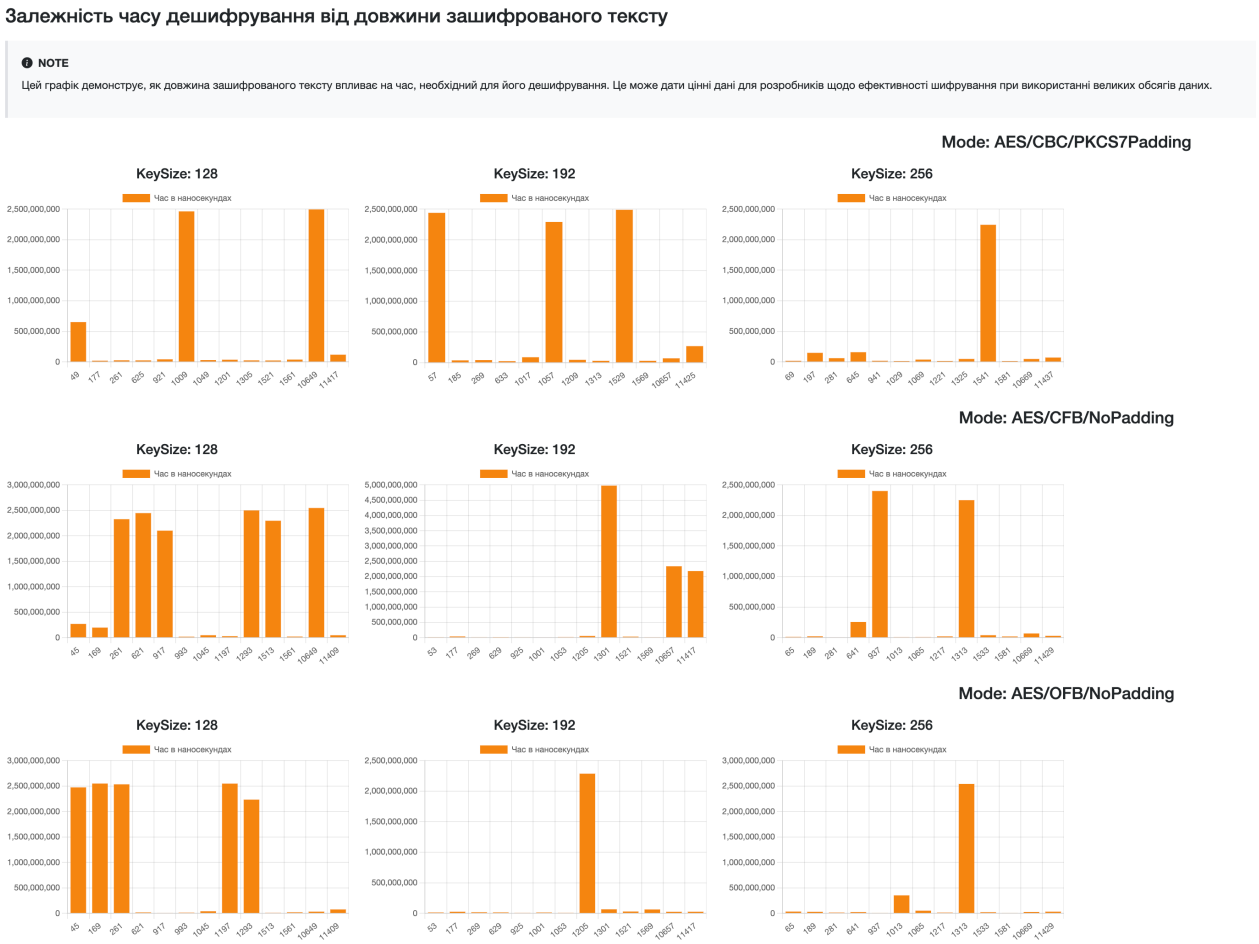


Рисунок 5.3 – Залежність часу дешифрування від довжини зашифрованого тексту

На рисунку 5.4 ми бачимо графіки залежності часу шифрування від довжини зашифрованого тексту. Цей графік ілюструє, як час шифрування варіюється в залежності від довжини зашифрованого тексту. Це дозволяє зрозуміти, як обробка даних впливає на швидкість шифрування, що може бути критично важливим для додатків з високими вимогами до продуктивності. Враховуючи що час шифрування, як і час дешифрування до волі малі, то ми знову маємо до волі хаотичні графіки, але все одно можна відслідкувати малопомітну тенденцію, що часу все ж таки витрачається більше при більшій довжині тексту, найбільше це помітно при 192 та 256 розміру ключа.

Рисунок 5.4 – Залежність часу шифрування від довжини зашифрованого тексту

Наступними ми маємо порівняння часу шифрування та дешифрування за довжиною відкритого тексту. Цей графік показує співвідношення часу

шифрування та дешифрування. Розуміння цього співвідношення допомагає оцінити ефективність алгоритму в різних сценаріях, що може бути корисним для вибору правильного методу шифрування в залежності від потреб проекту. Відслідковується тенденція що шифрування відбувається більш повільно ніж дешифрування, але все одно ці значення неможливо помітити людським оком під час реалізації програми в графічному інтерфейсі, але помітні в результатах, що зображені на рисунку 5.5:

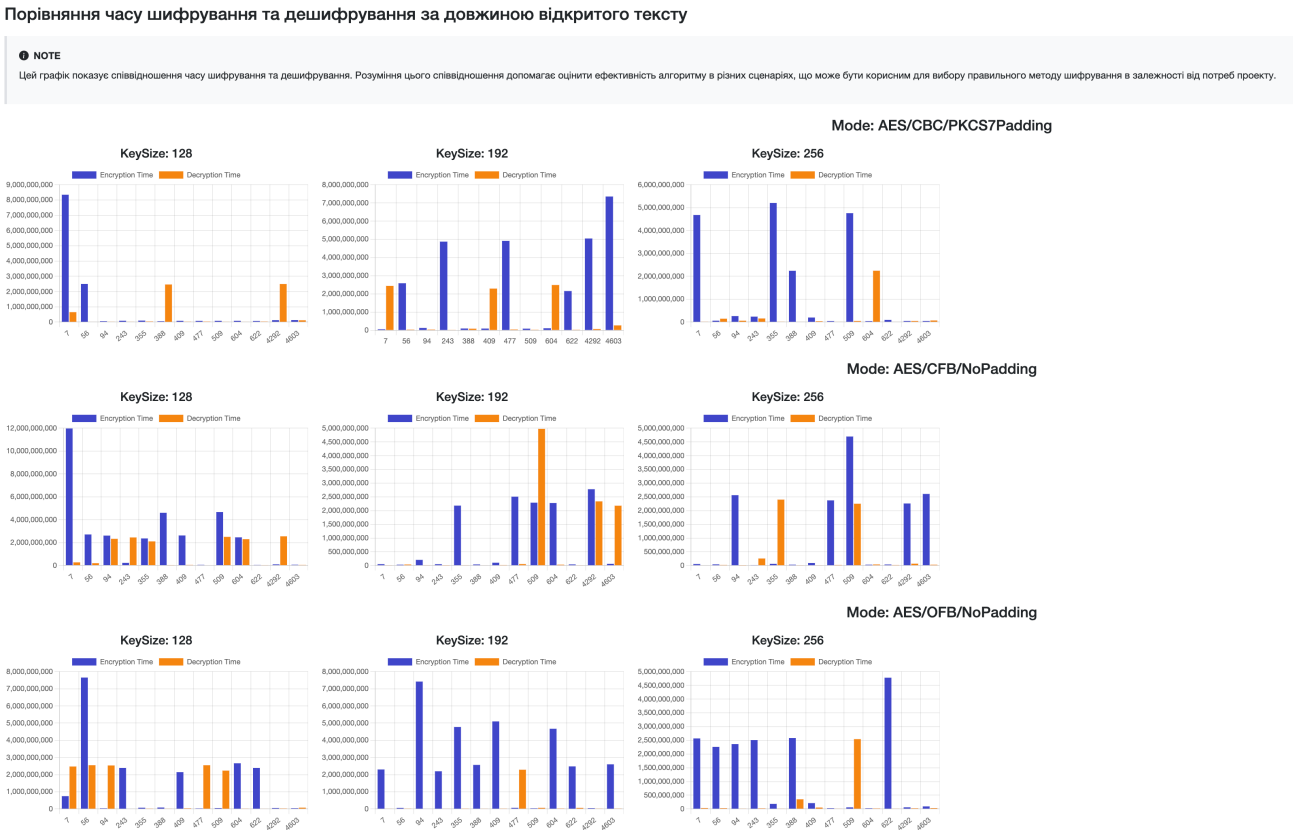


Рисунок 5.5 – Порівняння часу шифрування та дешифрування за довжиною відкритого тексту

Середній час шифрування та дешифрування за довжиною відкритого тексту, саме це назва останніх графіків для порівняння, графічний вигляд котрих зображений на рисунку 5.6. Цей графік ілюструє середні часи шифрування та

дешифрування в залежності від довжини відкритого тексту. Це може допомогти в плануванні ресурсів та оцінці швидкості обробки в системах, де шифрування є критично важливим аспектом. Як і в попередніх ми бачимо хаотичні значення, але відслідковується тенденція що 192 та 256 розмір ключів більш повільний, а також мод OFB повільніший за інші.

Рисунок 5.6 – Середній час шифрування та дешифрування за довжиною відкритого тексту

5.3 Виконання симуляції перебору ключів, використання сторінки Bruteforce web додатку та аналіз результату

На цій сторінці мною була розроблена інформація про можливі варіанти злому шифрувань, наведені нелеликі теоретичні відомості та розроблено алгоритм брутфорсу, або використовуючи іншу назву – перебор ключів. Перебор

ключів був обраний як один з найпростіших в реалізації методів взлому що не потребує додаткового обладнання крім комп'ютера, а також він самий довгий що забезпечить малу ймовірність використання програмного забезпечення для справжніх несанкціонованих атак. Цей алгоритм створений лише в науково-навчальних цілях для людей, котрі не розуміють як працює метод брутфорсу, лише для їх більшої обізнаності. За використання алгоритму в злочинних цілях я, як розробник, не несу відповідальності.

В верхній частині нашої сторінки можна потітити деякі теоретичні відомості про типи атак, це можна побачити на рисунку 5.7:

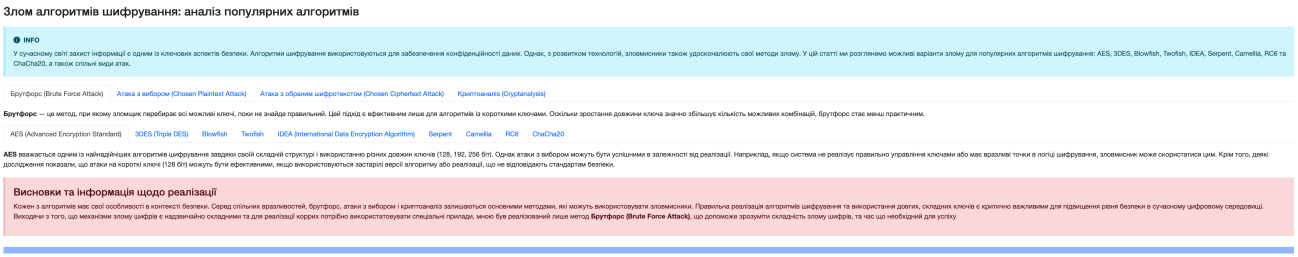


Рисунок 5.7 – Графічний інтерфейс теоретичних відомостей про атаки на алгоритми

Друга частина має в собі певну симуляцію брутфорсу що може використовуватися для навчальних цілей, а також для справжніх атак, що заборонено. Графічний інтерфейс представлено на рисунку 5.8:

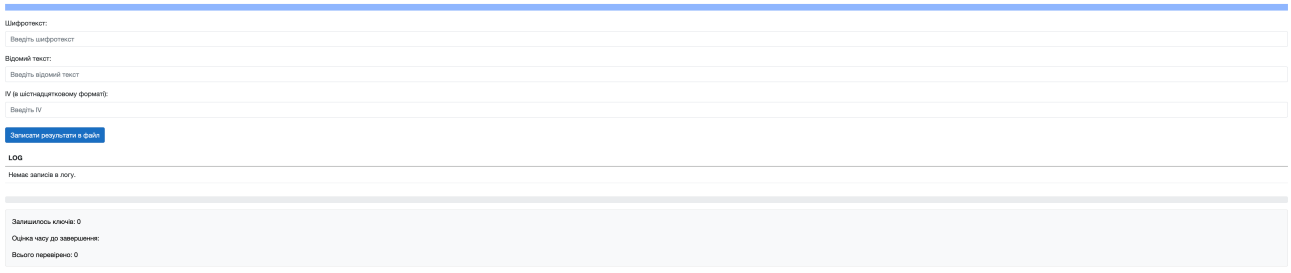


Рисунок 5.8 – Графічний інтерфейс функціоналу брутформу

Далі ми спробуємо використати наш алгоритм брутфорсу, для цього візьмемо один з результатів роботи алгоритму AES, що можна знайти в таблиці результатів на сторінці “AES All Results”. Візьмемо найпростіге зашифроване повідомлення, а саме «Привіт!». Котрий має:

- Зашифрований текст: l+zRPhzd82jClt6ynzBYMQ==
- Вектор ініціалізації: 76bf1a3496b41a4a1f478b5c845d3d7e
- Секретний ключ: pTTQrZLS6LuUAVAz0M39Cg==

Як можна побачити з рисунку 5.9 нижче, орієнтовний час злому, що є приблизним часом перебору всіх можливих секретних ключів, сягає 2167733 хв, що є 36128 годин або 1505 днів. Виходячи з цього, можна сказати що з обмеженими ресурсами звичайного комп’ютера задача злому методом перебору ключів майже неможлива.

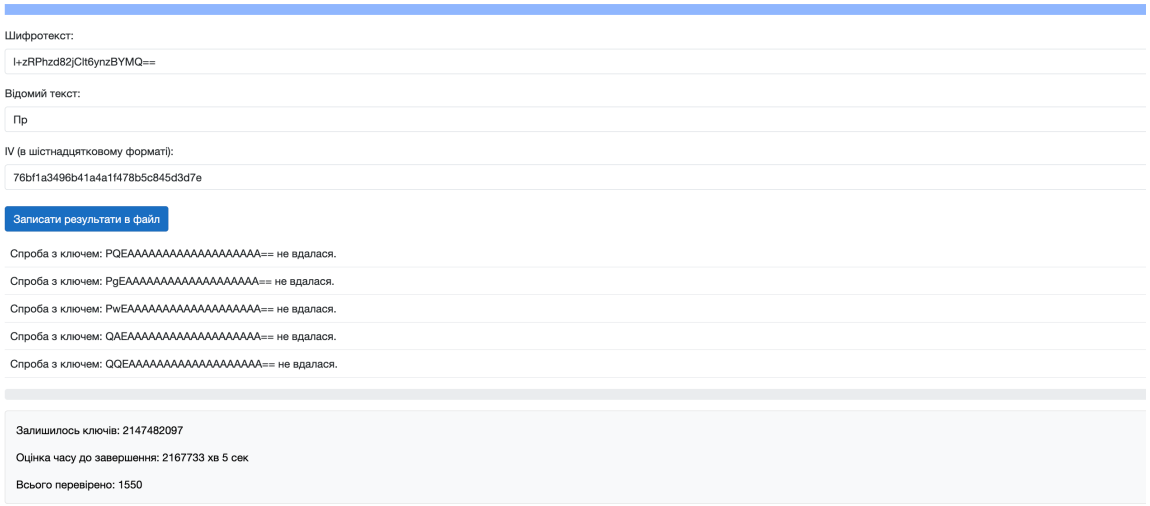


Рисунок 5.9 – Графічний інтерфейс під час виконання методу перебору ключів

ВИСНОВКИ

У дипломній роботі було розроблено програмне забезпечення для порівняння ефективності популярних алгоритмів симетричного блочного шифрування. Основними завданнями роботи розробка програмного забезпечення для аналіз існуючих алгоритмів з точки зору швидкодії, стійкості до атак та ресурсоємності, створення зручного веб-додатка для шифрування, дешифрування та тестування захисту даних, а також підвищення рівня обізнаності користувачів у сфері кібербезпеки.

Одним з головних досягнень є розробка інтерактивного веб-додатку, який дозволяє користувачам детально ознайомитися з принципами роботи сучасних криптографічних алгоритмів. Окрім демонстрації теоретичних аспектів криптографії, він також дозволяє тестувати алгоритми в різних сценаріях. Інтуїтивно зрозумілий інтерфейс робить додаток доступним як для фахівців з інформаційної безпеки, так і для нетехнічних користувачів. Особливу увагу було приділено зручності та зрозумілості системи, що сприяє її широкому використанню.

Важливим аспектом цієї роботи є освітня складова. У програму інтегровано довідковий матеріал, що пояснює основи криптографії, її історію та роль у забезпеченні інформаційної безпеки в сучасному цифровому суспільстві. Це дозволяє користувачам не лише отримати практичний досвід, а й підвищити рівень цифрової грамотності.

Практична цінність дипломної роботи полягає у створенні універсального інструменту, який може використовуватися в різних сферах. Розроблений веб-додаток є корисним для навчальних закладів, де він може стати частиною освітніх програм із кібербезпеки. Крім того, він може використовуватися

компаніями, які працюють із конфіденційними даними, для вибору оптимальних методів шифрування та підвищення безпеки інформаційних систем.

Таким чином, результати дипломної роботи сприяють кращому розумінню особливостей сучасних криптографічних алгоритмів і створюють базу для подальших досліджень. Розроблене програмне забезпечення не лише виконує освітню функцію, але й відкриває нові можливості для практичного застосування криптографії у сфері інформаційної безпеки.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. W. Stallings. 2017. Cryptography and Network Security: Principles and Practice. Pearson.

2. D. Eastlake, J. Schiller, and S. Crocker. 2005. Randomness Requirements for Security. RFC 4086, Internet Engineering Task Force (IETF). https://doi.org/10.17487/RFC4086 (дата звернення: 23.11.23).

3. B. Schneier. 1996. Applied Cryptography: Protocols, Algorithms, and Source Code in C. John Wiley & Sons.

4. J. Daemen and V. Rijmen. 2002. The Design of Rijndael: AES – The Advanced Encryption Standard. Springer Science & Business Media.

5. A. Bogdanov, D. Khovratovich, and C. Rechberger. 2011. Biclique Cryptanalysis of the Full AES. In Proceedings of ASIACRYPT 2011, Springer, 344-371.

6. P. Rogaway. 2011. Evaluation of Some Block Ciphers in the Context of AEAD Modes. Cryptology ePrint Archive, Report 2011/680. https://eprint.iacr.org/2011/680 (дата звернення: 23.11.23).

7. B. Gladman. 2000. A Comparison of Performance Characteristics of the AES Candidate Algorithm. NIST AES Round 2 Evaluations. Режим доступу: https://csrc.nist.rip/encryption/aes/round2/conf3/papers/25-mtakenaka.pdf (дата звернення: 23.11.23).

8. National Institute of Standards and Technology (NIST). 2001. Advanced Encryption Standard (AES). FIPS Publication 197. Режим доступу: https://doi.org/10.6028/NIST.FIPS.197 (дата звернення: 23.11.23).

9. B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, and N. Ferguson. 1998. Twofish: A 128-Bit Block Cipher. AES Candidate Submission. Режим доступу: https://www.schneier.com/academic/archives/1998/06/twofish_a_128-bit.html (дата звернення: 23.11.23).

10. A. Menezes, P. van Oorschot, and S. Vanstone. 1996. Handbook of Applied Cryptography. CRC Press.

11. A. Andoni, S. Zhou, and A. Thorpe. 2020. Analyzing the Performance of Symmetric Key Algorithms: AES, DES, and Blowfish. International Journal of Computer Applications 175(1), 23-30.

12. C. Dworkin. 2001. Recommendation for Block Cipher Modes of Operation. NIST Special Publication 800-38A. Режим доступу: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a.pdf (дата звернення: 23.11.23).

13. P. Zimmermann. 1995. The Official PGP User's Guide. MIT Press.

14. ECRYPT II. 2012. ECRYPT II Yearly Report on Algorithms and Key Sizes(2011-2012). Режим доступу: www.ecrypt.eu.org/ecrypt2/documents/D.SPA.28.pdf (дата звернення: 23.11.23).

ДОДАТОК А

NEW AREAS OF SCIENTIFIC RESEARCH: EXPLORING NEW FRONTIERS
INFORMATION TECHNOLOGIES AND CYBERSECURITY

SOFTWARE FOR COMPARING THE EFFECTIVENESS OF POPULAR CRYPTOGRAPHIC DATA ENCRYPTION ALGORITHMS IN TERMS OF SPEED, ATTACK RESISTANCE, AND RESOURCE USAGE

Maksym Tendit
Master's student at the Department of Information Systems and Technologies Security,
V. N. Karazin Kharkiv National University, Ukraine
xa12284134@student.karazin.ua
Oleksiy Nariezhnii
Candidate of Technical Sciences, Associate Professor at the Department of Information Systems and Technologies Security,
V. N. Karazin Kharkiv National University, Ukraine
o.nariezhnii@karazin.ua

Problem Statement. In the modern digital landscape, data security and integrity are critical, especially in the context of rapidly evolving cybersecurity threats. Encryption algorithms play a central role in ensuring the confidentiality and protection of data transmitted across networks and stored on devices. However, users, especially those without a technical background, often lack awareness of how these algorithms function and differ in their effectiveness. This knowledge gap is compounded by the growing complexity of algorithms and the diverse environments in which they are applied.

Addressing this challenge, the thesis explores the development of a web-based service that provides an interactive platform for understanding, comparing, and experimenting with popular symmetric block encryption algorithms, such as AES, Blowfish, RC6, and others. It also addresses a crucial issue: the need for effective tools to educate users and guide them in selecting suitable algorithms based on their specific requirements.

However, the practical implementation of this technology requires the creation of specialized web resources capable of processing and providing access to this data. Currently, there is a significant threat in Ukraine of various attacks on state and industrial institutions with the aim of reducing accuracy or disrupting services that rely on GPS information. One of the most common cases is the execution of spoofing attacks. Therefore, it is necessary to investigate the development of a secure A-GPS communication to ensure the stable operation of critical infrastructure.

Aims. The aim of this study is to develop and implement an innovative web-based service that combines educational and analytical functionalities to enhance

user understanding and application of symmetric block encryption algorithms. The project seeks to provide an intuitive platform where users can explore how these algorithms operate, compare their efficiency based on performance metrics like speed, resource usage, and resistance to attacks, and experiment with simulated attacks to evaluate their resilience. Additionally, the service includes features for storing and visually analyzing test results, empowering users to make informed decisions about algorithm selection tailored to specific data protection needs. The overarching goal is to bridge the knowledge gap in cybersecurity, equipping users with practical tools and foundational knowledge to address contemporary challenges in information security.

Results and discussion. The web application developed within the project effectively meets its objectives. It features an intuitive user interface that caters to users of varying technical expertise, from students to cybersecurity professionals. The application facilitates direct comparison of algorithms under diverse scenarios, offering insights into their strengths and weaknesses. The experimental module provides a unique opportunity for users to assess the resilience of encryption methods against simulated attacks, further enhancing their understanding.

Additionally, the platform serves as a valuable educational tool, with integrated resources explaining fundamental cryptographic principles and the practical significance of encryption in safeguarding data. The system's modular design supports potential expansion to include advanced encryption standards or integrations with quantum-resistant algorithms, ensuring its relevance in future developments.

The findings underscore the importance of accessible, interactive tools in promoting cybersecurity awareness and informed decision-making. This project demonstrates the potential of combining educational and practical approaches to address the challenges of data security in an increasingly digital world.

Here are some security measures that have been implemented:

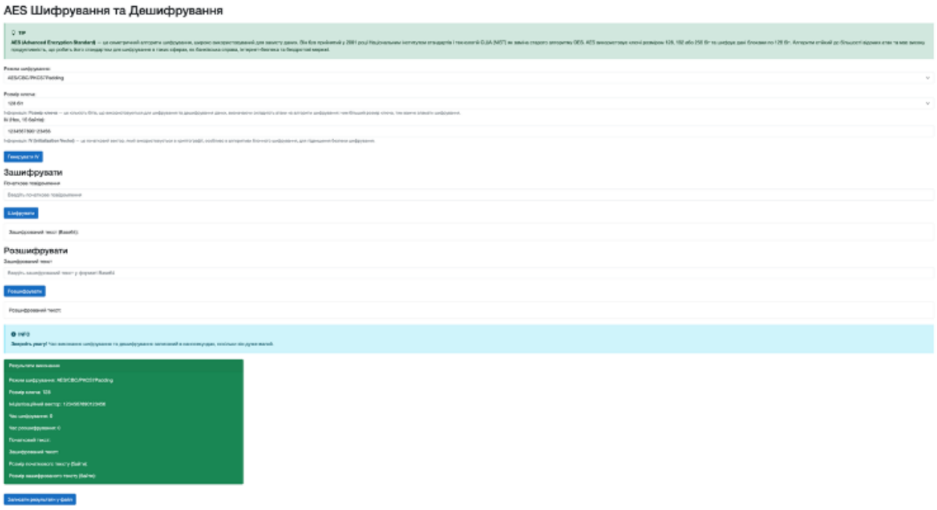
– Cryptographic protection of encrypted data. A robust cryptographic framework ensures the secure transmission of encrypted text between the web service and users. Digital signatures and encryption techniques verify the integrity and authenticity of the data exchanged, preventing tampering or unauthorized access during transmission.

– Protection against brute force and cryptographic attacks. In addition to regular encryption, the web application incorporates defenses against common cryptographic attacks, such as brute force attempts or cryptanalysis. Rate limiting and other countermeasures are applied to mitigate these risks.

– Real-time attack simulation. The system enables the simulation of common cryptographic attacks in real-time, such as brute force and chosen-plaintext attacks. This helps users understand how each algorithm performs under adversarial conditions and the level of protection it provides against such threats.

– User authentication and access control. To ensure secure interaction with the web application, user authentication protocols have been implemented, including

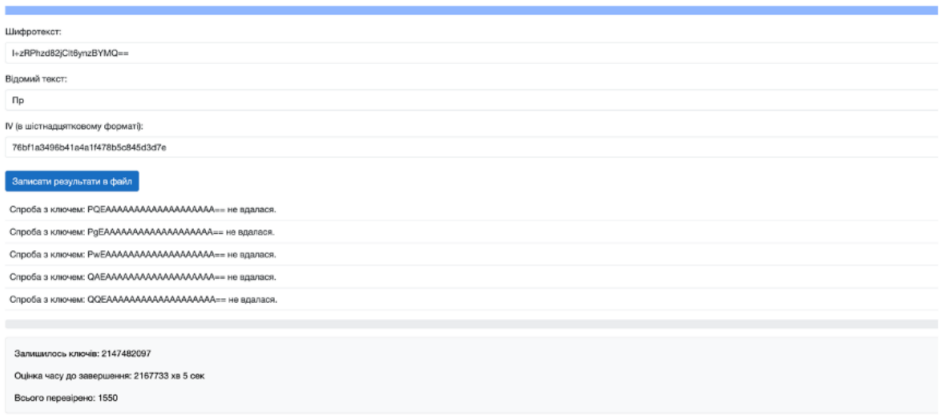
password-based access and, where applicable, stronger multi-factor authentication methods to prevent unauthorized access.



Picture 1 - User interface for encryption and decryption of AES algorithm data

In picture 1, the web pages for data encryption and decryption contain 9 pages, i.e. each algorithm has its own page. Each page contains general information about the algorithm, the function of encrypting and decrypting data, displaying the encryption results and saving them to the database. An example of the user interface of this type of page can be seen in Figure 4.2, which contains the encryption and decryption interface for the AES algorithm.

The following developed page serves as a demonstration tool for studying the practical application and analysis of the stability of cryptographic algorithms. The main idea is based on the implementation of the brute-force attack algorithm, which is the simplest and most versatile method of cracking encrypted messages.



Picture 2 - User interface for encryption and decryption of AES algorithm data

In picture 2, the main functionality of this page provides the following features, first of all, it is an interface for setting up attack parameters, such as ciphertext, initialization vector and encryption key, as well as displaying the attack process and the found key.

Next, it is appropriate to describe the functions presented on this page, first of all, the StartBruteforce function, which starts the key search process, clears the logs, initializes the startup parameters, starts the timer and calls the BruteforceAES method.

Conclusions. One of the main achievements is the development of an interactive web application that allows users to learn in detail about the principles of modern cryptographic algorithms. In addition to demonstrating the theoretical aspects of cryptography, it also allows users to test algorithms in various scenarios. The intuitive interface makes the application accessible to both information security professionals and non-technical users. Particular attention was paid to the convenience and clarity of the system, which contributes to its widespread use.

An important aspect of this work is the educational component. The program includes reference material that explains the basics of cryptography, its history, and its role in ensuring information security in the modern digital society. This allows users not only to gain practical experience but also to improve their digital literacy.

The practical value of the work lies in the creation of a universal tool that can be used in various fields. The developed web application is useful for educational institutions, where it can become part of cybersecurity education programs. In addition, it can be used by companies that work with confidential data to select the best encryption methods and improve the security of information systems.

Thus, the results of the work contribute to a better understanding of the features of modern cryptographic algorithms and create a basis for further research. The developed software not only performs an educational function, but also opens up new opportunities for the practical application of cryptography in the field of information security.

References:

1. W. Stallings. 2017. Cryptography and Network Security: Principles and Practice. Pearson.

2. National Institute of Standards and Technology (NIST). 2001. Advanced Encryption Standard (AES). FIPS Publication 197.

3. A. Andoni, S. Zhou, and A. Thorpe. 2020. Analyzing the Performance of Symmetric Key Algorithms: AES, DES, and Blowfish. International Journal of Computer Applications 175(1), 23-30.

ДОДАТОК Б

Власна публікація (стаття на ResearchGate)

УДК 004.056.55

М.Є Тендіт

Харківський національний університет імені В. Н. Каразіна, Україна

Вступ

У сучасному світі інформаційні системи відіграють ключову роль у зберіганні, обробці та передачі даних, що зумовлює зростаючі вимоги до захисту цих даних. З огляду на значний прогрес у розвитку технологій, а також на еволюцію загроз у сфері кібербезпеки, організації змушені постійно вдосконалювати свої методи шифрування. У цьому контексті симетричні блочні шифри займають важливе місце серед засобів захисту інформації. Вони є одними з найбільш поширених і ефективних інструментів, що забезпечують конфіденційність і цілісність даних під час їх зберігання та передачі.

Симетричні блочні шифри функціонують на основі одного і того ж ключа для шифрування та дешифрування даних, що забезпечує швидкість і ефективність обробки. Однак, незважаючи на їх численні переваги, ці алгоритми піддаються різним типам атак, зокрема диференційному та лінійному криптоаналізу, які можуть знизити їхню стійкість. Внаслідок цього, виникає необхідність у детальному дослідженні стійкості симетричних шифрів до нових загроз, а також у оцінці їх продуктивності в умовах сучасних вимог до безпеки.

Важливим аспектом цього дослідження є аналіз криптографічної стійкості, що включає в себе оцінку довжини ключів, структури алгоритмів та їх вразливості до відомих атак. Також слід врахувати швидкість виконання шифрування та дешифрування, оскільки продуктивність є критично важливою для практичного застосування шифрів в реальних системах.

У цій статті буде здійснено комплексний аналіз захищеності симетричних блочних шифрів від сучасних атак, досліджуючи їх ключові характеристики, такі як криптографічна стійкість, швидкість виконання та вразливість до атак. Результати дослідження мають на меті не лише виявити слабкі місця в існуючих алгоритмах, а й запропонувати можливі шляхи для їх вдосконалення. В умовах стрімкого розвитку технологій та

збільшення кількості кіберзагроз, ефективність методів захисту інформації є надзвичайно важливою для забезпечення безпеки сучасних інформаційних систем.

Теоретичні основи симетричних блочних шифрів

Симетричні блочні шифри є основою сучасних криптографічних систем, що використовуються для захисту інформації в цифровому середовищі. Вони функціонують за принципом, що один і той же ключ використовується як для шифрування, так і для розшифрування даних. Це дозволяє зберігати секретність інформації, забезпечуючи при цьому швидку і ефективну обробку даних. Основні характеристики симетричних блочних шифрів включають структуру алгоритму, довжину блоку та ключа, а також кількість раундів шифрування.

Серед найбільш відомих симетричних блочних шифрів можна виділити AES (Advanced Encryption Standard), DES (Data Encryption Standard) та 3DES (Triple DES). DES, впроваджений у 1977 році, спочатку став стандартом для шифрування даних. Проте, через коротку довжину ключа (56 біт), DES був зламаний за допомогою нових технологій обчислень і атак. На цю проблему відреагували, розробивши 3DES, який застосовує DES тричі з трьома різними ключами, що значно підвищує його стійкість до криптоаналізу.

Проте, у 2001 році AES став новим стандартом, обраним Національним інститутом стандартів і технологій США (NIST) для заміни DES. AES використовує блоки даних розміром 128 біт і підтримує різні довжини ключів — 128, 192 і 256 біт. Це дозволяє налаштовувати рівень безпеки відповідно до вимог користувачів. AES побудований на основі комбінації математичних функцій, які забезпечують високу криптографічну стійкість. Наприклад, його структура включає підходи, такі як заміна, перестановка та множення в полях Галуа, що ускладнює можливість криптоаналізу.

Крім того, криптографічна стійкість блочних шифрів безпосередньо залежить від їхньої архітектури. Алгоритми з більшою кількістю раундів шифрування зазвичай мають вищу стійкість, оскільки кожен раунд додає рівень складності до процесу шифрування. Наприклад, AES має 10, 12 або 14 раундів, залежно від довжини ключа, що робить його одним із найбільш надійних алгоритмів. Однак слід зазначити, що підвищення кількості раундів також може призводити до зниження швидкості виконання алгоритму.

Симетричні блочні шифри також можуть бути реалізовані з різними режимами роботи, такими як ECB (Electronic Codebook), CBC (Cipher

1. Аналіз стійкості до атак

Аналіз стійкості симетричних блочних шифрів до атак є критично важливим етапом у процесі оцінки їхньої безпеки. Стійкість алгоритмів шифрування визначається їхньою здатністю протистояти різним видам криптоаналітичних атак, які можуть використовуватися зловмисниками для зламу шифрів. Основними типами атак на симетричні блочні шифри є атаки з відомим текстом, атаки з вибраним текстом та атаки з обраним шифротекстом.

Атаки з відомим текстом передбачають, що зловмисник має доступ до деякої кількості даних, які були зашифровані разом з відповідними відкритими текстами. Це дозволяє атакуючому аналізувати зв'язок між відкритим текстом та шифротекстом для витягнення секретної інформації, зокрема ключа шифрування. Симетричні блочні шифри повинні бути розроблені таким чином, щоб ускладнити можливість такого аналізу, застосовуючи методи заміни та перестановки, які дезорієнтують зв'язки між текстами.

Атаки з вибраним текстом дозволяють зловмиснику обирати відкриті тексти для шифрування та аналізувати отримані шифротексти. Цей тип атаки є особливо небезпечним, оскільки надає криптоаналітикам можливість отримати більше інформації про структуру алгоритму шифрування. Для захисту від атак з вибраним текстом шифри повинні мати вбудовані механізми для захисту від аналізу, наприклад, завдяки використанню складних структур раундів і параметрів, які варіюються під час шифрування.

Атаки з обраним шифротекстом є ще одним серйозним ризиком, оскільки в цьому випадку зловмисник може вибирати шифротексти для дешифрування, намагаючись витягти відкритий текст. Відповідно до цього типу атаки, шифри повинні забезпечувати захист не лише від безпосереднього дешифрування, але й від можливих структурних вразливостей, які можуть бути виявлені через обрані шифротексти.

Крім цих атак, важливими аспектами, що впливають на стійкість блочних шифрів, є довжина ключа та кількість раундів. Довші ключі забезпечують більшу складність для зловмисника, оскільки зростає кількість можливих комбінацій, що потрібно перебрати під час атак. Проте, використання надто коротких ключів або недостатньої кількості

Block Chaining) та CTR (Counter Mode). Ці режими визначають, як блоки даних обробляються під час шифрування і розшифрування, а також можуть впливати на загальну стійкість алгоритму до атак. Наприклад, режим ECB не забезпечує достатньої безпеки для однотипних блоків, оскільки він шифрує їх однаково, що робить їх уразливими до криптоаналізу.

Усе це підкреслює важливість теоретичних основ симетричних блочних шифрів, які необхідні для розуміння їхніх можливостей та вразливостей. Оцінка стійкості та продуктивності алгоритмів шифрування є важливим етапом у забезпеченні інформаційної безпеки в сучасних умовах, коли зростає кількість загроз і атак на криптографічні системи.

раундів може призвести до того, що алгоритм стане вразливим для атак, таких як метод брутфорс, де зловмисник пробує всі можливі ключі до тих пір, поки не знайде правильний.

Однією з методик, що використовуються для оцінки стійкості блочних шифрів, є криптоаналіз. Цей підхід включає вивчення шифрів з метою виявлення вразливостей у їхній структурі. Дослідники постійно проводять тести на стійкість алгоритмів, розробляючи нові методи криптоаналізу, такі як диференційний та лінійний криптоаналіз, які виявляють патерни в даних, що шифруються.

Слід зазначити, що постійний розвиток квантових обчислень також підвищує актуальність аналізу стійкості симетричних блочних шифрів. З появою квантових комп'ютерів, алгоритми, які базуються на класичних принципах шифрування, можуть стати вразливими до атак, які раніше були недоступні. Це викликає необхідність у дослідженні нових стандартів і підходів до криптографії, щоб забезпечити необхідний рівень захисту в умовах розвитку технологій.

Аналіз стійкості симетричних блочних шифрів до атак є складним і багатогранним процесом, який вимагає не лише теоретичного підходу, але й практичного тестування та верифікації. Дослідження в цій сфері допомагають виявити уразливі місця, розробити нові методи захисту та забезпечити надійність криптографічних систем у сучасному цифровому середовищі.

2. Продуктивність симетричних блочних шифрів

Продуктивність симетричних блочних шифрів є одним з ключових аспектів, що впливають на їх ефективність у реальних умовах використання. Вона визначається кількома факторами, включаючи швидкість шифрування та дешифрування, вимоги до обчислювальних ресурсів, а також затримку, яка може виникати при обробці даних. У сучасних системах, де швидкість передачі даних і обробки інформації є критично важливими, продуктивність шифрів має особливе значення.

Швидкість шифрування та дешифрування залежить від алгоритму, який використовується, а також від довжини блоку даних, що шифрується. Наприклад, алгоритми, такі як AES (Advanced Encryption Standard), показують високу продуктивність завдяки ефективним конструкціям і оптимізації під сучасні апаратні платформи. AES використовує 128-бітні блоки даних і підтримує ключі довжиною 128, 192 та 256 біт, що дозволяє досягти різного рівня безпеки без значного впливу на швидкість.

Однак, не всі блочні шифри можуть похвалитися такою ж продуктивністю. Наприклад, деякі старіші або менш оптимізовані алгоритми можуть бути повільнішими через свою структуру, яка включає велику кількість раундів або складні операції. Продуктивність також може значно варіюватися в залежності від апаратного забезпечення, на якому виконується шифрування. Сучасні процесори часто мають вбудовані апаратні прискорювачі для виконання криптографічних операцій, що істотно підвищує швидкість шифрування. Наприклад, інструкції, що реалізують AES-NI (AES New Instructions), дозволяють виконувати операції шифрування значно швидше, ніж традиційні програмні реалізації.

Витрати ресурсів також є важливим аспектом продуктивності шифрів. Алгоритми шифрування, які потребують великих обсягів пам'яті або складних обчислень, можуть бути непридатними для використання в середовищах з обмеженими ресурсами, таких як мобільні пристрої або IoT (Internet of Things). У таких випадках оптимізація алгоритмів шифрування для зниження споживання пам'яті та обчислювальної потужності є критично важливою.

Затримка при обробці даних також грає роль у загальній продуктивності. У системах реального часу, де швидкість передачі даних має вирішальне значення, велика затримка може вплинути на продуктивність усього

додатку. Багато симетричних блочних шифрів проектуються з урахуванням потреб у зниженні затримки, що дозволяє досягти швидшого обміну даними без втрати безпеки.

Важливо також зазначити, що продуктивність шифрів можна поліпшити через паралельну обробку. Багато сучасних алгоритмів підтримують паралельне шифрування блоків, що дозволяє використовувати кілька потоків або ядер процесора для обробки даних одночасно. Це особливо важливо для застосувань, які обробляють великі обсяги даних, таких як відео- або аудіообробка.

Для оцінки продуктивності симетричних блочних шифрів використовуються різні тестові набори і стандартизовані методи вимірювання. Наприклад, може бути проведено тестування на швидкість шифрування та дешифрування, а також на ефективність використання ресурсів у різних умовах.

Таким чином, продуктивність симетричних блочних шифрів є критично важливою для їх практичного використання в реальних системах. Оптимізація шифрів для досягнення високої швидкості, зниження витрат ресурсів і мінімізації затримок може значно підвищити ефективність криптографічних рішень у сучасному інформаційному середовищі. З розвитком нових технологій і зростанням потреби в безпеці важливо продовжувати дослідження в цій сфері для покращення існуючих стандартів шифрування та розробки нових, більш продуктивних алгоритмів.

Висновок

У процесі дослідження було встановлено, що симетричні блочні шифри продовжують залишатися надійним інструментом для захисту даних у сучасному інформаційному середовищі. Проте їхня стійкість до нових типів атак вимагає постійного моніторингу та вдосконалення. Аналіз показав, що оптимальний баланс між криптографічною стійкістю та продуктивністю є важливим фактором для ефективного використання шифрів. У майбутньому необхідно продовжувати дослідження у цій галузі, зокрема, розвивати нові методи шифрування та вдосконалювати існуючі алгоритми, щоб забезпечити захист інформаційних систем у умовах зростаючих загроз.

Список використаних джеред

1. Stallings W. Cryptography and Network Security: Principles and Practice [Electronic resource] / William Stallings. – 7th ed. – Pearson, 2017. – 800 p. – Mode of access: https://www.pearson.com/store/p/cryptography-and-network-security-principles-and-practice/P100000438169 (date of access: 10.11.2023).

2. Katz J., Lindell Y. Introduction to Modern Cryptography: Principles and Protocols [Electronic resource] / Jonathan Katz, Yehuda Lindell. – 3rd ed. – CRC Press, 2020. – 816 p. – Mode of access: https://www.crcpress.com/Introduction-to-Modern-Cryptography-Principles-and-Protocols-Third/Katz-Lindell/p/book/9781138412449 (date of access: 10.11.2023).

3. Ferguson N., Schneier B., Kohno T. Cryptography Engineering: Design Principles and Practical Applications [Electronic resource] / Niels Ferguson, Bruce Schneier, Tadayoshi Kohno. – Wiley, 2010. – 528 p. – Mode of access: https://www.wiley.com/en-us/Cryptography+Engineering%3A+Design+Principles+and+Practical+Applications-p-9780470474242 (date of access: 10.11.2023).

4. Diffie W., Landau S. Privacy on the Line: The Politics of Wiretapping and Encryption [Electronic resource] / Whitfield Diffie, Susan Landau. – MIT Press, 2007. – 376 p. – Mode of access: https://mitpress.mit.edu/books/privacy-line (date of access: 10.11.2023).

5. NIST Special Publication 800-38A. Recommendation for Block Cipher Modes of Operation: Methods and Techniques [Electronic resource] / National Institute of Standards and Technology. – 2001. – 96 p. – Mode of access: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a.pdf (date of access: 10.11.2023).

6. Beaulieu R., Hoang T., Nandi M., et al. The SIMON and SPECK Families of Lightweight Block Ciphers [Electronic resource] / Robbe Beaulieu, Thanh Hoang, Mouhoub Nandi, et al. // 2015 IEEE International Conference on Electronics, Circuits, and Systems (ICECS). – 2015. – P. 90–94. – Mode of access: https://ieeexplore.ieee.org/document/7399471 (date of access: 10.11.2023).

Тендіт Максим Євгенович – студент, магістр кафедри кібербезпеки інформаційних систем, мереж і технологій Харківського національного університету імені В.Н.Каразіна, e-mail: xa12284134@student.karazin.ua, ORCID: 0009-0006-7013-3003.