Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя
(повне найменування вищого навчального закладу)
Факультет комп'ютерно-інформаційних систем і програмної інженерії
(назва факультету)
Кафедра кібербезпеки
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(освітній рівень)

на тему: "Дослідження ефективності сучасних симетричних криптографічних алгоритмів"

Виконав: студент (ка) 4 курсу, групи СБ - 41
Спеціальності:
125 «Кібербезпека»
(шифр і назва напряму підготовки, спеціальності)
Семчишин Олександр Володимирович
підпис (прізвище та ініціали)

Керівник Загородна Н.В.
підпис (прізвище та ініціали)

Нормоконтроль Тимощук Д. І.
підпис (прізвище та ініціали)

Завідувач кафедри Загородна Н.В.
підпис (прізвище та ініціали)

Рецензент
підпис (прізвище та ініціали)

м. Тернопіль – 2024

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра кібербезпеки
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Загородна Н.В.
(підпис) (прізвище та ініціали)
«__» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)

за спеціальністю 125 Кібербезпека
(шифр і назва спеціальності)

Студенту Семчишину Олександру Володимировичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження ефективності сучасних симетричних криптографічних алгоритмів.

Керівник роботи Загородна Наталя Володимирівна, к.т.н., доцент, завідувач кафедри КБ
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «15» 04 2024 року № 4/7–350

2. Термін подання студентом завершеної роботи 09.06.2024

3. Вихідні дані до роботи Характеристики сучасних симетричних криптографічних алгоритмів

4. Зміст роботи (перелік питань, які потрібно розробити)
Знайти та описати алгоритми, що використовуються при шифрувані.
Проаналізувати властивості, характеристику та принцип роботи алгоритмів шифрування.
Розробити програму, що реалізує досліджуванні алгоритми шифрування та виконує заміри їх ефективності.
Безпека життєдіяльності, основи охорони праці.
Висновки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Мета та завдання роботи. Основні поняття. Класифікація криптосистем. Симетричні криптосистеми. AES. DES. RC4. Blowfish. Тестування в ОС Windows. Тестування в ОС Linux. Висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи хорони праці	Мариненко С.Ю., к.т.н., доцент кафедри МТ		

7. Дата видачі завдання 29.01.2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	29.01 – 31.01	Виконано
2.	Підбір джерел для аналізу симетричних криптосистем	01.02 – 05.02	Виконано
3.	Опрацювання джерел в галузі дослідження	06.02 – 15.02	Виконано
4.	Провести аналіз симетричних криптоалгоритмів	22.02 – 12.03	Виконано
5.	Оформлення розділу «Симетричні криптосистеми та їх значення для інформаційної безпеки»	10.02 – 05.03	Виконано
6.	Оформлення розділу «Аналіз сучасних симетричних криптоалгоритмів»	26.03 – 04.05	Виконано
7.	Оформлення розділу «Практичне дослідження ефективності симетричних алгоритмів шифрування»	12.04 – 20.04	Виконано
8.	Виконання завдання до підрозділу «Безпека життєдіяльності, основи охорони праці»	25.04 – 10.05	Виконано
9.	Оформлення кваліфікаційної роботи	23.05 – 08.06	Виконано
10.	Нормоконтроль	09.06 – 11.06	Виконано
11.	Перевірка на плагіат	12.06 – 14.06	Виконано
12.	Попередній захист кваліфікаційної роботи	18.06	Виконано
13.	Захист кваліфікаційної роботи	26.06.2024	

Студент _____ Семчишин О.В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Загородна Н. В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дослідження ефективності сучасних симетричних криптографічних алгоритмів // Кваліфікаційна робота ОР «Бакалавр» // Семчишин Олександр Володимирович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра кібербезпеки, група СБ-41 // Тернопіль, 2024 // с. – 74, рис. – 21, табл. – 6, лістинги – 16, бібліогр. – 12.

КЛЮЧОВІ СЛОВА: криптографія, шифрування, симетричні криптосистеми, інформаційна безпека, конфіденційність, захист інформації.

Кваліфікаційна робота присвячена дослідженню ефективності сучасних симетричних криптографічних алгоритмів. З метою забезпечення безпеки інформаційних систем, вибір надійного криптографічного алгоритму є критично важливим, оскільки від цього залежить захист конфіденційних даних. У роботі розглянуто чотири популярні симетричні криптографічні алгоритми: AES, DES, RC4 та Blowfish.

У процесі дослідження було проведено теоретичний аналіз алгоритмів, де розглянуто їх архітектуру та основні параметри конфігурації. Також було проведено експериментальні дослідження для оцінки продуктивності алгоритмів з точки зору часу виконання, використання пам'яті та завантаження процесора.

Розроблений підхід та отримані результати можуть бути використані системними адміністраторами та розробниками для покращення захисту даних у різних інформаційних системах. Кваліфікаційна робота також буде корисною для студентів та фахівців у сфері кібербезпеки, які прагнуть поглибити свої знання про симетричні алгоритми та їх застосування для захисту даних.

Висновки та рекомендації, представлені у роботі, сприятимуть підвищенню рівня безпеки інформаційних систем шляхом вибору найефективнішого симетричного криптографічного алгоритму відповідно до конкретних вимог та обмежень.

ANNOTATION

Research on the Effectiveness of Modern Symmetric Cryptographic Algorithms // Qualification Work for the Bachelor's Degree // Semchyshyn Oleksandr Volodymyrovych // Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Cybersecurity, Group SB-41 // Ternopil, 2024 p. – 74, fig. – 21, tab. – 6, listings. – 16 bibliography. – 12.

KEYWORDS: cryptography, encryption, symmetric cryptosystems, information security, confidentiality, data protection.

This qualification work is dedicated to researching the efficiency of modern symmetric cryptographic algorithms. Ensuring the security of information systems critically depends on selecting a reliable cryptographic algorithm, as this directly impacts the protection of confidential data. The work examines four popular symmetric cryptographic algorithms: AES, DES, RC4 and Blowfish.

The research involved a theoretical analysis of the algorithms, focusing on their architecture and key configuration parameters. Additionally, experimental studies were conducted to evaluate the performance of the algorithms in terms of execution time, memory usage, and CPU load.

The developed approach and obtained results can be utilized by system administrators and developers to enhance data protection in various information systems. This qualification work will also be valuable for students and professionals in the field of cybersecurity who seek to deepen their knowledge of symmetric algorithms and their application for data protection.

The conclusions and recommendations provided in this work will help improve the security level of information systems by selecting the most effective symmetric cryptographic algorithm based on specific requirements and constraints.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ІБ	—	Інформаційна Безпека
AES	—	Advanced Encryption Standard
DES	—	Data Encryption Standard
RC4	—	Rivest cipher 4
MAC	—	Message Authentication Code
ECB	—	Electronic Codebook Mode
CBC	—	Cipher Block Chaining Mode
CFB	—	Cipher Feedback Mode
OFB	—	Output Feedback Mode
CTR	—	Counter Mode
CPU	—	Central Processing Unit
RAM	—	Random Access Memory
ОС	—	Операційна Система

ВСТУП

У сучасному світі стрімкого розвитку інформаційних технологій і глобалізації інформаційних потоків, захист конфіденційних даних стає одним з найважливіших аспектів безпеки інформаційних систем. Криптографія, як наука про методи шифрування і розшифрування інформації, є основою для забезпечення цілісності, автентичності та конфіденційності даних. Одним із найбільш поширених підходів у криптографії є використання симетричних криптографічних алгоритмів, де одна і та сама секретна ключова інформація використовується як для шифрування, так і для розшифрування даних.

Симетричні криптографічні алгоритми, такі як AES, DES та їх модифікації, довели свою ефективність і надійність у численних сферах застосування, від захисту персональних даних до забезпечення безпеки фінансових транзакцій. Проте, зі збільшенням обсягів даних і підвищенням потужності обчислювальних ресурсів, зростає необхідність у постійному вдосконаленні існуючих алгоритмів та розробці нових методів шифрування, що здатні протистояти сучасним кіберзагрозам.

Метою даної роботи є дослідження ефективності сучасних симетричних криптографічних алгоритмів, визначення їхніх сильних та слабких сторін. У роботі будуть розглянуті теоретичні аспекти симетричної криптографії, проведено аналіз існуючих алгоритмів, а також здійснено порівняльний аналіз їх продуктивності та стійкості до різних типів атак. А саме буде виконано такі пункти:

- огляд сучасних симетричних криптоалгоритмів, таких як AES, DES, RC4, Blowfish;
- аналіз їх криптографічних властивостей;
- програмна реалізація криптоалгоритмів на мові Python, з метою дослідження ективності;
- проведення порівняльного аналізу характеристик, таких як безпека, продуктивність і придатність для практичного використання.

Актуальність теми даного дослідження обумовлена постійним зростанням кіберзагроз та необхідністю забезпечення надійного захисту інформації в умовах сучасного цифрового середовища.

Предметом дослідження є сучасні симетричні криптоалгоритми, які використовуються для шифрування повідомлень і файлів з метою захистити конфіденційність. Ці алгоритми є невідємною складовою інформаційної безпеки оскільки вони захищають інформацію від несанкціонованого доступу.

1 СИМЕТРИЧНІ КРИПТОСИСТЕМИ ТА ЇХ ЗНАЧЕННЯ ДЛЯ ІНФОРМАЦІЙНОЇ БЕЗПЕКИ

1.1 Поняття та задачі інформаційної безпеки та роль симетричних криптосистем в ній

Перш ніж обговорювати інформаційну безпеку, потрібно визначити поняття "інформація". У сфері інформаційних систем це зазвичай означає відомості, які є об’єктом зберігання, передавання та оброблення. Інформація може бути представлена в різних формах: комп'ютерні дані, листи, записи, досьє, формули, креслення, діаграми, моделі продукції, дисертації, судові документи тощо [1].

Інформаційна безпека, своєю чергою, означає стан захищеності життєво важливих інтересів людини, суспільства і держави. Це досягається шляхом запобігання шкоді, яка може виникнути через неповноту, невчасність або недостовірність інформації; негативний інформаційний вплив; негативні наслідки використання інформаційних технологій; несанкціоноване розповсюдження, використання, порушення цілісності, конфіденційності та доступності інформації. Інформаційна безпека є складною сферою діяльності, успіх у якій можливий лише за умови систематичного і комплексного підходу.

Розглянемо роль симетричних криптосистем в забезпеченні інформаційної безпеки, вони є дуже важливими завдяки своїм характеристикам, що дозволяють забезпечити конфіденційність, цілісність, автентичність та доступність даних. Вони є невід'ємною частиною багатьох сучасних інформаційних систем і застосувань, включаючи фінансові транзакції, комунікаційні протоколи, зберігання даних та інші сфери, де необхідний захист інформації.

Ознайомимось з основними задачами інформаційної безпеки:

- забезпечення доступності інформації;
- забезпечення цілісності інформації;
- забезпечення конфіденційності інформації;

Розглянемо детальніше кожну з них та роль симетричних криптосистем у цих задачах.

Захист конфіденційності – це забезпечення того, щоб доступ до інформації мали лише уповноважені особи, що запобігає несанкціонованому розголошенню даних [1].

Симетричні криптосистеми забезпечують конфіденційність інформації, шифруючи дані таким чином, що вони стають недоступними для неавторизованих осіб. Це особливо важливо в комунікаційних протоколах, таких як SSL/TLS, що використовуються для захисту переданих через Інтернет даних. Шифрування даних гарантує, що навіть якщо повідомлення буде перехоплене, зловмисник не зможе зрозуміти його зміст без відповідного ключа.

Крім захисту даних під час передачі, симетричні криптосистеми також використовуються для захисту даних при зберіганні. Шифрування дисків, файлів та баз даних дозволяє захистити інформацію від несанкціонованого доступу в разі крадіжки або втрати фізичних носіїв. Наприклад, повнодискове шифрування за допомогою AES використовується у багатьох сучасних операційних системах для захисту конфіденційних даних користувачів.

Цілісність даних є гарантією того, що інформація залишається точною та незмінною без дозволу уповноважених осіб. Це є одним із ключових аспектів інформаційної безпеки.

Симетричні криптосистеми дозволяють захистити дані від несанкціонованої модифікації. Використання таких методів, як Message Authentication Code (MAC), забезпечує перевірку цілісності даних, дозволяючи виявити будь-які зміни, внесені під час передачі або зберігання даних.

У сучасних інформаційних системах, передача даних через небезпечні мережі є повсякденною необхідністю. Симетричні криптосистеми, такі як AES, забезпечують високий рівень захисту даних під час передачі, шифруючи повідомлення до їх відправки. Це дозволяє забезпечити конфіденційність і цілісність переданих даних навіть у відкритих мережах, таких як Інтернет.

Доступність – це забезпечення своєчасного та безперебійного доступу до інформації для тих, хто має на це право.

Автентифікація є ключовим елементом інформаційної безпеки, оскільки вона дозволяє перевірити автентичність джерела інформації. Симетричні криптосистеми використовуються для створення криптографічних хешів або MAC, що дозволяє переконатися, що повідомлення було створене відправником, який володіє спільним секретним ключем, і що воно не було змінене під час передачі.

Щодо термінології, варто зауважити, що термін "комп'ютерна безпека" є надто вузьким. Комп'ютери – лише одна з частин інформаційних систем. Хоча наша увага буде зосереджена переважно на інформації, яка зберігається, обробляється і передається за допомогою комп'ютерів, її безпека визначається всіма складовими системи, особливо найслабкішою ланкою, якою найчастіше є люди.

Інформаційна безпека залежить не тільки від комп'ютерів, але й від підтримуючої інфраструктури, включаючи системи електро-, водо- і теплопостачання, кондиціонери, засоби комунікацій і, звичайно, обслуговуючий персонал. Хоча ця інфраструктура має самостійну цінність, нас цікавить, як вона впливає на здатність інформаційної системи виконувати свої функції.

Для забезпечення безпеки в інформаційних системах необхідно:

- захистити інформацію під час зберігання, обробки і передачі мережею
- підтвердити дійсність об’єктів даних і користувачів (автентифікація сторін, що встановлюють зв’язок);
- виявити і запобігти порушенням цілісності даних;
- захистити технічні пристрої і приміщення;
- захистити конфіденційну інформацію від витоків та несанкціонованого зняття за допомогою вбудованих електронних пристроїв;
- захистити програмне забезпечення від вірусів та програмних закладок;
- захистити від несанкціонованого доступу до інформаційного ресурсу і технічних засобів мережі, зокрема, до засобів керування, щоб уникнути зниження рівня інформаційної безпеки і самої мережі в цілому;
- організувати заходи для збереження конфіденційних даних.

Інформаційна безпека в рамках забезпечення працездатності ІС повинна забезпечувати захист від:

- порушення функціонування інформаційної системи через вплив на інформаційні канали, канали сигналізації, керування, віддалене завантаження баз даних, комутаційне обладнання, системне і прикладне програмне забезпечення;
- несанкціонованого доступу до інформаційних ресурсів і спроб використання мережевих ресурсів, що можуть призвести до витоку даних, порушення цілісності мережі та інформації, зміни функціонування підсистем розподілу інформації, доступності баз даних;
- руйнування вбудованих і зовнішніх засобів захисту;
- неправомірних дій користувачів і обслуговуючого персоналу мережі.

Пріоритети серед перерахованих задач інформаційної безпеки визначаються індивідуально для кожної конкретної ІС і залежать від вимог, що висуваються безпосередньо до інформаційних систем.

Також варто відзначити, що однаєї з основних переваг симетричних криптосистем є їх висока швидкість і ефективність у порівнянні з асиметричними алгоритмами. Це робить їх придатними для використання в умовах обмежених ресурсів, таких як мобільні пристрої, вбудовані системи та інші середовища, де важлива висока продуктивність і низьке споживання ресурсів.

Як висновок можна сказати, що інформаційна безпека є критично важливою складовою сучасного цифрового суспільства, що забезпечує захист конфіденційності, цілісності та доступності інформації. В умовах постійного розвитку технологій та зростання кількості кіберзагроз, ефективне управління інформаційною безпекою стає все більш важливим для організацій та окремих осіб, а симетричні криптосистеми є важливим інструментом для забезпечення інформаційної безпеки у багатьох сучасних застосуваннях. Їх використання дозволяє захистити конфіденційність, цілісність та автентичність даних, забезпечуючи високий рівень захисту в умовах зростаючих кіберзагроз.

1.2 Симетричні криптосистеми

Симетричні криптографічні алгоритми є фундаментальним аспектом сучасної криптографії і використовуються для захисту конфіденційності даних шляхом шифрування і дешифрування інформації за допомогою одного і того ж секретного ключа [2]. Далі наведено детальний опис роботи симетричних криптографічних алгоритмів також принцип зображено на рисунку 2.1.

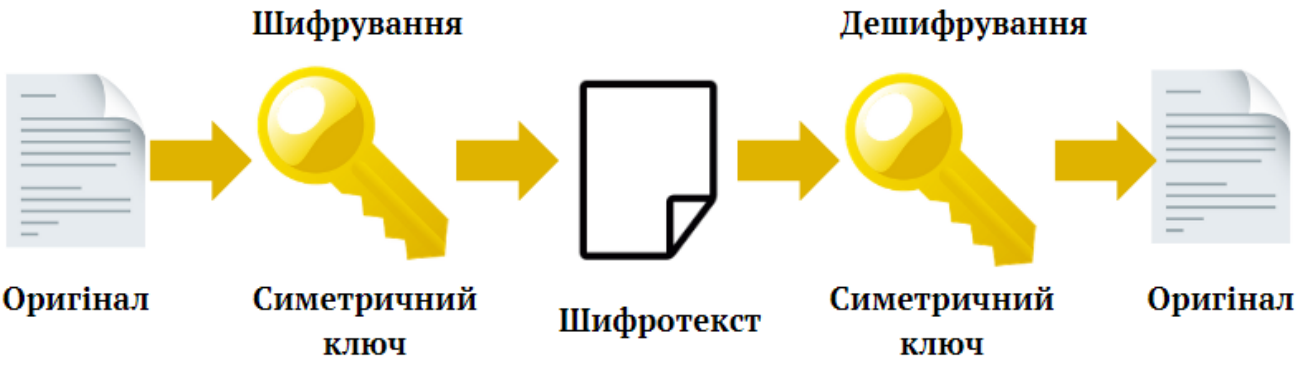

Рисунок 2.1 Принцип роботи симетричного шифрування

Основні поняття:

- Секретний ключ: Єдиний ключ, який використовується як для шифрування, так і для дешифрування даних. Обидві сторони (відправник і отримувач) повинні зберігати ключ у секреті.
- Блокове шифрування: Процес, при якому дані розбиваються на блоки фіксованої довжини (наприклад, 64 або 128 біт), і кожен блок шифрується окремо.
- Потокове шифрування: Процес, при якому дані шифруються по одному біту або байту за раз, часто використовуючи псевдовипадковий генератор чисел [2].

Блокові шифри:

- DES (Data Encryption Standard): Класичний алгоритм блокового шифрування, який використовує 56-бітовий ключ. Шифрування виконується за допомогою 16 раундів перестановок і замін.

- AES (Advanced Encryption Standard): Сучасний стандарт блокового шифрування, що підтримує ключі розміром 128, 192 і 256 біт. Використовує підстановки, перестановки та матричні операції, виконані за 10, 12 або 14 раундів залежно від розміру ключа.
- Blowfish: Підтримує змінний розмір ключа (від 32 до 448 біт) і використовує 16 раундів шифрування.

Потокові шифри:

- RC4: Алгоритм потокового шифрування, який використовує змінний розмір ключа (до 2048 біт). Шифрує дані, генеруючи псевдовипадковий ключовий потік, який комбінується з відкритим текстом.

Операційні режими блокових шифрів використовуються для підвищення безпеки блокові шифри використовуються в різних операційних режимах:

- ECB (Electronic Codebook Mode): Кожен блок шифрується незалежно, що може призводити до однакових шифротекстів для однакових блоків відкритого тексту, що знижує безпеку.
- CBC (Cipher Block Chaining Mode): Кожен блок відкритого тексту комбінується з попереднім блоком шифротексту перед шифруванням. Перший блок використовує вектор ініціалізації (IV).
- CFB (Cipher Feedback Mode): Перетворює блоковий шифр на потоковий, використовуючи попередній блок шифротексту для шифрування наступного блоку відкритого тексту.
- OFB (Output Feedback Mode): Подібний до CFB, але використовує вихід шифрувального процесу для шифрування наступного блоку.
- CTR (Counter Mode): Використовує лічильник, який інкрементується для кожного блоку, шифруючи значення лічильника і комбінуючи з відкритим текстом.

Симетричні алгоритми забезпечують високу швидкість шифрування та дешифрування, однак їх безпека залежить від секретності ключа. Поширені атаки:

- Брутфорс атака: Повний перебір усіх можливих ключів.

- Аналіз відомих відкритих текстів: Використання пар відомих відкритих текстів і відповідних шифротекстів для знаходження ключа.
- Аналіз вибраних відкритих текстів: Атака, коли атакуючий може отримати шифротексти для вибраних відкритих текстів.

Практичне застосування:

- VPN: Шифрування даних при передаванні через мережу.
- Диск шифрування: Захист даних на жорстких дисках.
- Електронна пошта: Захист листів від несанкціонованого доступу.
- Мобільні комунікації: Захист повідомлень і дзвінків.

2 АНАЛІЗ СУЧАСНИХ СИМЕТРИЧНИХ КРИПТОАЛГОРИТМІВ

Ми будемо досліджувати AES, DES, RC4 і BlowFish через їхнє історичне значення та вплив на розвиток криптографії. DES, прийнятий у 1977 році, став основою для сучасних симетричних алгоритмів, хоча сьогодні його вважають недостатньо безпечним. AES, стандарт з 2001 року, забезпечує високий рівень захисту і широко використовується у різних сферах. RC4, створений у 1987 році, був популярним завдяки простоті та швидкості, але виявився вразливим. BlowFish, розроблений у 1993 році, пропонував безпеку і гнучкість, але поступився місцем новішим алгоритмам. Дослідження цих алгоритмів дозволяє зрозуміти еволюцію криптографії та сучасні стандарти безпеки.

Шифри AES, DES, RC4 і Blowfish описані в різних криптографічних стандартах і специфікаціях. AES, описаний у федеральному стандарті США FIPS 197, став офіційним стандартом у 2001 році. DES був стандартизований у FIPS 46 у 1977 році з подальшими оновленнями і також згадується у стандартах ISO/IEC 18033-3. RC4 не має офіційного стандарту через його комерційну секретність, але був опублікований у різних технічних і академічних джерелах, включаючи RFC 7465. Blowfish, розроблений Брюсом Шнайєром у 1993 році, не має офіційного стандарту, але описаний у його книзі "Applied Cryptography" і широко використовується в програмних бібліотеках і додатках.

2.1 Криптосистема AES

AES (Advanced Encryption Standard) є одним з найпопулярніших і найбільш широко використовуваних алгоритмів симетричного шифрування у сучасній криптографії. Він був розроблений як заміна DES (Data Encryption Standard) та став офіційним стандартом шифрування для уряду США в 2001 році. Створений бельгійськими криптографами Вінсентом Рейменом та Жоаном Дейменом під назвою Rijndael, AES забезпечує високий рівень безпеки та продуктивності[3].

Основною особливістю AES є використання блокового шифрування з фіксованим розміром блоку 128 біт і можливістю вибору ключа довжиною 128,

192 або 256 біт. Ці три варіанти ключів визначають кількість раундів шифрування: 10 раундів для 128-бітного ключа, 12 раундів для 192-бітного ключа та 14 раундів для 256-бітного ключа[4].

Для кращого розуміння продемонструємо всі параметри криптосистеми у таблиці 2.1.

Таблиця 2.1 – Параметри криптосистеми AES

Параметр	AES-128	AES-192	AES-256
Розмір ключа	128 біт (16 байт)	192 біт (24 байт)	256 біт (32 байт)
Розмір блоку	128 біт (16 байт)	128 біт (16 байт)	128 біт (16 байт)
Кількість раундів	10	12	14
Розмір розширеного ключа	176 байт	208 байт	240 байт
Структура даних	4x4 матриця байтів	4x4 матриця байтів	4x4 матриця байтів
Операції в кожному раунді	SubBytes, ShiftRows, MixColumns, AddRoundKey	SubBytes, ShiftRows, MixColumns, AddRoundKey	SubBytes, ShiftRows, MixColumns, AddRoundKey
Операції в останньому раунді	SubBytes, ShiftRows, AddRoundKey	SubBytes, ShiftRows, AddRoundKey	SubBytes, ShiftRows, AddRoundKey

Опис параметрів:

- Розмір ключа: Кількість бітів у секретному ключі, який використовується для шифрування і дешифрування. Більший розмір ключа забезпечує вищий рівень безпеки.

- Розмір блоку: Кількість бітів у блоці даних, який шифрується за один раз. У AES розмір блоку завжди 128 біт.
- Кількість раундів: Кількість раундів шифрування, які виконуються для кожного блоку даних. Кількість раундів залежить від розміру ключа і впливає на безпеку і продуктивність алгоритму.
- Розмір розширеного ключа: Загальна кількість байтів, до якої розширюється початковий ключ для використання в кожному раунді шифрування.
- Структура даних: Представлення даних у вигляді 4x4 матриці байтів, яка використовується під час шифрування і дешифрування.
- Операції в кожному раунді: Основні кроки, які виконуються в кожному раунді шифрування: підстановка байтів (SubBytes), перестановка рядків (ShiftRows), змішування стовпців (MixColumns) і додавання раундового ключа (AddRoundKey).
- Операції в останньому раунді: Кроки, які виконуються в останньому раунді шифрування. В останньому раунді не виконується змішування стовпців (MixColumns) [3].

Процес шифрування в AES можна поділити на кілька основних етапів: підготовка ключів, початкова заміна, раундове шифрування та фінальна стадія.

Першим етапом є підготовка ключів або ключовий розклад (Key Expansion). Початковий ключ, обраний користувачем, розширюється до масиву ключів для кожного раунду шифрування. Цей процес включає в себе застосування підстановок і перестановок, які гарантують складність і безпеку шифрування.

Другим етапом є початкова заміна, де початковий блок відкритого тексту організовується у форму матриці розміром 4x4 байти. Ця матриця, відома як "стан" (state), є основною структурою даних, з якою працює AES протягом усіх раундів шифрування.

Кожен раунд шифрування включає кілька кроків: підстановку байтів (SubBytes), перестановку рядків (ShiftRows), змішування стовпців (MixColumns) і додавання раундового ключа (AddRoundKey).

Розглянемо кожен крок детальніше.

Підстановка байтів (SubBytes) – кожен байт у матриці стану замінюється на відповідний байт з таблиці підстановок (S-box). Ця таблиця розроблена таким чином, щоб забезпечити нелінійність та підвищити стійкість до криптоаналітичних атак[4].

Перестановка рядків (ShiftRows) – байти у кожному рядку матриці зсуваються циклічно вліво на певну кількість позицій. Це додає дифузії до алгоритму, змішуючи дані між стовпцями.

Змішування стовпців (MixColumns) – кожен стовпець матриці піддається лінійному перетворенню, що перемішує байти всередині кожного стовпця. Цей етап додає додаткову дифузію та гарантує, що зміни в одному байті поширюються на інші байти.

Додавання раундового ключа (AddRoundKey) – до матриці стану додається ключ поточного раунду за допомогою операції XOR. Це забезпечує тісний зв'язок між ключем і шифрованим текстом.

Останній раунд шифрування пропускає крок змішування стовпців (MixColumns), що забезпечує симетрію процесів шифрування та дешифрування.

Процес дешифрування в AES є зворотним до процесу шифрування. Він включає аналогічні кроки, але у зворотному порядку і з використанням обернених операцій. Замість підстановки байтів (SubBytes) використовується обернена підстановка (InvSubBytes), замість перестановки рядків (ShiftRows) використовується обернена перестановка (InvShiftRows), і замість змішування стовпців (MixColumns) використовується обернене змішування (InvMixColumns) [5].

AES є надзвичайно ефективним і може бути реалізований як у програмному, так і в апаратному забезпеченні. Його архітектура дозволяє паралельну обробку даних, що робить його швидким і продуктивним для великих обсягів даних. Завдяки своїй стійкості до різноманітних атак, включаючи атаки на основі відомих і вибраних відкритих текстів, AES залишається надійним вибором для захисту конфіденційної інформації у

багатьох додатках, від інтернет-комунікацій до захисту даних у хмарних сховищах.

Детально відображено процес шифрування та дешифрування на рисунку 2.2.

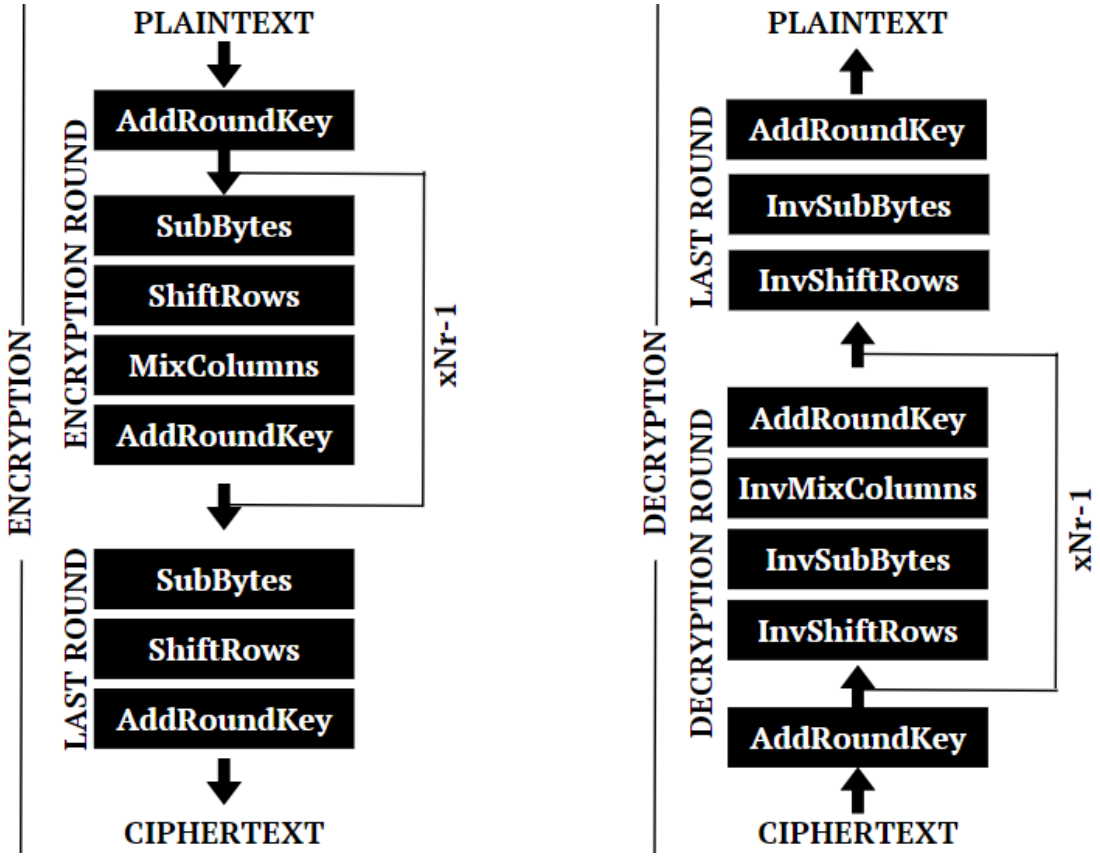

Рисунок 2.1 – Повний процес шифрування AES

Таким чином, AES став стандартом де-факто у галузі криптографії, забезпечуючи високу безпеку і продуктивність, що робить його ключовим елементом у захисті інформації у сучасному цифровому світі.

2.2 Криптосистема DES

Data Encryption Standard (DES) є одним із найвідоміших симетричних криптографічних алгоритмів, який було розроблено в середині 1970-х років. DES став офіційним стандартом шифрування, затвердженим Національним інститутом стандартів і технологій (NIST) США, і використовувався в багатьох сферах, включаючи банківські транзакції та захист урядових даних. Незважаючи

на те, що DES сьогодні вважається застарілим через свою вразливість до сучасних методів криптоаналізу, його структура та принципи роботи залишаються важливими для розуміння симетричних криптосистем[2].

Основою роботи DES є блокове шифрування, де дані обробляються у фіксованих блоках розміром 64 біти. Ключова довжина DES становить 56 біт, хоча початковий ключ, який вводиться користувачем, містить 64 біти. Вісім бітів використовуються як біти парності для виявлення помилок, що зменшує ефективну довжину ключа до 56 біт [2].

Процес шифрування за допомогою DES включає кілька етапів. Спочатку блок відкритого тексту проходить початкову перестановку (Initial Permutation, IP), яка перемішує біти блоку відповідно до фіксованого шаблону. Після початкової перестановки блок піддається 16 раундам обробки, кожен з яких включає операції перестановки, заміни та XOR з підключем. Підключі генеруються з основного ключа за допомогою процесу, що включає перестановку і стиснення ключа[5].

Кожен раунд обробки в DES включає кілька основних операцій. Блок розділяється на дві половини: ліву (L) і праву (R). Права половина проходить через функцію F, яка складається з кількох підоперацій. Спочатку права половина розширюється з 32 до 48 біт за допомогою операції розширення (Expansion E), що дозволяє змішувати біти і підвищує складність шифрування. Розширений блок XOR-ується з поточним підключем, після чого отриманий результат проходить через набір замінних блоків (S-boxes). S-блоки виконують нелінійні перетворення, замінюючи вхідні біти на інші біти відповідно до фіксованих таблиць. Після заміни блок піддається остаточній перестановці (P), яка перемішує біти, підвищуючи дифузію. Отриманий результат об'єднується з лівою половиною блоку за допомогою операції XOR[5].

Після завершення 16 раундів, ліві і праві половини блоку знову об'єднуються і піддаються зворотній початковій перестановці (Inverse Initial Permutation, IP-1), яка є зворотною до початкової перестановки. Результатом цього процесу є зашифрований блок шифротексту.

Декодування в DES здійснюється за допомогою зворотного процесу, де підключі застосовуються в зворотному порядку. Це забезпечує симетричність алгоритму, дозволяючи використовувати той самий ключ для шифрування і розшифрування.

Весь процес шифрування DES показано на рисунку 2.2.

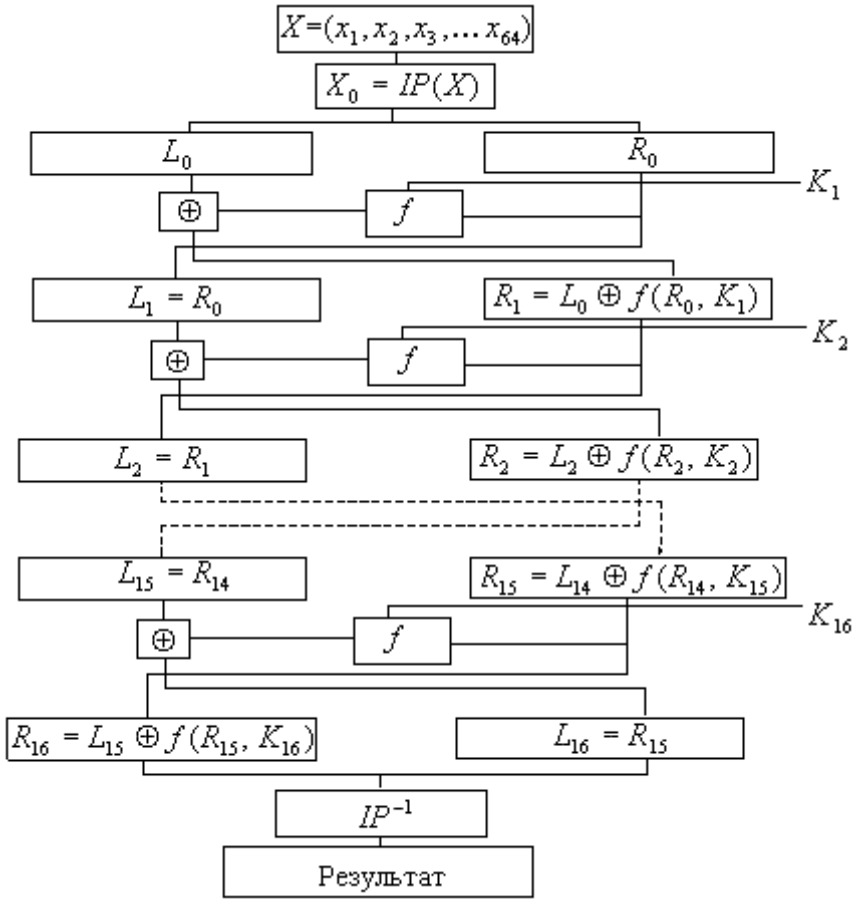

Рисунок 2.2 – Процес шифрування криптоалгоритму DES

Незважаючи на свою історичну значимість, DES має кілька відомих вразливостей. Найбільш серйозною є відносно невелика довжина ключа, яка робить алгоритм вразливим до атак перебору (brute-force attack). Сучасні обчислювальні потужності дозволяють швидко перебрати всі можливі ключі і знайти правильний. Крім того, існують відомі криптоаналітичні атаки, такі як диференціальний криптоаналіз, які можуть ефективно зламати DES з меншими зусиллями, ніж повний перебір ключів[5].

Для того щоб зрозуміти походження проблем у криптосистемі DES, в першу чергу потрібно ознайомитися з параметрами агоритму. В них можна буде зауважити слабкі сторони алгориму.

З парематрими крипто системи DES можна ознайомитися у таблиці 2.2.

Таблиця 2.2 Парметри алгоримту DES

Параметр	Опис
Тип алгоритму	Симетричний блоковий шифр
Розмір блоку	64 біти (8 байтів)
Довжина ключа	64 біти (8 байтів), з яких 56 біт використовуються для шифрування, а 8 біт є бітами парності
Кількість раундів	16
Структура	Мережа Фейстеля
Початкова перестановка (IP)	Є
Зворотна початкова перестановка (IP-1)	Є
Операції розширення (Expansion E)	Збільшує 32-бітну половину до 48 біт
S-блоки	8 блоків заміщення, кожен з яких перетворює 6-бітний вхід на 4-бітний вихід
P-перестановка	Перестановка 32-бітного виходу після S-блоків
Тип підключа	16 різних підключів, по 48 біт кожен, генеруються з основного ключа
Криптоаналіз	Вразливий до атак перебору (brute-force attack), диференціального криптоаналізу та інших методів
Стійкість	Невисока за сучасними стандартами через коротку довжину ключа і відомі вразливості
Швидкодія	Відносно висока, але нижча, ніж у сучасних алгоритмів, таких як AES

Опис параметрів:

- Тип алгоритму: DES є симетричним блоковим шифром, що означає, що він використовує один і той самий ключ для як шифрування, так і

розшифрування, а також обробляє вхідні дані блоками фіксованого розміру.

- Розмір блоку: Розмір блоку у DES становить 64 біти, що еквівалентно 8 байтам або 16 шістнадцятковим символам.
- Довжина ключа: Ключ у DES складається з 64 біт, але лише 56 біт використовуються для фактичного шифрування. Решта 8 біт використовуються для бітів парності, які використовуються для перевірки цілісності ключа.
- Структура: DES використовує мережу Фейстеля, що є спеціалізованою структурою для шифрування блоків даних.
- Тип підключа: DES використовує 16 різних підключів, кожен з яких генерується з основного ключа відповідно до внутрішньої процедури.
- Криптоаналіз: DES вразливий до атак перебору та диференціального криптоаналізу, що обмежує його застосування в сучасних системах.
- Стійкість: Хоча DES був розроблений з високими стандартами безпеки на момент його створення, сучасні обчислювальні можливості зробили його вразливим до криптоаналітичних атак.

З огляду на ці вразливості, DES було замінено більш безпечними алгоритмами, такими як Triple DES (3DES) і Advanced Encryption Standard (AES). Однак вивчення DES залишається важливим для розуміння принципів блокового шифрування та розвитку сучасних криптографічних методів.

2.3 Криптосистема RC4

RC4 (Rivest Cipher 4) є одним з найпопулярніших потокових шифрів, розроблених Роном Рівестом у 1987 році для компанії RSA Security. Його популярність обумовлена високою швидкістю шифрування і простотою реалізації, що зробило його широко використовуваним у різних додатках, включаючи протоколи SSL/TLS і WEP/WPA для захисту бездротових мереж [2].

RC4 працює на основі псевдовипадкової генерації потоку ключів, який потім комбінується з відкритим текстом для отримання шифротексту. Процес

шифрування і дешифрування в RC4 є симетричним, тобто застосовується той самий ключ для обох процесів. Основна ідея полягає в тому, щоб генерувати псевдовипадковий ключовий потік (keystream), який потім використовується для шифрування або дешифрування даних шляхом побітового додавання (XOR) з відкритим або шифрованим текстом[6].

Першим етапом у RC4 є ініціалізація ключового масиву (Key-Scheduling Algorithm, KSA). Вхідний ключ змінної довжини (зазвичай від 40 до 256 біт) використовується для ініціалізації масиву S, який містить усі можливі байти від 0 до 255. Початковий масив S заповнюється послідовністю чисел від 0 до 255. Потім масив S перемішується за допомогою вхідного ключа, що забезпечує початкове розміщення елементів у масиві.

Процес перемішування масиву S здійснюється через обмін елементів між різними індексами в масиві. Цей обмін базується на значеннях ключа і індексів масиву. Після цього етапу масив S стає псевдовипадковим, що гарантує складність передбачення розміщення байтів у масиві.

Другим етапом є генерація псевдовипадкового ключового потоку (Pseudo-Random Generation Algorithm, PRGA). Цей етап полягає у створенні послідовності байтів, яка використовується як ключовий потік. Для кожного байта відкритого тексту генерується відповідний байт ключового потоку, який комбінується з байтом відкритого тексту за допомогою операції XOR, що дає байт шифротексту. Процес генерування ключового потоку також базується на масиві S, який постійно змінюється під час шифрування.

Генерація ключового потоку виконується через обхід масиву S з використанням двох індексів, які постійно змінюються і обмінюються елементами масиву. Кожен обхід масиву генерує новий байт ключового потоку, який є псевдовипадковим і залежить від початкового ключа та попередніх значень масиву S[6].

З процесом шифрування можна ознайомитися на рисунку 2.3.

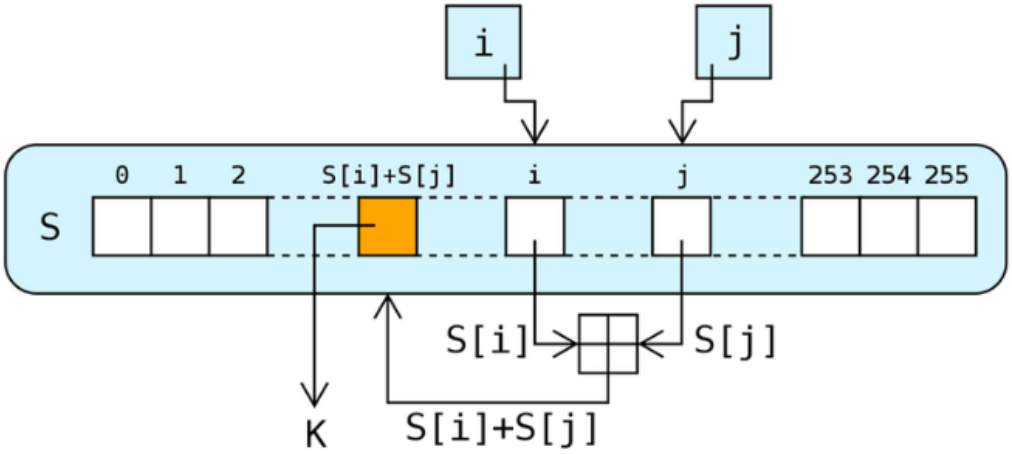

Рисунок 2.3 – Процес шифрування RC4

З парематрими крипто системи RC4 можна ознайомитися у таблиці 2.3.

Таблиця 2.3 Параметри криптосистеми RC4

Параметр	Значення
Розробник	Рон Рівест (Ron Rivest)
Рік розробки	1987
Тип алгоритму	Потоковий шифр
Розмір ключа	Змінний (зазвичай від 40 до 256 біт)
Розмір блоку	Потоковий (шифрує дані побайтово або побітово)
Ключовий масив (S)	Масив з 256 байтів (0-255)
Початкова ініціалізація	Ініціалізація ключового масиву (Key-Scheduling Algorithm, KSA)
Генерація ключового потоку	Псевдовипадкова генерація ключового потоку (Pseudo-Random Generation Algorithm, PRGA)
Продуктивність	Висока, ефективна для програмної реалізації
Основні операції	Обмін (swap) елементів масиву, побітове додавання (XOR)
Вразливості	Слабкості у перших байтах ключового потоку, атаки на WEP/WPA
Типові застосування	SSL/TLS, WEP/WPA (застарілі), деякі протоколи швидкого шифрування

Опис параметрів:

- Тип алгоритму: RC4 є потоковим шифром, що означає, що він шифрує дані побайтово або побітово.
- Розмір ключа: Ключ у RC4 має змінну довжину, зазвичай від 40 до 256 біт, що дозволяє адаптувати алгоритм до різних рівнів безпеки.
- Розмір блоку: RC4 не використовує блоки фіксованого розміру, оскільки є потоковим шифром.
- Ключовий масив (S): Масив S містить 256 байтів (значення від 0 до 255), які використовуються для генерації ключового потоку.
- Початкова ініціалізація: Ініціалізація ключового масиву (KSA) – це процес перемішування масиву S на основі вхідного ключа.
- Генерація ключового потоку: Псевдовипадкова генерація ключового потоку (PRGA) – це процес створення ключового потоку для шифрування даних.
- Основні операції: Основні операції включають обмін (swap) елементів масиву S і побітове додавання (XOR) для шифрування даних.

RC4 забезпечує високу швидкість шифрування завдяки своїй простій структурі, що робить його ефективним для використання у програмному забезпеченні.

2.4 Криптосистема Blowfish

Blowfish є одним з найвідоміших симетричних блокових шифрів, розроблених Брюсом Шнайером у 1993 році. Він був створений як швидкий і ефективний алгоритм для шифрування даних, що не тільки забезпечує високий рівень безпеки, але й легко реалізується як у програмному, так і в апаратному забезпеченні. Blowfish є вільно доступним для використання, що сприяло його широкому впровадженню у різноманітних додатках[6].

Основною особливістю Blowfish є його змінний розмір ключа, який може бути від 32 до 448 біт. Ця гнучкість дозволяє налаштовувати алгоритм для досягнення необхідного балансу між безпекою і продуктивністю. Blowfish використовує блоки даних розміром 64 біти, що означає, що він шифрує дані по 8 байтів за один раз.

Процес шифрування в Blowfish складається з двох основних частин: ініціалізація ключових таблиць і сам процес шифрування, який включає кілька раундів перетворень. Початково, алгоритм генерує кілька ключових таблиць на основі вхідного ключа. Ці таблиці, відомі як підключі (subkeys), використовуються для обробки даних під час шифрування та дешифрування.

Перший етап ініціалізації включає генерування двох масивів: один масив P-Box, який містить 18 32-бітових значень, і чотири S-Box, кожен з яких містить 256 32-бітових значень. Початкові значення цих масивів є фіксованими і визначені алгоритмом. Вхідний ключ використовується для зміни цих початкових значень у процесі ключового розкладу.

Ключовий розклад починається з послідовного додавання байтів вхідного ключа до елементів масиву P-Box. Потім, використовуючи змінені значення P-Box і початкові значення S-Box, алгоритм проходить через кілька раундів шифрування з нульовими вхідними даними, щоб створити остаточні значення масивів P-Box і S-Box. Цей процес гарантує, що ключові таблиці добре перемішані і залежать від вхідного ключа.

Після ініціалізації ключових таблиць починається процес шифрування. Дані, які треба зашифрувати, розділяються на блоки по 64 біти. Кожен блок проходить через 16 раундів перетворень, що складаються з різних нелінійних і лінійних операцій. У кожному раунді блок даних поділяється на дві частини: ліву і праву. Ліва частина блоку комбінується з одним з підключів P-Box, а потім проходить через послідовність нелінійних перетворень, які включають використання S-Box. Результат цих перетворень комбінується з правою частиною блоку за допомогою операції XOR, після чого ліва і права частини блоку міняються місцями. Цей процес повторюється для кожного з 16 раундів, причому на останньому раунді ліва і права частини не міняються місцями.

Після завершення всіх раундів, обидві частини блоку комбінуються з останніми двома підключами P-Box, що завершує процес шифрування. В результаті, кожен 64-бітний блок відкритого тексту перетворюється на зашифрований блок тієї ж довжини.

Процес дешифрування в Blowfish є зворотним до процесу шифрування і використовує ті ж самі ключові таблиці. Він також включає 16 раундів перетворень, але у зворотному порядку, з відповідними обміном лівої і правої частин блоку[6].

Повний процес шифрування продемонстровано на рисунку 2.4

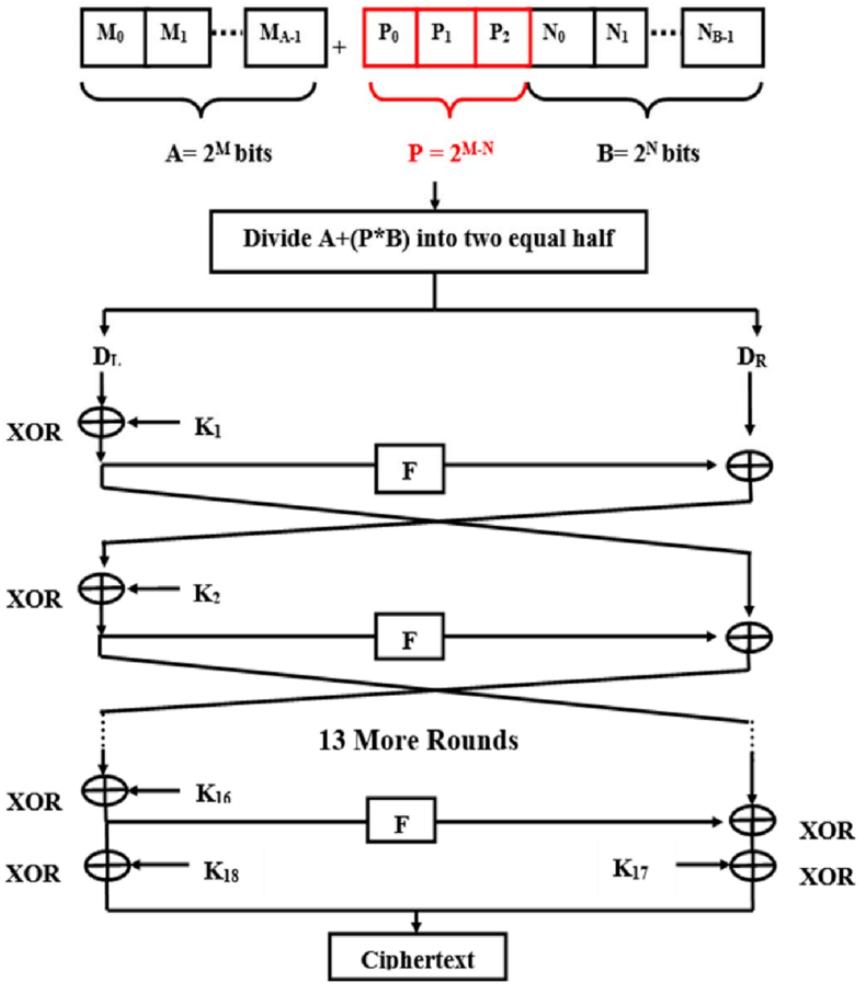

Рисунок 2.4 – Процес шифрування криптоалгоритму Blowfish

Також варто ознайомитися з параметрами цього криптоалгоритму, див. таблицю 2.4.

Таблиця 2.4 Параметри криптоалгоритму Blowfish

Параметр	Значення
Розробник	Брюс Шнайер (Bruce Schneier)
Рік розробки	1993
Тип алгоритму	Симетричний блоковий шифр
Розмір ключа	Змінний (від 32 до 448 біт)
Розмір блоку	64 біти (8 байтів)
Кількість раундів	16
P-Box	Масив з 18 32-бітних значень
S-Box	4 масиви по 256 32-бітних значень кожен
Основні операції	Підстановка (S-Box), перестановка (P-Box), XOR
Продуктивність	Висока, ефективна для програмної і апаратної реалізації
Безпека	Висока, за умови правильного вибору ключа
Вразливості	Розмір блоку 64 біти може бути недоліком для великих обсягів даних

Опис параметрів:

- Тип алгоритму: Blowfish є симетричним блоковим шифром, що означає використання одного ключа для шифрування і дешифрування.
- Розмір ключа: Алгоритм дозволяє використовувати ключі різної довжини, від 32 до 448 біт, що надає гнучкість у налаштуванні рівня безпеки.
- Розмір блоку: Blowfish шифрує дані блоками по 64 біти (8 байтів), що є типовим для блокових шифрів.
- Кількість раундів: Процес шифрування включає 16 раундів перетворень, які забезпечують високу стійкість до атак.
- Основні операції: Blowfish використовує підстановку і перестановку для створення складного шифрованого тексту. Вони включають

використання S-Box для нелінійної підстановки і P-Box для лінійної перестановки, а також операції XOR для комбінування даних.

- Продуктивність: Blowfish відомий своєю високою продуктивністю і ефективністю як у програмній, так і в апаратній реалізації.
- Безпека: Алгоритм забезпечує високий рівень безпеки, за умови правильного вибору довжини ключа і його належної секретності.

2.5 Криптостійкість обраних алгоритмів шифрування

Криптостійність шифрів – це властивість шифрувальних алгоритмів залишатися стійкими до злому навіть у випадку, коли зловмисник має доступ до шифрованого тексту, знає алгоритм шифрування і навіть може використовувати обмежені ресурси (такі як час або обчислювальну потужність) для атаки.

Основні аспекти криптостійкості включають:

- Секретність ключа: Шифрувальний ключ повинен бути важким до вгадування для зловмисника, який намагається дешифрувати дані.
- Стійкість до криптоаналізу: Це означає, що атака на шифр, яка використовує методи криптоаналізу (наприклад, знаходження ключа шифрування або шукання вразливостей в самому алгоритмі), має бути важкою або практично неможливою.
- Різноманітність ключів і довжина ключа: Використання довгих ключів і забезпечення їхньої випадковості підвищує стійкість шифру.
- Секретність алгоритму: Сам алгоритм шифрування повинен бути відомий лише відповідним сторонам (наприклад, виробникам програмного забезпечення і криптографам) і не повинен бути доступним для загального використання або аналізу.
- Стійкість до відомих атак: Шифр має захищатися від таких атак, як brute-force (груба сила), знання відкритого тексту, адаптивний аналіз і т.д.

Загалом, криптостійкість шифру є критично важливою для забезпечення безпеки конфіденційної інформації та захисту від несанкціонованого доступу до даних.

Оцінка криптостійкості шифрів AES, DES, RC4 і Blowfish від 1 до 10 базується на сучасних знаннях про їх безпеку, включаючи відомі атаки та їх здатність забезпечувати конфіденційність даних.

- AES: 10/10. Він є сучасним стандартом шифрування, затвердженим NIST. Він використовує ключі довжиною 128, 192 або 256 біт, що робить його дуже стійким до атак повного перебору (brute force). Відсутність серйозних уразливостей та його широке прийняття у всьому світі підтверджують його високу криптостійкість.
- DES: 3/10. Він був стандартом шифрування протягом багатьох років, але його 56-бітний ключ робить його вразливим до атак повного перебору, які можуть бути здійснені за відносно короткий час з використанням сучасних обчислювальних потужностей. Через ці вразливості DES більше не вважається безпечним для більшості застосувань.
- Blowfish: 7/10. Є досить стійким шифром і використовує змінну довжину ключа до 448 біт. Він не має відомих серйозних вразливостей, але є деякі питання щодо його безпеки у використанні з малими блоками даних. Хоча він залишається надійним варіантом, існують новіші і більш ефективні алгоритми, такі як AES, які пропонують вищий рівень безпеки.
- RC4: 2/10. Він був популярним потоковим шифром через його простоту і швидкість, але виявлені вразливості, такі як атаки на відновлення ключа та відомі слабкості у генерації ключів, значно знизили його криптостійкість. Більшість сучасних протоколів більше не рекомендують використовувати RC4.

У цьому розділі було проведено детальне дослідження ключових аспектів сучасних методів шифрування: оцінка криптостійкості, ефективності та потенційних вразливостей алгоритмів.

3 ПРАКТИЧНЕ ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ СИМЕТРИЧНИХ АЛГОРИТМІВ ШИФРУВАННЯ

3.1 Вибір середовища розробки програми та технічні характеристики пристроїв для аналізу алгоритмів шифрування

Python – мова програмування та VS-Code – інтегроване середовище розробки, були обрані для програмної реалізації алгоритмів і методів дослідження їх ефективності.

Python – високорівнева мова програмування, відома своєю простотою та чіткістю синтаксису. Python, який був створений Гвідо ван Россумом і випущений у 1991 році, швидко набув популярності див. рисунок 3.1.

Завдяки своїй потужності для досвідчених програмістів і простоті використання для новачків. Багато парадигм програмування підтримуються в Python, включаючи об’єктно-орієнтоване, функціональне та імперативне програмування.

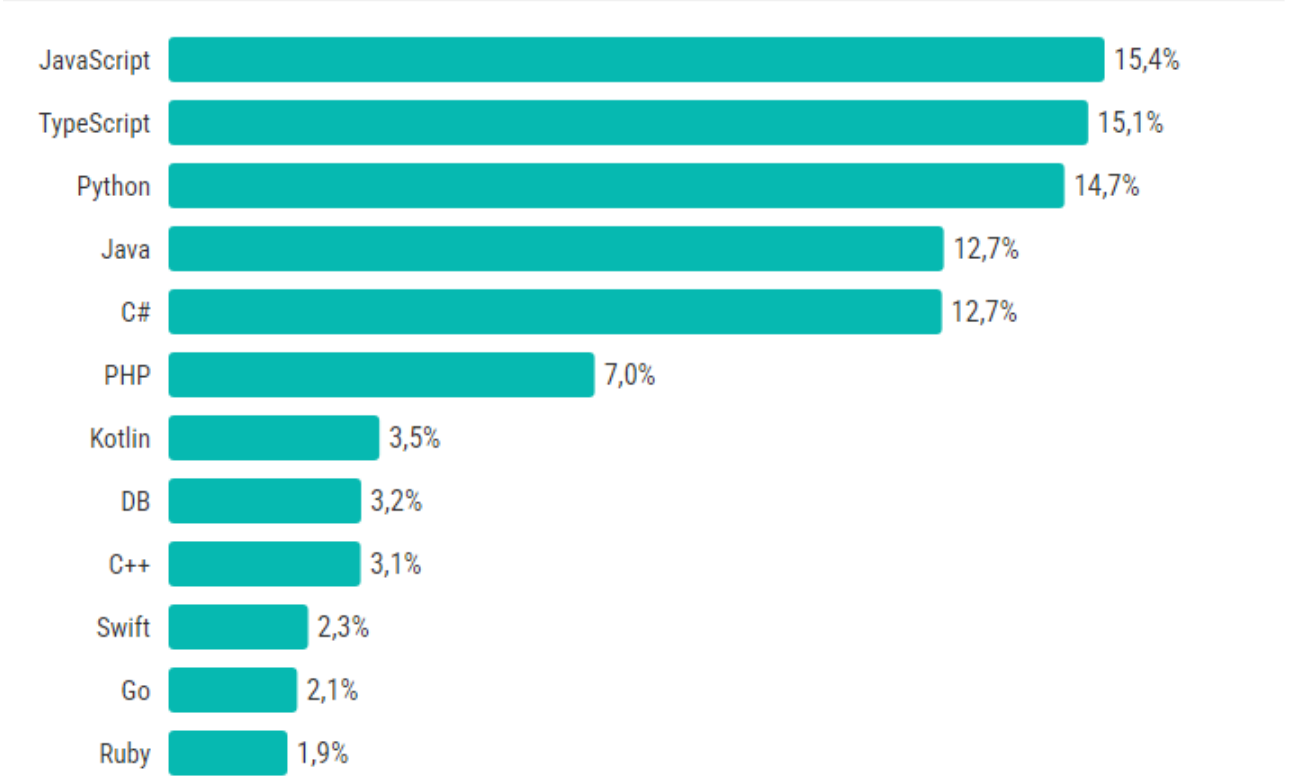

Рисунок 3.1 – Рейтинг мов програмування на 2024 рік

Для отримання об’єктивних і репрезентативних результатів необхідно вибрати методи оцінки ефективності алгоритмів шифрування. Час виконання, використання CPU та використання пам'яті були обрані для цієї роботи.

Час виконання є важливою метрикою для шифрування. Швидкість шифрування впливає на загальну продуктивність системи. Якщо процес шифрування займає багато часу, це може сповільнити передачу даних або відповідь системи.

Використання CPU є важливою метрикою, тому що системи які використовують шифрування не завжди є продуктивними, тому при великому навантаженні може створити незручності, абож взагалі відмовити система.

Використання памятті також є дуже важливою метрикую для шифрування з причини того, що дуже багато систем дял шифрування не володіють великим запасом оперативної пам’яті, але мають виконувати багато задач по шифруванню паралельно, тому потрібно мімінізувати використання пам’яті з цілью зберегти продуктивність.

Також роль грає те на якому девайсі буде запущено процес шифрування, тобто яка система використвується, запаси оперативної пам’яті, продуктивність процесора. Всі ці параметри є невідємною частиною виконання любого процесу на компютері, сервері, мікро компютері та інші.

Для цього дослідження ми будемо використовувати декалька різних систем, починаючи з перслнального компютера закінчуючи хмарними сервером їхні характеристики будуть вказані в процесі досладження.

3.2 Розробка програми, що реалізує досліджуванні алгоритми шифрування та вимірює ефективність алгоритмів

Як зазначалося раніше, Python був обраний для проведення тестів ефективності алгоритмів шифоування через свою простоту, широку підтримку криптографічних бібліотек і потужні інструменти для аналізу продуктивності. Для цієї роботи використовуються такі основні бібліотеки: pycryptodome, base64, os, timeit, psutil.

В програмі присутні декілька функцій які відповідають за шифрування і дешифрування алгоритмом AES у режимі CBC і оцінювання витрат русурсів компютера, з цілю оцінювання ефективності криптоалгоритму.

Функціїями якими реалізується AES є pad(data) яка додає заповнення до тексту, щоб його довжина була кратною 16 байтам див. лістинг 3.1.

Лістинг 3.1 Код функції заповнення тексту

```
def pad(data):
    padding_length = 16 - len(data) % 16
    padding = chr(padding_length) * padding_length
    return data + padding
```

Далі функція unpad(data), яка видаляє заповнення з тексту – код програми наведено в лістингу 3.2.

Лістинг 3.2 Код функції видалення заповнення тексту

```
def unpad(data):
    padding_length = ord(data[-1])
    return data[:-padding_length]
```

Наступна функція encrypt(plain_text, key) воно відповідає за шифрування, і виконує саме додає заповнення до відкритого тексту, генерує випадковий вектор ініціалізації (IV), створює новий AES-шифр з ключем і IV у режимі CBC, шифрує заповнений текст, повертає результат у форматі Base64 – код програми наведено в лістинку 3.3.

Лістинг 3.3 Функція шифрування AES

```
def encrypt(plain_text, key):
    padded_text = pad(plain_text)
    iv = get_random_bytes(16)
    cipher = AES.new(key, AES.MODE_CBC, iv)
    encrypted_text = cipher.encrypt(padded_text.encode('utf-8'))
    return base64.b64encode(iv + encrypted_text).decode('utf-8')
```

Функція дешифрування – decrypt(encrypted_text, key) декодує Base64-закодований текст, витягує IV та зашифрований текст, створює новий AES-шифр з ключем і IV у режимі CBC, дешифрує зашифрований текст, видаляє заповнення, повертає відкритий текст, див. лістинг 3.4.

Лістинг 3.4 Функція дешифрування AES

```
def decrypt(encrypted_text, key):
    encrypted_data = base64.b64decode(encrypted_text)
    iv = encrypted_data[:16]
    encrypted_text = encrypted_data[16:]
    cipher = AES.new(key, AES.MODE_CBC, iv)
    padded_text = cipher.decrypt(encrypted_text).decode('utf-8')
    plain_text = unpad(padded_text)
    return plain_text
```

Функція, що відносяться до оцінки ефективності, а саме measure_performance(func, *args) – використовує бібліотеку psutil для вимірювання використання пам'яті та процесора, вимірює час виконання функції з використанням timeit, обчислює середнє значення завантаження процесора до і після виконання функції, повертає час виконання, середнє використання процесора та використану пам'ять, ця функція буде надалі використовуватися у всіх тестах див. лістинг 3.5.

Лістинг 3.5 Функція вимірювання ефективності

```
def measure_performance(func, *args):
    process = psutil.Process(os.getpid())
    start_memory = process.memory_info().rss
    start_cpu_percent = psutil.cpu_percent(interval=None)
    execution_time = timeit.timeit(lambda: func(*args), number=1)
    end_memory = process.memory_info().rss
    end_cpu_percent = psutil.cpu_percent(interval=None)
    avg_cpu_percent = (end_cpu_percent - start_cpu_percent)
    memory_used = end_memory - start_memory
    return execution_time, avg_cpu_percent, memory_used
```

Що стосується покового шифру RC4 програма виконує такі функції pad(data): функція додає заповнення (padding) до даних, щоб їх довжина була кратною 16 байтам. Заповнення необхідне, оскільки деякі шифри вимагають, щоб довжина вхідних даних була кратною певному числу байт див лістинг 3.6.

Лістинг 3.6 Функція додає заповнення до даних

```
def pad(data):
    padding_length = 16 - len(data) % 16
    padding = chr(padding_length) * padding_length
    return data + padding
```

Функція unpad видаляє заповнення з даних після їх дешифрування наведено в лістингу 3.7.

Лістинг 3.7 Функції видалення заповнення тексту

```
def unpad(data):
    padding_length = ord(data[-1])
    return data[:-padding_length]
```

Функція encrypt(plain_text, key): Шифрує вхідний текст за допомогою ключа, спочатку додається заповнення до тексту, створюється новий RC4 шифр з використанням ключа, текст шифрується і кодується у base64 формат код наведено в лістингу 3.8.

Лістинг 3.8 Функції шифрування тексту

```
def encrypt(plain_text, key):
    padded_text = pad(plain_text)
    cipher = ARC4.new(key)
    encrypted_text = cipher.encrypt(padded_text.encode('utf-8'))
    return base64.b64encode(encrypted_text).decode('utf-8')
```

Функція decrypt(encrypted_text, key): дешифрує зашифрований текст за допомогою ключа, декодує зашифрований текст з base64 формату, створюється новий RC4 шифр з використанням того ж ключа, текст дешифрується, видаляється заповнення код наведено в лістингу 3.9.

Лістинг 3.9 Функції шифрування тексту

```
def decrypt(encrypted_text, key):
    encrypted_data = base64.b64decode(encrypted_text)
    cipher = ARC4.new(key)
    padded_text = cipher.decrypt(encrypted_data).decode('utf-8')
    plain_text = unpad(padded_text)
    return plain_text
```

Фукція measure_performance(func, *args) Вимірює продуктивність вказаної крипто системи і є не зміною для всіх реалізацій код наведено в лістингу 3.5.

В свою чергу програмна реалізація DES, працює таким чином – pad(data): Додає заповнення до тексту, щоб його довжина була кратною 8 байтам (блоковий розмір для DES).код функції наведено в лістингу 3.10.

Лістинг 3.10 Функції заповнення тексту

```
def pad(data):
    padding_length = 8 - len(data) % 8
    padding = chr(padding_length) * padding_length
    return data + padding
```

Функція unpad(data) – видаляє додане заповнення після дешифрування див лістинг 3.11.

Лістинг 3.11 Функції видалення заповнення тексту

```
def unpad(data):
    # Removing the padding
    padding_length = ord(data[-1])
    return data[:-padding_length]
```

Фукція encrypt(plain_text, key) – додає заповнення до тексту, генерує випадковий вектор ініціалізації (IV), створює новий об'єкт DES cipher з використанням ключа та IV, шифрує заповнений текст, повертає результат у вигляді base64-кодування, що містить IV та зашифрований текст код наведено в лістингу 3.12.

Лістинг 3.12 Функція шифрування DES

```
def encrypt(plain_text, key):
    padded_text = pad(plain_text)
    iv = get_random_bytes(8)
    cipher = DES.new(key, DES.MODE_CBC, iv)
    encrypted_text = cipher.encrypt(padded_text.encode('utf-8'))
    return base64.b64encode(iv + encrypted_text).decode('utf-8')
```

Функція decrypt(encrypted_text, key) – декодує base64-кодовані дані, витягує IV та зашифрований текст, створює новий об'єкт DES cipher з використанням ключа та IV, дешифрує текст, видаляє заповнення, щоб отримати початковий текст код наведено в лістингу 3.13.

Лістинг 3.13 Функція дешифрування DES

```
def decrypt(encrypted_text, key):
    encrypted_data = base64.b64decode(encrypted_text)
    iv = encrypted_data[:8]
    encrypted_text = encrypted_data[8:]
    cipher = DES.new(key, DES.MODE_CBC, iv)
    padded_text = cipher.decrypt(encrypted_text).decode('utf-8')
    plain_text = unpad(padded_text)
    return plain_text
```

Функція замірювання ефективності реалізована як в попередніх криптосистемах, код наведено в лістингу 3.5.

Програмна реалізація крипто системи Blowfish має такий вигляд функція pad(data) додає заповнення до тексту, щоб його довжина була кратною 8 байтам (блоковий розмір для Blowfish). Наведено в лістингу 3.14.

Лістинг 3.14 Функції заповнення тексту

```
def pad(data):
    padding_length = 8 - len(data) % 8
    padding = chr(padding_length) * padding_length
    return data + padding
```

Функція unpad(data): Видаляє додане заповнення після дешифрування код наведено в лістингу 3.15.

Лістинг 3.15 Функції заповнення тексту

```
def unpad(data):
    padding_length = ord(data[-1])
    return data[:-padding_length]
```

Функція encrypt(plain_text, key) – додає заповнення до тексту, генерує випадковий вектор ініціалізації (IV), створює новий об'єкт Blowfish cipher з використанням ключа та IV, шифрує заповнений текст, повертає результат у вигляді base64-кодування, що містить IV та зашифрований текст код наведено у лістингу 3.16.

Лістинг 3.16 Функції шифрування тексту

```
def encrypt(plain_text, key):
    padded_text = pad(plain_text)
    iv = get_random_bytes(8)
```

Продовження лістингу 3.16

```
cipher = Blowfish.new(key, Blowfish.MODE_CBC, iv)
encrypted_text = cipher.encrypt(padded_text.encode('utf-8'))
return base64.b64encode(iv + encrypted_text).decode('utf-8')
```

Фукція decrypt(encrypted_text, key) відповідає за шифрування а саме виконує, декодування base64-кодовані дані, витягує IV та зашифрований текст, створює новий об'єкт Blowfish cipher з використанням ключа та IV, дешифрує текст, видаляє заповнення, щоб отримати початковий текст.

Функція яка відповідає за заміри ефективності ідентична попереднім реалізація код наведно у лістнгу 3.5.

В цілому реалізація цих алгоритмів схожа, але враховує всі особливості, тому дослідження будуть проходити в рівних умовах.

3.3 Тестування та оцінка ефективності шифрувальних алгоритмів

Для оцінки ефективності шифрувальних алгоритмів AES, DES, RC4 і Blowfish було проведено дослідження. Ми використовували спеціально розроблену програму для вимірювання часу виконання, споживання процесорного часу, використання пам’яті та використання CPU для кожного алгоритму.

Ми проведемо тестування на різних пристроях. Це дозволить зібрати дані про продуктивність і використання ресурсів цих алгоритмів у різних умовах і на різному обладнанні. Такий підхід забезпечить більш повне розуміння їхніх переваг і недоліків, що є ключовим для прийняття обґрунтованих рішень щодо їхнього використання в реальних системах з різними технічними характеристиками.

Для початку проведемо дослідження на звичайному домашньому компютері, його характеристики:

- Процесор: Intel core i5-9400f
- Оперетивна память: 16Gb
- Опереаційна система: Windows

Першим проведемо заміри крипто алгоритму AES, результати тестуваня зображено на рисунку 3.2.

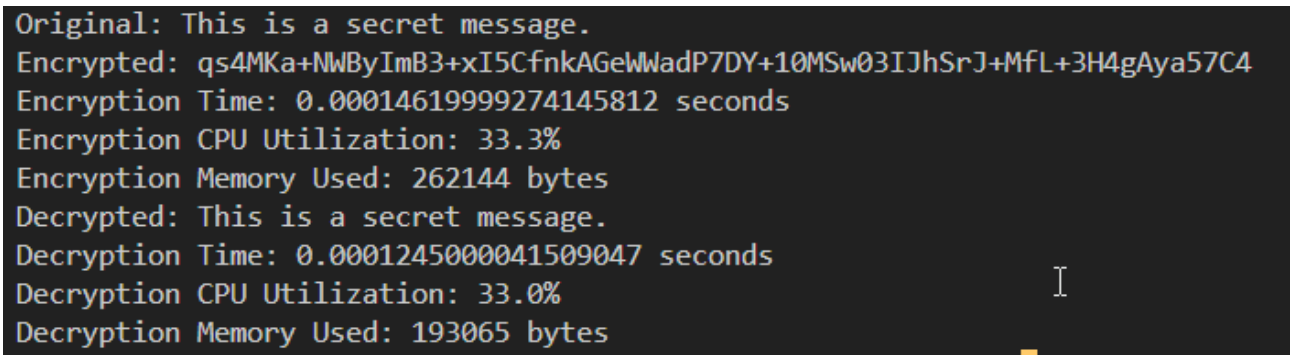

Рисунок 3.2 – Результат виконання програми алгоритму AES

Під час тестування шифру AES було виявлено, що цей алгоритм демонструє високу ефективність з низьким використанням ресурсів. На першому пристрої AES використовував 33.3% CPU і 262144 B RAM з часом виконання 0.00014 секунд для шифрування, тоді як для дешифрування показники були ще кращими: 33.0% CPU і 193065 B RAM з часом виконання 0.00012 секунд. Це підтверджує високу швидкість і ефективність AES.

Наступним на черзі буде криптоалгоритм DES, результати тестування зображено на риснку 3.3.

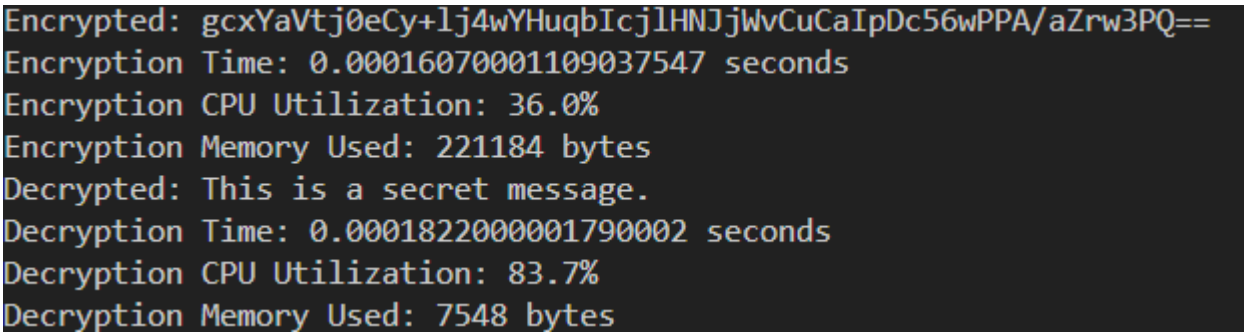

Рисунок 3.3 – Результат виконання програми алгоритму DES

Шифр DES, хоча й застарілий, показав дещо гірші результати порівняно з AES. На першому пристрої DES використовував 36.0% CPU і 225280 B RAM з часом виконання 0.00015 секунд при шифруванні, а при дешифруванні - 28.0% CPU і 203452 B RAM з часом виконання 0.00016 секунд. Це свідчить про його

меншу ефективність та вищі ресурси витрат порівняно з AES, що робить його менш придатним для використання в сучасних системах.

Далі було проведено заміри такого криптоалгоритму як RC4, деталі замірів зображено на рисунку 3.4.

```
Encrypted: Qkkw7TYscMiweg/b/8XuTOlmaqv/YAcOZOejcaEu7Nw=
Encryption Time: 0.00019680000094871968 seconds
Encryption CPU Utilization: 43.0%
Encryption Memory Used: 262144 bytes
Decrypted: This is a secret message.
Decryption Time: 0.00015740000051446258 seconds
Decryption CPU Utilization: 44.0%
Decryption Memory Used: 210327 bytes
```

Рисунок 3.4 – Результат виконання програми алгоритму RC4

RC4, хоча й швидкий потоковий шифр, показав найвищі показники використання CPU серед усіх протестованих алгоритмів. На першому пристрої RC4 використовував 43.0% CPU і 262144 B RAM з часом виконання 0.00019 секунд при шифруванні, тоді як при дешифруванні він споживав ті ж 43.0% CPU, але з меншим використанням RAM у 210327 B і часом виконання 0.00015 секунд. Незважаючи на високу швидкість, значне споживання ресурсів і відомі вразливості роблять його менш привабливим для сучасних застосувань.

І останім в цьому дослідженні буде криптосистема Blowfish результати замірів зображено на рисунку 3.5.

```
Original: This is a secret message.
Encrypted: p8bbRSIVUr/NJECWmoDnRkLzLxiYzWdKG4ipV033S1WWD/J4zqaMug==
Encryption Time: 0.00017470000127796083 seconds
Encryption CPU Utilization: 37.1%
Encryption Memory Used: 266240 bytes
Decrypted: This is a secret message.
Decryption Time: 0.00018900000830180942 seconds
Decryption CPU Utilization: 32.4%
Decryption Memory Used: 209345 bytes
```

Рисунок 3.5 – Результат виконання програми алгоритму Blowfish

Blowfish продемонстрував хороші результати, хоча й потребував трохи більше ресурсів, ніж AES. На першому пристрої він використовував 37.1% CPU і 266240 B RAM з часом виконання 0.00017 секунд при шифрувані, а при дешифрувані - 32.4% CPU і 209345 B RAM з часом виконання 0.00018 секунд. Це вказує на його надійність і ефективність, особливо для систем, де AES може бути недоступним або складним для реалізації.

Для оцінки продуктивності та ефективності шифрувальних алгоритмів AES, DES, Blowfish і RC4 ми провели серію тестів на пристрої з операційною системою Windows. Вимірювалися ключові параметри: використання процесора (CPU), обсяг використаної оперативної пам'яті (RAM) і час виконання для кожного алгоритму. Результати вимірювань допоможуть розробникам та системним адміністраторам зрозуміти компроміси між швидкістю, ресурсозатратністю і рівнем безпеки, які пропонує кожен алгоритм. На основі цих даних було створено графіки, які наочно демонструють ефективність кожного шифру в різних умовах. Далі представлені графіки ефективності шифрування для кожного з протестованих алгоритмів спочатку розглянемо процесор див. рисунок 3.6.

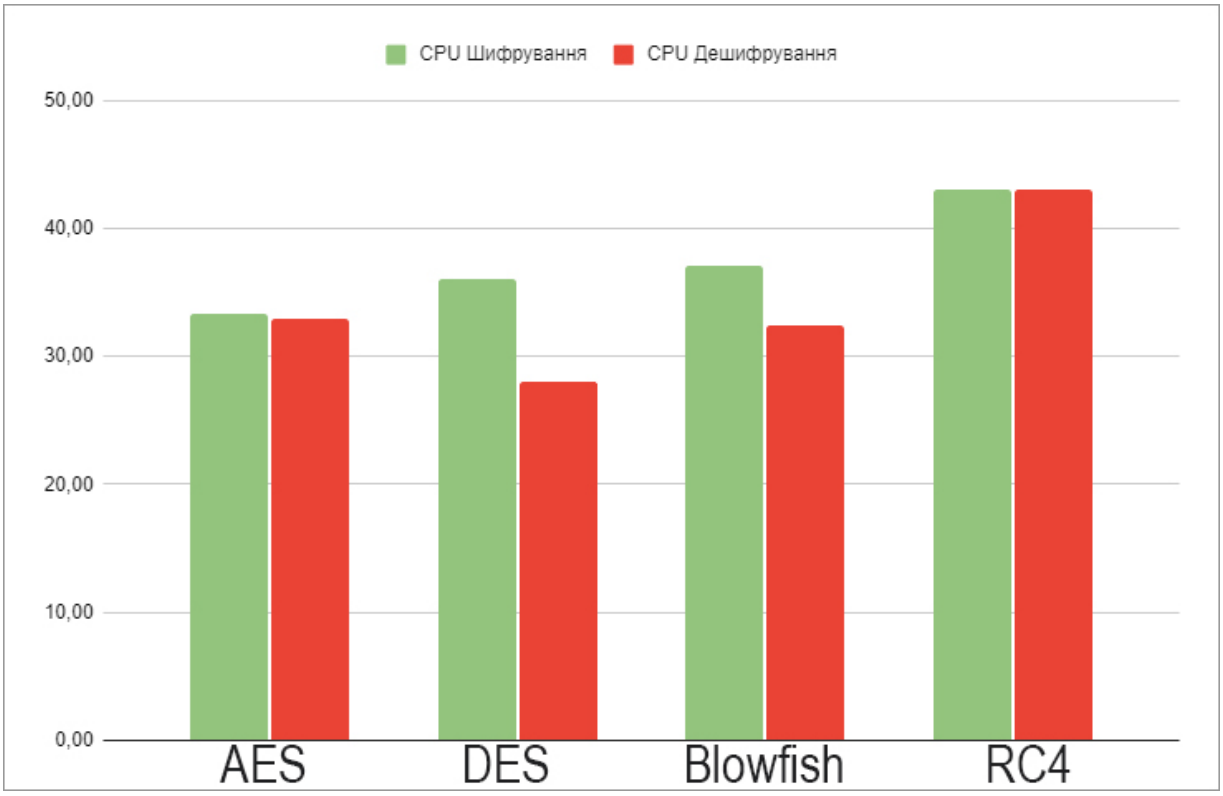

Рисунок 3.6 – Графік результатів використання процесора

На представленому графіку показано використання процесора (CPU) під час шифрування та дешифрування для алгоритмів AES, DES, Blowfish та RC4. Зелені стовпчики відображають використання CPU під час шифрування, а червоні – під час дешифрування. AES демонструє стабільне використання CPU як під час шифрування (близько 33%), так і під час дешифрування (близько 33%), що свідчить про високу ефективність та симетричність роботи алгоритму. DES показує трохи вище використання CPU під час шифрування (близько 36%), ніж під час дешифрування (близько 28%), що може свідчити про більшу потребу в ресурсах для шифрування. Blowfish має порівняно високе використання CPU під час шифрування (близько 37%) та дещо менше під час дешифрування (близько 32%), що також вказує на ефективність алгоритму, хоча він потребує більше ресурсів під час шифрування. RC4 виявляє найвище використання CPU серед усіх алгоритмів як під час шифрування (близько 43%), так і під час дешифрування (близько 43%), що свідчить про його високу ресурсоємність, яка може бути недоліком для застосувань з обмеженими ресурсами.

Отже, AES показав найкращі результати з точки зору балансу між шифруванням та дешифруванням, забезпечуючи високу ефективність при відносно низькому використанні CPU. DES і Blowfish потребують більше ресурсів для шифрування, ніж для дешифрування, але все ще залишаються ефективними. RC4 виявився найресурсоємнішим алгоритмом, що робить його менш придатним для сучасних застосувань, де важлива ефективність використання ресурсів.

Для подальшої оцінки ефективності шифрувальних алгоритмів AES, DES, Blowfish і RC4 ми виміряли обсяг використаної оперативної пам'яті (RAM) під час шифрування та дешифрування. Цей аспект є критично важливим, оскільки використання пам'яті впливає на загальну продуктивність системи, особливо в умовах обмежених ресурсів. На графіку нижче представлені результати цих вимірювань, що допомагає наочно порівняти ефективність кожного алгоритму в різних умовах роботи див. рис. 3.7.

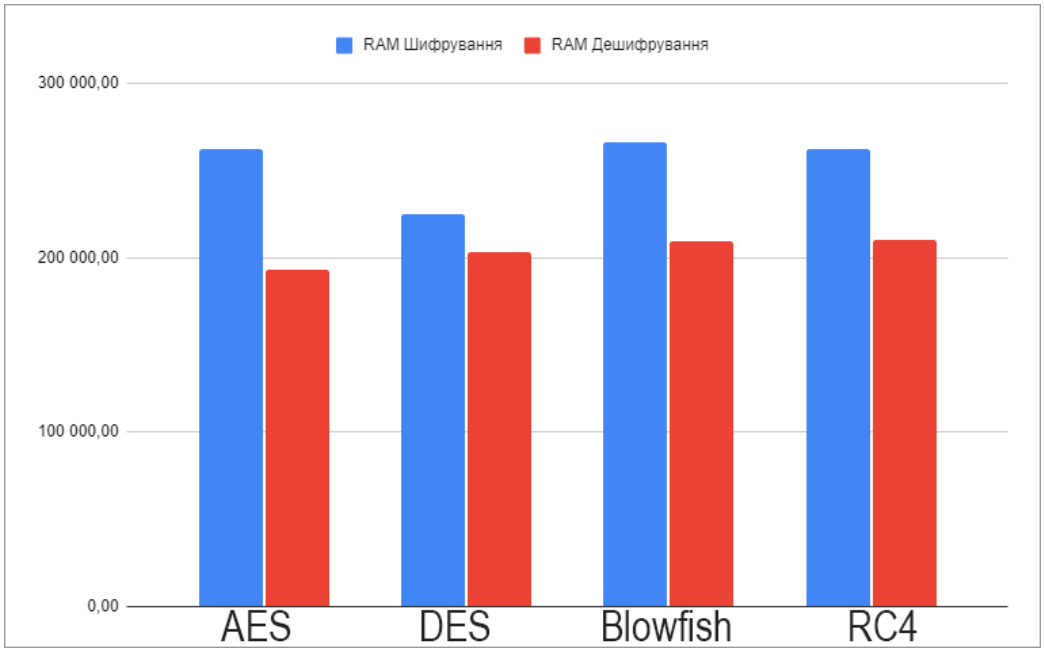

Рисунок 3.7 – Графік результатів використання оперативної пам’яті

На представленому графіку показано використання оперативної пам'яті (RAM) під час шифрування та дешифрування для алгоритмів AES, DES, Blowfish та RC4. Сині стовпчики відображають використання RAM під час шифрування, а червоні – під час дешифрування.

Для алгоритму AES було виявлено, що використання RAM під час шифрування становить близько 262144 B, тоді як під час дешифрування – близько 193065 KB. Це свідчить про те, що AES використовує більше пам'яті під час шифрування, ніж під час дешифрування.

DES показав використання RAM на рівні близько 225280 B під час шифрування та близько 203452 KB під час дешифрування. Ці результати вказують на те, що DES потребує менше пам'яті порівняно з AES, хоча різниця між шифруванням і дешифруванням також є помітною.

Blowfish використовував приблизно 266240 B RAM під час шифрування та близько 209345 B під час дешифрування. Це свідчить про те, що Blowfish також споживає більше пам'яті під час шифрування, але загалом потребує трохи більше ресурсів, ніж DES.

RC4 показав використання RAM на рівні близько 262144 B під час шифрування та близько 210327 B під час дешифрування. Цей алгоритм

використовує значну кількість пам'яті, що може бути обмеженням для деяких систем.

З результатів можна зробити кілька ключових висновків. AES споживає більше пам'яті під час шифрування, ніж під час дешифрування, але загалом є дуже ефективним з точки зору використання ресурсів. DES потребує менше пам'яті порівняно з AES, але також показує різницю у споживанні пам'яті між шифруванням і дешифруванням. Blowfish має найбільше споживання пам'яті під час шифрування, але демонструє хорошу ефективність під час дешифрування. RC4, хоча і швидкий потоковий шифр, споживає значну кількість пам'яті, що може бути недоліком для систем з обмеженими ресурсами. Ці результати допомагають зрозуміти компроміси між ефективністю та споживанням пам'яті для кожного алгоритму, що є важливим для оптимізації їх використання у різних умовах.

Для оцінки продуктивності шифрувальних алгоритмів AES, DES, Blowfish і RC4 було проведено вимірювання часу, необхідного для виконання шифрування та дешифрування. Час виконання є критично важливим параметром, оскільки він безпосередньо впливає на швидкість обробки даних та загальну продуктивність системи. На графіку нижче наведені результати цих вимірювань, які наочно демонструють різницю в ефективності кожного алгоритму під час виконання криптографічних операцій, див. рис. 3.8.

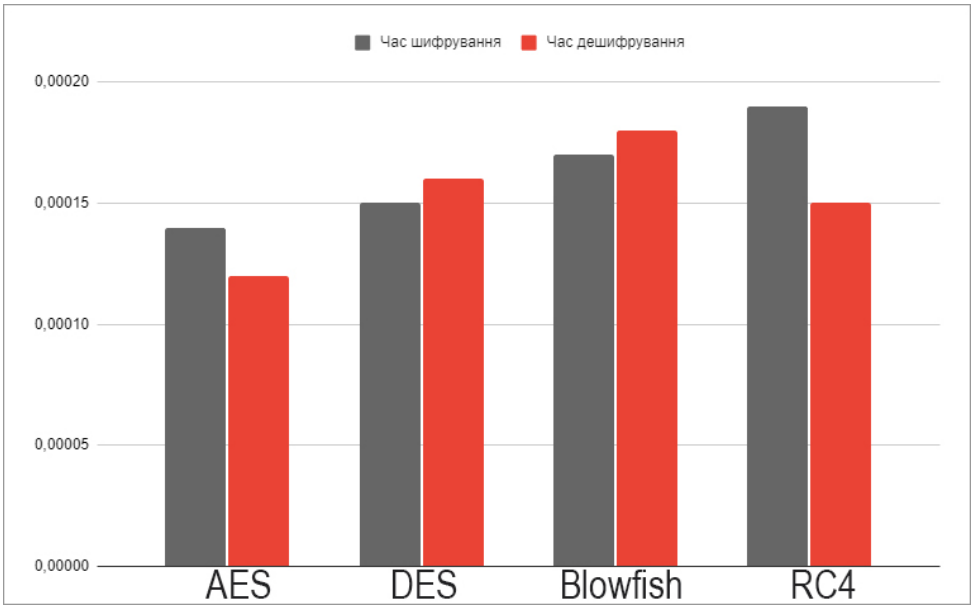

Рисунок 3.8 – Графік результатів часу винонання

На представленому графіку показано час, необхідний для шифрування та дешифрування даних алгоритмами AES, DES, Blowfish та RC4. Сірі стовпці відображають час шифрування, а червоні – час дешифрування.

Результати показують, що AES має найнижчий час виконання серед усіх алгоритмів: час шифрування становить приблизно 0.00014 секунд, а час дешифрування – 0.00012 секунд. Це свідчить про високу ефективність AES як у шифруванні, так і в дешифруванні.

Алгоритм DES має час шифрування близько 0.00015 секунд, а дешифрування – 0.00016 секунд. Хоча різниця між шифруванням і дешифруванням незначна, DES все ж потребує трохи більше часу для дешифрування.

Blowfish показує трохи більший час виконання, ніж AES та DES, з часом шифрування близько 0.00017 секунд і дешифруванням – 0.00018 секунд. Це вказує на те, що Blowfish потребує трохи більше часу на обробку даних.

RC4 має час шифрування близько 0.00019 секунд і час дешифрування – 0.00015 секунд. Хоча RC4 відомий своєю швидкістю, ці результати показують, що він потребує більше часу на шифрування порівняно з іншими алгоритмами, але дешевше дешифрує дані.

Загалом, AES показав найкращі результати з точки зору часу виконання, роблячи його найбільш ефективним алгоритмом серед розглянутих. DES та Blowfish також демонструють прийнятні результати, хоча потребують трохи більше часу на обробку даних. RC4, хоча і є швидким потоковим шифром, показав дещо вищий час шифрування, що може впливати на його продуктивність у деяких випадках.

Далі проведемо тестування на системі Kali Linux для об’єктивності вимірювань.

Проведемо дослідження на віртуальній машині яка запущена на тому самому домашньому компютері, його характеристики:

- Процесор: Intel core i5-9400f (2 ядра)
- Оперетивна память: 4Gb
- Опереаційна система: Kali Linux

Першим проведемо заміри крипто алгоритму AES, результати тестуваня зображено на рисунку 3.9.

```
┌──(venv)─(kali㉿kali)-[~/Desktop]
└─$ python3 AES.py
Original: This is a secret message.
Encrypted: us/h25G8R6dBz/ORtKGGIs22mjkLzNiMLtyUI2TlS7f0o0MMk4O7zx7RZTwMyqdu
Encryption Time: 0.00017892199994344992 seconds
Encryption CPU Utilization: 15.0%
Encryption Memory Used: 262144 bytes
Decrypted: This is a secret message.
Decryption Time: 0.00012744900002151552 seconds
Decryption CPU Utilization: 12.0%
Decryption Memory Used: 199398 bytes
```

Рисунок 3.9 – Результат виконання програми алгоритму AES в ОС Kali Linux

Алгоритм AES демонструє високий рівень ефективності як у шифруванні, так і в дешифруванні. Використання процесора для шифрування становить 15%, а для дешифрування – 12%. Обсяг оперативної пам'яті, необхідний для шифрування, складає 262144 байти, а для дешифрування – 199398 байтів. Час шифрування AES становить 0,00017 секунд, а дешифрування – 0,00012 секунд.

Наступним на черзі алгоритм шифрування DES, його результати наведені на рисунку 3.10.

```
┌──(venv)─(kali㉿kali)-[~/Desktop]
└─$ python3 DES.py
Original: This is a secret message.
Encrypted: vDEhXu/bTTpGUWrxV3M1T2ibaPOIoQPJVHqc8jTVYdyTlW6L8GShzQ==
Encryption Time: 0.00019413000000790816 seconds
Encryption CPU Utilization: 18.0%
Encryption Memory Used: 183404 bytes
Decrypted: This is a secret message.
Decryption Time: 0.00014193699999177624 seconds
Decryption CPU Utilization: 20.0%
Decryption Memory Used: 174892 bytes
```

Рисунок 3.10 – Результат виконання програми алгоритму DES в ОС Kali Linux

DES показує дещо вищий рівень використання процесора під час шифрування та дешифрування, відповідно 18% і 20%. Обсяг оперативної пам'яті для шифрування становить 183404 байти, а для дешифрування – 174892 байти. Час шифрування DES складає 0,00019 секунд, а дешифрування – 0,00014 секунд.

Це свідчить про те, що DES потребує більше ресурсів процесора порівняно з AES, але менше оперативної пам'яті.

Далі буде розглянуто такий алгорити шифрування як Blowfish, результати його вимірювань наведено на рисунку 3.11.

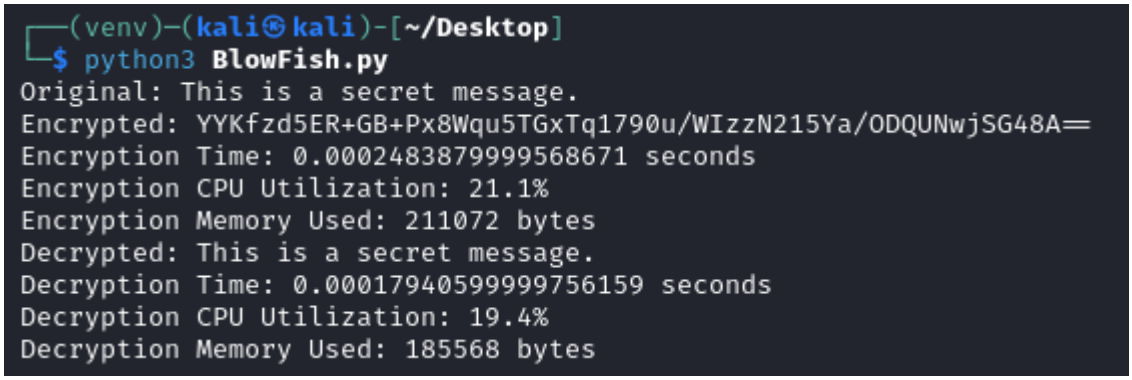

Рисунок 3.11 – Результат виконання програми алгоритму Blowfish в ОС Kali Linux

Blowfish демонструє високий рівень використання процесора як під час шифрування (21,10%), так і дешифрування (19,40%). Обсяг оперативної пам'яті для шифрування складає 211072 байти, а для дешифрування – 185568 байтів. Час шифрування Blowfish становить 0,00024 секунд, а дешифрування – 0,00017 секунд. Це вказує на те, що Blowfish потребує більше часу та ресурсів для шифрування порівняно з AES і DES.

Останім на розгляді в рамках цього дослідження буде потоковий алгоритм шифрування RC4, результати його замірів наведено на рисунку 3.12.

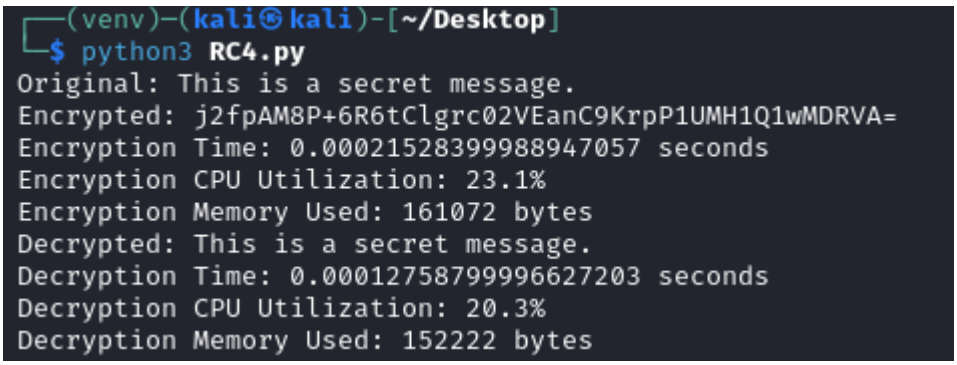

Рисунок 3.12 – Результат виконання програми алгоритму RC4 в ОС Kali Linux

RC4 показує найвищий рівень використання процесора під час шифрування (23,10%) та дешифрування (20,30%). Обсяг оперативної пам'яті для шифрування становить 161072 байти, а для дешифрування – 152222 байти. Час шифрування RC4 складає 0,00021 секунд, а дешифрування – 0,00012 секунд. Хоча RC4 відомий своєю швидкістю, він потребує більше ресурсів процесора для шифрування порівняно з іншими алгоритмами, але споживає найменше оперативної пам'яті.

Для наочної демонстрації результатів тестування алгоритмів шифрування AES, DES, Blowfish та RC4 на системі Linux було створено кілька графіків. Завдяки цим графікам ми можемо краще зрозуміти ефективність кожного алгоритму в різних умовах, порівняти їхні переваги та недоліки, а також зробити обґрунтований вибір для конкретних задач, використання CPU наведено на рисунку 3.13.

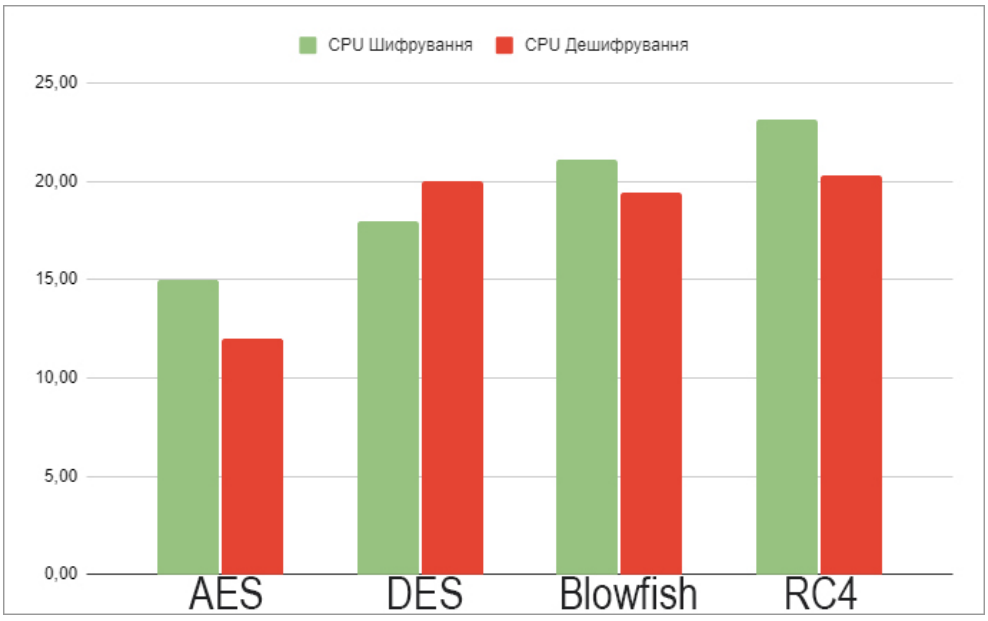

Рисунок 3.13 – Графік результатів використання процесора на ОС Linux

На представленому графіку показано використання процесора (CPU) під час шифрування та дешифрування для алгоритмів AES, DES, Blowfish та RC4. AES демонструє стабільне використання CPU як під час шифрування (близько 15%), під час дешифрування дещо менше (близько 12%), що свідчить про високу ефективність та симетричність роботи алгоритму. DES показує трохи вище

використання CPU під час шифрування (близько 18%), ніж під час дешифрування (близько 20%), що може свідчити про більшу потребу в ресурсах для шифрування. Blowfish має порівняно високе використання CPU під час шифрування (близько 21%) та дещо менше під час дешифрування (близько 19.4%), що також вказує на ефективність алгоритму, хоча він потребує більше ресурсів під час шифрування. RC4 виявляє найвище використання CPU серед усіх алгоритмів як під час шифрування (близько 23.1%), так і під час дешифрування (близько 20.3%), що свідчить про його високу ресурсоємність, яка може бути недоліком для застосувань з обмеженими ресурсами.

Порівнюючи результати використання процесора для шифрування та дешифрування на Linux і Windows, можна зробити кілька цікавих висновків про продуктивність алгоритмів AES, DES, Blowfish та RC4.

Загалом, результати свідчать про те, що всі чотири алгоритми мають свої специфічні характеристики та вимоги до ресурсів на різних платформах. AES залишається найбільш ефективним і симетричним у використанні ресурсів.

Використання опереативно пам’яті алгоритамими шифрування в ОС Linux наведено на рисунку 3.14.

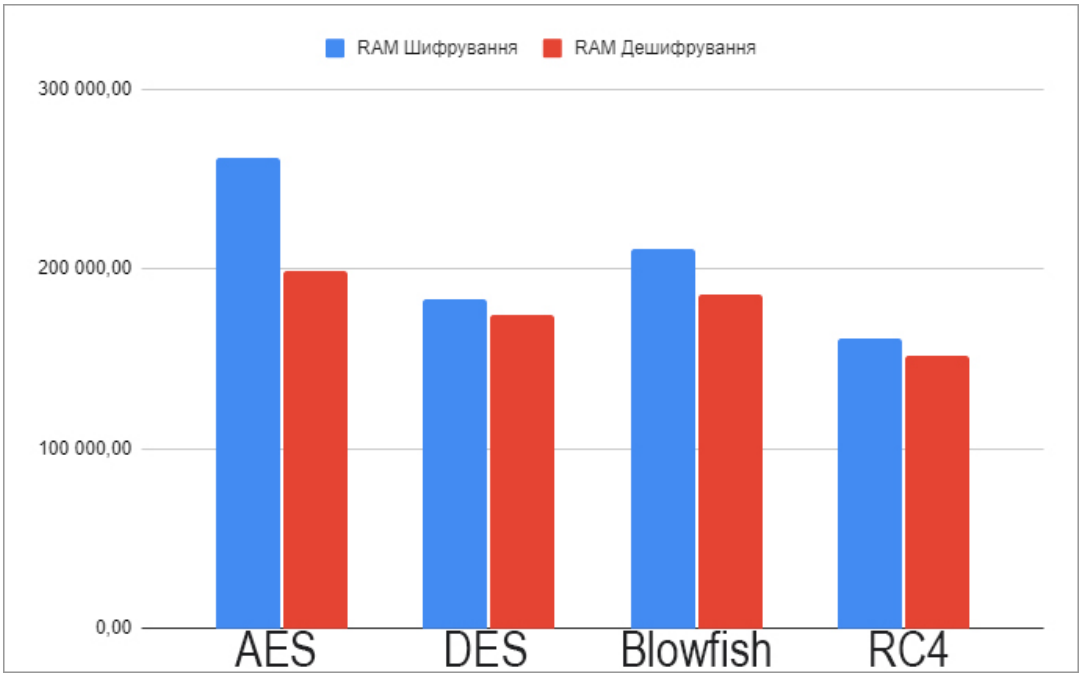

Рисунок 3.14 – Графік результатів використання оперативнох пам’яті на ОС Linux

На представленому графіку показано використання оперативної пам'яті (RAM) під час шифрування та дешифрування для алгоритмів AES, DES, Blowfish та RC4.

Для алгоритму AES було виявлено, що використання RAM під час шифрування становить близько 262144 B, тоді як під час дешифрування – близько 199398 KB. Це свідчить про те, що AES використовує більше пам’яті під час шифрування, ніж під час дешифрування.

DES показав використання RAM на рівні близько 183404 B під час шифрування та близько 174892 B під час дешифрування. Ці результати вказують на те, що DES потребує менше пам'яті порівняно з AES.

Blowfish використовував приблизно 211072 B RAM під час шифрування та близько 185568 B під час дешифрування. Це свідчить про те, що Blowfish також споживає менше пам’яті ніж AES під час шифрування, але загалом потребує трохи більше ресурсів, ніж DES.

RC4 показав використання RAM на рівні близько 161072 B під час шифрування та близько 152222 B під час дешифрування. Цей алгоритм використовує найменшу кількість оперативної пам’яті в ОС Linux, в той час як на Windows він використовував найбільше.

Порівняння використання оперативної пам’яті (RAM) алгоритмами AES, DES, Blowfish та RC4 на платформах Linux і Windows виявляє цікаві закономірності та відмінності в їхній продуктивності.

На Linux, алгоритм AES під час шифрування використовує близько 262144 байтів RAM, а під час дешифрування – близько 199398 байтів. На Windows, AES показує аналогічні тенденції, але з дещо більшими значеннями: близько 262144 байтів під час шифрування і 193065 байтів під час дешифрування. Це свідчить про те, що AES має стабільно високе споживання пам'яті під час шифрування на обох платформах, але трохи менше потребує пам’яті під час дешифрування на Windows.

DES на Linux використовує близько 183404 байтів RAM під час шифрування і 174892 байтів під час дешифрування. На Windows ці значення трохи вищі: близько 225280 байтів під час шифрування і 203452 байтів під час

дешифрування. Таким чином, DES споживає менше пам'яті на Linux порівняно з Windows, але все одно демонструє менше використання пам'яті порівняно з AES на обох платформах.

Blowfish на Linux використовує приблизно 211072 байтів RAM під час шифрування і 185568 байтів під час дешифрування. На Windows Blowfish також показує схожі тенденції, але з дещо більшими значеннями: близько 266240 байтів під час шифрування і 209345 байтів під час дешифрування. Це вказує на те, що Blowfish споживає більше ресурсів пам'яті на обох платформах порівняно з DES, але менше, ніж AES під час шифрування.

RC4 на Linux використовує близько 161072 байтів RAM під час шифрування і 152222 байтів під час дешифрування, що робить його найбільш ефективним з точки зору споживання пам'яті серед усіх алгоритмів на цій платформі. На Windows, однак, RC4 показує найбільше споживання пам'яті: 262144 байтів під час шифрування і 210327 байтів під час дешифрування. Це значна різниця, яка вказує на те, що RC4 є більш ресурсоємним на Windows, ніж на Linux.

Загальний аналіз використання оперативної пам'яті алгоритмами шифрування на платформах Linux і Windows вказує на суттєві відмінності в їхній продуктивності. AES стабільно споживає багато пам'яті під час шифрування на обох платформах, але трохи менше під час дешифрування. DES є більш економним з точки зору пам'яті на Linux, але все одно потребує більше ресурсів на Windows. Blowfish показує тенденцію до вищого споживання пам'яті, ніж DES, але меншого, ніж AES, на обох платформах. RC4, хоча і є найменш ресурсоємним на Linux, виявляється найбільш ресурсоємним на Windows. Ці результати підкреслюють важливість врахування специфіки платформи та доступних ресурсів при виборі алгоритму шифрування, щоб забезпечити оптимальну продуктивність і ефективність системи.

Результати вимирівання часу виконання в ОС Linux наведено на рисунку 3.15.

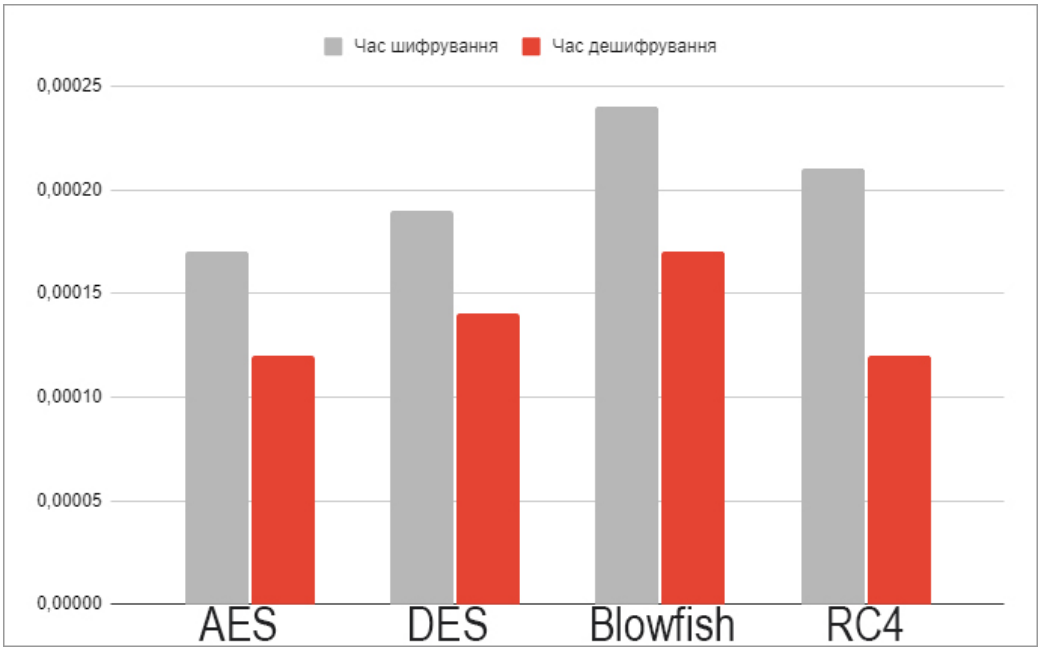

Рисунок 3.15 – Графік результатів часу винонання на ОС Linux

На платформі Linux було проведено вимірювання часу, необхідного для шифрування та дешифрування даних за допомогою алгоритмів AES, DES, Blowfish та RC4.

Для алгоритму AES час шифрування становить 0,00017 секунд, тоді як час дешифрування – 0,00012 секунд. Це вказує на високу швидкість роботи AES, особливо під час дешифрування, що робить його ефективним для застосувань, де важлива швидкість обробки даних.

Алгоритм DES показав час шифрування 0,00019 секунд і час дешифрування 0,00014 секунд. DES є дещо повільнішим у порівнянні з AES, що може бути важливим фактором при виборі алгоритму для завдань, що потребують швидкої обробки даних.

Blowfish має час шифрування 0,00024 секунд і час дешифрування 0,00017 секунд. Це робить його найповільнішим серед розглянутих алгоритмів, що може обмежувати його використання в сценаріях, де потрібна висока швидкість шифрування та дешифрування.

RC4 показав час шифрування 0,00021 секунд і час дешифрування 0,00012 секунд. Хоча час шифрування RC4 дещо більший, ніж у AES, його час дешифрування дорівнює AES, що вказує на ефективну роботу цього алгоритму під час дешифрування.

Загалом, результати показують, що AES є найшвидшим алгоритмом як для шифрування, так і для дешифрування на Linux. DES і Blowfish є повільнішими, з Blowfish як найповільніший серед розглянутих. RC4 демонструє прийнятну швидкість, особливо під час дешифрування, але є трохи повільнішим під час шифрування порівняно з AES.

Підсумовуючи результати аналізу всіх параметрів – використання процесора (CPU), оперативної пам'яті (RAM) та часу – можна зробити кілька ключових висновків щодо ефективності та універсальності алгоритмів шифрування AES, DES, Blowfish та RC4.

Алгоритм AES демонструє високу ефективність та стабільність на обох платформах, Linux і Windows. Він має оптимальне використання процесора та пам'яті під час шифрування і дешифрування, а також швидкий час обробки даних. AES вимагає трохи більше пам'яті під час шифрування, але це компенсується його високою швидкістю та безпекою. Ці характеристики роблять AES найуніверсальнішим шифром, придатним для широкого спектра застосувань, від захисту даних на мобільних пристроях до забезпечення безпеки в корпоративних мережах.

DES показує трохи вищі показники використання процесора та пам'яті порівняно з AES, а також дещо повільніший час обробки. DES може бути використаний у старих системах або там, де сумісність з існуючими стандартами є критично важливою. Проте, через нижчий рівень безпеки порівняно з сучасними алгоритмами, його використання обмежене.

Blowfish демонструє вищі вимоги до ресурсів пам'яті та часу шифрування, що робить його менш придатним для застосувань, де важлива висока швидкість. Однак, Blowfish може бути корисним у випадках, де необхідна додаткова гнучкість у виборі ключів або де високий рівень безпеки виправдовує додаткові витрати на ресурси.

RC4 показує найвищі показники використання процесора під час шифрування і дещо менші під час дешифрування на Linux, але на Windows він є найресурсоємнішим. Це робить його менш придатним для ресурсно-обмежених систем. Тим не менш, RC4 може бути використаний у спеціалізованих

застосуваннях, де його потоковий характер і швидке дешифрування є перевагами.

Отже, з огляду на всі параметри: CPU, RAM та час – AES є найуніверсальнішим і найбільш ефективним алгоритмом шифрування, підходящим для широкого спектра застосувань. Інші алгоритми мають свої специфічні переваги і можуть бути використані в залежності від вимог до сумісності, гнучкості та специфіки ресурсів.

4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Значення адаптації в трудовому процесі

Праця людини безпосередньо пов'язана з виробничим середовищем. Лише тоді, коли умови зовнішнього середовища відповідають оптимальним умовам, працівники можуть виконувати нормальну трудову діяльність. Якщо вони змінюються і стають несприятливими, організм має спеціальний механізм протидії їм, таким чином підтримуючи внутрішнє середовище постійним або змінюючи його в допустимих межах. Цей механізм називається адаптацією. Адаптація є важливим засобом запобігання травматизму та нещасним випадкам на виробництві та відіграє важливу роль в охороні праці.

Адаптація – пристосування організму до певних умов існування, які постійно змінюються, що формувалося в процесі еволюційного розвитку. Без адаптації неможлива була б підтримка нормальної життєдіяльності та пристосування до різних чинників зовнішнього середовища: кліматичних і температурних, гіпоксії, дії на організм інфекційних агентів та інших. Вона має велике значення для організму людини і тварин, дозволяє не тільки переносити різні зміни в навколишньому середовищі, але й активно перебудовувати свої фізіологічні функції та поведінку відповідно до цих змін. Завдяки адаптації підтримується постійність внутрішнього середовища організму, такі константи крові, як кислотно-лужний баланс, осмотичний тиск. В свою чергу в трудовій діяльності вона поділяється на фізіологічну, психічну, соціальну та професійну.

Фізіологічна адаптація – це сукупність фізіологічних реакцій, які служать основою пристосування організму до змін зовнішніх умов, спрямованих на підтримку відносної сталості свого внутрішнього середовища – гомеостазу.

Гомеостаз – відносна динамічна сталість внутрішнього середовища та деяких фізіологічних функцій організму людини і тварин. Гомеостаз забезпечується складною системою координованих адаптаційних механізмів, спрямованих на усунення чи обмеження дії на організм зовнішніх та внутрішніх чинників.

Суть механізму адаптації полягає в зміні межі чутливості аналізатора, розширенні діапазону фізіологічних резервів організму, зміні в певних межах параметрів фізіологічних функцій. Завдяки фізіологічним адаптаціям фізичні та біохімічні показники, що визначають життєдіяльність організму, змінюються у вузькому діапазоні порівняно зі значними змінами зовнішніх умов: підвищується стійкість організму до холоду, тепла, гіпоксії, зміни атмосферного тиску та інших факторів. Велике значення в фізіологічній адаптації має реактивність організму, його вихідний функціональний стан і відповідна реакція організму на різні форми поведінки. Процес фізіологічної адаптації до аномальних і екстремальних умов буде проходити в кілька етапів: спочатку переважає декомпенсація, потім неповна адаптація і, нарешті, стадія відносно стабільної адаптації.

Фізіологічна адаптація до праці має активний характер і за сприятливих умов виробничого середовища та оптимальних навантажень веде до підвищення стійкості та працездатності організму, збільшення його резервних можливостей, зменшення захворювань і травматизму. Проте коливання умов середовища, в яких відбувається фізіологічна адаптація, має певну межу, характерну для кожного організму. Якщо працівник потрапляє в умови, коли інтенсивність впливу чинників виробничого середовища переважає можливості його адаптації, настають патологічні зміни фізіологічних систем, захворювання організму.

Психічна адаптація – це процес встановлення оптимальної відповідності особистості до навколишнього середовища в процесі діяльності. Зрозуміло, що такі властивості, як гальмування мислення та низька швидкість переробки інформації, обмежений діапазон сприйняття, порушення функції пам’яті гальмують адаптацію; висока рухливість нервових процесів, навпаки, її підвищує.

Соціальна адаптація – це пристосування працюючої людини до системи відносин у робочому колективі з його нормами, правилами, традиціями, ціннісними орієнтаціями. Під час соціальної адаптації працівник поступово отримує різнобічну інформацію про колектив, де він працює, про систему ділових та особистих взаємовідносин. При несприятливому протіканні

соціальної адаптації підвищується рівень стресу на роботі, наслідки якого позначаються на поведінці працівника та можуть призвести до міжособових конфліктів, нещасних випадків, а це все впливає на його працездадність.

Професійна адаптація – це адаптація до трудової діяльності з усіма її складовими: адаптація до робочого місця, знарядь та засобів праці, об’єктів та предметів праці, особливостей технологічного процесу, часових параметрів роботи тощо.

Кожен розглянутий вид адаптації впливає на працездатність і здоров'я працівника, створюючи в нього певний ступінь чутливості і стійкості до психоемоційних перевантажень, завдяки розвитку яких може наступити значне різноманіття надійності професійної діяльності.

4.2 Вимоги ергономіки до організації робочого місця оператора ПК

Робоче місце – це зона простору, що оснащена необхідним устаткуванням, де відбувається трудова діяльність одного працівника чи групи працівників.

Правильне планування робочого місця повинно забезпечувати: оптимальне розміщення інструментів і предметів праці, запобігання загального дискомфорту, зниження втоми працівника, підвищення ефективності праці. Зони робочого місця повинні забезпечувати, щоб працівники не робили зайвих рухів і не відчували дискомфорту під час роботи. Також важливо вміти змінювати робочу позу, тобто положення тіла, рук і ніг. Однак рекомендується виключити або звести до мінімуму всі фізіологічно неприродні і незручні положення тіла.

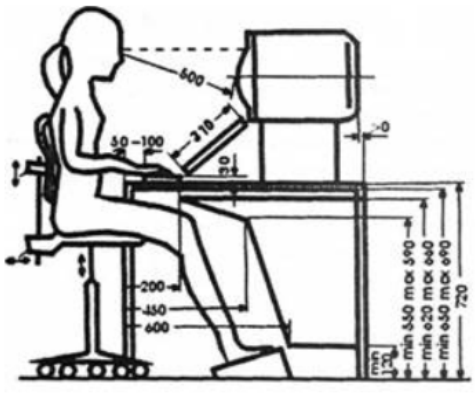

Рисунок 4.1 – Робочий стіл і розміщення користувача ПК

Дослідження вчених показують, що при раціональній організації робочих місць продуктивність праці має зростати на 15-25% [8].

Організація робочого місця передбачає:

- правильне розміщення робочого місця у виробничому приміщенні;
- вибір ергономічно обґрунтованого робочого положення, виробничих меблів з урахуванням антропометричних характеристик людини;
- раціональне компонування обладнання на робочих місцях;
- урахування характеру та особливостей трудової діяльності.

Загальні принципи організації робочого місця:

- на робочому місці не повинно бути нічого зайвого;
- ті предмети, якими користуються частіше, розташовуються ближче, ніж ті предмети, якими користуються рідше;
- предмети, які беруть лівою рукою, повинні бути зліва, а ті предмети, які беруть правою рукою – справа;
- якщо використовують обидві руки, то місце розташування пристосувань вибирається з урахуванням зручності захоплювання його двома руками.

Робоча поза – це основна поза працівників у просторі: зручна робоча поза повинна забезпечувати стійкість тіла, ніг, рук і голови працівників під час роботи, мінімізувати витрати енергії та підвищити ефективність праці.

Найпоширенішими на даний момент положеннями під час роботи є сидячи і стоячи.

При проектуванні робочого місця необхідно враховувати, що для фізично важких робіт найкраще підходить положення стоячи, а для менших навантажень – сидячи. Працювати стоячи втомлює більше, ніж працювати сидячи. Це вимагає на 10% більше енергії, що призводить до підвищення артеріального і венозного тиску крові, розширення вен ніг, травм ніг і викривлення хребта. При роботі сидячи нижня частина тіла розслаблена і основне статичне навантаження припадає на м'язи шиї, спини, таза і сідниць [8].

Неправильна поза сидячи може спричинити дуже багато проблем, як застій крові в ногах і призвести до запалення суглобів, якщо пальці виконують велику роботу.

4.3 Гігієнічні вимоги до параметрів виробничого середовища приміщень з ВДТ

Гігієнічні вимоги до параметрів виробничого середовища включають вимоги до параметрів мікроклімату, освітлення, шуму і вібрації, рівнів електромагнітного та іонізуючого випромінювання.

У виробничих приміщеннях на робочих місцях з ВДТ мають забезпечуватись оптимальні значення параметрів мікроклімату: температури, відносної вологості й рухливості повітря, наведено у таблиці 4.1.

Таблиця 4.1 – Норми мікроклімату для приміщень з ВТД ЕОМ та ПЕОМ

Пора року	Категорія робіт	Температура повітря, °С, не більше	Відносна вологість повітря, %	Швидкість руху повітря, м/с
Холодна	легка-1а	22-24	40-60	0,1
	легка-1б	21-23	40-60	0,1
Тепла	легка-1а	23-25	40-60	0,1
	легка-1б	22-24	40-60	0,2

До категорії 1а належать роботи, що виконуються сидячи і не потребують фізичного напруження, при яких витрати енергії складають до 139 Вт, до категорії 16 належать роботи, що виконуються сидячи, стоячи або пов’язані з ходінням та супроводжуються деяким фізичним напруженням, при яких витрати енергії становлять від 140 до 174 Вт [8].

Рівні позитивних і негативних іонів у повітрі приміщень з ВДТ мають відповідати санітарно-гігієнічним нормам №2152-80 наведено у таблиці 4.2.

Таблиця 4.2 – Рівні іонізації повітря приміщень при роботі на ВДТ

Рівні	Число іонів в 1 $см^3$ повітря	
	n+	n-
Мінімально необхідні	400	600
Оптимальні	1500-3000	3000-5000
Максимальні	50000	50000

Штучне освітлення на робочих місцях з ВДТ повинно забезпечуватися системою загального рівномірного освітлення. У виробничих, адміністративних і громадських приміщеннях можуть застосовуватися комбіновані системи освітлення (загальна система освітлення плюс додаткові світильники місцевого освітлення), якщо робота пов'язана з великим обсягом роботи з документами.

Освітленість поверхні робочого столу, де розміщуються файли, повинна бути 300-500 лк. При відсутності системи загального освітлення допускається місцеве освітлення. При цьому при установці світильників місцевого освітлення слід уникати відблисків на поверхні екрана, а яскравість екрану не повинна перевищувати 300 лк.

Як джерела світла для штучного освітлення мають застосовуватись переважно люмінесцентні лампи типу ЛБ. У разі влаштування відбитого освітлення у виробничих та адміністративно-громадських приміщеннях допускається застосування металогалогенних ламп потужністю 250 Вт. Допускається застосування ламп розжарювання у світильниках місцевого освітлення.

Рівні звукового тиску в октавних смугах частот, рівні звуку та еквівалентні рівні звуку на робочих місцях, обладнаних ВДТ, мають відповідати вимогам.

ВИСНОВКИ

Загальні висновки роботи об'єднують теоретичний і практичний аналіз сучасних симетричних криптосистем, їх значення для інформаційної безпеки та результати проведених тестувань ефективності алгоритмів шифрування. Спершу розглянуто основні поняття інформаційної безпеки та роль симетричних криптосистем у забезпеченні захисту даних. Симетричні криптосистеми, включаючи AES, DES, RC4 та Blowfish, аналізувалися з точки зору їх криптографічної стійкості та практичної ефективності.

Проведені заміри показали різні показники використання процесора, оперативної пам'яті та часу шифрування та дешифрування для кожного з алгоритмів на платформах Windows і Linux. AES виявився найефективнішим з точки зору балансу між швидкістю роботи та споживанням ресурсів, показуючи стабільну роботу на обох операційних системах. DES, хоча і менш ефективний, все ще використовується в деяких застарілих системах, де важлива сумісність із старими стандартами. Blowfish, попри більші вимоги до ресурсів, залишається важливим через свою гнучкість у виборі ключів і високий рівень безпеки. RC4, хоча і найменш ефективний з точки зору ресурсів, може бути корисним у специфічних застосуваннях через свій потоковий характер.

Практичні вимірювання підтвердили теоретичні припущення про переваги та недоліки кожного алгоритму. AES продемонстрував найкращі результати, що робить його найбільш універсальним і придатним для широкого спектру застосувань, забезпечуючи високу швидкість і безпеку. Інші алгоритми мають свої специфічні області застосування, де їх особливості можуть бути найбільш корисними. Загалом, робота підкреслює важливість ретельного вибору криптосистеми залежно від конкретних вимог до безпеки та доступних ресурсів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Лужецький В. А., Кожухівський А. Д., Войтович О. П. Основи інформаційної безпеки. Вінниця, 2009. 268 с. URL: https://studfile.net/preview/6012701/ (дата звернення: 05.04.2024).

2. Корченко О. Г., Сіденко В. П., Дрейс Ю. О. ПРИКЛАДНА КРИПТОЛОГІЯ: системи шифрування. Житомир, 2014. 448 с.

3. Karnaukhov, A., Tymoshchuk, V., Orlovska, A., & Tymoshchuk, D. (2024). USE OF AUTHENTICATED AES-GCM ENCRYPTION IN VPN. Матеріали конференцій МЦНД, (14.06. 2024; Суми Україна), 191-193. https://doi.org/10.62731/mcnd-14.06.2024.004

4. Symmetric Encryption Algorithms in a Polynomial Residue Number System / I. Yakymenko та ін. Journal of Applied Mathematics. 2024. Т. 2024. С. 1–12. URL: https://doi.org/10.1155/2024/4894415 (дата звернення: 20.05.2024).

5. Тимощук, В., & Стебельський, М. (2023). Шифрування даних в операційних системах. Матеріали VI Міжнародної студентської науково-технічної конференції „Природничі та гуманітарні науки. Актуальні питання “, 183-184.

6. Enhancing Cryptographic Primitives through Dynamic Cost Function Optimization in Heuristic Search / O. Kuznetsov та ін. Electronics. 2024. Т. 13, № 10. С. 1825. URL: https://doi.org/10.3390/electronics13101825 (дата звернення: 21.05.2024).

7. Awati R. What is Blowfish and how is it used in cryptography?. Security. URL: https://www.techtarget.com/searchsecurity/definition/Blowfish (дата звернення: 15.03.2024).

8. Букатка, С., & Тимощук, В. (2023). ХЕШ-алгоритм шифрування паролів користувачів ос Linux. Матеріали VI Міжнародної студентської науково-технічної конференції „Природничі та гуманітарні науки. Актуальні питання “, 112-113.

9. Alomar S. I. Comparative analysis of modern methods and algorithms of cryptographic protection of information. Journal of Computer Science IJCSIS. 2017. Vol. 15, No. 12. URL: http://surl.li/uokxy (дата звернення: 02.05.2024).

10. Ebrahim M., Khan S., Bin Khalid U. Symmetric Algorithm Survey: A Comparative Analysis. International Journal of Computer Applications. 2013. Volume 61, No.20. URL: https://arxiv.org/ftp/arxiv/papers/1405/1405.0398.pdf (дата звернення: 25.04.2024).

11. Nataliya Zagorodna, Iryna Kramar (2020). Economics, Business and Security: Review of Relations. Business Risk in Changing Dynamics of Global Village BRCDGV-2020: Monograph / Edited by Pradeep Kumar, Mahammad Sharif. India, Patna: Novelty & Co., Ashok Rajpath,. 446 p., pp.25-39.

12. Skorenkyy, Y., Kozak, R., Zagorodna, N., Kramar, O., & Baran, I. (2021, March). Use of augmented reality-enabled prototyping of cyber-physical systems for improving cyber-security education. In Journal of Physics: Conference Series (Vol. 1840, No. 1, p. 012026). IOP Publishing.

ДОДАТКИ

Додаток А Програмна реалізація шифру AES із заміром ефективності

```
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes
import base64
import timeit
import psutil
import subprocess
import tracemalloc
import os

def pad(data):
    padding_length = 16 - len(data) % 16
    padding = chr(padding_length) * padding_length
    return data + padding

def unpad(data):
    padding_length = ord(data[-1])
    return data[:-padding_length]

def encrypt(plain_text, key):
    padded_text = pad(plain_text)
    iv = get_random_bytes(16)
    cipher = AES.new(key, AES.MODE_CBC, iv)
    encrypted_text = cipher.encrypt(padded_text.encode('utf-8'))
    return base64.b64encode(iv + encrypted_text).decode('utf-8')

def decrypt(encrypted_text, key):
    encrypted_data = base64.b64decode(encrypted_text)
    iv = encrypted_data[:16]
    encrypted_text = encrypted_data[16:]
    cipher = AES.new(key, AES.MODE_CBC, iv)
    padded_text = cipher.decrypt(encrypted_text).decode('utf-8')
    plain_text = unpad(padded_text)
    return plain_text

def measure_performance(func, *args):
    process = psutil.Process(os.getpid())
    start_memory = process.memory_info().rss
    start_cpu_percent = psutil.cpu_percent(interval=None)

    execution_time = timeit.timeit(lambda: func(*args), number=1)
    end_memory = process.memory_info().rss
    end_cpu_percent = psutil.cpu_percent(interval=None)
    avg_cpu_percent = (end_cpu_percent - start_cpu_percent)
    memory_used = end_memory - start_memory
    return execution_time, avg_cpu_percent, memory_used

key = get_random_bytes(16)
plain_text = "This is a secret message."

print("Original:", plain_text)
```

```
enc_time, enc_cpu_percent, enc_memory_used =
measure_performance(encrypt, plain_text, key)
encrypted_text = encrypt(plain_text, key)
print("Encrypted:", encrypted_text)
print(f"Encryption Time: {enc_time} seconds")
print(f"Encryption CPU Utilization: {enc_cpu_percent}%")
print(f"Encryption Memory Used: {enc_memory_used} bytes")

dec_time, dec_cpu_percent, dec_memory_used =
measure_performance(decrypt, encrypted_text, key)
decrypted_text = decrypt(encrypted_text, key)
print("Decrypted:", decrypted_text)
print(f"Decryption Time: {dec_time} seconds")
print(f"Decryption CPU Utilization: {dec_cpu_percent}%")
print(f"Decryption Memory Used: {dec_memory_used} bytes")
```

Додаток Б Програмна реалізація шифру DES із заміром ефективності

```
from Crypto.Cipher import DES
from Crypto.Random import get_random_bytes
import base64
import timeit
import psutil
import os

def pad(data):
    padding_length = 8 - len(data) % 8
    padding = chr(padding_length) * padding_length
    return data + padding

def unpad(data):
    padding_length = ord(data[-1])
    return data[:-padding_length]

def encrypt(plain_text, key):
    padded_text = pad(plain_text)
    iv = get_random_bytes(8)
    cipher = DES.new(key, DES.MODE_CBC, iv)
    encrypted_text = cipher.encrypt(padded_text.encode('utf-8'))
    return base64.b64encode(iv + encrypted_text).decode('utf-8')

def decrypt(encrypted_text, key):
    encrypted_data = base64.b64decode(encrypted_text)
    iv = encrypted_data[:8]
    encrypted_text = encrypted_data[8:]
    cipher = DES.new(key, DES.MODE_CBC, iv)
    padded_text = cipher.decrypt(encrypted_text).decode('utf-8')
    plain_text = unpad(padded_text)
    return plain_text

def measure_performance(func, *args):
    process = psutil.Process(os.getpid())
    start_memory = process.memory_info().rss
    start_cpu_percent = psutil.cpu_percent(interval=None)

    execution_time = timeit.timeit(lambda: func(*args), number=1)
    end_memory = process.memory_info().rss
    end_cpu_percent = psutil.cpu_percent(interval=None)
    avg_cpu_percent = (end_cpu_percent - start_cpu_percent)
    memory_used = end_memory - start_memory
    return execution_time, avg_cpu_percent, memory_used

key = get_random_bytes(8)
plain_text = "This is a secret message."

print("Original:", plain_text)

encrypted_text = encrypt(plain_text, key)
enc_time, enc_cpu_percent, enc_memory_used =
measure_performance(encrypt, plain_text, key)
```

```
print("Encrypted:", encrypted_text)
print(f"Encryption Time: {enc_time} seconds")
print(f"Encryption CPU Utilization: {enc_cpu_percent}%")
print(f"Encryption Memory Used: {enc_memory_used} bytes")

decrypted_text = decrypt(encrypted_text, key)
dec_time, dec_cpu_percent, dec_memory_used =
measure_performance(decrypt, encrypted_text, key)
print("Decrypted:", decrypted_text)
print(f"Decryption Time: {dec_time} seconds")
print(f"Decryption CPU Utilization: {dec_cpu_percent}%")
print(f"Decryption Memory Used: {dec_memory_used} bytes")
```

Додаток В Програмна реалізація шифру RC4 із заміром ефективності

```
from Crypto.Cipher import ARC4
from Crypto.Random import get_random_bytes
import base64
import timeit
import psutil
import os

def pad(data):
    padding_length = 16 - len(data) % 16
    padding = chr(padding_length) * padding_length
    return data + padding

def unpad(data):
    padding_length = ord(data[-1])
    return data[:-padding_length]

def encrypt(plain_text, key):
    padded_text = pad(plain_text)
    cipher = ARC4.new(key)
    encrypted_text = cipher.encrypt(padded_text.encode('utf-8'))
    return base64.b64encode(encrypted_text).decode('utf-8')

def decrypt(encrypted_text, key):
    encrypted_data = base64.b64decode(encrypted_text)
    cipher = ARC4.new(key)
    padded_text = cipher.decrypt(encrypted_data).decode('utf-8')
    plain_text = unpad(padded_text)
    return plain_text

def measure_performance(func, *args):
    process = psutil.Process(os.getpid())
    start_memory = process.memory_info().rss
    start_cpu_percent = psutil.cpu_percent(interval=None)
    execution_time = timeit.timeit(lambda: func(*args), number=1)
    end_memory = process.memory_info().rss
    end_cpu_percent = psutil.cpu_percent(interval=None)
    avg_cpu_percent = (end_cpu_percent - start_cpu_percent)
    memory_used = end_memory - start_memory
    return execution_time, avg_cpu_percent, memory_used

key = get_random_bytes(16)  # RC4 key size can vary but here we
use 16 bytes
plain_text = "This is a secret message."

print("Original:", plain_text)

enc_time, enc_cpu_percent, enc_memory_used =
measure_performance(encrypt, plain_text, key)
encrypted_text = encrypt(plain_text, key)
print("Encrypted:", encrypted_text)
print(f"Encryption Time: {enc_time} seconds")
print(f"Encryption CPU Utilization: {enc_cpu_percent}%")
```

```
print(f"Encryption Memory Used: {enc_memory_used} bytes")

dec_time, dec_cpu_percent, dec_memory_used =
measure_performance(decrypt, encrypted_text, key)
decrypted_text = decrypt(encrypted_text, key)
print("Decrypted:", decrypted_text)
print(f"Decryption Time: {dec_time} seconds")
print(f"Decryption CPU Utilization: {dec_cpu_percent}%")
print(f"Decryption Memory Used: {dec_memory_used} bytes")
```

Додаток Г Програмна реалізація шифру Blowfish із заміром ефективності

```
from Crypto.Cipher import Blowfish
from Crypto.Random import get_random_bytes
import base64
import timeit
import psutil
import os

def pad(data):
    padding_length = 8 - len(data) % 8
    padding = chr(padding_length) * padding_length
    return data + padding

def unpad(data):
    padding_length = ord(data[-1])
    return data[:-padding_length]

def encrypt(plain_text, key):
    padded_text = pad(plain_text)
    iv = get_random_bytes(8)
    cipher = Blowfish.new(key, Blowfish.MODE_CBC, iv)
    encrypted_text = cipher.encrypt(padded_text.encode('utf-8'))
    return base64.b64encode(iv + encrypted_text).decode('utf-8')

def decrypt(encrypted_text, key):
    encrypted_data = base64.b64decode(encrypted_text)
    iv = encrypted_data[:8]
    encrypted_text = encrypted_data[8:]
    cipher = Blowfish.new(key, Blowfish.MODE_CBC, iv)
    padded_text = cipher.decrypt(encrypted_text).decode('utf-8')
    plain_text = unpad(padded_text)
    return plain_text

def measure_performance(func, *args):
    process = psutil.Process(os.getpid())
    start_memory = process.memory_info().rss
    start_cpu_percent = psutil.cpu_percent(interval=None)
    execution_time = timeit.timeit(lambda: func(*args), number=1)
    end_memory = process.memory_info().rss
    end_cpu_percent = psutil.cpu_percent(interval=None)
    avg_cpu_percent = (end_cpu_percent - start_cpu_percent)
    memory_used = end_memory - start_memory
    return execution_time, avg_cpu_percent, memory_used

key = get_random_bytes(16)
plain_text = "This is a secret message."
print("Original:", plain_text)

encrypted_text = encrypt(plain_text, key)
enc_time, enc_cpu_percent, enc_memory_used =
measure_performance(encrypt, plain_text, key)
print("Encrypted:", encrypted_text)
print(f"Encryption Time: {enc_time} seconds")
```

```
print(f"Encryption CPU Utilization: {enc_cpu_percent}%")
print(f"Encryption Memory Used: {enc_memory_used} bytes")

decrypted_text = decrypt(encrypted_text, key)
dec_time, dec_cpu_percent, dec_memory_used =
measure_performance(decrypt, encrypted_text, key)
print("Decrypted:", decrypted_text)
print(f"Decryption Time: {dec_time} seconds")
print(f"Decryption CPU Utilization: {dec_cpu_percent}%")
print(f"Decryption Memory Used: {dec_memory_used} bytes")
```