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

ПОМАЗИБІДА Василь Миколайович

Алгоритми гомоморфного шифрування для безпечних хмарних обчислень / Homomorphic encryption algorithms for secure cloud computing

спеціальність: 125 – Кібербезпека та захист інформації
освітньо-професійна програма – Кібербезпека

Кваліфікаційна робота

Виконав студент групи КБм -21
В.М. Помазибіда

Науковий керівник
д. філософії, С. В. Кулина

Кваліфікаційну роботу допущено
до захисту:

«____» _____ ___2025 р.

Завідувач кафедри
_____ В. В. Яцків

ТЕРНОПІЛЬ – 2025

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

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ В. В. Яцків
«____» _____ 2025 року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
ПОМАЗИБІДА Василь Миколайович
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи:
Алгоритми гомоморфного шифрування для безпечних хмарних обчислень / Homomorphic encryption algorithms for secure cloud computing
керівник роботи д. філософії С. В. Кулина
затверджені наказом по університету від 20 грудня 2024 року № 938

2. Строк подання студентом закінченої кваліфікаційної роботи 5 грудня 2025 р.

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

4. Основні питання, які потрібно розробити:
– провести аналіз методів шифрування даних;
– проаналізувати основні криптографічні алгоритми;
– проаналізувати основні гомоморфні криптографічні алгоритми;
– дослідити сфери застосування гомоморфних алгоритмів;
– провести аналіз принципів роботи гомоморфного шифрування;
– побудувати алгоритми гомоморфного шифрування даних
– реалізувати модулі шифрування та дешифрування пакетів даних на основі запропонованих алгоритмів гомоморфного шифрування.

5. Перелік графічного матеріалу у роботі:
– принципи шифрування та дешифрування;
– загальна схема порівняння гомоморфного шифрування;
– алгоритми шифрування даних;
– зображення і характеристики інфраструктури
– модуль шифрування;
– модуль дешифрування;
– модуль перевірки часових залежностей шифрування і дешифрування.

6. Консультанти розділів кваліфікаційної роботи

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

7. Дата видачі завдання 20 грудня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строки виконання етапів кваліфікаційної роботи	Примітка
1	Основи криптографічного захисту інформації	12.2024 р. – 03.2025 р.	
2	Алгоритми та моделі гомоморфного шифрування	03.2025 р. – 06.2025 р.	
3	Практична реалізація гомоморфного шифрування	06.2025 р. – 11.2025 р.	

Студент _____ Помазибіда В. М.
підпис

Керівник роботи _____ д. філософії, КУЛИНА С. В.
підпис

АНОТАЦІЯ

Помазибіда В. М. Алгоритми гомоморфного шифрування для безпечних хмарних обчислень. – Рукопис.

Дослідження на здобуття освітнього ступеня «магістр» за спеціальністю 125 «Кібербезпека та захист інформації», освітньо-професійна програма «Кібербезпека». – Західноукраїнський національний університет, Тернопіль, 2025.

У роботі досліджено поширені алгоритми гомоморфного шифрування даних, запропоновано та реалізовано алгоритм гомоморфного шифрування на основі обчислення шифрованих компонентів за допомогою відкритого ключа

Ключові слова: ГОМОМОРФНЕ ШИФРУВАННЯ, БЕЗПЕЧНІ ХМАРНІ ОБЧИСЛЕННЯ, КОНФІДЕНЦІЙНІСТЬ ДАНИХ, ОБРОБКА ЗАШИФРОВАНИХ ДАНИХ, ХМАРНЕ ЗБЕРІГАННЯ, БЕЗПЕКА ДАНИХ, СХЕМИ ШИФРУВАННЯ, ШИФРОТЕКСТ.

ABSTRACT

Pomazybida V. M. Homomorphic encryption algorithms for secure cloud computing. – Manuscript.

Doctoral studies for the education level «Master» with the title 125 «Cybersecurity and Information Protection». – West Ukrainian National University, Ternopil, 2025.

The thesis investigates common homomorphic data encryption algorithms, and proposes and implements a homomorphic encryption algorithm based on computing encrypted components using a public key.

Keywords: HOMOMORPHIC ENCRYPTION, SECURE CLOUD COMPUTING, DATA CONFIDENTIALITY, ENCRYPTED DATA PROCESSING, CLOUD STORAGE, DATA SECURITY, ENCRYPTION SCHEMES, CIPHERTEXT.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

RSA - Rivest–Shamir–Adleman.

AES - Advanced Encryption Standard. DES - Data Encryption Standard.

SHA - Secure Hash Algorithm.

PHE - Partially Homomorphic Encryption.

SHE - Somewhat Homomorphic Encryption.

FHE - Fully Homomorphic Encryption.

BGV - Brakerski-Gentry-Vaikuntanathan.

BFV - Brakerski-Fan-Vercauteren.

RLWE - Ring learning with errors.

YASHE - Yet Another Somewhat Homomorphic Encryption.

NLP - Natural Language Processing.

SSH - Secure Shell.

CKKS - Cheon-Kim-Kim-Song

HTTP - Hypertext Transfer Protocol.

HTTPS - Hypertext Transfer Protocol Secure.

REST API - Representational State Transfer Application Programming Interface.

JSON - JavaScript Object Notation.

TLS/SSL - Transport Layer Security / Secure Sockets Layer.

JWT - JSON Web Token.

ВСТУП

Перехід до хмарних технологій є наступним етапом розвитку проведення обчислень. Хмарні обчислення пропонують низку переваг своїм клієнтам. Оплачуючи використання апаратних та програмних ресурсів (серверів, розміщених у дата-центрах, додатків, програмного забезпечення тощо) користувачі можуть отримати через Інтернет доступ до потужних обчислювальних систем або систем зберігання даних великої ємності і без необхідності їх придбання та сплати додаткових витрат на обслуговування обладнання застосовувати їх у своїй діяльності. Хмарні технології повинні надавати користувачам гарантії щодо захисту конфіденційних даних, що зберігаються в їхніх центрах обробки. Спільне використання таких центрів декількома клієнтами за допомогою концепції віртуалізації не повинно загрожувати витоком даних, тому задля забезпечення захисту інформації застосовуються алгоритми шифрування, зокрема гомоморфне шифрування.

Гомоморфне шифрування представляє собою прогресивний підхід до забезпечення безпеки хмарних обчислень, дозволяючи виконувати обчислення безпосередньо над зашифрованими даними без необхідності їх розшифрування.

У роботі проведено аналіз поширених схем гомоморфного шифрування, що застосовуються для підвищення захищеності та конфіденційності інформації у процесі її зберігання та обробки при хмарних обчисленнях. Запропоновано алгоритм, який забезпечує шифрування даних протягом усього обчислювального процесу та гарантує, що конфіденційні дані залишаються захищеними від несанкціонованого доступу у процесі обробки.

Мета і завдання дослідження. Метою кваліфікаційної роботи є розробка алгоритму гомоморфого шифрування даних при хмарних обчисленнях.

Досягнення визначеної мети передбачає вирішення таких завдань:

– проаналізувати основні методи і цілі шифрування;

– проаналізувати способи застосування криптографічних алгоритмів та їх завдання;

– провести аналіз роботи гомоморфних алгоритмів для шифрування даних;

– запропонувати оптимальний алгоритм використання гомоморфного методу шифрування;

– створити алгоритм шифруваннння даних для безпечних хмарних обчислень;

– реалізувати модулі шифрування та дешифрування даних на основі запропонованого алгоритму гомоморфого шифрування.

Об’єкт дослідження – процеси шифрування та дешифрування даних у хмарних обчисленнях на стороні довіреної та недовіреної особи і обмін ключами.

Предмет дослідження – принципи та алгоритми гомоморфного шифрування, реалізація складових компонент для виконання обчислень на віддаленому чи хмарному пристрої.

Методи досліджень. Для розв’язання поставлених задач у даній кваліфікаційній роботі використано: теоретичний аналіз, порівняльний аналіз, аналіз часового та ресурсного навантаження, шифрування даних при надсиланні, обробці та зберіганні.

Наукова новизна одержаних результатів. Проведено аналіз поширених алгоритмів гомоморфного шифрування хмарних обчислень, запропоновано та реалізовано алгоритм гомоморфного шифрування на основі обчислення суми над шифрованими частинами даних і алгоритму їх оптимізації швидкодії і криптостійкості.

Практичне значення отриманих результатів. Реалізовано модулі відправника та отримувача на основі запропонованого у роботі алгоритму гомоморфного шифрування.

Публікації та апробація до магістерської роботи

1. Помазибіда В., Кулина С. Алгоритми гомоморфного шифрування для безпечних хмарних обчислень. Кібербезпека та комп’ютерно-інтегровані технології (КБКІТ-2025): Збірник матеріалів науково-практичної конференції молодих вчених, аспірантів та студентів. – Тернопіль, 2025. – С. 85–87.

2. Помазибіда В., Нетребяк М. Аналіз розвитку хмарних обчислень та проблеми їх безпеки. Захист інформації: Збірник матеріалів науково-практичного симпозіуму, 28.11.2025. – Тернопіль, 2025. – С. 83–85.

1. ТЕОРЕТИЧНІ ОСНОВИ КРИПТОГРАФІЧНОГО ЗАХИСТУ ІНФОРМАЦІЇ

1.1. Поняття криптографії

Криптографія – це техніка захисту інформації та комунікацій за допомогою кодів для забезпечення конфіденційності, цілісності та автентичності. Таким чином, запобігається несанкціонований доступ до інформації [1].

Слово «криптографія» походить від грецького слова kryptos, що означає «прихований». Префікс «crypt-» означає «прихований» або «сховище», а суфікс «-graphy» означає «письмо». Походження криптографії зазвичай датується приблизно 2000 роком до н. е., коли в Єгипті використовували ієрогліфи. Вони складалися зі складних піктограм, повне значення яких було відомо лише небагатьом обраним.

Першим відомим використанням сучасного шифру був Юлій Цезар (100 р. до н. е. – 44 р. до н. е.), який не довіряв своїм посланцям під час спілкування зі своїми губернаторами та офіцерами. З цієї причини він створив систему, в якій кожен символ у його повідомленнях замінювався символом, що стояв на три позиції попереду нього в латинській абетці.

Останнім часом криптографія перетворилася на поле битви деяких з найкращих математиків і комп'ютерних вчених світу. Здатність безпечно зберігати та передавати конфіденційну інформацію виявилася критичним фактором успіху у війні та бізнесі [2]. Оскільки уряди не хочуть, щоб певні організації в їхніх країнах та за їх межами мали доступ до засобів отримання та надсилання прихованої інформації, яка може становити загрозу національним інтересам, криптографія піддається різним обмеженням у багатьох країнах, починаючи від обмежень використання та експорту програмного забезпечення до публічного поширення математичних концепцій, які можуть бути використані для розробки криптосистем.

Однак Інтернет дозволив поширити потужні програми і, що ще важливіше, основні техніки криптографії, завдяки чому сьогодні багато найсучасніших криптосистем та ідей є загальнодоступними. У криптографії техніки, що

використовуються для захисту інформації, базуються на математичних концепціях та наборі розрахунків на основі правил, відомих як алгоритми, для перетворення повідомлень таким чином, щоб їх було важко розшифрувати. На рисунку 1.1 зображено основний процес роботи з даними в криптографії.

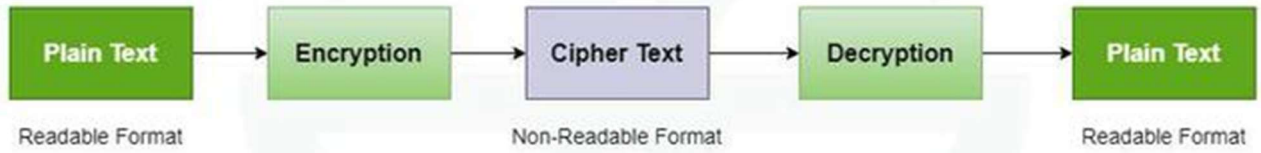

Рисунок 1.1 - Процес шифрування

Особливості криптографії, які роблять її популярним вибором у різних сферах застосування, можна перерахувати так:

1) Конфіденційність - доступ до інформації має лише особа, для якої вона призначена, і ніхто інший, крім неї, не може отримати до неї доступ.

2) Відмовостійкість - автор / відправник інформації не може пізніше заперечувати свій намір надіслати інформацію.

3) Цілісність - інформація не може бути змінена під час зберігання або передачі між відправником і призначеним одержувачем без виявлення будь-яких доповнень до інформації.

4) Адаптивність - криптографія постійно розвивається, щоб випереджати загрози безпеці та технологічні досягнення.

5) Сумісність - криптографія забезпечує безпечний обмін даними між різними системами та платформами.

6) Аутентифікація - підтверджується особистість відправника та одержувача, також підтверджується місце призначення/походження інформації.

1.2. Основні криптографічні алгоритми

Криптографічний алгоритм - це набір кроків, які можна використовувати для перетворення відкритого тексту в зашифрований текст. Криптографічний алгоритм також відомий як алгоритм шифрування [12].

Криптографічний алгоритм використовує ключ шифрування для приховування інформації та перетворення її в нечитабельний формат. Аналогічно, ключ дешифрування можна використовувати для перетворення її назад у відкритий читабельний текст (рис. 1.2).

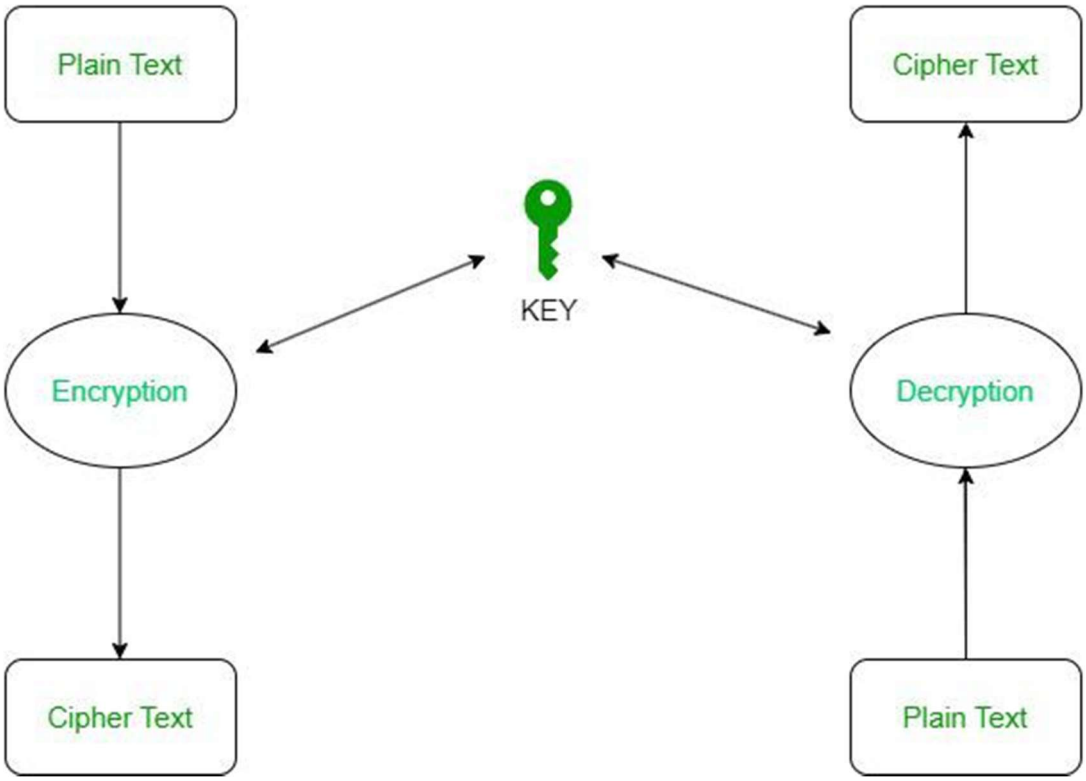

Рисунок 1.2 - Процес шифрування з ключем

Для захисту конфіденційних даних і розмов криптографія використовує складні алгоритми. Ці математичні формули забезпечують процеси шифрування, дешифрування, підпису та перевірки, які захищають секретні дані під час передачі та зберігання [3].

Існують різні типи криптографічних алгоритмів, але можна узагальнити 4 основні типи криптографічних алгоритмів.

1. Стандарт шифрування даних (DES).

2. Розширений стандарт шифрування (AES).

3. Алгоритм RSA (алгоритм Рівеста, Шаміра, Адлемана).

4. Алгоритм безпечного хешування (SHA).

1. DES - це старий алгоритм шифрування, який використовується для перетворення 64-бітних даних у вигляді відкритого тексту в 48-бітні зашифровані дані. Він використовує симетричні ключі (тобто один і той самий ключ для шифрування та дешифрування). За сучасними стандартами він є дещо застарілим, але може використовуватися як базовий елемент для вивчення нових алгоритмів шифрування (рис. 1.3) [6].

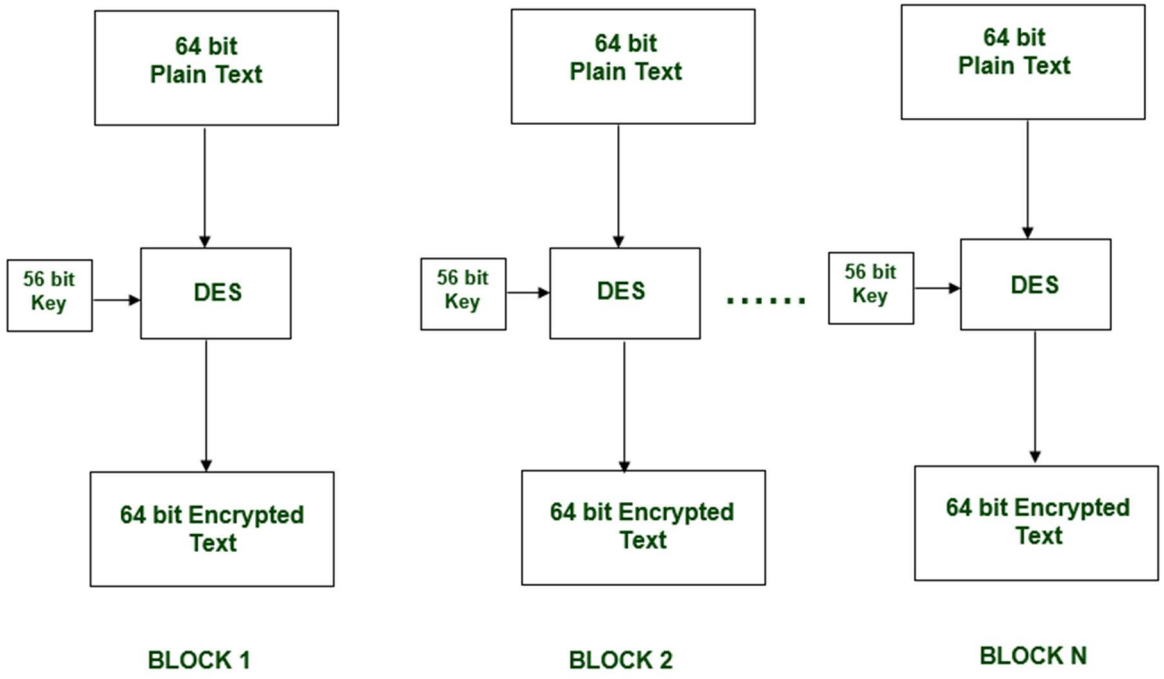

Рисунок 1.3 – Процес шифрування з DES

Основними характеристиками DES можна назвати:

1) Симетричний тип - DES використовує один і той самий симетричний ключ, тому шифрування та дешифрування можна виконувати за допомогою одного ключа, використовуючи той самий алгоритм.

2) Простіша реалізація - DES був розроблений для апаратного забезпечення, а не для програмного, і демонструє ефективність та швидку реалізацію в апаратному забезпеченні.

3) Техніка шифрування - використовується шифрування транспозицією та підстановкою: цей алгоритм використовує як шифрування транспозицією, так і шифрування підстановкою.

4) Будівельний блок - техніка DES виступає будівельним блоком для інших криптографічних алгоритмів.

2. AES – популярний алгоритм шифрування, який використовує один і той самий ключ для шифрування та дешифрування. Це симетричний алгоритм блокового шифрування з розміром блоку 128 біт, 192 біт або 256 біт. Алгоритм AES широко вважається заміною алгоритму DES (Data encryption standard) [6, 17].

В таблиці 1.1 наведено поширені типи алгоритму AES в залежності від кількості раундів і розміру ключа

Таблиця 1.1

Порівняння основних розмірів блоків AES

Тип AES	Ключ (біт)	Кількість раундів
AES-128	128	10
AES-192	192	12
AES-256	256	14

Чим більше раундів, тим безпечніше шифрування. Ось чому AES-256 вважається найбезпечнішим шифруванням (рис. 1.4).

Основними характеристиками AES можна назвати:

1) Багато розмірів ключів - доступні три розміри ключів: 128, 192 і 256 біт.

2) Безпека - комплексні заходи безпеки для захисту від загроз.

3)Універсальність - універсальний, оскільки може використовуватися як для апаратного, так і для програмного забезпечення.

4) Широке застосування - широко використовується в різних додатках, включаючи Google Cloud, Facebook і менеджери паролів.

3. RSA – це базовий асиметричний криптографічний алгоритм, який використовує два різних ключі для шифрування [13]. Алгоритм RSA працює за принципом блокового шифрування, яке перетворює звичайний текст на зашифрований і навпаки.

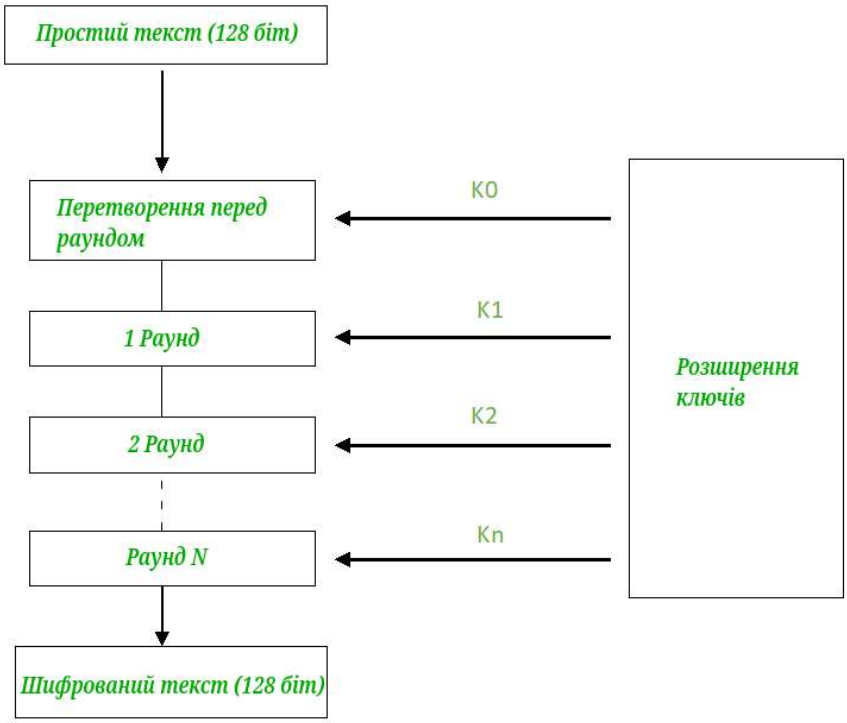

Рисунок 1.4 – Процес шифрування з AES

Алгоритм RSA є асиметричним криптографічним алгоритмом, асиметричний означає, що він працює з двома різними ключами, а саме: відкритим ключем і закритим ключем [11]. На рисунку 1.5 показано звичайний процес шифрування тексту по RSA.

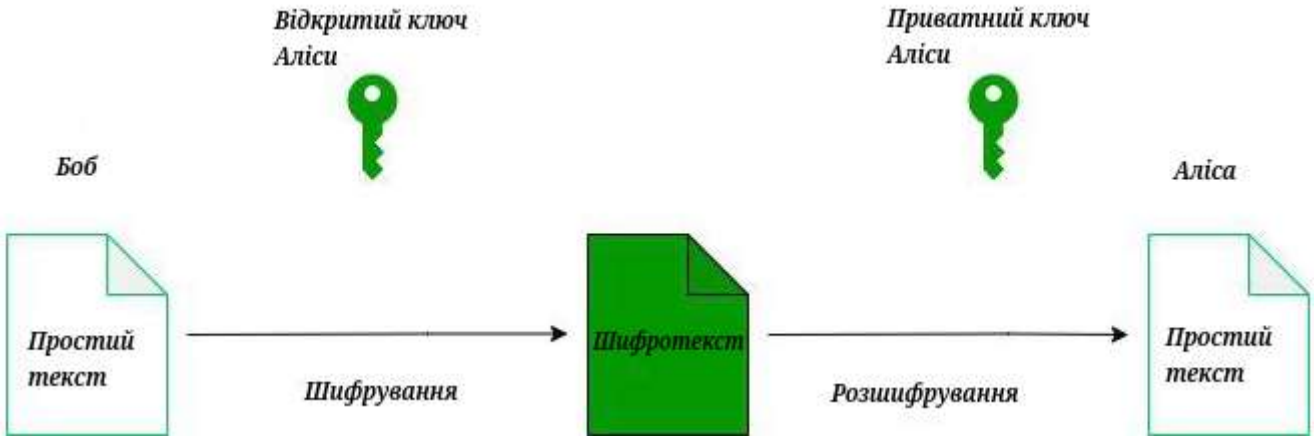

Рисунок 1.5 - Процес шифрування з RSA

Основними характеристиками RSA можна назвати:

1) Безпека - багато хто вважає метод RSA дуже безпечним і широко використовує його для передачі даних.

2) Висока швидкість - підхід RSA відомий своєю швидкістю. Його можна швидко впровадити, коли виникає потреба в криптографії.

3) Різні ключі - у RSA для шифрування та дешифрування даних використовуються два ключі. Відкритий ключ - для шифрування інформації, а закритий ключ – для дешифрування.

4) Обмін ключами - за допомогою RSA можна забезпечити безпечний обмін, що дозволяє двом сторонам обмінюватися ключами без передачі їх через мережу.

4. SHA використовується для генерації унікальних цифрових відбитків вхідних

даних фіксованої довжини, відомих як хеші. Варіанти SHA, такі як SHA-2 і SHA-3, зазвичай використовуються для забезпечення цілісності та автентичності даних. Найменша зміна вхідних даних різко змінює вихідний хеш, що вказує на втрату цілісності. Хешування – це процес зберігання пар ключ-значення за допомогою хеш-функції в хеш-таблиці.

Основними характеристиками SHA можна назвати:

1) Безпека - SHA 256 широко визнаний серед хеш-алгоритмів завдяки своїм надійним функціям безпеки. Він ефективно запобігає колізійним атакам, гарантуючи, що різні вхідні дані не дають однакового хеш-значення. Веб-сайти надають пріоритет конфіденційності користувачів, зберігаючи паролі у зручному форматі [11].

2) Одностороннє хешування - використання алгоритмів SHA для одностороннього хешування дозволяє зберігати таку інформацію, як паролі. Хешування даних у вихідні дані фіксованої довжини спрощує індексацію та порівняння. Навіть незначна зміна в повідомленні призводить до зміни хеш-значення при використанні алгоритмів SHA, що полегшує виявлення пошкоджених даних.

3) Ефект лавини - невелика зміна вхідного значення, навіть одного біта, повністю змінює результуюче хеш-значення.

4) Змінна довжина вхідних даних і фіксована довжина вихідних даних - алгоритм SHA складається із змінної довжини вхідних даних (тобто довжина вхідних даних є динамічною) і фіксованої довжини вихідних даних.

1.3. Розвиток хмарних обчислень та проблеми безпеки

Хмарні обчислення за останні роки значно зросли в популярності та застосуванні. Переваги хмарних обчислень численні, серед них економія коштів, підвищена масштабованість, покращена доступність та спрощене управління IT. Компанії будь-якого розміру переходять на хмарні технології завдяки їхній здатності надавати обчислювальні ресурси, сховища та програмні послуги на вимогу без необхідності розбудови великої внутрішньої інфраструктури.

Хмарні обчислення вже існували під різними назвами, такими як «аутсорсинг» та «хостинг серверів» [22]. Але низька продуктивність використовуваних процесорів, повільне підключення до Інтернету та надмірна вартість використовуваних матеріалів не дозволяють використовувати послуги та місця для зберігання даних. Однак останні досягнення в сучасних технологіях (завдяки віртуалізації) проклали шлях для цих операцій із більш швидкою обробкою. Проблеми безпеки хмарних обчислень також є актуальними для багатьох дослідників; першочерговим завданням було зосередитися на безпеці, яка є найбільшою проблемою для організацій, що розглядають можливість переходу до хмарних технологій. Використання хмарних обчислень дає багато переваг, включаючи зниження витрат, легке обслуговування та повторне надання ресурсів.

Вперше концепція хмарних обчислень була реалізована в 2002 році компанією Amazon Web Services, коли вона здавала в оренду свої ресурси компаніям у періоди свят (коли не було пікового навантаження на її IT-системи) за запитом (рис. 1.6).

Багато людей щодня користуються хмарою, навіть не знаючи про це. Наприклад, у всіх версіях електронної пошти (Gmail або Webmail) та доступі до програм, які фізично не встановлені на локальному ПК, таких як Excel, Microsoft Word і тому подібне.

Це використання здійснюється завдяки Інтернету, але клієнти можуть не знати місцезнаходження серверів, на яких зберігаються їхні електронні листи та розміщений вихідний код програм, якими вони користуються [3, 23].

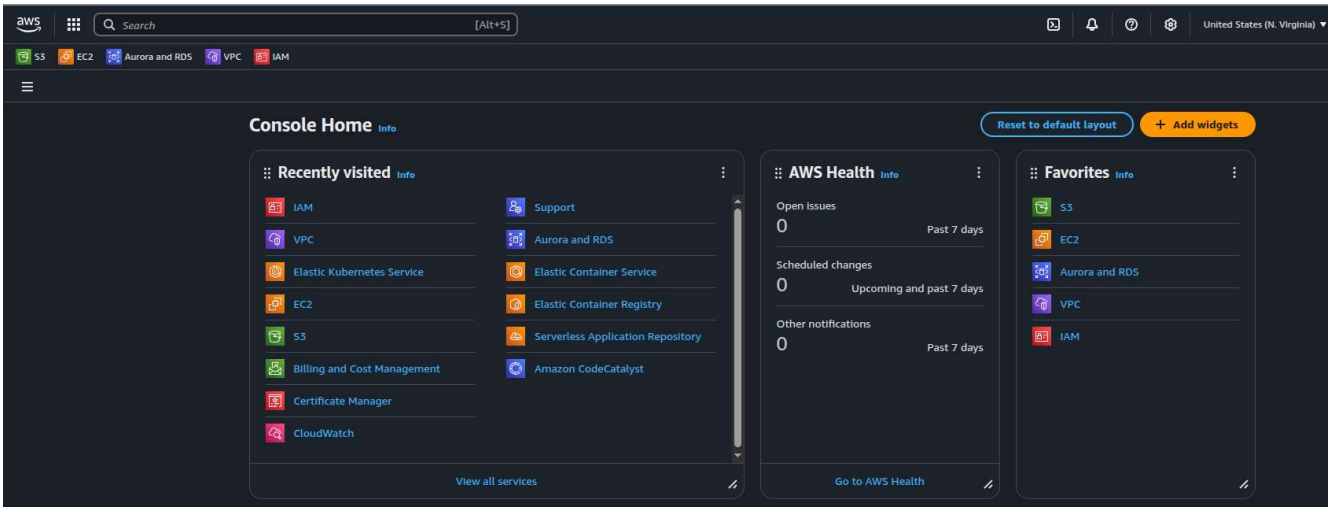

Рисунок 1.6 - Електронна комерція Amazon Web Services в сучасний час

Послуги, що пропонуються постачальниками хмарних обчислень, надходять з величезних цифрових станцій, які називаються дата-центрами, і використовують технології, засновані на віртуалізації [3].

Віртуалізація – це весь технічний матеріал та/або програмне забезпечення, яке може працювати на одній машині з декількома операційними системами та/або декількома додатками, окремо один від одного, як якщо б вони працювали на окремих фізичних машинах. Віртуалізація та консолідація можуть спростити управління парком серверів, зменшивши кількість машин, що потребують обслуговування, за рахунок оптимізації використання ресурсів та забезпечення високої доступності. Але впровадження та перехід до хмарних обчислень можливі лише за умови забезпечення безпеки. Як гарантувати кращий захист даних і як зберегти конфіденційність особистої інформації клієнтів? Існує два основних питання, які становлять виклик для постачальників хмарних обчислень [11].

Однак зростаюча залежність від хмарних обчислень також викликала значні проблеми з безпекою. Коли організації передають свої дані та обчислення хмарним постачальникам послуг, вони відмовляються від прямого контролю над своєю конфіденційною інформацією. Це створює новий набір ризиків для безпеки, оскільки дані тепер зберігаються та обробляються на інфраструктурі, яка не належить організації та не контролюється нею повністю [10].

Система хмарних обчислень поділяється на дві частини: фронтенд і бекенд. Фронтенд – це частина, через яку користувач може взаємодіяти з сервером, а бекенд

– це сервер, який надає дані клієнту. Між сервером і клієнтом мережа працює як проміжне програмне забезпечення (рис. 1.7).

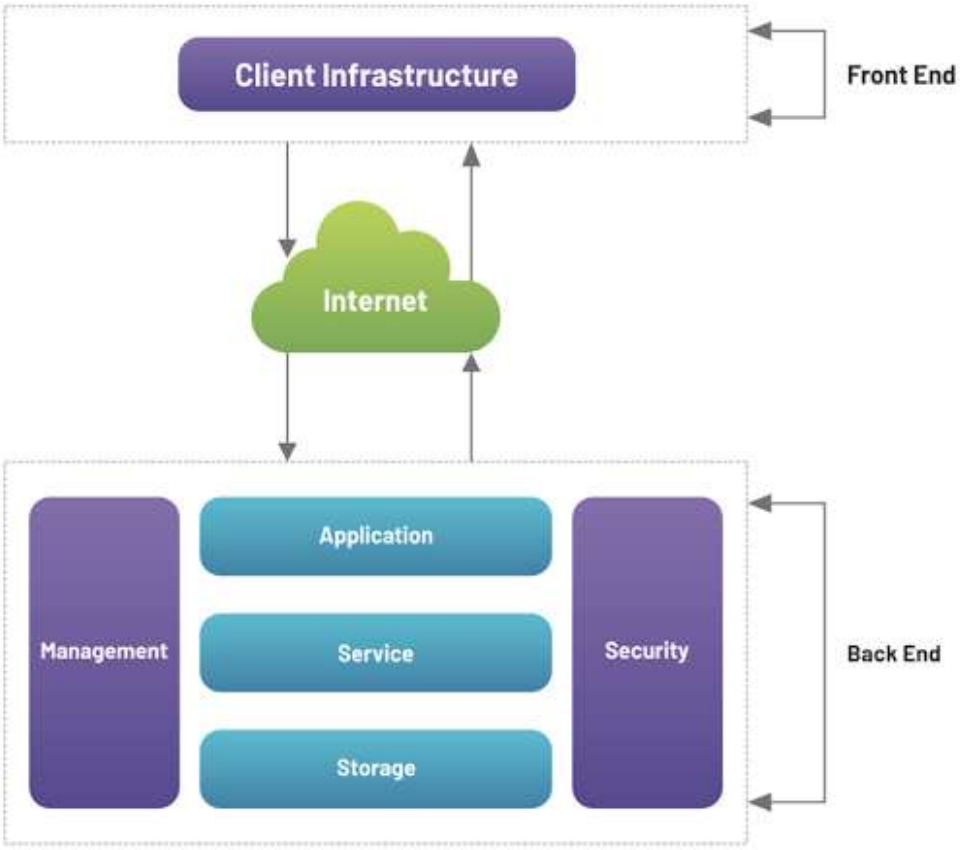

Рисунок 1.7 - Система хмарної архітектури

Хмарні обчислення є одним з найвизначніших досягнень у світі інформаційних технологій за останнє десятиліття. IT-компанії надають гнучкі хмарні послуги та інфраструктуру практично для будь-яких цілей, що дозволяє клієнтам швидко масштабувати свої послуги за допомогою моделей надання послуг з оплатою за фактичне використання.

Питання про потенційні ризики, пов'язані з втратою конфіденційності, цілісності та доступності даних користувачів у хмарних мережах через технічні збої або навіть цілеспрямовані атаки як ззовні, так і зсередини мережі хмарного провайдера, обговорювалися в наукових працях [15, 21].

Хмарні провайдери зазвичай використовують загальні моделі захисту безпеки, які вимагають довіри до хмарного провайдера, та класичні прозорі для користувача заходи безпеки для захисту своїх мереж та задоволення цілей безпеки своїх клієнтів. Ці заходи все ще перебувають на початковій стадії та схильні до

загальних загроз безпеки, таких як примус хмарних провайдерів під тиском місцевої (політичної) влади розкривати дані користувачів хмари [25].

Обчислення зашифрованих даних базуються на двох незалежних гомоморфних алгоритмах шифрування, що забезпечують гомоморфність для додавання та множення. Модель складається з агента, який виступає делегатором, та принаймні одного працівника в хмарі, який виконує обчислення, однак необхідність повторного шифрування даних для переходу між операціями додавання та множення обмежувала застосовність.

Визначення хмарних обчислень, про які згадувалося раніше, не згадує жодних понять безпеки даних, що зберігаються в хмарних обчисленнях, навіть незважаючи на те, що це нещодавнє визначення. Тому ми розуміємо, що хмарним обчисленням бракує безпеки, конфіденційності та прозорості. Надання інфраструктури (IaaS), платформи (PaaS) або програмного забезпечення (SaaS) як послуги є недостатнім, якщо хмарний провайдер не гарантує кращої безпеки та конфіденційності даних клієнтів.

Зазвичай, ми вважаємо хмарними обчисленнями будь-яку обробку або зберігання особистої або професійної інформації, що здійснюється поза відповідною структурою (тобто поза компанією) забезпечити безпеку хмари означає забезпечити безпеку обробки (обчислень) та зберігання (баз даних, що розміщуються у хмарного провайдера).

Хмарні провайдери, такі як IBM, Google та Amazon, використовують віртуалізацію на своїх хмарних платформах, і на одному сервері можуть співіснувати віртуалізовані простори зберігання та обробки, що належать різним підприємствам.

Аспект безпеки та конфіденційності повинен бути врахований для захисту даних кожного з підприємств. Безпечне зберігання та обробка даних вимагають використання сучасних методів криптографії, що має критерії обробки, такі як необхідний час для відповіді на будь-який запит, надісланий клієнтом та розмір зашифрованих даних, які будуть зберігатися на хмарному сервері.

Передача обробки ваших даних третій стороні також означає передачу частини відповідальності, пов'язаної з їхньою безпекою та відповідністю вимогам.

Не дивно, що фахівці з безпеки нервують. Тому дуже важливо, щоб ви повністю довіряли своєму хмарному провайдеру. Переваги хмарних обчислень включають зниження витрат, легке обслуговування та повторне надання ресурсів, а отже, збільшення прибутку. Тому загалом прийнято шифрувати дані перед надсиланням їх хмарному провайдеру, але для виконання обчислень дані слід розшифровувати щоразу, коли нам потрібно з ними працювати. Дотепер було неможливо шифрувати дані і довіряти третій стороні їх безпеку та можливість виконувати віддалені обчислення на них [26].

Отже, щоб дозволити хмарному провайдеру виконувати операції з зашифрованими даними без їх розшифрування, потрібно використовувати криптосистеми на основі гомоморфного шифрування (рис. 1.8) [10].

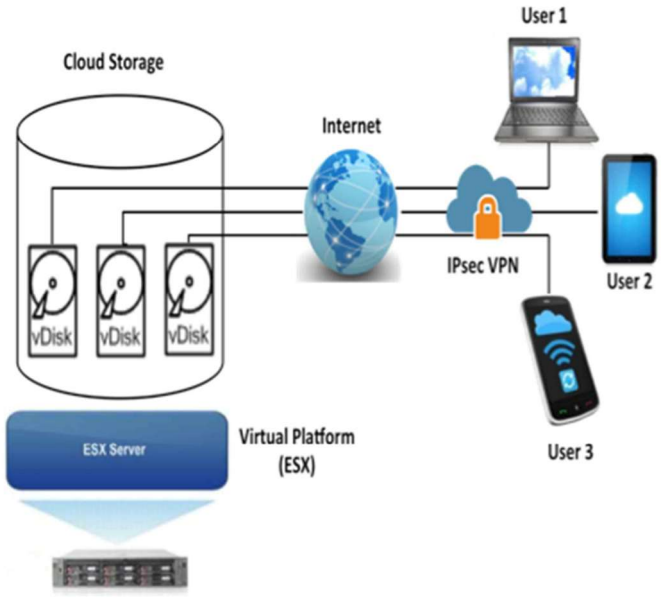

Рисунок 1.8 - Схема базової архітектури хмарних обчислень

1.4. Характеристики хмарних обчислювальних послуг

Під хмарними обчисленнями ми розуміємо модель інформаційних технологій (IT) для обчислень, яка складається з усіх IT-компонентів (апаратного забезпечення, програмного забезпечення, мереж та послуг), що необхідні для розробки та надання хмарних послуг через Інтернет або приватну мережу.

Серед характеристик хмари можна навести:

1) Еластичність - це одна з найважливіших характеристик нашого бачення хмари. Вона визначає здатність певної інфраструктури динамічно адаптуватися до масштабу.

2) Здатність до адаптації - хмара повинна забезпечувати набір засобів автоматизації, що дозволяють їй самостійно управляти собою. Її адміністрування повинно вимагати мінімального втручання людини.

3) Якість обслуговування - це ще один ключовий аспект хмари, що використовує такі показники, як час відгуку, кількість операцій за секунду; послуга надає гарантії своїм користувачам. Користувач більше не повинен вирішувати, які ресурси розгортати, а лише визначати термінали, які послуга повинна обслуговувати [32].

4) Висока доступність - використовуючи репліковані дані в різних центрах обробки даних, хмара повинна забезпечувати надійність, нечутливу до відмови інстанції або центру обробки даних.

5) Зниження витрат - Pay Per Use (оплата за використання) означає, що користувач платить за послугу тільки на основі її використання.

6) Екологічний підхід - розподіл ресурсів за суворою необхідністю для зменшення енергоспоживання IT-парків.

Окрім економічного аспекту, ці скорочення дозволяють зменшити екологічний вплив компанії на енергоспоживання [28].

2. АЛГОРИТМИ ТА МОДЕЛІ ГОМОМОРФНОГО ШИФРУВАННЯ

2.1. Поняття та властивості гомоморфного шифрування

Гомоморфне шифрування означає шифрування, при якому відкриті тексти та зашифровані тексти обробляються за допомогою еквівалентної алгебраїчної функції. Гомоморфне шифрування дозволяє серверу виконувати операції над зашифрованими даними, не знаючи оригінального відкритого тексту.

На рисунку 2.1 показано загальну схему гомоморфного шифрування.

Рисунок 2.1 - Загальна структура захисту даних у хмарі

Гомоморфне шифрування дозволяє виконувати складні математичні операції над зашифрованими даними без використання вихідних даних. Для відкритих текстів X1 і X2 та відповідних шифрованих текстів Y1 і Y2 гомоморфна схема шифрування дозволяє обчислювати X1 Θ X2 з Y1 і Y2 без використання P1 Θ P2. Криптосистема є мультиплікативною або адитивною гомоморфною залежно від операції Θ, яка може бути множенням або додаванням [31].

Основними типами гомоморфого шифруваня є:

1) Адитивне гомоморфне шифрування;
2) Мультиплікативне гомоморфне шифрування

В таблиці 2.1 наведено порівняльну схему гомоморфного шифрування відповідно до своїх властивостей.

Таблиця 2.1

Гомоморфні схеми шифрування

Схема	**Гомоморфні властивості**	**Алгоритм**
RSA	Multiplicative	Асиметричний
Elgaml	Multiplicative	Асиметричний
Goldwasser Micali	XOR	Асиметричний
Benalh	Additive	Симетричний
Paillier	Additive	Асиметричний
Okamoto uchiyama	Additive	Асиметричний

Гомоморфне шифрування є адитивним, якщо:

$$Enc(x \oplus y) = Enc(x) \oplus Enc(y)$$

$$Enc(\sum_{i=1}^{1} m_i) = \prod_{i=1}^{1} Enc(m_i)$$

Хорошим прикладом є криптосистема Пайє, яка реалізує властивість адитивного гомоморфного шифрування. Одним із застосувань адитивного гомоморфного шифрування є електронне голосування: кожен голос шифрується, але розшифровується лише «сума». Послідовність дій для проведення такого голосування представлено в таблиці 2.2 [24].

Таблиця 2.2

Криптосистема Пайє (1999)

Генерація ключа: KeyGen(p,q)	Шифрування: Enc(m, pk)	Дешифрування: Dec(c, sk)
Вхідні дані: p , q € P	Вхідні дані: м € Zn	Вхідні дані: C € Zn
Обчислити: n=p*q, і λ=lcm(p-1)(q-1). Вибрати g € Zn таким чином, щоб Gcd(L(g^λ mod n2), n)=1 З L(u)=(u-1)/n	Вибрати r € Zn Обчисліть: c= gm * rn mod n2	Обчислити: m= mod n [L(c^λ mod n2)/L(g^λ mod n2)]
Вихідні дані: (pk , sk) Відкритий ключ: pk=(n, g) Таємний ключ: sk=(p,q)		Вихідні дані: m € Zn

В алгоритмі Пає для для демонстрації адитивної гомоморфності необхідно обчислити добуток двох шифротекстів, наприклад, припустимо, що ми маємо два повідомлення m1 і m2, та відповідні їм шифротексти C1 і C2, такі що:

$C_1 = gm_1\ r_1\ n \bmod n^2$

$C_2 = gm_2\ r_2\ n \bmod n^2$

Якщо перемножити ці два шифри, ми отримаємо:

$C_1\ C_2 = gm_1\ r_1\ n\ gm_2\ r_2\ n \bmod n^2 = gm_1 + m_2\ (r_1\ r_2)\ n \bmod n^2$

Гомоморфне шифрування є мультиплікативним, якщо:

$$Enc(\prod_{i=1}^{1} m_i) = \prod_{i=1}^{1} Enc(m_i)$$

$$Enc(x \otimes y) = Enc(x) \otimes Enc(y)$$

В таблиці 2.3 показана криптосистема RSA, яка має властивості мультиплікативного гомоморфного шифрування, але вона не відповідає належним поняттям безпеки, оскільки криптосистема RSA працює з властивістю мультиплікативного гомоморфного шифрування

Таблиця 2.3

Криптосистема RSA для гомоморфного шифрування (1978)

Генерація ключа KeyGen(p,q)	Шифрування: Enc(m,pk)	Дешифрування: Dec(c, sk)
Вхідні дані: p, q € P	Вхідні дані: м€ Zn	Вхідні дані: c € Zn
Обчислити: n=p*q, і Ø(n)=(p1)(q-1)	Обчислити: c = me mod n	Обчислити: m=cd mod n
Вибрати e таке, що Gcd(e,□(n))=1 Визначте d таке, що e * d□1mod □(n)	Вихід: c € Zn	Вихід: м € Zn
Вихідні дані: (pk ,sk) Відкритий ключ: pk=(e,n) Таємний ключ: sk=(d)		

Якщо припустити, що два шифри C1, C2 відповідають повідомленням m1, m2 відповідно, то:

$C_1 = m_1^e \bmod n$

$C_2 = m_2^e \bmod n$

Якщо виконати операцію множення над цими двома шифротекстами, отримаємо:

$$C_1 C_1 = m_1^e m_2^e \bmod n = (m_1 m_1)^e \bmod n$$

Передавач надсилає пару (C_1, C_2) на хмарний сервер, сервер виконує розрахунки, запитувані клієнтом, і надсилає зашифрований результат (C_1 x C_2) клієнту [29]. Якщо зловмисник перехопить два шифри C_1 і C_2, які зашифровані одним і тим же ключем, він зможе розшифрувати всі повідомлення, обмінені між двома комунікаціями, оскільки гомоморфне шифрування є мультиплікативним,тобто добуток шифрів дорівнює шифру добутку тому і для забезпечення захищеного комунікування слід використовувати надійніші і стійкіші алгоритми (рис. 2.2) [4].

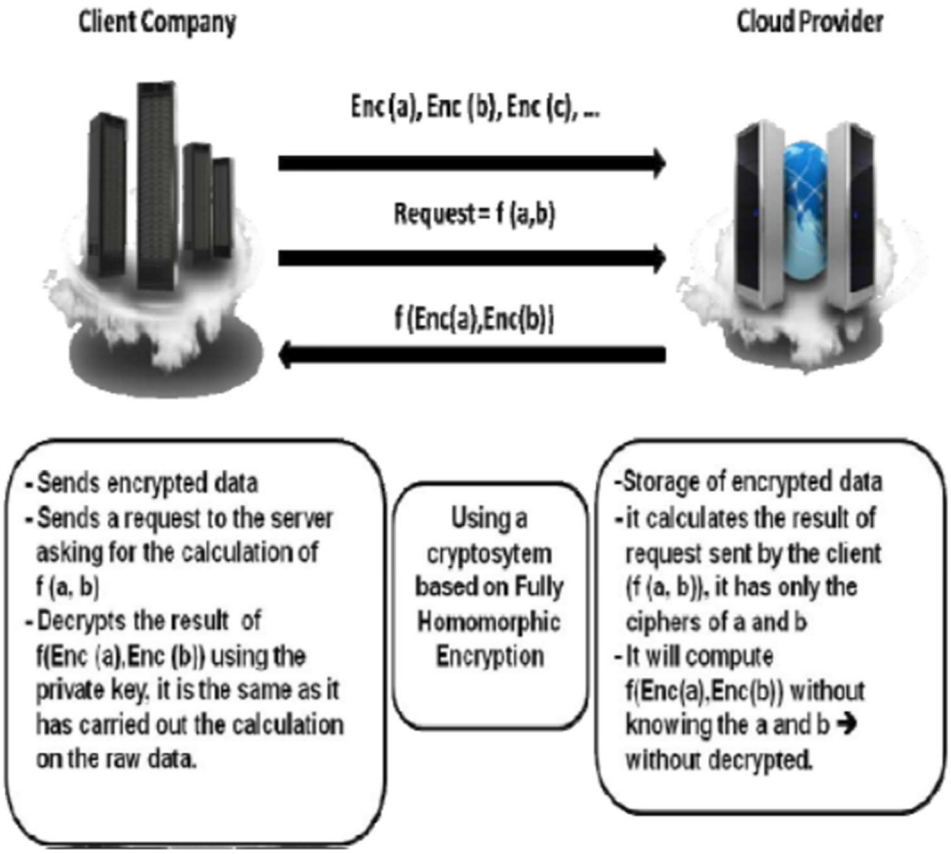

Рисунок 2.2 - Загальна структура хмарних послуг із захистом даних

2.2. Класифікація гомоморфних систем

Гомоморфна криптосистема функціонує аналогічно до інших типів публічного шифрування в тому сенсі, що вона використовує публічний ключ для шифрування даних і дозволяє отримати доступ до незашифрованих даних лише особі, яка має відповідний приватний ключ. Однак її особливість полягає у використанні алгебраїчної системи, яка дозволяє виконувати ряд обчислень (або операцій) над зашифрованими даними.

Більшість гомоморфних систем шифрування оптимально функціонують з даними, вираженими у вигляді цілих чисел, та при використанні операційних функцій, таких як додавання та множення. Це дозволяє аналізувати та обробляти зашифровані дані так, ніби вони знаходяться у форматі відкритого тексту, без фактичного їх розшифрування. Зашифровані дані можна обчислювати та обробляти для отримання зашифрованої відповіді, але тільки власник приватного ключа може розшифрувати шифрований текст і зрозуміти його значення.

Гомоморфне шифрування класифікується на три категорії залежно від виду та частоти математичних обчислень, які можуть виконуватися над шифрованим текстом (рис. 2.3) [10].

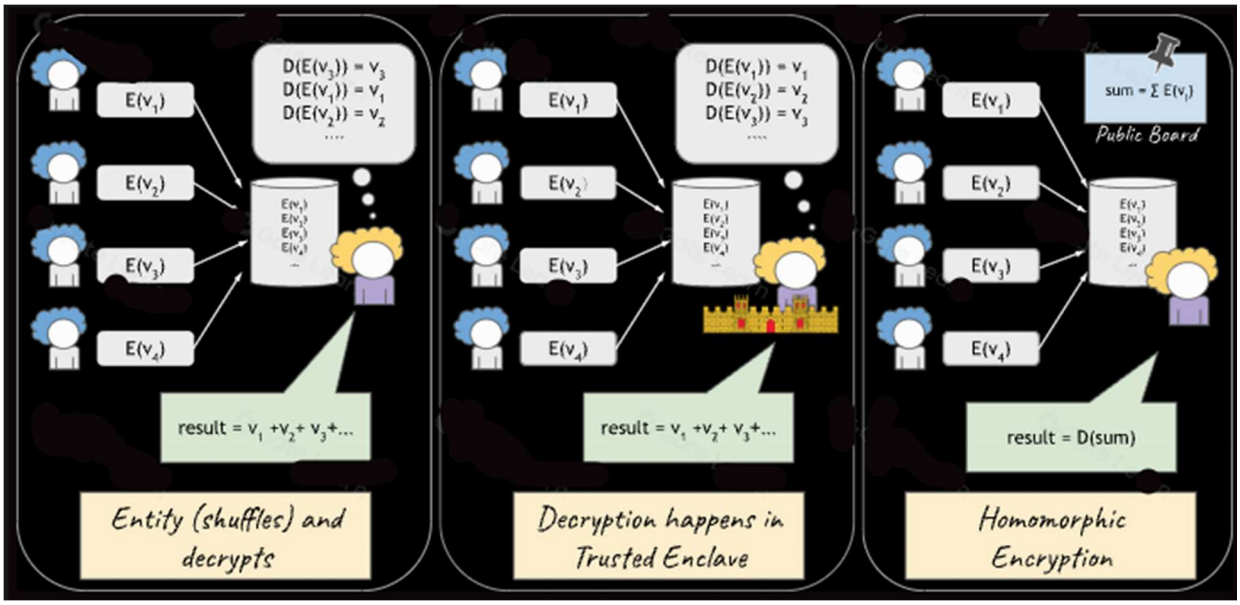

Рисунок 2.3 – Сценарії роботи системи шифрування

1) **Частково гомоморфне шифрування** (PHE) - позначає схему шифрування, яка оцінює схеми, що складаються тільки з одного типу математичних логічних елементів. Прикладами математичних логічних елементів є додавання і множення. Хоча це обмежує потенційні можливості застосування, схеми частково гомоморфного шифрування відносно прості в розробці [14].

Прикладом адитивної частково гомоморфної криптосистеми є використання криптосистеми Paillier, яка є криптосистемою з відкритим ключем. Криптографія з відкритим ключем – це криптографічна система, яка використовує пари ключів для шифрування та дешифрування. Відкритий ключ може бути відомий іншим, але приватний ключ відомий тільки власнику. У цій системі будь-яка особа може зашифрувати повідомлення за допомогою відкритого ключа, але зашифроване повідомлення можна розшифрувати тільки за допомогою приватного ключа.

Гомоморфне шифрування є кроком у правильному напрямку для досягнення безпеки даних без заплутування інформації. Більшість схем гомоморфного шифрування, навіть частково гомоморфні схеми, такі як схема Рівіста-Шаміра-Адлемана (RSA), є захищеними від традиційних атак [10]. Хоча частково гомоморфне шифрування є найпростішою формою гомоморфного шифрування, воно все одно може забезпечити безпечний обмін даними. Сучасне PHE базується на криптосистемах, які не захищені від квантових обчислень, тоді як найновіше повністю гомоморфне шифрування (FHE) вважається квантово-безпечним [8, 27].

PHE підтримує операції піднесення до степеня як розширення підтримки мультиплікативних операцій.

Наприклад, алгоритм шифрування RSA є мультиплікативно гомоморфним алгоритмом, який використовується комп'ютерами для шифрування та дешифрування повідомлень і є формою криптографії з відкритим ключем (рис. 2.4) [19].

З шифруванням RSA множення двох шифрованих текстів, зашифрованих одним і тим же ключем, еквівалентно піднесенню до степеню секретного ключа добутку відкритих текстів. Обмежувальним математичним фактором PHE є не складність операцій, а кількість їх типів.

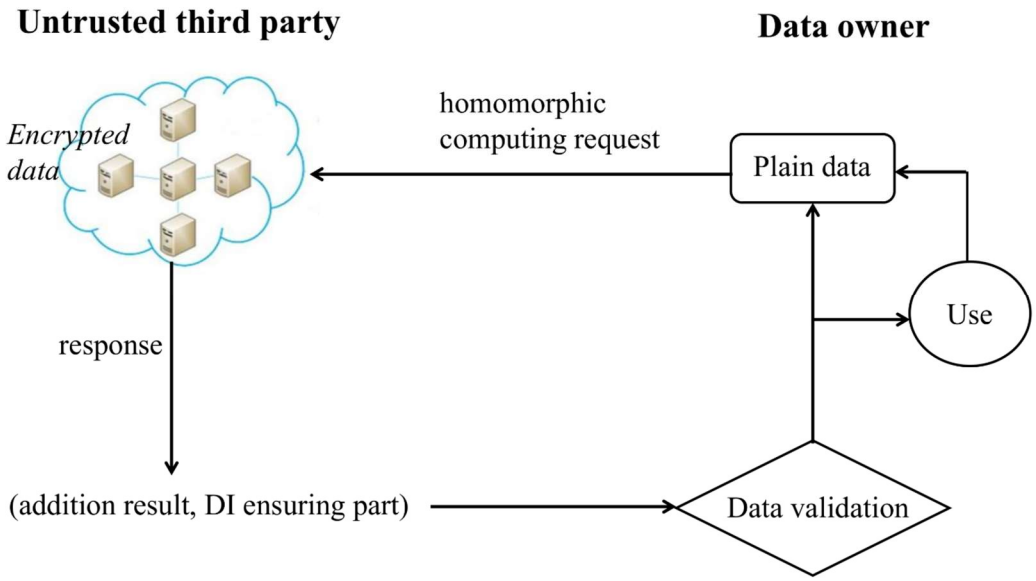

Рисунок 2.4 – Багатоключове PHE з схемою для пристроїв низького класу

2) **Дещо гомоморфне шифрування** (SHE) - відноситься до схеми шифрування, яка може оцінювати два типи операцій, але тільки для підмножини схем. Обмеження дещо гомоморфного шифрування виникає, коли шифрований текст генерує занадто багато шуму в даних. Коли в шифрованому тексті більше шуму, обчислювальні накладні витрати збільшуються, і схема SHE працює повільніше(рис. 2.5).

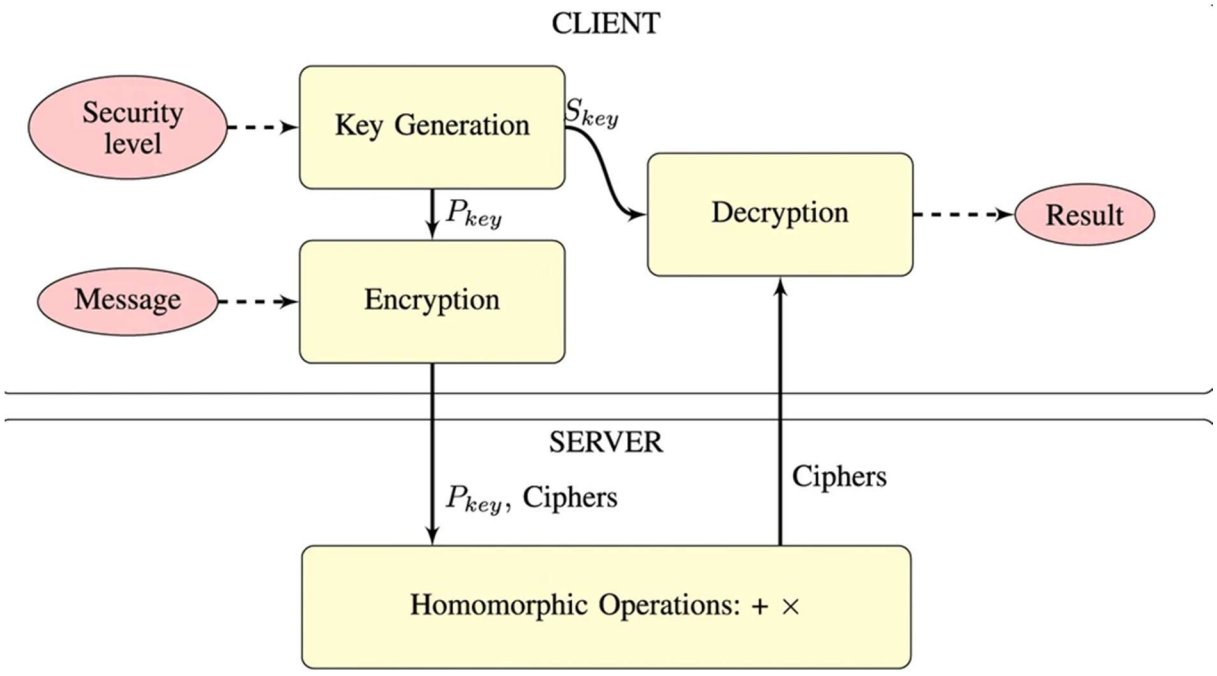

Рисунок 2.5 – Практичні параметри для SHE на бінарних схемах

Іншим обмеженням схеми SHE є мультиплікативна глибина. Це максимальна кількість множень, для виконання яких була створена схема SHE. Навіть з цими обмеженнями SHE все ще може ефективно використовуватися у фізичних додатках, що передбачають когерентні стани.

3) **Повністю гомоморфне шифрування (FHE)** - дозволяє оцінювати довільні схеми, що складаються з декількох типів логічних елементів необмеженої глибини. Іншими словами, FHE є найнадійнішим з трьох типів гомоморфного шифрування [16].

FHE може використовуватися в більш складних ситуаціях забезпечення безпеки даних і допомагає подолати проблеми мобільних мереж (рис.2.6).

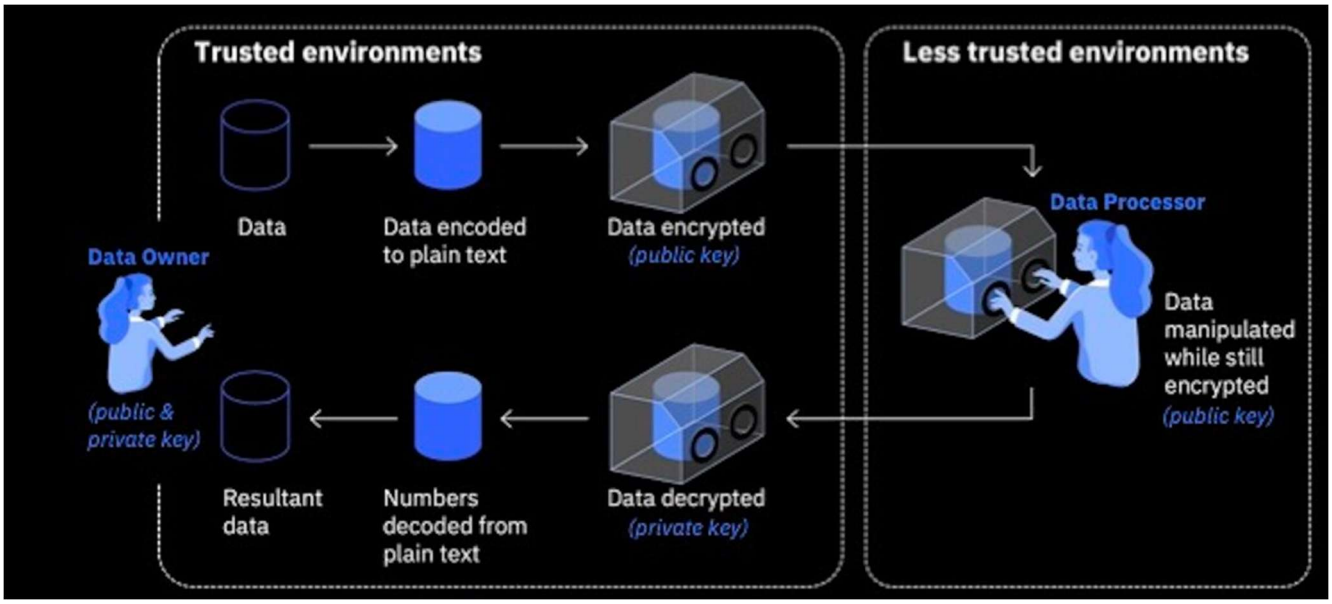

Рисунок 2.6 – Схема роботи повністю гомоморфного шифрування

FHE та PHE служать різним цілям і мають відмінні характеристики в області безпеки даних та криптографії. Оригінальна конструкція FHE починалася з дещо гомоморфної схеми шифрування як основного будівельного блоку. Хоча математичні основи цих двох схем шифрування однакові, їхні можливості досить різні. Основна відмінність між частково та повністю гомоморфним шифруванням полягає в обмеженнях їхньої ємності. PHE обмежується обчисленням поліномів низького ступеня над зашифрованими даними через накопичення шуму в шифрованому тексті [30].

Ці різні схеми шифрування працюють у різних сферах, іноді в одній і тій самій галузі. Прикладом цього є фіксований та мобільний характер мереж 5G. SHE добре працює в фіксованих мережах, які мають більш обмежені канали передачі даних і менше шлюзів. Для мобільних мереж використовується повністю гомоморфне шифрування, оскільки воно не має таких самих обмежень гомоморфних операцій. FHE може виконувати необмежену кількість операцій, тоді як SHE – ні.

FHE дозволяє виконувати довільні обчислення над зашифрованими даними без необхідності їх попереднього розшифрування, що дає можливість оцінювати будь-яку функцію над шифрованими текстами. [9] На відміну від цього, PHE підтримує тільки певні операції (або додавання, або множення, але не обидві) над зашифрованими даними [27].

В таблиці 2.4 наведено схеми порівняння PHE і FHE методів шифрування відповідно до своїх властивостей.

Таблиця 2.4

Порівняння PHE і FHE методів шифрування

Характеристика	(PHE)	(FHE)
Допустимі операції	Підтримує тільки одну арифметичну операцію: або додавання, або множення (але не обидві одночасно).	Підтримує обидві операції (додавання та множення) необмежену кількість разів, що дозволяє обчислювати будь-яку функцію.
Обчислювальна потужність	Обмежена. Може виконувати лише специфічні завдання (для прикладу, сумування голосів).	Необмежена. Може виконати будь-яку програму над зашифрованою інформацією.
Швидкодія	Висока. Операції виконуються відносно швидко, накладні витрати ресурсів помірні.	Низька. Операції можуть бути у 1000+ разів повільніші за роботу з незашифрованими даними (через складність математики).
Проблема зашумлення	Загалом відсутня. Операції не накопичують критичної помилки.	Критична. Кожна операція (особливо множення) додає шум. Потрібна процедура очищення шуму, яка є дуже ресурсоємною.

Однією з основних відмінностей між FHE і PHE є складність відповідних процесів шифрування та дешифрування. FHE, як правило, є значно складнішим через необхідність додавання шуму до зашифрованих даних для підвищення безпеки, що може призвести до компромісів у продуктивності. Як результат, операції FHE можуть бути повільнішими, ніж виконання еквівалентних операцій без шифрування. [11, 13] З іншого боку, PHE зазвичай забезпечує вищу продуктивність завдяки простішому оперативному обсягу [34].

FHE розроблено таким чином, що довіра потрібна лише до базової математики, а не до системи, адміністраторів або використовуваного програмного забезпечення. [11] Це може забезпечити більшу безпеку для додатків, що обробляють конфіденційні дані. На відміну від цього, PHE може вимагати вищого рівня довіри до середовища хоста через обмежену операційну структуру, що потенційно може піддавати дані більшому ризику під час обробки.

FHE є значним проривом у галузі криптографії, що дозволяє виконувати обчислення над зашифрованими даними без необхідності їх попереднього розшифрування. Концептуальні основи FHE можна простежити до 1978 року, коли Рівст, Адлман і Дертузос вперше запропонували цю ідею.

Однак лише в 2009 році Крейг Джентрі представив практичну схему, що використовує підхід на основі решітки та техніку, яка називається «бутстрепінг», що ознаменувало перехід FHE від теорії до потенційних реальних застосувань [15]. Інновація Джентрі стимулювала вдосконалення, спрямовані на підвищення практичності та ефективності FHE, особливо з огляду на зростання попиту на надійні рішення для захисту конфіденційності даних у цифрову епоху [15].

Однією з основних проблем FHE є її обчислювальна складність. Операції, що виконуються за допомогою FHE, є повільнішими за ті, що виконуються на незашифрованих даних, часто на кілька порядків [16]. Ця проблема продуктивності пов'язана зі складною представленістю даних та додатковою обробкою, необхідною для центральних та графічних процесорів для виконання обчислень FHE. Розширення даних, пов'язане з гомоморфно зашифрованими даними, ще більше ускладнює ситуацію, оскільки зазвичай вимагає більше місця для зберігання порівняно з незашифрованими даними [16].

FHE використовує різні математичні конструкції, включаючи поліноми та систему чисел залишків, для полегшення зашифрованих обчислень. Наприклад, такі схеми, як BGV та BFV, використовують поліноміальну арифметику для управління операціями з зашифрованими даними [11]. Ці методи дозволяють виконувати операції, такі як додавання та множення, безпосередньо над шифрованими текстами, хоча вони вводять додаткові рівні складності щодо модулів коефіцієнтів та змін бази під час множення [17].

Бутстрепінг – це критично важлива техніка в FHE, яка дозволяє оновлювати зашифровані дані, що дає можливість виконувати більш складні обчислення без значної втрати безпеки (рис. 2.7).

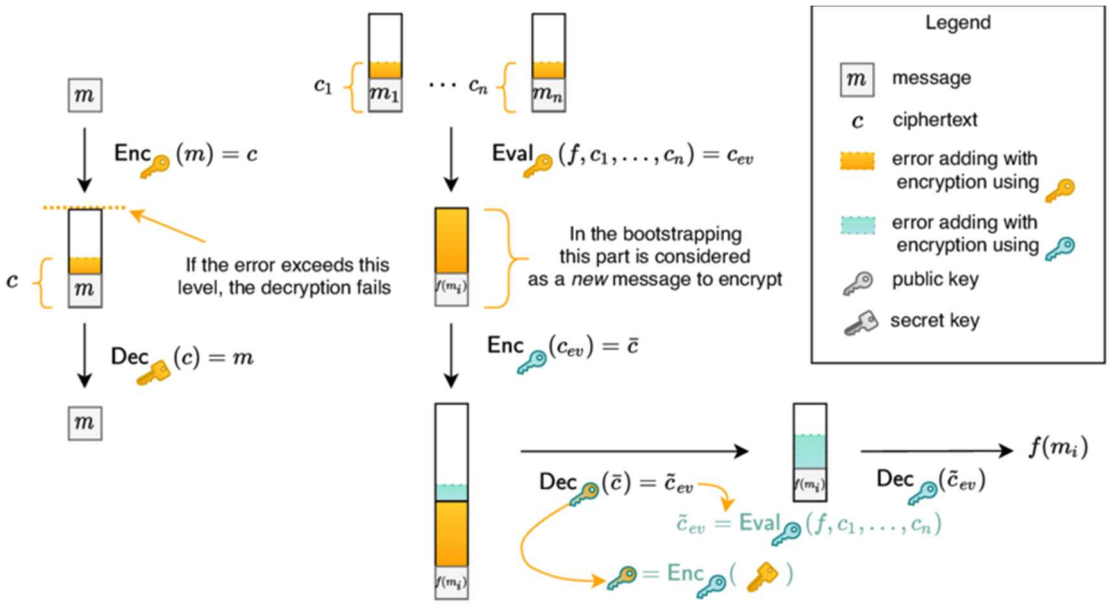

Рисунок 2.7 - Техніка бутстрепінгу

Цей процес зменшує «шум», який накопичується під час операцій шифрування, забезпечуючи збереження дійсності та працездатності даних в межах схеми шифрування [15]. Можливість виконання бутстрепінгу відрізняє FHE від PHE, де над зашифрованими даними можна виконувати лише певні операції (додавання або множення, але не обидві одночасно) [7].

Дослідження в області FHE значно еволюціонували в останні роки, зосередившись, зокрема, на підвищенні ефективності та практичності схем FHE. Комплексна класифікація існуючих методів прискорення для FHE виявляє дві основні категорії: алгоритмічне прискорення та апаратне прискорення [4, 15].

Останні дослідження акцентують увагу на алгоритмічних методах, спрямованих на оптимізацію кількості операцій, необхідних для шифрування, дешифрування та гомоморфних операцій. Наприклад, дослідники вивчали такі оптимізації, як використання числового теоретичного перетворення (NTT) та методів скорочення Баррета для підвищення продуктивності. Хоча були досягнуті певні успіхи, багато схем алгоритмічного прискорення стикаються з обмеженнями в досягненні істотного прискорення, причому значна увага приділяється NTT та операціям бутстрепінгу [35]. Це вказує на гостру необхідність теоретичних проривів у розробці алгоритмів, особливо для бутстрепінгу, щоб ефективно задовольнити вимоги практичного застосування.

З боку апаратного забезпечення спеціалізовані конструкції, такі як спеціалізовані інтегральні схеми (ASIC), продемонстрували значні ефекти прискорення. Ці апаратні рішення забезпечують підвищену гнучкість налаштування, що дозволяє оптимізувати можливості зберігання та обробки даних, адаптовані для операцій FHE [18]. Дослідження показали, що підходи на основі ASIC все частіше інтегруються в глибокі нейронні мережі (DNN) для задоволення практичних потреб, що ще раз підтверджує їх корисність для прискорення впровадження FHE [18]. Крім того, були запропоновані нові апаратні архітектури, такі як Processing-in-Memory (PiM), що пропонують іншу парадигму, дозволяючи виконувати обчислення безпосередньо в пам'яті, тим самим скорочуючи час передачі даних і підвищуючи продуктивність. Однак застосування PiM залишається в основному теоретичним і стикається з проблемами практичної реалізації [18].

Сучасний стан досліджень FHE вказує на різні потенційні майбутні напрямки, включаючи дослідження нових алгоритмів FHE, розробку гібридних схем прискорення, що поєднують алгоритмічні та апаратні методи, а також вдосконалення нових апаратних архітектури [35]. Висвітлюючи ці напрямки для подальших досліджень, вчені прагнуть розширити межі технології FHE, сприяючи її застосуванню в різних галузях, де збереження конфіденційності має вирішальне значення [18]. У міру того, як наукова спільнота продовжує вирішувати ці завдання,

очікується підвищення ефективності та застосовності повністю гомоморфного шифрування, що зробить його більш придатним для практичного використання.

Визнаючи складність FHE, ведуться спільні зусилля з метою встановлення глобальних стандартів та найкращих практик для його впровадження. Ініційований Intel і продовжуваний в рамках ISO/IEC, стандарт ISO/IEC 28033 має на меті створити багатокомпонентний стандарт, що охоплює визначення, основні техніки та стандарти застосування для FHE, що сприятиме ширшому впровадженню та використанню в різних сферах [11].

Хоча FHE ще не повністю оптимізовано для високомасштабних додатків загального призначення, воно почало знаходити своє місце в різних галузях, які надають пріоритет конфіденційності та безпеці даних [33]. Наприклад, FHE є вигідним для обробки конфіденційних даних у таких сферах, як охорона здоров'я та фінанси, де обмін даними викликає занепокоєння щодо конфіденційності [21]. Однак PHE залишається актуальним для конкретних сценаріїв, де достатньо обмежених операцій і де швидкість є більш критичним фактором, ніж гнучкість або повнота [5].

Деякі з найкращих досягнень у впровадженні дещо гомоморфних схем шифрування були зроблені в контексті схем на основі кілець поліномів [9]. Безпека та продуктивність практичних гомоморфних схем шифрування базуються на проблемі RLWE - це обчислювальна проблема, яка вважається безпечною для квантових комп'ютерів і базується на арифметиці поліномів. Схеми RLWE є дещо гомоморфними, оскільки глибина значень не може бути нескінченною.

Існує багато дещо гомоморфних схем шифрування, які базуються на кільцях поліномів через безпеку, яку забезпечують такі схеми. У дослідженні проведеному в Брістольському університеті, дві схеми на основі кілець поліномів піднялися на вершину списку з шести дещо гомоморфних схем шифрування після ретельних тестувань на межі.

Для невеликих відкритих текстів найефективнішою є схема YASHE (рис. 2.8).

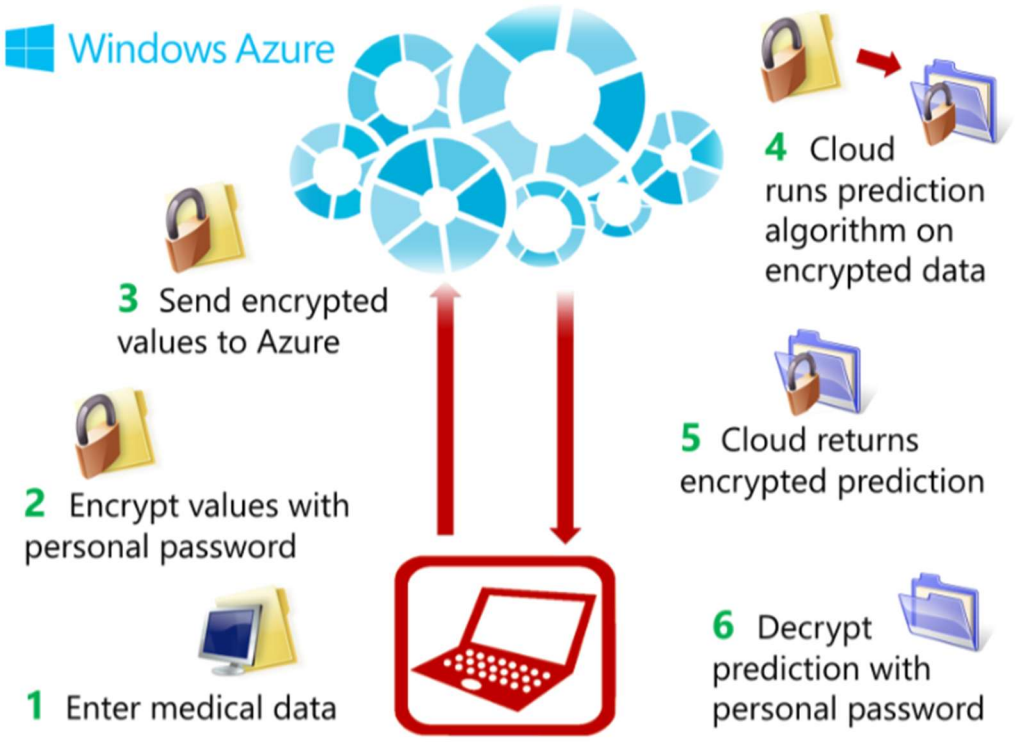

Рисунок 2.8 – Практича ралізація схеми YASHE в Microsoft Azure

Зі збільшенням розміру відкритого тексту схема BGV швидко випереджає інші тестовані схеми на основі кілець поліномів.

Існує кілька важливих обмежуючих факторів для використання схеми гомоморфного шифрування. Будь-яка гомоморфна криптосистема буде обчислювально інтенсивною на математичному рівні, що може бути неефективним для виконання [9].

Крім того, багато гомоморфних алгоритмів є приватними або запатентованими, тому доступ до них обмежений. Виходячи з проблеми доступу, якщо кілька груп шифрують різні частини набору даних, можуть використовуватися різні схеми шифрування. Оцінка схем, де одна частина даних була зашифрована за іншою схемою ніж решта, стає складною.

Обробка природної мови (NLP) - це галузь інформатики, пов'язана зі штучним інтелектом. NLP поєднує лінгвістику, машинне навчання та глибоке навчання, щоб «розуміти» письмовий або усний текст. Дані, які NLP може швидко надати, є величезними. Проте в багатьох випадках, як-от у випадку з медичними

даними, організація, що має доступ до інформації, не володіє технічними знаннями або обчислювальною потужністю для використання NLP.

Це приклад того, де можна використовувати гомоморфне шифрування. Дані можна зашифрувати, а медичну інформацію можна відокремити від особистої інформації пацієнта. Зашифровані дані можна потім передати організації, що має можливості NLP, і провести аналіз даних без ризику для конфіденційності.

2.3. Застосування гомоморфного шифрування до безпеки хмарних обчислень

Коли дані передаються в хмару, ми використовуємо стандартні методи шифрування для забезпечення безпеки операцій і зберігання даних. Основна концепція полягала в тому, щоб шифрувати дані перед їх надсиланням до хмарного провайдера. Але останній повинен розшифровувати дані при кожній операції. Клієнт повинен надати приватний ключ серверу (хмарному провайдеру) для розшифрування даних перед виконанням необхідних обчислень, що може вплинути на конфіденційність і приватність даних, що зберігаються в хмарі.

У роботі запропоновано застосування методу для виконання операцій над зашифрованими даними без їх розшифрування, що забезпечить ті самі результати після розрахунків, якби ми працювали безпосередньо з відкритими даними.

Системи гомоморфного шифрування використовуються для виконання операцій над зашифрованими даними без знання приватного ключа (без розшифрування), клієнт є єдиним власником секретного ключа. Коли ми розшифровуємо результат будь-якої операції, він є таким самим, якби ми виконали обчислення над відкритими даними.

Шифрування є гомоморфним, якщо: з Enc(a) та Enc(b) можна обчислити Enc(f (a, b)), де f може бути: +, ×, □ і без використання приватного ключа. Серед гомоморфного шифрування ми розрізняємо, відповідно до операцій, які дозволяють оцінювати необроблені дані, адитивне гомоморфне шифрування (тільки додавання необроблених даних) – це Pailler [12] і Goldwasser-Micalli криптосистеми, а мультиплікативне гомоморфне шифрування (тільки добутки на необроблених даних) – це криптосистеми RSA [14] та El Gamal [15].

- Ek – алгоритм шифрування з ключем k.

- Dk – алгоритм дешифрування.

Dk (Ek (n) × Ek (m)) = n×m OR Enc (x□y) = Enc(x) □ Enc(y)

DL (EL (n) × EL (m)) = n+m OR Enc (x□y) = Enc(x) □ Enc(y)

Перша властивість називається адитивним гомоморфним шифруванням, а друга – мультиплікативним гомоморфним шифруванням.

Алгоритм є повністю гомоморфним, якщо обидві властивості задовольняються одночасно, припустимо, що x1 і x2 є відкритими текстами. Тоді

$ek(x_1)\ ek(x_2) = x_1\ b\ x_2\ b \bmod n = (x_1\ x_2)\ b \bmod n = ek\ (x_1\ x_2)$, на рисунку 2.9 показано застосування такого методу.

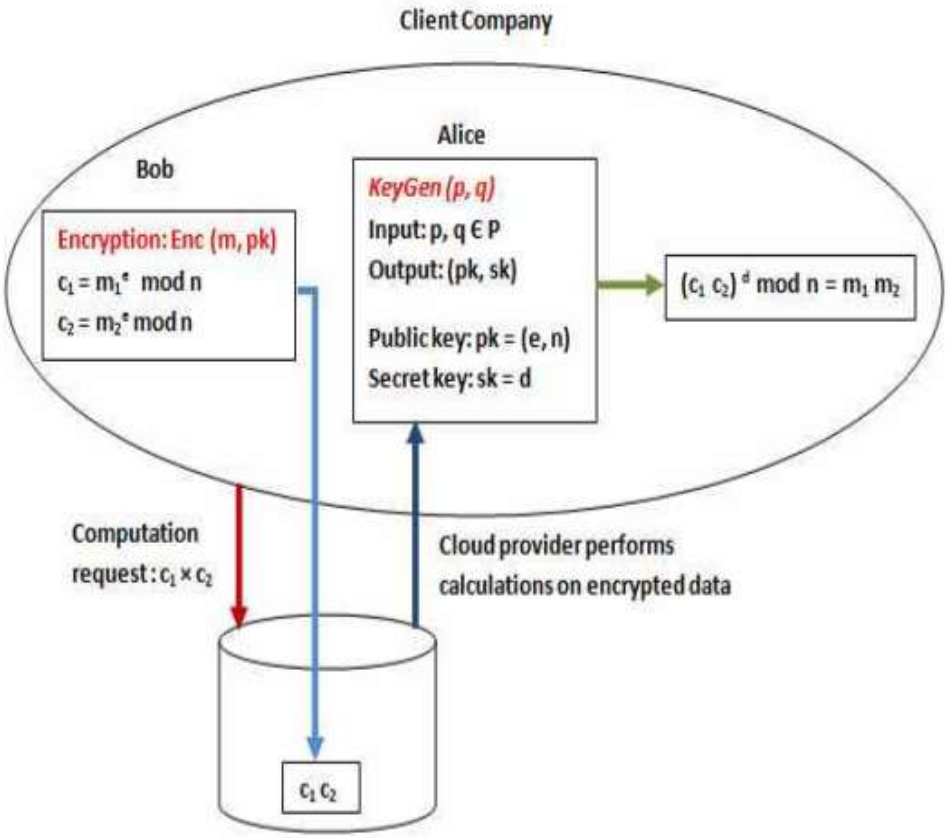

Рисунок 2.9 - Мультиплікативне гомоморфне шифрування, застосоване до хмарних обчислень

Поширеним методом також є аддитивно гомоморфне шифрування у криптосистемі Пайє (рис. 2.10).

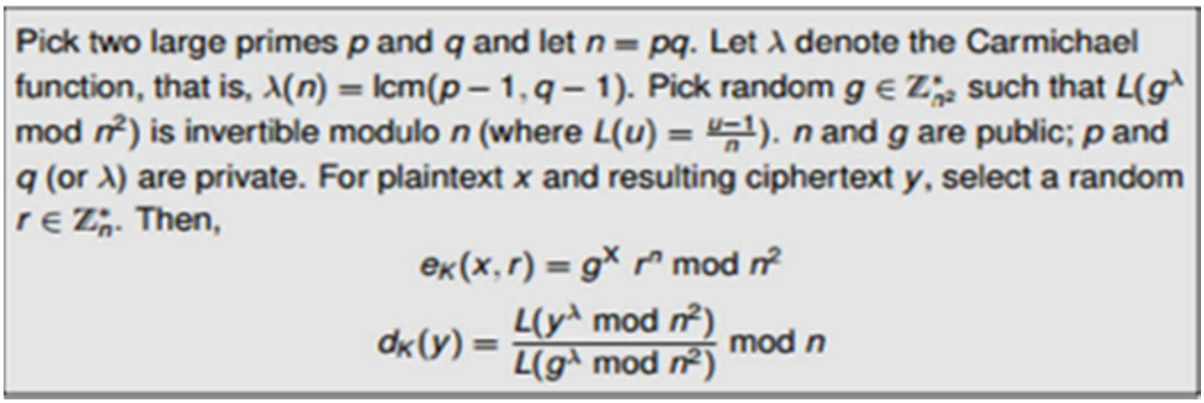

Рисунок 2.10 - Адитивно гомоморфне шифрування (криптосистема Пайє)

Алгоритм є гомоморфним, якщо обидві властивості задовольняються чергов, припустимо, що x_1 і x_2 є відкритими текстами. Тоді:

$$ek(x_1,r1)\ ek(x_2, r_2) = gx_1\ r_1\ n\ .\ gx_2\ r_1\ n \bmod n_2$$

$$= gx_1 + x_2\ (r_1\ r_2)\ n \bmod n_2$$

$$= ek(x_1 + x_2, r_1 r_2)$$

Для виконання додавання та множення зашифрованих даних, що зберігаються у хмарному провайдерові, клієнт повинен мати два різних генератори ключів (один для RSA та один для Paillier) [9]. Далі буде представлено криптосистему El Gamal, яка в основному є мультиплікативною гомоморфною криптосистемою, але шляхом модифікації режиму кодування ми можемо зробити її адитивною (рис. 2.11) [4].

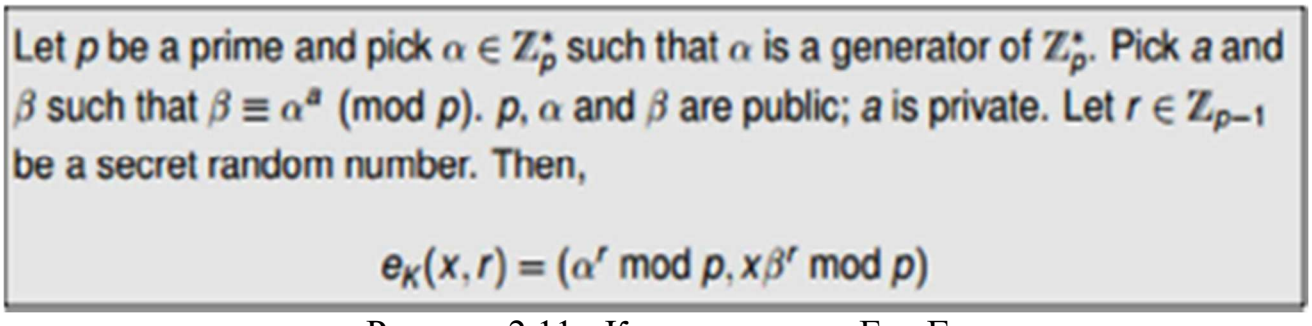

Рисунок 2.11 - Криптосистема Ель Гамаля

Криптосистема Ель Гамаля виконує властивість мультиплікативного гомоморфного шифрування: Нехай x_1 і x_2 – відкриті тексти. Тоді:

$ek(x_1, r_1)\ ek(x_2, r_2) = (\alpha r_1 \bmod p, x_1\ \beta\ r_1 \bmod p)(\alpha r_2 \bmod p, x_2\ \beta\ r_2 \bmod p)$

$= (\alpha r_1 + r_2 \bmod p, (x_1\ x_2)\ \beta\ r_1 + r_2 \bmod p$

$= ek(x_1\ x_2, r_1\ r_2)$

Якщо ми помістимо відкритий текст в експоненту, отримаємо:

$ek(x, r) = (\alpha r \bmod p, \alpha x\ \beta r \bmod p)$

Тоді гомоморфізм є адитивним:

$ek(x_1, r_1)\ ek(x_2, r_2) = (\alpha r_1 \bmod p, \alpha x_1\ \beta\ r_1 \bmod p)(\alpha r_1 \bmod p, \alpha x_2\ \beta\ r_2 \bmod p)$

$= (\alpha r_1 + r_2 \bmod p, \alpha x_1 + x_2\ \beta\ r_1 + r_2) \bmod p$

$= ek(x_1 + x_2, r_1 + r_2$

Для всіх типів обчислень на даних, що зберігаються в хмарі, ми повинні вибрати повністю гомоморфне шифрування, яке здатне виконувати всі типи операцій над зашифрованими даними без розшифрування [7].

Тому проаналізувавши всі дані у додатку Б представлена таблиця, де проведено порівняння різних криптосистеми гомоморфного шифрування за ключовими характеристиками, а саме:

- тип гомоморфного шифрування;
- дотримання конфіденційності чутливих даних;
- чи застосовується захист у провайдера хмарних послуг або у клієнта;
- хто використовує ключі шифрування та дешифрування?

Згідно представленого порівняння оптимальним для забезпечення необхідного рівня захисту є поєднання генерація ключів, виконання гомоморфних операцій та отримання вірного кінцевого результату, тому розроблений алгоритм виконуватиме такі функції:

1) Ініціалізація та генерація ключів - на стороні клієнта створюється сесія, під час якої встановлюється з’єднання з хмарним сервером. Відбувається генерація пари ключів: публічного та приватного. Приватний ключ залишається тільки у клієнта, а публічий надсилається на сервер.

2) Налаштування середовища – після надання серверу публічного ключа він використовує його для налаштування обчислювального середовища, необхідного для виконання гомоморфних операцій, не маючи при цьому можливості дешифрувати отримані дані.

3) Підготовка та шифрування даних - клієнт локально перетворює вхідний масив даних у набір шифротекстів. Процес виконується з використанням приватного ключа.

4) Передача та гомоморфні обчислення - зашифровані дані разом із командою обробки даних надсилаються на сервер. Сервер отримує їх і виконує гомоморфну обробку зашифрованих часток згідно з логікою роботи. Важливо підмітити, що сервер оперує виключно шифротекстами, не маючи доступу до реальних значень.

5) Отримання та дешифрування результату - сервер повертає клієнту результат обчислень у зашифрованому вигляді. отримувач, використовуючи свій секретний ключ, дешифрує отримане повідомлення, додатково з обчисленнями можуть виконуватися функції які фіксують часові затримки на кожному етапі виконання алгоритмую.

Це дозволяє оцінити масштабованість методу для великих проєктів та запропонувати наступну схему алгоритму гомоморфного шифрування (рис. 2.12).

Рисунок 2.12 – Схема алгоритму гомоморфного шифрування

Тому запропонований алгоритм успішно вирішує задачу захисту даних під час свого використання завдяки обрахунку шифрованих даних без їх розкривання третій стороні. Окрім цього, алгоритм має чітку модульну структуру, яка дозволить масштабувати систему під майбутні завдання.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ ГОМОМОРФНИХ АЛГОРИТМІВ ШИФРУВАННЯ ДЛЯ БЕЗПЕЧНИХ ХМАРНИХ ОБЧИСЛЕНЬ

На основі запропонованого у пункті 2.3 алгоритму та проведених у другому розділі досліджень проведена практична реалізація гомоморфного шифрування із обробкою зашифрованих даних у віртуальній хмарній інфраструктурі. Розроблено робочий прототип системи обчислення над зашифрованими даними. Він ілюструє можливість виконання математичних операцій, на прикладі додавання, над зашифрованими даними, при цьому секретний ключ залишається лише у клієнта.

3.1 Вимоги, апаратне та програмне оточення

В якості апаратної платформи для розгортання клієтської частини системи було обрано одноплатний комп’ютер Raspberry Pi 3 Model B (рис. 3.1).

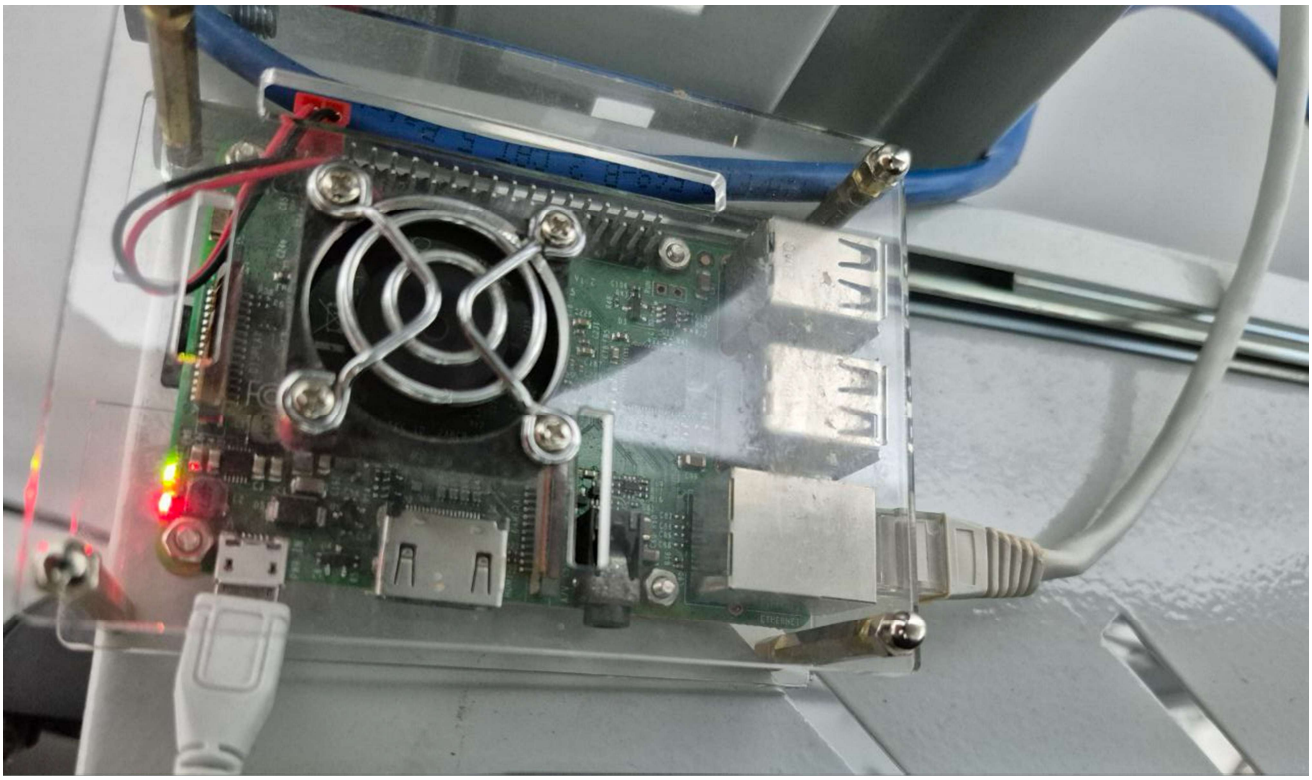

Рисунок 3.1 – Raspberry Pi 3 Model B

Основу обчислювального потенціалу пристрою складає 64-бітна система на кристалі (SoC) з чотириядерним процесором ARM Cortex-A53 (тактова частота 1.2

ГГц). На практиці це дозволяє системі обробляти вхідні запити та виконувати криптографічні операції без критичних затримок. Важливим фактором також є наявність 1 ГБ оперативної пам’яті. Цього обсягу цілком достатньо для стабільної роботи операційної системи (наприклад, дистрибутивів на базі Linux без графічного інтерфейсу) та фонових процесів, що відповідають за логіку роботи додатку..

Цей вибір не є випадковим, адже дана модель демонструє оптимальне співвідношення ціни, енергоефективності та обчислювальної потужності, необхідної для виконання поставлених у роботі завдань.

На рисунку 3.2 представлено схему будови пристрою Raspberry Pi 3, основні характеристики системи та принципи підключення сторонніх компонентів.

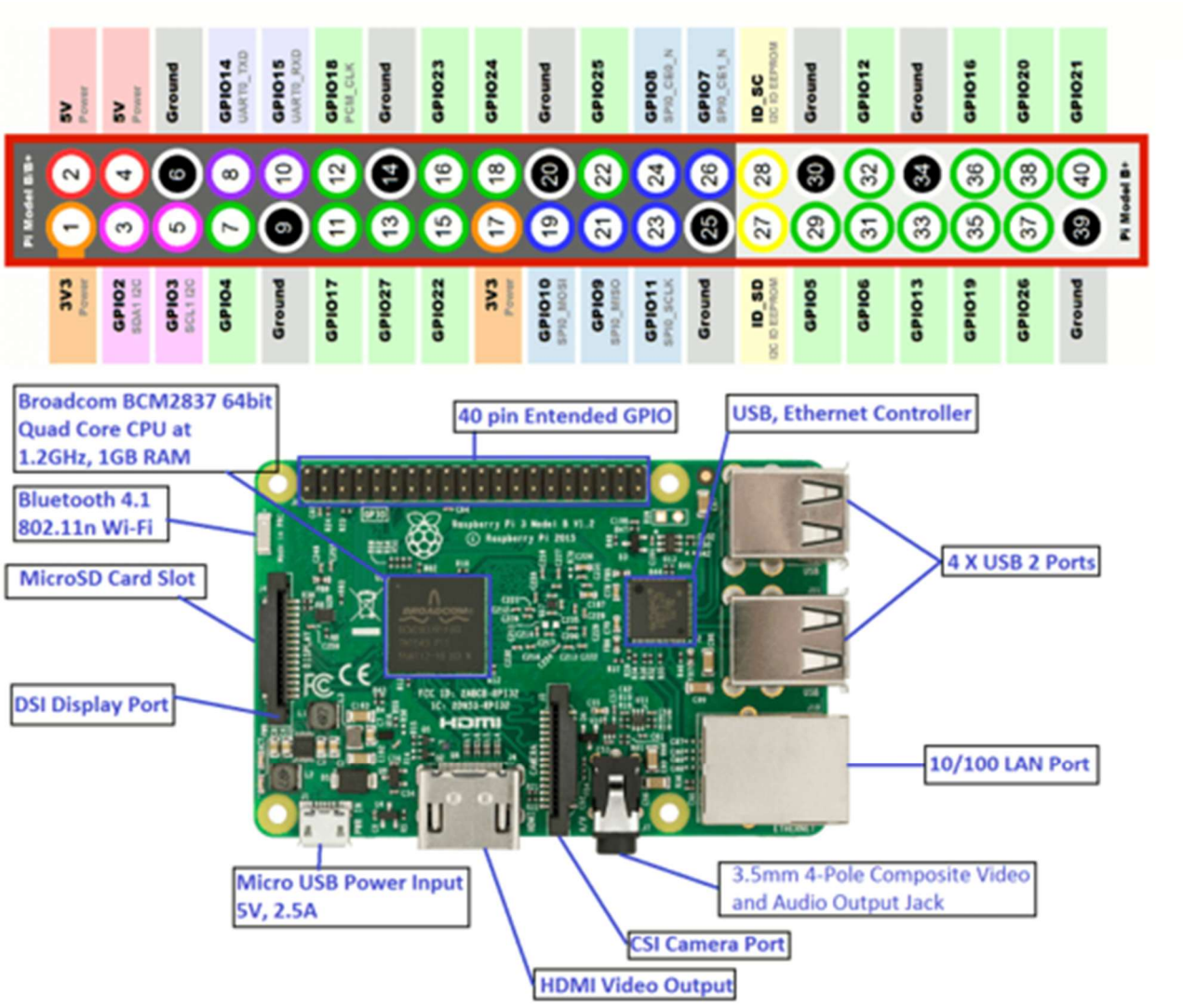

Рисунок 3.2 – Схема будови Raspberry Pi 3

За допомогою протоколу захищеного зв’язку SSH здійснюється підключення до пристрою. На рисунку 3.3 зображено характеристики операційної системи, що розгорнута на Raspberry Pi 3.

Рисунок 3.3 – Характеристики ОС на Raspberry Pi 3

Окрему увагу при проектуванні системи було приділено стабільності мережевої взаємодії. Для підключення Raspberry Pi до мережі передбачено використання вбудованого Ethernet-порту, який забезпечує надійний канал передачі даних зі швидкістю до 1000 Мбіт/с. Це є пріоритетним варіантом підключення, оскільки дротове з'єднання мінімізує ризик втрати пакетів. Однак, архітектура системи допускає і використання інтегрованого модуля Wi-Fi (стандарт 802.11n) у випадках, коли фізичне підключення кабелем є неможливим, за умови забезпечення достатнього рівня сигналу.

Питання безпечного зберігання даних реалізовано на рівні файлової системи носія (MicroSD-карти). Критично важливим аспектом тут є захист ключів шифрування та конфігураційних файлів. Безпека забезпечується не тільки фізичним контролем над пристроєм, а й програмним розмежуванням прав доступу. У файловій системі налаштовано суворі політики (права читання/запису лише для власника процесу), що унеможливлює несанкціонований доступ до чутливої інформації з боку інших користувачів системи або сторонніх процесів.

Реалізація програмного комплексу системи виконувалася з використанням мови програмування Python 3. Цей вибір обумовлений розвиненою екосистемою

бібліотек для роботи з криптографією та мережевими протоколами, а також кросплатформеністю, що дозволяє використовувати ідентичний код як на стороні клієнта (ARM-архітектура Raspberry Pi), так і на стороні сервера (x86/x64).

Клієнтська складова На боці IoT-пристрою логіка роботи побудована навколо збору телеметричних даних та їх подальшого шифрування. Для отримання показників стану системи (навантаження на ЦП, використання пам'яті тощо) використовується бібліотека `psutil`. Вона дозволяє зчитувати системні параметри в реальному часі, які потім виступають вхідними даними для гомоморфних обчислень. Мережева взаємодія з сервером організована за допомогою бібліотеки `requests`, яка забезпечує передачу зашифрованих пакетів через HTTP/HTTPS протоколи, гарантуючи надійність доставки повідомлень.

Криптографічне ядро Ключовим елементом програмної реалізації є бібліотека Pyfhel, яка виступає високорівневою обгорткою для ефективних C++ реалізацій гомоморфного шифрування. Вибір саме цього інструменту пояснюється простотою прототипування та підтримкою сучасних схем, таких як CKKS та BFV. Варто зазначити, що хоча в в минулому розділі роботи було розглянуто класифікацію різних підходів, включаючи схеми Paillier, RSA та BGV, у практичній частині акцент зроблено саме на схемі CKKS. Це рішення продиктоване необхідністю виконання операцій над дійсними числами (числами з плаваючою комою), що є типовим для обробки сигналів та статистичних даних з датчиків. Схема Paillier (або її аналоги) розглядається в контексті адитивних сценаріїв, де достатньо лише операції додавання зашифрованих значень.

Для імітації хмарної інфраструктури буде реалізована віддалена віртуальна машина з операційною системою Linux Ubuntu Server 24 у вигляді RESTful-сервісу на базі мікрофреймворку Flask, що буде основою серверної архітектури, характеристики системи представлені на рисунку 3.4. Завдання сервера полягає в прийомі зашифрованих даних, виконанні над ними математичних операцій без розшифрування та поверненні результату клієнту.

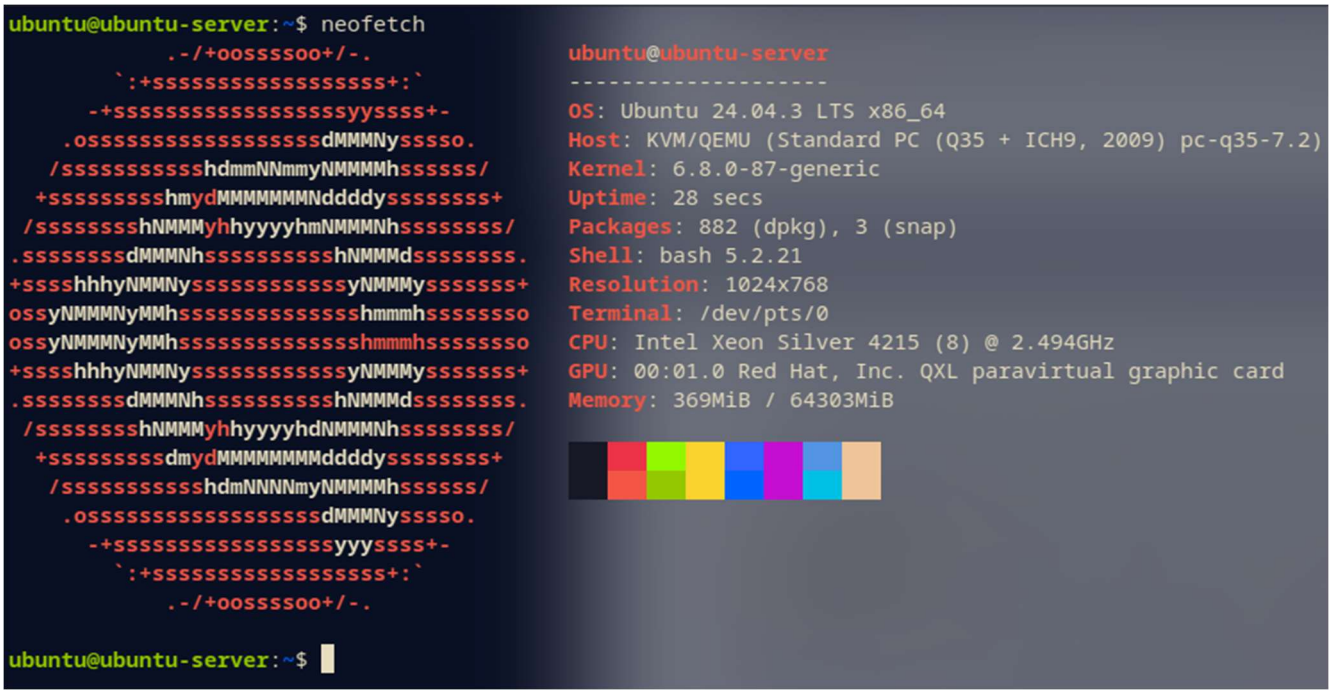

Рисунок 3.4 – Характеристики серверної частини

Для забезпечення стабільної роботи веб-додатку у виробничому середовищі використовується Python-сервер Flask, який дозволяє ефективно керувати потоками обробки запитів та балансувати навантаження. Така комбінація технологій забезпечує необхідну гнучкість та масштабованість системи при роботі з гомоморфним шифруванням.

Розроблена система базується на дворівневій клієнт-серверній архітектурі, де чітко розмежовано зони відповідальності та рівні довіри. Така структура дозволяє реалізувати концепцію "Privacy-Preserving Computation", де сторона, що виконує обчислення, не має доступу до вихідних даних у відкритому вигляді, система представлена такими компонентами:

1) **Клієнтська сторона** – це роль довіреного вузла виконує мікрокомп'ютер Raspberry Pi. Його основним завданням є управління життєвим циклом криптографічних ключів та первинна обробка даних. Процес починається з генерації пари ключів: приватного та публічного. Критично важливою умовою безпеки є те, що приватний ключ ніколи не покидає межі клієнтського пристрою і зберігається локально. Клієнт відповідає за шифрування вхідних векторів даних або окремих чисел, використовуючи публічний ключ, а також за фінальне дешифрування отриманих від сервера результатів.

2) **Серверна сторона** - розгорнута у хмарному середовищі, виступає у ролі обчислювального оракула. Вона отримує від клієнта лише публічний ключ та зашифровані дані. Оскільки сервер не володіє секретним ключем, він не може прочитати вміст даних. Натомість, використовуючи властивості гомоморфного шифрування, сервер виконує математичні операції над шифротекстами: додавання, множення на скаляр, скалярний добуток векторів або обчислення поліномів. Результатом цих обчислень є новий шифротекст, який повертається клієнту.

3) **Мережева взаємодія та транспортний канал** - комунікація між клієнтом та сервером здійснюється через протокол HTTP, захищений шифруванням транспортного рівня (TLS/SSL). Хоча самі дані вже зашифровані гомоморфно, використання HTTPS є обов'язковим стандартом для захисту від атак типу "Man-in-the-Middle" та забезпечення цілісності каналу зв'язку. Для передачі даних використовується формат JSON. Оскільки шифротексти являють собою бінарні дані, які не можуть бути безпосередньо вбудовані у JSON, застосовується механізм серіалізації з кодуванням у Base64. Це дозволяє безпечно передавати складні криптографічні структури у текстовому вигляді.

Логіка роботи системи реалізована за покроковим алгоритмом дій, відображеним на схемі (рис. 3.5).

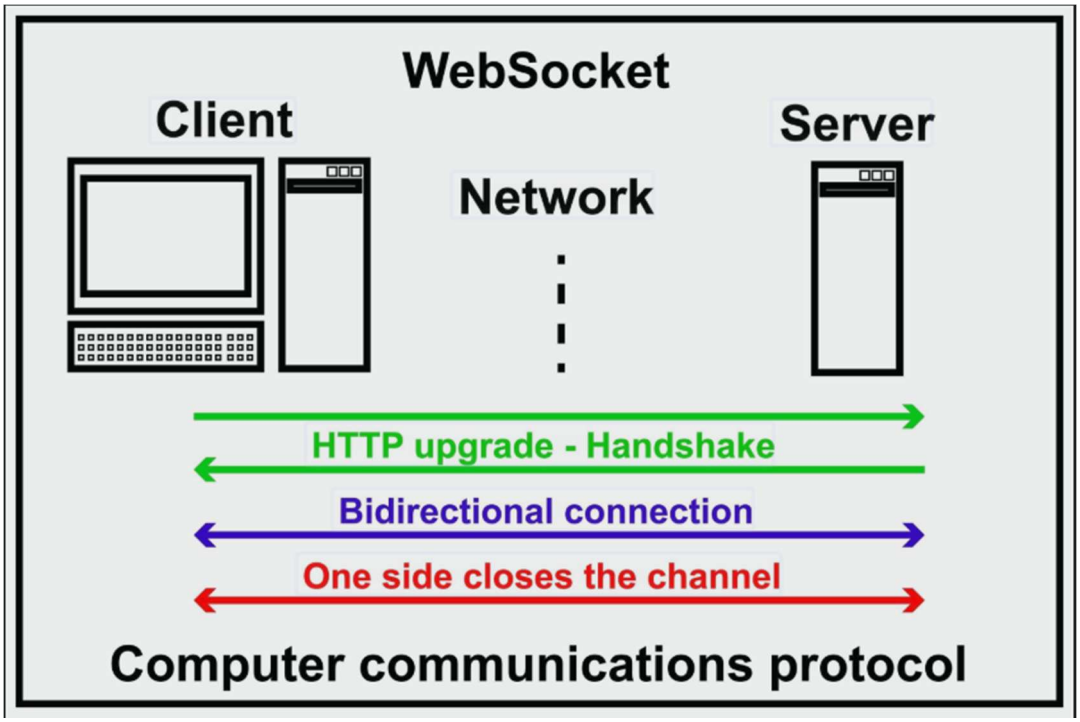

Рисунок 3.5 – Сценарій взаємодії проєкту

Процес ініціюється на клієнті етапом KeyGen, де створюється контекст шифрування. Наступним кроком є передача публічного ключа - Upload PK на сервер, що дозволяє останньому налаштувати середовище для обчислень.

Після ініціалізації відбувається етап Encrypt & Send, де клієнт шифрує дані та відправляє їх у вигляді серіалізованого запиту. Сервер, отримавши пакет, переходить до етапу Compute, виконуючи необхідні гомоморфні операції (HE operations) згідно з бізнес-логікою задачі. Сформований результат (новий шифротекст) повертається клієнту (Return result), де відбувається фінальний етап Decrypt – перетворення результату у зрозумілий для користувача вигляд за допомогою збереженого секретного ключа.

3.2. Програмна реалізація гомоморфного шифрування

Для забезпечення відтворюваності результатів експерименту та спрощення процедури розгортання системи на нових вузлах було розроблено сценарій автоматизованого встановлення install_pi.sh. Використання Bash-скрипта дозволяє мінімізувати вплив людського фактору та гарантувати наявність усіх необхідних залежностей у коректних версіях.

На рисунку 3.6 наведено вміст скрипта, що виконує підготовку операційної системи Raspberry Pi OS (Debian-based) до роботи з модулями гомоморфного шифрування:

```
 GNU nano 7.2                                                start.sh
#!/bin/bash
set -e
sudo apt update && sudo apt upgrade -y
sudo apt install -y build-essential cmake git python3-dev python3-pip libgmp-dev libmpfr-dev libfftw3-dev
python3 -m pip install --upgrade pip
python3 -m venv env && source env/bin/activate
pip3 install pyfhel psutil requests
```

Рисунок 3.6 – Скрипт підготовки ОС

На рисунку 3.7 наведено вміст програмної реалізації, що виконує функції сервера для обрахунку і дешифрування шифротекстів Також оптимізовано для швидкого старту та мінімізації часу кінцевого обрахунку,:

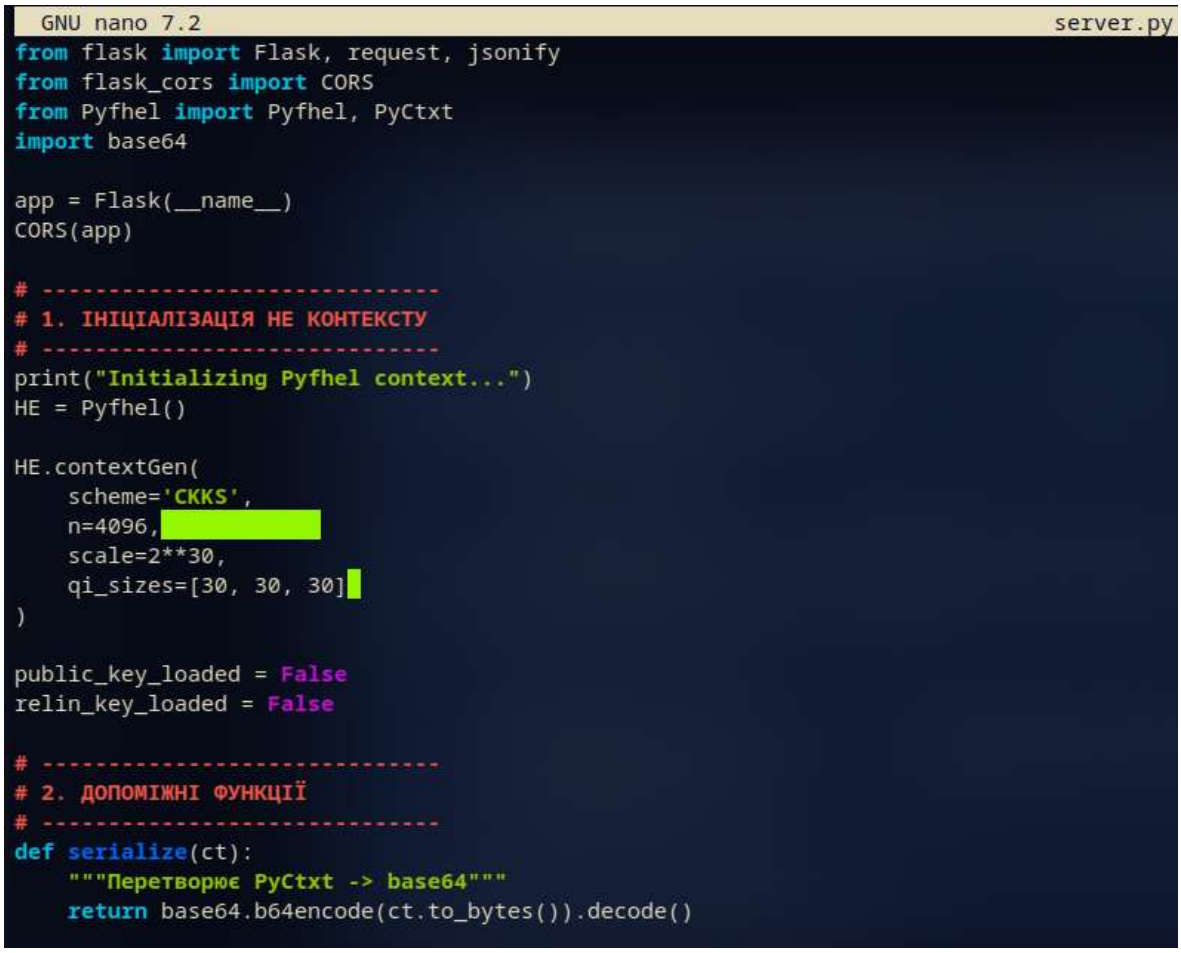

Рисунок 3.7 – Створення сервер отримання і обчислення даних

Для реалізації серверної частини проекту необхідно наступний перелік бібліотек що оголошені у файлі requirements.txt:

- *Flask;*
- *Pyfhel;*
- *base64;*
- *psutil.*

Повний вміст ключових функцій сервера для обрахунку і дешифрування шифротекстів надісланих клієнтом наведено в додатку А.

На рисунку 3.8 наведено вміст програмної реалізації клієнта, що виконується на Raspberry Pi і виступає довіреною стороною у схемі обчислень.

Його архітектура побудована навколо п’яти послідовних етапів: від ініціалізації криптосистеми до отримання та інтерпретації результатів, нижче наведено етап шифрування на стороні клієнта, повний вміст етапів розміщено в додатку А.

```
# ------------------------------------------------------------
# 3. ШИФРУВАННЯ
# ------------------------------------------------------------
vec = np.array([1.5, 2.25, 3.75, 0.5], dtype=np.float64)
expected_sum = np.sum(vec)
print(f" 3. Шифрування: {vec}")

ctxts_b64 = []
for v in vec:
    v_arr = np.array([v], dtype=np.float64)
    c = HE.encryptFrac(v_arr)
    ctxts_b64.append(base64.b64encode(c.to_bytes()).decode())

# ------------------------------------------------------------
# 4. ОБЧИСЛЕННЯ
# ------------------------------------------------------------
print(" 4. Запит на обчислення...")
try:
    resp = requests.post(f"{SERVER_URL}/compute", json={'ctxts': ctxts_b64, 'operation': 'sum'})
    resp.raise_for_status() # Викине помилку, якщо сервер поверне 500
    res_b64 = resp.json()['result']
    print("    Відповідь отримана.")
except Exception as e:
    print(f"    Помилка обчислень: {e}")
    # Виводимо текст помилки сервера для діагностики
    if hasattr(e, 'response') and e.response is not None:
        print(f"   Деталі сервера: {e.response.text}")
    exit(1)
```

Рисунок 3.8 – Створення сервер отримання і обчислення даних

Серверний компонент виступає в ролі недовіреного обчислювача (Untrusted Evaluator). Його основна задача – надати інтерфейс для прийому зашифрованих даних та виконання над ними арифметичних дій згідно із запитом клієнта.

На рисунку 3.9 наведено фрагмент коду, що демонструє реалізацію REST API на базі фреймворку Flask, повний вміст розміщено в додатку А.

```
 GNU nano 7.2                                                server.py *
# 4. ОБЧИСЛЕННЯ
# ------------------------------
@app.route('/compute', methods=['POST'])
def compute():
    global public_key_loaded
    try:
        if not public_key_loaded:
            return jsonify({"error": "Public key not loaded."}), 400

        j = request.json
        operation = j.get('operation', 'суми')
        ctxts_b64 = j.get('ctxts', [])

        if not ctxts_b64:
            return jsonify({"error": "No ciphertexts provided"}), 400

        # Десеріалізація списку
        ctxts = []
        for c in ctxts_b64:
            ct_obj = PyCtxt(pyfhel=HE)
            ct_obj.from_bytes(base64.b64decode(c))
            ctxts.append(ct_obj)

        print(f"Отримано запит на обчислення: {operation} на {len(ctxts)} шифрованих елементи")

        if operation == "sum":
            result = ctxts[0]
            for ct in ctxts[1:]:
                result += ct
        elif operation == "mul_scalar":
            scalar = float(j.get("scalar", 1.0))
```

Рисунок 3.9 - Реалізація прийому даних

Сам процес обчислення шифротексту представлено на рисунку 3.10.

```
        print(f"Отримано запит на обчислення: {operation} на {len(ctxts)} шифрованих елементи")

        if operation == "sum":
            result = ctxts[0]
            for ct in ctxts[1:]:
                result += ct

        elif operation == "mul_scalar":
            scalar = float(j.get("scalar", 1.0))
            result = ctxts[0] * scalar

        else:
            return jsonify({"error": f"Unknown operation: {operation}"}), 400

        res_b64 = serialize(result)
        return jsonify({"result": res_b64})

    except Exception as e:
        print(f"Computation error: {e}")
        return jsonify({"error": str(e)}), 500
```

Рисунок 3.10 - Реалізація обчислення даних

Для додаткових досліджень створено скрипт для тестування швидкості шифрування з різною кількістю елементів, на рисунку 3.11 наведено вміст ключових функцій скрипта моніторингу, повний вміст розміщено в додатку А:

```
 GNU nano 7.2                                                metric_client.py
    return HE

def measure_encrypt_performance(HE, runs=100):
    """Вимірює середній час шифрування одного числа"""
    print(f"\n[BENCH] Тест швидкості шифрування ({runs} ітерацій)...")

    # Використовуємо numpy масив, бо Pyfhel не любить голі float
    val = np.array([1.123], dtype=np.float64)

    times = []
    for _ in range(runs):
        t0 = time.time()
        HE.encryptFrac(val) # Шифрування
        t1 = time.time()
        times.append(t1 - t0)

    avg_time = sum(times) / len(times)
    print(f"   -> Середній час шифрування: {avg_time*1000:.4f} ms")
    return avg_time

def measure_roundtrip(HE, vector_size=10):
    print(f"\n[BENCH] Тест повного циклу (Roundtrip) для вектора з {vector_size} елементів...")

    # Генеруємо випадковий вектор
    vec = np.random.random(vector_size)

    # 1. Етап шифрування
    t_start = time.time()
    ctxts_b64 = []
    for v in vec:
        v_arr = np.array([v], dtype=np.float64)
        c = HE.encryptFrac(v_arr)
        ctxts_b64.append(base64.b64encode(c.to_bytes()).decode())
```

Рисунок 3.11 - Реалізація вимірювання використання ресурсів системи

Оскільки шифротексти у схемах гомоморфного шифрування (зокрема CKKS) представляють собою складні бінарні структури (поліноми з великими коефіцієнтами), їх неможливо передати через текстові протоколи напряму. Для інтеграції з REST API було реалізовано наступний ланцюжок перетворень:

1) Бінарна конвертація - об’єкт шифротексту спочатку перетворюється на потік байтів за допомогою вбудованого методу бібліотеки (.to_bytes()).
2) Кодування - отриманий байтовий масив кодується у формат Base64. Це перетворює бінарні дані у ASCII-рядок, придатний для передачі у тілі JSON-запиту.
3) Інкапсуляція - текстовий рядок додається до JSON-структури разом із метаданими (тип операції, ідентифікатор запиту) та відправляється через HTTP POST.

Варто зазначити, що розмір одного шифротексту може досягати десятків або навіть сотень кілобайт (залежно від параметрів полінома N).

У рамках даного прототипу використання JSON є виправданим завдяки простоті реалізації. Однак, при масштабуванні системи для передачі великих масивів даних (наприклад, для шифрування зображень або відеопотоку) рекомендується перехід на потокову передачу або використання multipart/form-data, щоб уникнути надмірного використання оперативної пам'яті при парсингу гігантських JSON-об'єктів.

Ключовим принципом розробленої архітектури є Zero-Knowledge Privacy стосовно сервера – хмарний вузол виконує обчислення, не маючи жодної можливості дізнатися зміст даних. Цей принцип реалізується через низку організаційних та технічних заходів.

Слід зауважити, що секретний ключ генерується виключно на стороні клієнта і за жодних обставин не передається мережею. Його зберігання організовано локально у захищеній ділянці файлової системи. Для захисту від локальних атак застосовано принцип мінімальних привилеїв: файл ключа має права доступу *600* (читання/запис тільки для власника) або *400* (тільки читання). Це унеможливлює доступ до ключа для інших користувачів системи або скомпрометованих процесів, що працюють під іншими обліковими записами. Для захисту каналу зв'язку від атак

типу "Man-in-the-Middle" (MitM) та перехоплення трафіку, комунікація між клієнтом та сервером повинна відбуватися через захищений протокол TLS (HTTPS) із перевіркою валідності серверного сертифіката, також слід відзначити що у рамках даного дипломного проекту для демонстрації функціональності прототипу використовується стандартний протокол HTTP, однак, у роботі чітко визначено, що перехід на HTTPS є обов'язковою вимогою для будь-якого промислового застосування системи, оскільки Base64-кодування саме по собі не є шифруванням і не захищає дані від перехоплення (хоча самі дані всередині Base64 є криптографічно стійкими, метадані запиту залишаються відкритими).

Аутентифікація та авторизація – важлива тим щоб запобігти несанкціонованому використанню обчислювальних ресурсів сервера сторонніми особами, передбачено механізм аутентифікації запитів. Рекомендованим підходом є використання токенів (JWT – JSON Web Tokens) або API-ключів, які передаються у заголовку Authorization кожного запиту до ендпоінту /compute. Це дозволяє серверу відхиляти запити від неавторизованих джерел ще до початку ресурсоємних гомоморфних обчислень, захищаючи систему від DoS-атак. Сервер налаштовано таким чином, щоб у журналах подій (logs) фіксувалися лише метадані запитів (час отримання, IP-адреса клієнта, тип операції, статус виконання). Збереження тіла запитів або відповідей (навіть у зашифрованому вигляді) у відкритих логах заборонено для мінімізації ризиків витоку інформації та дотримання принципу мінімізації даних.

3.3. Тестування програмної реалізації гомоморфного шифрування

Першим етапом слід оновити систему і отримати всі необхідні пакунки для роботи з програмами, оскільки на Raspberry Pi потрібні компілятор/залежності для Pyfhel необхідно встановити наступні компоненти:

1) *sudo apt update && sudo apt upgrade -y* – оновлення системи*;*
2) *sudo apt install -y build-essential cmake git python3-dev python3-pip libgmp-dev libmpfr-dev libfftw3-dev* - необхідні пакети*;*
3) *sudo pip3 install psutil requests* – для моніторингу*;*

4) *sudo pip3 install pyfhel* - встановлення *Pyfhel.*

Процес інсталяції зазначених компонентів зображено на рисунку 3.12.

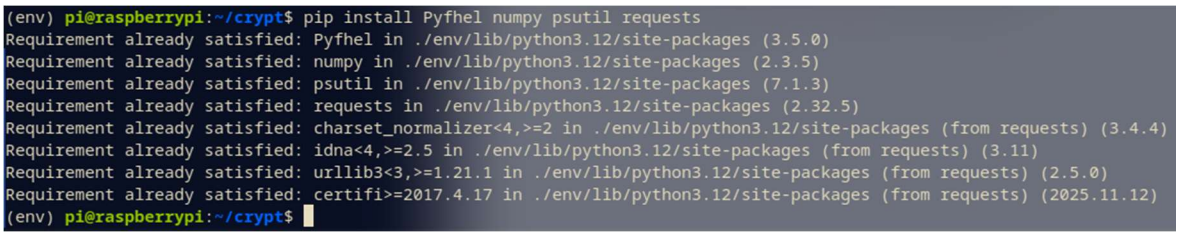

Рисунок 3.12 – Успішне встановлення бібліотек Python

Наступним етапом є оновлення системи на стороні сервера і отриманням всіх необхідних пакетів і бібліотек за допомогою команд:

1) *sudo apt update – оновлення системи*
2) *sudo pip3 install flask pyfhel – необхідні бібліотеки*

Вдале встановлення залежностей показано на рисунку 3.13.

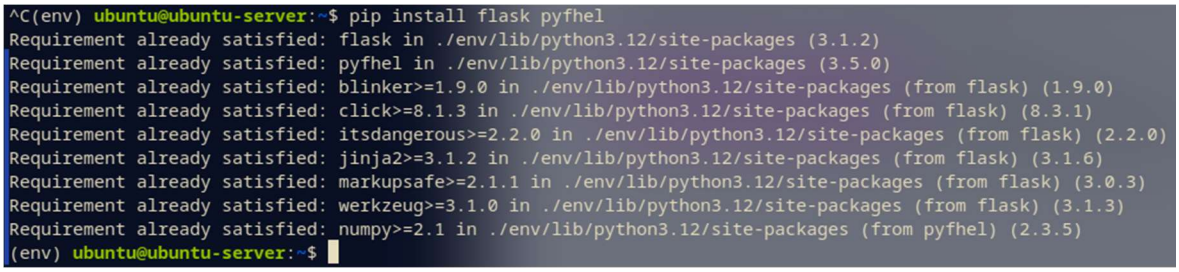

Рисунок 3.13 - Розгортання бібліотек Python на хмарному сервері

Першим тестовим сценарієм буде відправлення зашифрованого масиву [1.5, 2.25, 3.75, 0.5] на сервер для обрахунку і отримання результату його суми.

На рисунку 3.14 показано успішне випробування отримання результату операції.

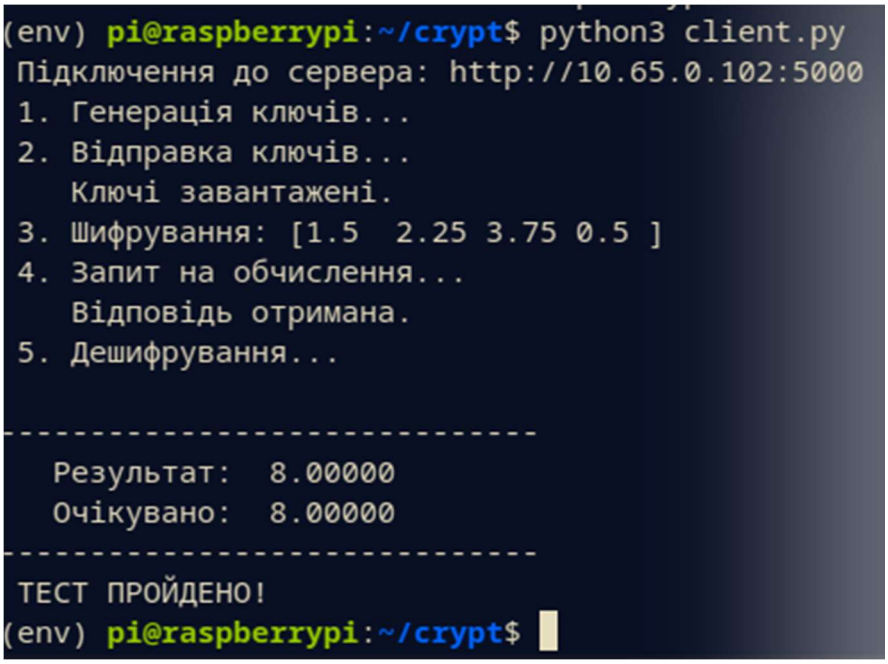

Рисунок 3.14 – Отримання істинного результату операції

На сервері також отримано успішне виконання завдання (рис. 3.15):

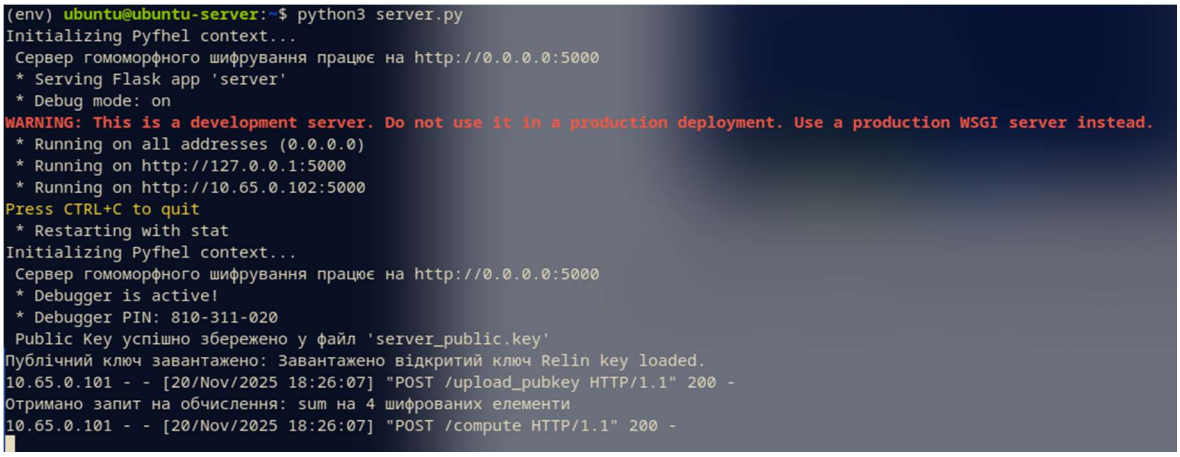

Рисунок 3.15 – Виконання операції хмарним сервером

Сам алгоритм обрахунку завдяки гомоморфним обчисленням буде складатися з таких етапів:

1) Клієнт створює пару ключів. Сервер готує математичну основу для обрахунку.

2) Клієнт надіслав публічний ключ серверу і тепер сервер може здійснити операції над шифротекстами, але не може їх читати.

3) Клієнт локально перетворив числа у масиві у шифротекст.

4) Клієнт відправив 4 зашифровані частки на сервер, далі сервер отримав їх, побачив команду `sum` і додав їх.

5) Сервер відправив результат, теж зашифрований назад.

6) Клієнт своїм секретним ключем відкрив результат і отримав результат 8.0, що є вірним.

Також варто перевірити часові залежності виконання коду для розуміння можливостей подальшого застосування цього методу для масштабування проєкту, це буде виконано за допомогою скрипту metric_client.py (рис. 3.16).

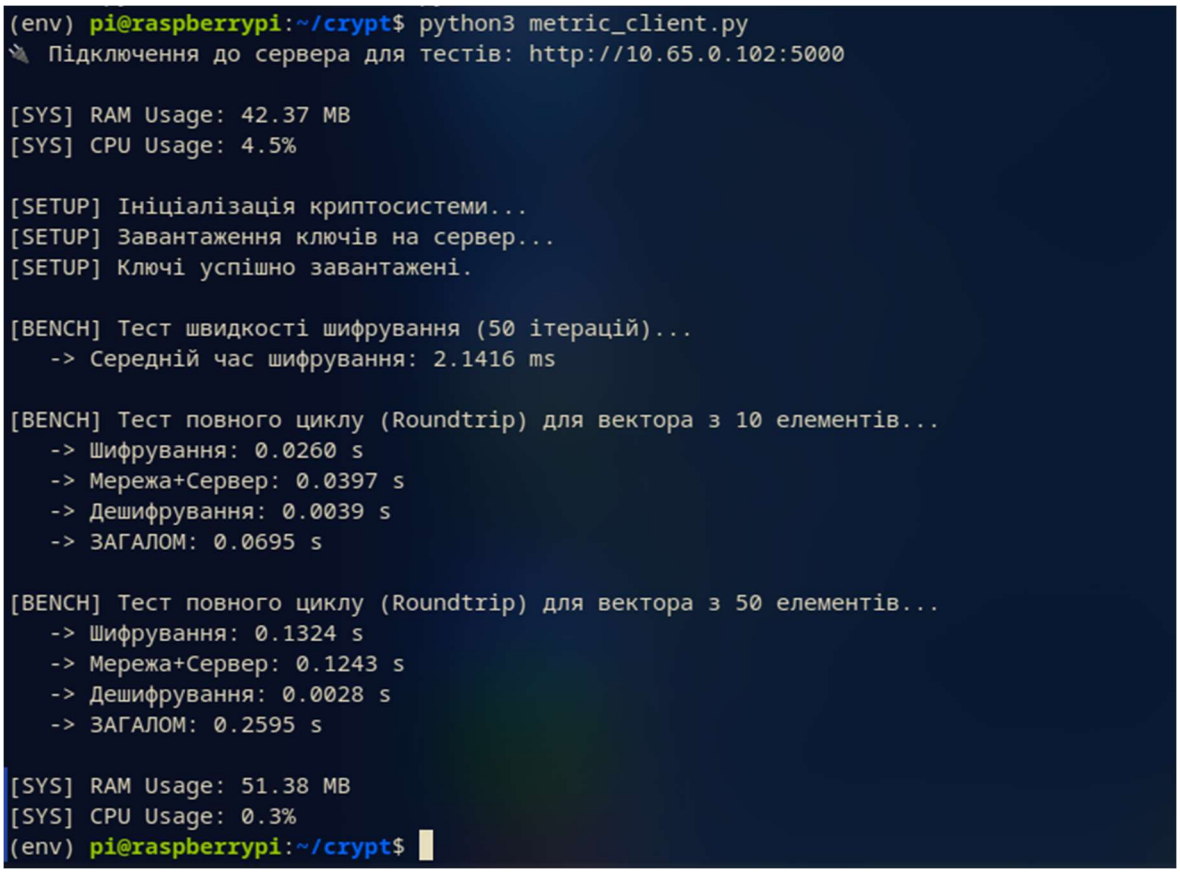

Рисунок 3.16 – Виконання операції хмарним сервером

З представленого на рисунку 3.16 результату можна побачити, що шифрування це найбільш ресурснозатратна операція на стороні клієнта, тому зі збільшенням кількості елементів у масиві час витрачений на шифрування, зростає лінійно, тому при масштабуванні критичним ресурсом стає обчислювальна потужність процесора.

Останнім етапом тестування буде перевірка на зашифрованість переданих даних через мережу, для цього буде використана утиліта tcpdump для перехоплення пакетів даних між клієнтом і хмарним сервером, і подальшого його аналізу у графічній програмі Wireshark.

Запускаємо на клієнті перехоплення пакетів через утиліту tcpdump у файл для зберігання мережевого трафіку (рис. 3.17).

```
sudo tcpdump --interface enp1s0 src 10.65.0.101 and dst 10.65.0.102 and dst port 5000 -w dumpfile.pcap
[sudo] password for pi:
tcpdump: listening on enp1s0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
^C415 packets captured
415 packets received by filter
0 packets dropped by kernel
```

Рисунок 3.17 – Перехоплення пакетів даних у файл

Після проведення декількох обчислень за допомогою гомоморфного шифрування, отримаємо якомога більшу кількість пакетів даних (рис. 3.18).

```
 * Running on http://10.65.0.102:5000
Press CTRL+C to quit
 * Restarting with stat
Initializing Pyfhel context...
 Сервер гомоморфного шифрування працює на http://0.0.0.0:5000
 * Debugger is active!
 * Debugger PIN: 111-804-320
 Public Key успішно збережено у файл 'server_public.key'
Публічний ключ завантажено: Завантажено відкритий ключ Relin key loaded.
10.65.0.101 - - [22/Nov/2025 19:27:29] "POST /upload_pubkey HTTP/1.1" 200 -
Отримано запит на обчислення: sum на 4 шифрованих елементи
10.65.0.101 - - [22/Nov/2025 19:27:29] "POST /compute HTTP/1.1" 200 -
 Public Key успішно збережено у файл 'server_public.key'
Публічний ключ завантажено: Завантажено відкритий ключ Relin key loaded.
10.65.0.101 - - [22/Nov/2025 19:28:57] "POST /upload_pubkey HTTP/1.1" 200 -
Отримано запит на обчислення: sum на 4 шифрованих елементи
10.65.0.101 - - [22/Nov/2025 19:28:57] "POST /compute HTTP/1.1" 200 -
 Public Key успішно збережено у файл 'server_public.key'
Публічний ключ завантажено: Завантажено відкритий ключ Relin key loaded.
10.65.0.101 - - [22/Nov/2025 19:29:09] "POST /upload_pubkey HTTP/1.1" 200 -
Отримано запит на обчислення: sum на 10 шифрованих елементи
10.65.0.101 - - [22/Nov/2025 19:29:09] "POST /compute HTTP/1.1" 200 -
Отримано запит на обчислення: sum на 50 шифрованих елементи
10.65.0.101 - - [22/Nov/2025 19:29:09] "POST /compute HTTP/1.1" 200 -
 Public Key успішно збережено у файл 'server_public.key'
Публічний ключ завантажено: Завантажено відкритий ключ Relin key loaded.
10.65.0.101 - - [22/Nov/2025 19:35:29] "POST /upload_pubkey HTTP/1.1" 200 -
Отримано запит на обчислення: sum на 4 шифрованих елементи
10.65.0.101 - - [22/Nov/2025 19:35:29] "POST /compute HTTP/1.1" 200 -
 Public Key успішно збережено у файл 'server_public.key'
Публічний ключ завантажено: Завантажено відкритий ключ Relin key loaded.
10.65.0.101 - - [22/Nov/2025 19:35:33] "POST /upload_pubkey HTTP/1.1" 200 -
Отримано запит на обчислення: sum на 10 шифрованих елементи
10.65.0.101 - - [22/Nov/2025 19:35:33] "POST /compute HTTP/1.1" 200 -
Отримано запит на обчислення: sum на 50 шифрованих елементи
10.65.0.101 - - [22/Nov/2025 19:35:33] "POST /compute HTTP/1.1" 200 -
```

Рисунок 3.18 – Успішні запити до сервера

Тепер завантаживши файл з мережевим трафіком досліджуємо його у програмі Wireshark для розуміння трафіку (рис. 3.19).

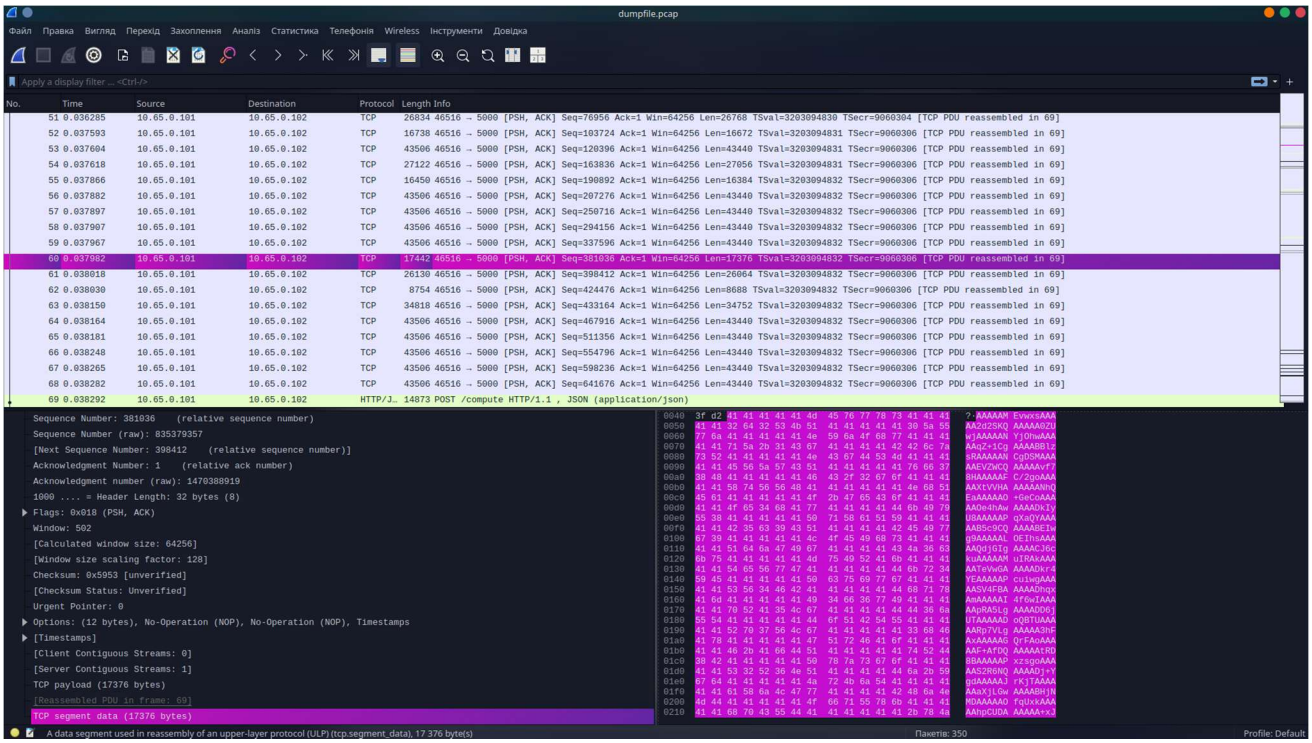

Рисунок 3.19 – Аналіз пакетів даних

На рисунку 3.19 можна оцінити, що передача даних здійснюється через протокол HTTP. Це дозволяє проаналізувати структуру запиту, проте корисним для дослідження є серіалізований шифротекст (Base64-рядок). Якщо буде спроба декодувати Base64, то буде отримано тільки бінарну "кашу" (через поліноми бібліотеки Pyfhel), без розкривання інформації, що підтверджує необхідний рівень захисту шифрованих даних.

Проведене тестування демонструє покрокову взаємодію між клієнтом та хмарним сервером, точність та узгодженість у отриманні, опрацюванні і відправці інформації; швидкодійність і ресурсоефективність виконаних операції, а також реалізоване розмежування доступу користувачів до незашифрованих даних.

Це підтверджує, що дані залишають клієнтський пристрій вже у зашифрованому вигляді, реалізуючи концепцію наскрізного шифрування.

ВИСНОВКИ

У магістерській роботі вирішено актуальне завдання розробки алгоритму шифрування даних та безпечного обчислень в хмарній інфраструктурі.

У результаті проведеного дослідження отримано такі основні результати:

1. Проведено аналіз основних методів і цілей шифрування даних, завдання криптографії та їх класифікації, розглянуто головні криптографічні алгоритми та різні способи реалізації гомоморфного шифрування. Проаналізовано способи застосування криптографічних алгоритмів та їх завдання, що виникають при обчисленні, транспортуванні та зберіганні даних у хмарних обчисленнях.

2. Проаналізовано специфіку роботи гомоморфних алгоритмів для шифрування даних і принципів роботи гомоморфного шифрування та його властивостей.

3. Запропоновано оптимальний і надійний алгоритм і складові для успішної реалізації завдання та обґрунтовано використання криптосистеми мультиплікативого та адитивного методу для гомоморфного шифрування, проведено порівняння характеристик при використанні запропонованих криптосистем і їх вплив на криптостійкість і часову залежність.

4. Розроблено алгоритм шифрування даних та безпечних обчислень за допомогою хмарної інфраструктури яка виступає третім лицем для обчислень без розкривання інформації іншій стороні, за допомогою публічного ключа, а також алгоритм дешифрування за допомогою приватного ключа на боці клієнта.

5. Приведено практичну реалізацію модулів шифрування та дешифрування даних обчислень на основі запропонованого алгоритму гомоморфного шифрування та тестування залежності часу від потужності обчислювальної інфраструктури для даного алгоритму шифрування. Запропонована система дозволяє проводити обчислення даних і їх передачу у хмарній інфраструктурі без розголошення їх відкритого тексту, обчислення проводяться за допомогою публічного ключа і шифротекстів, які потім розшифровуються приватним ключем на стороні клієнта, тому дана модель шифрування дозволяє швидко обраховувати великі масиви даних без небезпеки їх розголошення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Помазибіда В., Кулина С. Алгоритми гомоморфного шифрування для безпечних хмарних обчислень. Кібербезпека та комп'ютерно-інтегровані технології(КБКІТ-2025): Збірник матеріалів науково-практичної конференції молодих вчених, аспірантів та студентів. – Тернопіль, 2025. – С. 85–87.
2. Помазибіда В., Нетребяк М. Аналіз розвитку хмарних обчислень та проблеми їх безпеки. Захист інформації: Збірник матеріалів науково-практичного симпозіуму, 28.11.2025. – Тернопіль, 2025. – С. 83–85.
3. Aiyanyo, Imatitikua D., et al. "A Systematic Review of Defensive and Offensive Cybersecurity with Machine Learning." Applied Sciences, vol. 10, no. 17, Aug. 2020, p. 5811. https://doi.org/10.3390/app10175811.
4. Alloghani, Mohamed, et al. "A systematic review on the status and progress of homomorphic encryption technologies." Journal of Information Security and Applications 48 (2019): 102362.
5. Balli, B.P. Rösler, and S. Vaudenay, "Determining the Core Primitive for Optimally Secure Ratcheting," in *Proceedings of ASIACRYPT 2020*, Part III, LNCS, vol. 12345, pp. 1–20, 2020.
6. "Basics of Cryptographic Algorithms" Geekforgeeks, 2025. [Online]. Available: https://www.geeksforgeeks.org/algorithms/
7. Boer, Derian, and Stefan Kramer. "Secure sum outperforms homomorphic encryption in (current) collaborative deep learning." arXiv preprint arXiv:2006.02894 (2020).
8. Bouti A. and J. Keller. Securing cloud-based computations against malicious providers. In SIGOPS Oper. Syst. Rev., Vol. 46, No. 2,pages 38–42, 2012.
9. B. Brenner, H. Perl and Matthew Smith. How Practical is Homomorphically Encrypted Program Execution? An Implementation and Performance Evaluation. In 11th IEEE Int. Conference on Trust, Security and Privacy in Computing and Communications (TrustCom-12), pages 375–382, 2012.
10. Chillotti I., Gama N., Georgieva M., Izabachène M.: TFHE: fast fully homomorphic encryption over the torus. J. Cryptol. 33(1), 34–91 (2021)

11. "Cryptography and its Types" Geekforgeeks, 2025. [Online]. Available: https://www.geeksforgeeks.org/computer-networks/cryptography-and-its-types/
12. Gaborit, Philippe, Olivier Ruatta, and Julien Schrek. "On the complexity of the rank syndrome decoding problem." IEEE Transactions on Information Theory 62.2 (2020): 1006-1019.
13. Gidney, Craig, and Martin Ekerå. "How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits." Quantum 5 (2021): 433.
14. "Homomorphic Encryption: What Is It, and Why Does It Matter?" InternetSociety, 2020. [Online]. Available: https://www.internetsociety.org/resources/doc/2
15. Kara, Mostefa, et al. "A novel delegated proof of work consensus protocol." 2021 International Conference on Artificial Intelligence for Cyber Security Systems and Privacy (AI-CSP). IEEE, 2021.
16. Marcolla, Chiara, et al. "Survey on fully homomorphic encryption, theory, and applications." Proceedings of the IEEE 110.10 (2022): 1572-1609.
17. Muhammed, R.K. "Comparative Analysis of AES, Blowfish, Twofish, Salsa20, and ChaCha20 Encryption Algorithms," *arXiv*, 2024. Available: https://arxiv.org/pdf/2407.16274
18. Munjal, Kundan, and Rekha Bhatia. "A systematic review of homomorphic encryption and its contributions in healthcare industry." Complex & Intelligent Systems 9.4 (2023): 3759-3786.
19. Pang, H., & Wang, B. (2020). Privacy-preserving association rule mining using homomorphic encryption in a multikey environment. IEEE Systems Journal, 15(2), 3131-3141.
20. Patel, Mr. Hitesh D. Patel, Prof. Pinal J. Patel, "A Secure Cloud using Homomorphic Encryption Scheme", International Journal of Computer Science Research & Technology (IJCSRT) Vol. 1 Issue 1, June-2023.
21. Ravindram, Parsi Kalpana, "Data Storage Security Using Partially Homomorphic Encryption in a Cloud", International Journal of Advanced Research in Computer Science and Software Engineering Volume 3, Issue 4, April 2021.
22. Sarker, Iqbal H., et al. "Cybersecurity data science: an overview from machinelearning perspective." Journal of Big data 7 (2020): 1-29.

23. Sarker, Iqbal H. "Machine learning for intelligent data analysis and automation in cybersecurity: current and future prospects." Annals of Data Science, 1473- 1498.
24. Shaukat, Kamran, et al. “Performance Comparison and Current Challenges of Using Machine Learning Techniques in Cybersecurity.” Energies, vol. 13, no. 10, May 2020, p. 2509. https://doi.org/10.3390/en13102509.
25. Sniatala, Pawel, S. S. Iyengar, and Sanjeev Kaushik Ramani. "Homomorphic Encryption." Evolution of Smart Sensing Ecosystems with Tamper Evident Security. Cham: Springer International Publishing, 2021. 69-76.
26. "Somewhat Homomorphic Encryption from Linear Homomorphism and Sparse LPN" Cryptology ePrint Archive, 2024. URL: https://eprint.iacr.org/2024/1760
27. "Types of Homomorphic Encryption" IEEE Digital Ptivacy, 2020. [Online]. Available: https://digitalprivacy.ieee.org/publications/t
28. "What is cryptography?" TechTarget, 2024. [Online]. Available: https://www.techtarget.com/searchsecurity/definition/cryptography\
29. "What is Homomorphic Encryption?" CyberArk, 2022. [Online]. Available: https://www.cyberark.com/what-is/homomorphic-encryption/
30. Белей, Олександр Ігорович. "Гомоморфне шифрування даних у хмарних сховищах методом матричних поліномів." Сучасний стан наукових досліджень та технологій в промисловості 4 (6) (2018): 5-14.
31. Ділмегані Д. Що таке гомоморфне шифрування? Переваги та проблеми / Д. Ділмегані. – 2021. – URL: https://research.com/homomorphic-encryption/
32. Ільєнко, Анна, Олександр Тищенко, and Ірина Кравчук. "Перспективи використання гомоморфного шифрування для хмарних обчислень."
33. Coinmarketcap (2024). Повністю гомоморфне шифрування [Електронний ресурс] – URL: https://incrypted.com/ua/jak-pratsjuje-povnistju
34. FinanceFeeds (2025). Повністю гомоморфне шифрування (FHE): Розблокування приватних обчислень для Web3, штучного інтелекту та безпеки даних. URL: https://financefeeds.com/uk/fully-homomorphic-encryption
35. Incrypted (2024). Як працює повністю гомоморфне шифрування (FHE) і в чому користь для криптовалют? [Електронний ресурс] – Режим доступу: https://incrypted.com/ua/jak-pratsjuje-povnistju-gomomorfne-shyfruvannja-fhe/

ДОДАТОК А

Код програмної реалізації модуля відправника, модуля отримувача хмарних обчислень, модуля отримувача з числовими даними

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import base64
import requests
import time
import numpy as np
from Pyfhel import Pyfhel, PyCtxt

SERVER_IP = "10.65.0.102"
SERVER_PORT = "5000"
SERVER_URL = f"http://{SERVER_IP}:{SERVER_PORT}"

print(f" Підключення до сервера: {SERVER_URL}")

# 1. Ініціалізація
print(" 1. Генерація ключів...")
HE = Pyfhel()
HE.contextGen(scheme='CKKS', n=4096, scale=2**30, qi_sizes= [30, 30, 30])
HE.keyGen()
HE.relinKeyGen()

# 2. Відправка ключів
print(" 2. Відправка ключів...")
try:
  payload = {
    'pubkey_b64': base64.b64encode(HE.to_bytes_public_key()).decode(),
    'relin_b64': base64.b64encode(HE.to_bytes_relin_key()).decode()
  }
  requests.post(f"{SERVER_URL}/upload_pubkey", json=payload).raise_for_status()
  print("  Ключі завантажені.")
except Exception as e:
  print(f"  Помилка ключів: {e}")
  exit(1)

# 3. Шифрування
```

```
vec = np.array( [1.5, 2.25, 3.75, 0.5], dtype=np.float64)
expected_sum = np.sum(vec)
print(f" 3. Шифрування: {vec}")

ctxts_b64 = []
for v in vec:
  v_arr = np.array( [v], dtype=np.float64)
  c = HE.encryptFrac(v_arr)
  ctxts_b64.append(base64.b64encode(c.to_bytes()).decode())

# 4. Обчислення
print(" 4. Запит на обчислення...")
try:
  resp = requests.post(f"{SERVER_URL}/compute", json={'ctxts': ctxts_b64, 'operation': 'sum'})
  resp.raise_for_status()
  res_b64 = resp.json() ['result']
  print("  Відповідь отримана.")
except Exception as e:
  print(f"  Помилка обчислень: {e}")
  if hasattr(e, 'response') and e.response is not None:
    print(f"  Деталі сервера: {e.response.text}")
  exit(1)

# 5. Дешифрування
print(" 5. Дешифрування...")
ct_res = PyCtxt(pyfhel=HE)
ct_res.from_bytes(base64.b64decode(res_b64))

decrypted_arr = HE.decryptFrac(ct_res)
final_res = decrypted_arr [0]

print(f"\n{'-'*30}")
print(f"  Результат: {final_res:.5f}")
print(f"  Очікувано: {expected_sum:.5f}")
print(f"{'-'*30}")

if np.isclose(final_res, expected_sum, atol=0.001):
  print(" ТЕСТ ПРОЙДЕНО!")
else:
  print(" Розбіжність у результатах")
```

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

from flask import Flask, request, jsonify
from flask_cors import CORS
from Pyfhel import Pyfhel, PyCtxt
import base64

app = Flask(__name__)
CORS(app)

# 1. Ініціалізація не контексту
print("Initializing Pyfhel context...")
HE = Pyfhel()

HE.contextGen(
  scheme='CKKS',
  n=4096,
  scale=2**30,
  qi_sizes= [30, 30, 30]
)

public_key_loaded = False
relin_key_loaded = False

# 2. Допоміжні функції
def serialize(ct):
  """Перетворює PyCtxt -> base64"""
  return base64.b64encode(ct.to_bytes()).decode()

# 3. Завантаження ключів
@app.route('/upload_pubkey', methods= ['POST'])
def upload_pubkey():
  global public_key_loaded, relin_key_loaded
  try:
    json_data = request.json
    pk_b64 = json_data.get('pubkey_b64')
    relin_b64 = json_data.get('relin_b64')

    if not pk_b64:
      return jsonify({"error": "Missing pubkey_b64"}), 400
    pk_bytes = base64.b64decode(pk_b64)
    with open("server_public.key", "wb") as f:
      f.write(pk_bytes)
```

```
    print(" Public Key успішно збережено у файл 'server_public.key'")
    HE.from_bytes_public_key(pk_bytes)

    public_key_loaded = True
    msg = "Завантажено відкритий ключ"

    if relin_b64
      HE.from_bytes_relin_key(base64.b64decode(relin_b64))
      relin_key_loaded = True
      msg += " Relin key loaded."

    print(f"Публічний ключ завантажено: {msg}")
    return jsonify({'status': 'ok', 'message': msg})

  except Exception as e:
    print(f"Key upload error: {e}")
    return jsonify({'error': str(e)}), 500
# 4. Обчислення
@app.route('/compute', methods= ['POST'])
def compute():
  global public_key_loaded
  try:
    if not public_key_loaded:
      return jsonify({"error": "Public key not loaded."}), 400

    j = request.json
    operation = j.get('operation', 'суми')
    ctxts_b64 = j.get('ctxts', [])

    if not ctxts_b64:
      return jsonify({"error": "No ciphertexts provided"}), 400

    # Десеріалізація списку
    ctxts = []
    for c in ctxts_b64:
      ct_obj = PyCtxt(pyfhel=HE)
      ct_obj.from_bytes(base64.b64decode(c))

      ctxts.append(ct_obj)

    print(f"Отримано запит на обчислення: {operation} на {len(ctxts)} шифрованих елементи")

    if operation == "sum":
      result = ctxts [0]
      for ct in ctxts [1:]:
```

```
            result += ct

        elif operation == "mul_scalar":
            scalar = float(j.get("scalar", 1.0))
            result = ctxts [0] * scalar

        else:
            return jsonify({"error": f"Unknown operation: {operation}"}), 400

        res_b64 = serialize(result)
        return jsonify({"result": res_b64})

    except Exception as e:
        print(f"Computation error: {e}")
        return jsonify({"error": str(e)}), 500

if __name__ == "__main__":
    print(" Сервер гомоморфного шифрування працює на http://0.0.0.0:5000")
    app.run(host="0.0.0.0", port=5000, debug=True)
```

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import time
import requests
import base64
import psutil
import numpy as np
import os
from Pyfhel import Pyfhel, PyCtxt

SERVER_IP = "10.65.0.102"
SERVER_URL = f"http://{SERVER_IP}:5000"

HE_PARAMS = {
  'scheme': 'CKKS',
  'n': 4096,
  'scale': 2**30,
  'qi_sizes': [30, 30, 30]
}

print(f" Підключення до сервера для тестів: {SERVER_URL}")

# 1. Функції вимірювання
def setup_infrastructure():
  print("\n [SETUP] Ініціалізація криптосистеми...")
  HE = Pyfhel()
  HE.contextGen(**HE_PARAMS)
  HE.keyGen()
  HE.relinKeyGen()

  print(" [SETUP] Завантаження ключів на сервер...")
  payload = {
    'pubkey_b64': base64.b64encode(HE.to_bytes_public_key()).decode(),
    'relin_b64': base64.b64encode(HE.to_bytes_relin_key()).decode()
  }
  try:
    requests.post(f"{SERVER_URL}/upload_pubkey", json=payload).raise_for_status()
    print(" [SETUP] Ключі успішно завантажені.")
  except Exception as e:
    print(f" [ERROR] Не вдалося завантажити ключі: {e}")
    exit(1)
```

```
  return HE

def measure_encrypt_performance(HE, runs=100):
  print(f"\n [BENCH] Тест швидкості шифрування ({runs} ітерацій)...")
  val = np.array( [1.123], dtype=np.float64)
  times = []
  for _ in range(runs):
    t0 = time.time()
    HE.encryptFrac(val) # Шифрування
    t1 = time.time()
    times.append(t1 - t0)

  avg_time = sum(times) / len(times)
  print(f"  -> Середній час шифрування: {avg_time*1000:.4f} ms")
  return avg_time

def measure_roundtrip(HE, vector_size=10):
  print(f"\n [BENCH] Тест повного циклу (Roundtrip) для вектора з {vector_size} елементів...")

  #2. Генеруємо випадковий вектор
  vec = np.random.random(vector_size)

  # 3. Етап шифрування
  t_start = time.time()
  ctxts_b64 = []
  for v in vec:
    v_arr = np.array( [v], dtype=np.float64)
    c = HE.encryptFrac(v_arr)
    ctxts_b64.append(base64.b64encode(c.to_bytes()).decode())

  t_enc_end = time.time()

  # 4. Етап Сервер + Мережа
  payload = {'ctxts': ctxts_b64, 'operation': 'sum'}
  try:
    resp = requests.post(f"{SERVER_URL}/compute", json=payload)
    res_b64 = resp.json() ['result']
  except Exception as e:
    print(f" [ERROR] Помилка мережі: {e}")
    return None

  t_server_end = time.time()

  # 5. Етап Дешифрування
  ct_res = PyCtxt(pyfhel=HE)
```

```
  ct_res.from_bytes(base64.b64decode(res_b64)) # encoding прибрано
  HE.decryptFrac(ct_res)

  t_dec_end = time.time()
  stats = {
    'elements': vector_size,
    't_encrypt': (t_enc_end - t_start),
    't_network_compute': (t_server_end - t_enc_end),
    't_decrypt': (t_dec_end - t_server_end),
    't_total': (t_dec_end - t_start)
  }

  print(f"  -> Шифрування: {stats ['t_encrypt']:.4f} s")
  print(f"  -> Мережа+Сервер: {stats ['t_network_compute']:.4f} s")
  print(f"  -> Дешифрування: {stats ['t_decrypt']:.4f} s")
  print(f"  -> ЗАГАЛОМ: {stats ['t_total']:.4f} s")

  return stats

def print_system_usage():
  process = psutil.Process(os.getpid())
  mem_info = process.memory_info()
  print(f"\n [SYS] RAM Usage: {mem_info.rss / 1024 / 1024:.2f} MB")
  print(f" [SYS] CPU Usage: {psutil.cpu_percent(interval=1)}%")

if __name__ == "__main__":
  print_system_usage()
  he_context = setup_infrastructure()
  measure_encrypt_performance(he_context, runs=50)

  # 6. Тест 2: Повний цикл для різних розмірів вектора
  for size in [10, 50]:
    measure_roundtrip(he_context, vector_size=size)

  print_system_usage()
```

ДОДАТОК Б

Таблиця Б1

Характеристики існуючих криптосистем гомоморфного шифрування для хмарних обчислень

Характеристика	RSA	Paillier	El Gamal	GoldwasserMicali	Boneh-GohNissim	Gentry
Платформа	Хмарні обчислення	Хмарні обчислення	Хмарні обчислення	Хмарні обчислення	Хмарні обчислення	Хмарні обчислення
Тип гомоморфного шифрування	Мультиплікативний	Адитивний	Мультиплікативний	Адитивний, але він може шифрувати тільки один біт	Необмежена кількість додавань, але тільки одне множення	Повне
Конфіденційність даних	Забезпечується в процесах комунікації та зберігання	Забезпечується в процесах комунікації та зберігання	Забезпечується в процесах комунікації та зберігання	Забезпечується в процесах комунікації та зберігання	Забезпечується в процесах комунікації та зберігання	Забезпечується в процесах комунікації та зберігання
Безпека застосовується до	Хмарний провайдер Сервер	Хмарний провайдер Сервер	Хмарний провайдер Сервер	Хмарний провайдер Сервер	Хмарний провайдер Сервер	Хмарний провайдер Сервер
Ключі, що використовуються	Клієнт (Для шифрування та дешифрування використовуються різні ключі)	Клієнт (Для шифрування та дешифрування використовуються різні ключі)	Клієнт (Для шифрування та дешифрування використовуються різні ключі)	Клієнт (Для шифрування та дешифрування використовуються різні ключі)	Клієнт (Для шифрування та дешифрування використовуються різні ключі)	Клієнт (Для шифрування та дешифрування використовуються різні ключі)

Додаток В

Копії публікацій




ЗАХІДНОУКРАЇНСЬКИЙНАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КІБЕРБЕЗПЕКИ
ГРОМАДСЬКА ОРГАНІАЦІЯ «КІБЕРБЕЗПЕКА І АВТОМАТИЗАЦІЯ»

Матеріали
науково-практичного симпозіуму

"ЗАХИСТ'ІНФОРМАЦІЇ'2025"

28 листопада 2025
Тернопіль

Збірник матеріалів науково-практичного симпозіуму «Захист інформації'2025», Тернопіль, 2025. – 118с.

Адреса редакції:
Громадська організація «Кібербезпека і автоматизація»
м. Тернопіль
Контактний телефон: (066)043-42-10
e-mail: conferencekb@gmail.com

УДК 004.62

Василь ПОМАЗИБІДА, Микола НЕТРЕБЯК

Західноукраїнський національний університет

АНАЛІЗ РОЗВИТКУ ХМАРНИХ ОБЧИСЛЕНЬ ТА ПРОБЛЕМИ ЇХ БЕЗПЕКИ

Вступ. Перехід до хмарних технологій є наступним етапом розвитку проведення обчислень. Хмарні обчислення пропонують низку переваг своїм клієнтам. Оплачуючи використання апаратних та програмних ресурсів (серверів, розміщених у дата–центрах, додатків, програмного забезпечення тощо) користувачі можуть отримати через Інтернет доступ до потужних обчислювальних систем або систем зберігання даних великої ємності і без необхідності їх придбання та сплати додаткових витрат на обслуговування обладнання застосовувати їх у своїй діяльності. Спільне використання таких центрів декількома клієнтами за допомогою концепції віртуалізації не повинно загрожувати витоком даних, тому задля забезпечення захисту інформації застосовуються алгоритми шифрування, зокрема гомоморфне шифрування.

Метою дослідження є дослідження сфери застосування гомоморфного шифрування, зокрема хмарних обчислень та аналіз проблем, що виникають при їх впровадженні.

1. Основні криптографічні алгоритми

Останнім часом криптографія перетворилася на поле битви деяких з найкращих математиків і комп'ютерних вчених світу. Здатність безпечно зберігати та передавати конфіденційну інформацію виявилася критичним фактором успіху у війні та бізнесі. Оскільки уряди не хочуть, щоб певні організації в їхніх країнах та за їх межами мали доступ до засобів отримання та надсилання прихованої інформації, яка може становити загрозу національним інтересам, криптографія піддається різним обмеженням у багатьох країнах, починаючи від обмежень використання та експорту програмного забезпечення до публічного поширення математичних концепцій, які можуть бути використані для розробки криптографічних алгоритмів.

Криптографічний алгоритм – це набір кроків, які можна використовувати для перетворення відкритого тексту в зашифрований текст. Криптографічний алгоритм також відомий як алгоритм шифрування [1].

Криптографічний алгоритм використовує ключ шифрування для приховування інформації та перетворення її в шифротекст, а ключ дешифрування можна використовувати для перетворення її назад у відкритий текст.

Існують різні типи криптографічних алгоритмів, але можна узагальнити 4 основні типи криптографічних алгоритмів:

- стандарт шифрування даних (DES);
- розширений стандарт шифрування (AES);
- алгоритм RSA (алгоритм Рівеста, Шаміра, Адлемана);
- алгоритм безпечного хешування (SHA).

Кожен з яких має свої переваги при реалізації, а гомоморфне шифрування дає змогу поєднати декілька алгоритмів у одну систему, що дозволяє підсилити переваги та нейтралізувати недоліки.

2. Розвиток хмарних обчислень та проблеми безпеки

Хмарні обчислення за останні роки значно зросли в популярності та застосуванні. Переваги хмарних обчислень численні, серед них економія коштів, підвищена масштабованість, покращена доступність та спрощене управління IT. Компанії будь–якого розміру переходять на хмарні технології завдяки їхній здатності надавати обчислювальні ресурси, сховища та програмні послуги на вимогу без необхідності розбудови великої внутрішньої інфраструктури.

Хмарні обчислення вже існували під різними назвами, такими як «аутсорсинг» та «хостинг серверів» [2]. Але низька продуктивність використовуваних процесорів, повільне підключення до Інтернету та надмірна вартість використовуваних матеріалів не дозволяють використовувати послуги та місця для зберігання даних. Однак останні досягнення в сучасних технологіях (завдяки віртуалізації) проклали шлях для цих операцій із більш швидкою обробкою. Проблеми безпеки хмарних обчислень також є актуальними для багатьох дослідників; першочерговим завданням було зосередитися на безпеці, яка є найбільшою проблемою для організацій, що розглядають можливість переходу до хмарних технологій. Використання хмарних обчислень дає багато переваг, включаючи зниження витрат, легке обслуговування та повторне надання ресурсів.

Однак зростаюча залежність від хмарних обчислень також викликала значні проблеми з безпекою. Коли організації передають свої дані та обчислення хмарним постачальникам послуг, вони відмовляються від прямого контролю над своєю конфіденційною інформацією. Це створює новий набір ризиків для безпеки, оскільки дані тепер зберігаються та обробляються на інфраструктурі, яка не належить організації та не контролюється нею повністю [3].

Система хмарних обчислень поділяється на дві частини: фронтенд і бекенд. Фронтенд – це частина, через яку користувач може взаємодіяти з сервером, а бекенд – це сервер, який надає дані клієнту. Між сервером і клієнтом мережа працює як проміжне програмне забезпечення (рис. 1).

Хмарні обчислення є одним з найвизначніших досягнень у світі інформаційних технологій за останнє десятиліття. IT–компанії надають гнучкі хмарні послуги та інфраструктуру практично для будь–яких цілей, що дозволяє клієнтам швидко масштабувати свої послуги за допомогою моделей надання послуг з оплатою за фактичне використання. Питання про потенційні ризики, пов'язані з втратою конфіденційності, цілісності та доступності даних користувачів у хмарних мережах через технічні збої або навіть цілеспрямовані атаки як ззовні, так і зсередини мережі хмарного провайдера [4].

Хмарні провайдери зазвичай використовують загальні моделі захисту безпеки, які вимагають довіри до хмарного провайдера, та класичні прозорі для користувача заходи безпеки для захисту своїх мереж та задоволення цілей безпеки своїх клієнтів.

Ці заходи все ще перебувають на початковій стадії та схильні до загальних загроз безпеки, таких як примус хмарних провайдерів під тиском місцевої (політичної) влади розкривати дані користувачів хмари [6].

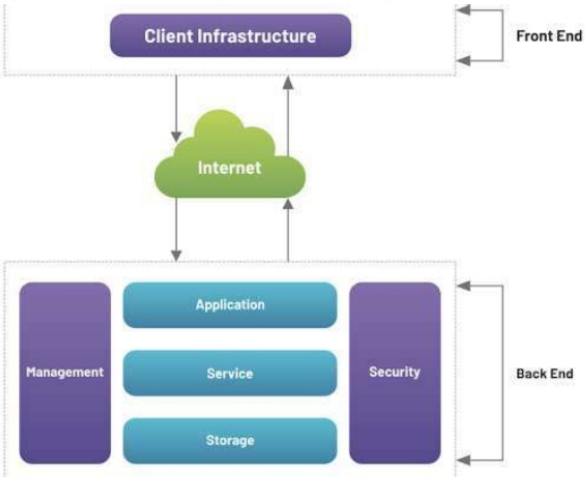

Рисунок 1 – Система хмарної архітектури

Обчислення зашифрованих даних базуються на двох незалежних алгоритмах шифрування, що забезпечують гомоморфність. Модель складається з агента, який виступає делегатором, та принаймні одного працівника в хмарі, який виконує обчислення, однак необхідність повторного шифрування даних для переходу між операціями додавання та множення обмежувала застосовність.

Висновок. Проведене дослідження показало, що визначення хмарних обчислень, про які згадувалося раніше, не згадують жодних понять безпеки даних, що зберігаються в хмарних обчисленнях, навіть незважаючи на те, що це нещодавнє визначення. Тому ми розуміємо, що хмарним обчисленням бракує безпеки, конфіденційності та прозорості. Надання інфраструктури, платформи або програмного забезпечення як послуги є недостатнім, якщо хмарний провайдер не гарантує кращої безпеки та конфіденційності даних клієнтів. В такій ситуації застосування гомомрфного шифрування на боці користувача дасть можливість безпечної обробки даних та дозволить забезпечити їх конфіденційність.

Перелік використаних джерел.

1. Gaborit, Philippe, Olivier Ruatta, and Julien Schrek. "On the complexity of the rank syndrome decoding problem." IEEE Transactions on Information Theory 62.2 (2020): 1006–1019.

2. Sarker, Iqbal H., et al. "Cybersecurity data science: an overview from machinelearning perspective." Journal of Big data 7 (2020): 1–29.

3. Chillotti I., Gama N., Georgieva M., Izabachène M.: TFHE: fast fully homomorphic encryption over the torus. J. Cryptol. 33(1), 34–91 (2021)

4. Marcolla, Chiara, et al. "Survey on fully homomorphic encryption, theory, and applications." Proceedings of the IEEE 110.10 (2022): 1572–1609.

5. Shaukat, Kamran, et al. “Performance Comparison and Current Challenges of Using Machine Learning Techniques in Cybersecurity.” Energies, vol. 13, no. 10, May 2020, p. 2509. https://doi.org/10.3390/en13102509.

ЗАХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ПОЛІТЕХНІКА»
ГАЛИЦЬКИЙ ФАХОВИЙ КОЛЕДЖ ІМ. В'ЯЧЕСЛАВА ЧОРНОВОЛА

КІБЕРБЕЗПЕКА ТА КОМП'ЮТЕРНО-ІНТЕГРОВАНІ ТЕХНОЛОГІЇ (КБКІТ – 2025)

науково-практична конференція
молодих вчених, аспірантів та студентів

28–29 серпня 2025
Тернопіль

Збірник матеріалів науково-практичної конференції молодих вчених, аспірантів та студентів «Кібербезпека та комп'ютерно-інтегровані технології» (КБКІТ - 2025), Тернопіль, 2025. - 154 с.

Адреса редакції:
Західноукраїнський національний університет, кафедра кібербезпеки,
вул. Олени Теліги 8, м. Тернопіль 46003
Контакти:
*e-mail:*conferencekb@gmail.com

УДК 004.421

Василь ПОМАЗИБІДА, Сергій КУЛИНА

Західноукраїнський національний університет

АЛГОРИТМИ ГОМОМОРФНОГО ШИФРУВАННЯ ДЛЯ БЕЗПЕЧНИХ ХМАРНИХ ОБЧИСЛЕНЬ

Вступ. У сучасному цифровому світі хмарні обчислення стали фундаментальною технологією для зберігання та обробки величезних масивів даних. Однак передача чутливої інформації - як-от медичні записи, фінансові дані чи комерційна таємниця - стороннім провайдерам створює значні ризики для безпеки та конфіденційності. Традиційне шифрування захищає дані під час зберігання та передачі, але вимагає їх розшифрування для будь-якої обробки, роблячи їх вразливими у хмарному середовищі.

Саме тому дослідження алгоритмів гомоморфного шифрування є надзвичайно важливим, оскільки вони дозволяють виконувати обчислення (наприклад, аналіз чи машинне навчання) безпосередньо над зашифрованими даними. Це дає змогу використовувати потужні ресурси хмари для обробки інформації, не розкриваючи при цьому самі дані, та є ключем до побудови справді безпечних хмарних сервісів, що гарантують повну конфіденційність.

Метою дослідження є підвищення рівня безпеки та ефективності обробки конфіденційних даних шляхом розробки або вдосконалення алгоритмів гомоморфного шифрування, що дозволяють збалансувати високий рівень захисту інформації з прийнятними обчислювальними витратами.

1. Дослідження існуючих алгоритмів гомоморфного шифрування

Дослідження існуючих алгоритмів гомоморфного шифрування зазвичай починається з їх класифікації за підтримуваними операціями та рівнем складності. Історично першими з'явилися частково гомоморфні системи (PHE), такі як Paillier (дозволяє необмежену кількість операцій додавання) або "чистий" RSA (дозволяє необмежену кількість операцій множення). Вони є обчислювально ефективними, але їхня функціональність обмежена вузькоспеціалізованими завданнями, як-от безпечне голосування чи агрегація статистичних даних. Наступним кроком стали дещо гомоморфні схеми (SHE), які підтримують обмежену кількість і додавань, і множень. Їхня головна проблема — неконтрольоване "зашумлення" даних: кожна операція збільшує "шум" у шифротексті, і після досягнення певного, заздалегідь визначеного порогу, дані стають неможливими для коректного розшифрування.

Сучасні дослідження зосереджені переважно на повністю гомоморфних (FHE) схемах, які вирішують проблему "шуму" за допомогою ресурсоємної операції "перешифрування" (bootstrapping). На практиці домінують кілька ключових "родин" алгоритмів, що базуються на проблемі складності (Ring) LWE. Схеми, як-от BGV та BFV, чудово підходять для точних обчислень над цілими числами (арифметичні схеми), що корисно для запитів до баз даних. З іншого боку, схема CKKS стала де-факто стандартом для прикладного машинного навчання та аналізу даних, оскільки вона працює з приблизними (дійсними) числами. Окремо стоять схеми TFHE/FHEW, оптимізовані для надшвидкого

перешифрування, що робить їх ідеальними для оцінки булевих схем. Таким чином, фундаментальна проблема, яку аналізує дослідження, — це компроміс: не існує єдиного "найкращого" алгоритму, і вибір (BGV, BFV, CKKS чи TFHE) залежить від конкретного хмарного завдання, при цьому всі вони все ще мають значні накладні витрати на продуктивність та розмір даних.

2. Розробка алгоритму гомоморфного шифрування

Варто зазначити, що гомоморфне шифрування — це не один конкретний алгоритм, а радше криптографічний протокол, що описує взаємодію між власником даних та обчислювальним середовищем. Розглянемо загальні кроки цього процесу на прикладі однієї з сучасних FHE-схем (як-от BFV або CKKS). Цей процес завжди включає щонайменше двох учасників: Клієнта (який володіє даними та секретним ключем) і Сервера (який виконує обчислення, маючи лише публічний ключ). Етапи роботи гомоморфного протоколу:

Крок 1. Ініціалізація та генерація ключів. На цьому етапі клієнт готує криптографічну систему та обирає набір криптографічних параметрів (наприклад, ступінь поліноміального кільця N, розмір модулів q і t). Від цих параметрів залежить рівень безпеки та "глибина" обчислень (скільки операцій можна виконати до того, як "шум" стане занадто великим).

Клієнт генерує три типи ключів:

– Секретний ключ (SK) - це приватний ключ, який клієнт нікому ніколи не передає. Він єдиний, що здатен розшифрувати дані.

– Публічний ключ (PK), який використовується для шифрування даних.

– Ключі оцінки (Evaluation Keys) - це спеціальний набір публічних даних (наприклад, "ключі перелінеаризації"), які необхідні Серверу для виконання складних операцій, найчастіше — множення шифротекстів.

Після чого клієнт надсилає Публічний ключ та Ключі оцінки на хмарний Сервер. Секретний ключ залишається у Клієнта.

Крок 2. Шифрування та передача даних.

Клієнт бере свої конфіденційні дані. Залежно від схеми (наприклад, BFV чи CKKS), дані кодуються у спеціальний формат (наприклад, у поліноми). Використовуючи свій Публічний ключ, Клієнт шифрує підготовлені дані та отримує шифротекст.

Клієнт надсилає зашифровані дані на сервер для зберігання та обробки.

Крок 3. Гомоморфна оцінка.

Сервер не має Секретного ключа і не може бачити дані. Сервер отримує від Клієнта (або авторизованого користувача) програму, яку потрібно виконати над даними. Ця програма має бути представлена у вигляді арифметичної схеми (послідовності операцій додавання та множення).

Сервер використовує властивості схеми для виконання обчислень. Кожна операція (особливо множення) додає до шифротексту "шум". Якщо "шум" перевищить певний поріг, дані буде неможливо розшифрувати. Сервер використовує Ключі оцінки для керування цим шумом (наприклад, перелінеаризація після множення).

Якщо програма дуже складна і "шум" стає критичним (у FHE-схемах), Сервер може виконати процедуру bootstrapping. Це операція, яка гомоморфно

"розшифровує" і "зашифровує" дані заново, використовуючи надані Клієнтом ключі. Вона "очищує" шифротекст від шуму, дозволяючи виконувати необмежену кількість операцій.

Виконавши всю програму, Сервер отримує кінцевий шифротекст.

Крок 4. Розшифрування результату.

Сервер надсилає зашифрований результат назад Клієнту.

Клієнт використовує свій Секретний ключ (SK) для розшифрування.

В результаті Клієнт отримує фінальний відкритий текст, який дорівнює результату обчислень, так, ніби вони виконувались на його власних незашифрованих даних.

Ефективність алгоритмів гомоморфного шифрування є головним компромісом та ключовим викликом для їхнього практичного впровадження. Хоча вони пропонують теоретично ідеальний рівень безпеки, дозволяючи обчислення на зашифрованих даних, їхня практична ефективність наразі залишається низькою порівняно з традиційною обробкою у відкритому вигляді. Це виражається у трьох основних аспектах: обчислювальна складність, роздуття даних та складність реалізації.

Висновок. На сучасному етапі розвитку, повністю гомоморфне шифрування не є ефективним для загальноцільових хмарних обчислень. Воно залишається нішевою технологією, ефективність якої є прийнятною лише для вузькоспеціалізованих завдань, де вимоги до конфіденційності є абсолютними і переважають будь-які витрати на продуктивність. Це можуть бути, наприклад, прості статистичні запити до медичних баз даних, приватне об'єднання фінансових наборів даних або виконання простих моделей машинного навчання.

Таким чином, головний напрямок досліджень сьогодні — це не стільки розробка нових криптографічних схем, скільки оптимізація існуючих: створення спеціалізованого апаратного забезпечення (FPGA, ASIC) для прискорення гомоморфних операцій та розробка кращих компіляторів і бібліотек, які б автоматизували складний процес оптимізації програм під FHE.

Перелік використаних джерел.

1. Dunuwila R. M. T. R., Senevirathne T., Weerasinghe P., Kankanamge C. A Comprehensive Survey on Fully Homomorphic Encryption for Cloud Computing: A Practical Perspective. IEEE Access. 2022. Vol. 10. P. 111494–111521.

2. Al-Khafaji S. A. G. G. N., Al-Naji A., Mohammed A. H. A survey of privacy-preserving SQL queries using homomorphic encryption in cloud databases. Journal of Cloud Computing. 2023. Vol. 12. Art. 48. DOI: 10.1186/s13677-023-00478-4.

3. Chillotti I., Gama N., Gkoulalas-Divanis A., Poly F. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. Journal of Cryptology. 2021. Vol. 34. № 4. Art. 28. DOI: 10.1007/s00145-021-09404-8.

4. Ghani M. K. A., Yusoff Z., Yunus M. A. M., Ariffin A. Homomorphic encryption in cloud computing: challenges and future directions. Multimedia Tools and Applications. 2023. Vol. 82. P. 3673–3697. DOI: 10.1007/s11042-022-13237-y.

5. Chen H., Ma M., Cui T., Han M., Wang Y. Efficient privacy-preserving machine learning framework in cloud computing using homomorphic encryption. Information Sciences. 2022. Vol. 609. P. 1007–1020. DOI: 10.1016/j.ins.2022.07.126.