НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА СИСТЕМНОГО ПРОГРАМУВАННЯ І
СПЕЦІАЛІЗОВАНИХ КОМП'ЮТЕРНИХ СИСТЕМ

«На правах рукопису»
УДК 004.021

«До захисту допущено»

Завідувач кафедри СПСКС

_____Віталій РОМАНКЕВИЧ
(підпис) (ім'я, прізвище)

"___"_____2020р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності 123 Комп'ютерна інженерія
(Системне програмування)

на тему: Спосіб гомоморфного шифрування даних на основі HElib для вебдодатків
_____

Виконав: студент ІІ курсу, групи КВ-92мп
Булах Олександр Віталійович _____ _____
(прізвище, ім'я, по батькові) (підпис)

Науковий керівник доцент, к.т.н Сапсай Т.Г._____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант з нормоконтролю доцент, с.н.с.,к.т.н. Юлія БОЯРІНОВА _____
_____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2020 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра системного програмування і спеціалізованих комп'ютерних систем

Рівень вищої освіти – другий (магістерський)
за освітньо-професійною програмою
Спеціальність 123 Комп'ютерна інженерія
(Системне програмування)

ЗАТВЕРДЖУЮ
Завідувач кафедри СПСКС
_____ Віталій РОМАНКЕВИЧ
(підпис)
«1» листопада 2019 р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Булаху Олександру Віталійовичу

1. Тема дисертації «Спосіб гомоморфного шифрування даних на основі HElib для вебдодатків», науковий керівник дисертації доцент, к.т.н. Сапсай Т.Г., затверджені наказом по університету від «12» листопада 2020 р. № 3298-С

2. Термін подання студентом дисертації 15 грудня 2020 р.

3. Об'єкт дослідження: процеси шифрування, параметри та алгоритми шифрування на основі бібліотеки HElib

4. Предмет дослідження: способи підвищення ефективності гомоморфного шифрування

5. Перелік завдань, які потрібно розробити: розробка способу зручного використання та налаштування параметрів алгоритму гомоморфного шифрування даних на основі бібліотеки HElib з застосуванням коефіцієнту пріоритетності, який дозволяє оптимізувати процес шифрування і надає можливість користувачу вибирати між швидкодією та крипостійкістю.

6. Перелік ілюстративного матеріалу: презентація

7. Перелік публікацій: «Спосіб гомоморфного шифрування даних на основі HElib для вебдодатків», XII наукова конференція магістрантів та аспірантів “Прикладна математика та комп’ютинг” ПМК-2020;

8. Дата видачі завдання 5 вересня 2019 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1.	Вивчення літератури за тематикою дисертації	11.10.2019	
2.	Підготовка матеріалів першого розділу магістерської дисертації	16.09.2020	
3.	Підготовка матеріалів другого розділу магістерської дисертації	22.09.2020	
4.	Підготовка матеріалів третього розділу магістерської дисертації	26.09.2020	
5.	Підготовка матеріалів четвертого розділу магістерської дисертації	30.09.2020	
6.	Підготовка матеріалів п’ятого розділу магістерської дисертації	05.10.2020	
7.	Розробка моделі сітки Петрі	09.10.2020	
8.	Розробка моделі задачі реального часу	15.10.2020	
9.	Оформлення документації магістерської дисертації	21.11.2020	
10.	Попередній розгляд магістерської дисертації на кафедрі	26.11.2020	

Студент _____ Олександр БУЛАХ
(підпис)

Науковий керівник дисертації _____ Тетяна САПСАЙ
(підпис)

РЕФЕРАТ

Актуальність теми

Наразі багато аспектів повсякденного життя все більше пов’язані з інформаційно-комунікаційними системами та сервісами, які повинні забезпечувати надійність обробки, зберігання та передачі даних. Обов’язковою частиною системного програмного забезпечення стали засоби криптографічних перетворень. Користувачі комп’ютерних систем і мереж можуть використовувати шифрування та розшифрування даних фактично без глибоких знань в області криптографії.

Одним із шляхів вирішення вказаної проблеми є використання механізму гомоморфного шифрування.

На сьогодні існує чимало алгоритмів гомоморфного шифрування, але вони не є достатньо ефективними для практичного застосування.

Оптимізація значень параметрів, при яких гомоморфне шифрування забезпечить необхідні криптографічні перетворення даних користувача, зберігаючи їх конфіденційність, є основою способу шифрування даних на базі HElib, що дозволяє керувати параметрами для кожного окремого випадку передачі інформації. Даний підхід сприяє подальшому розвитку механізму гомоморфного шифрування.

Мета і задачі дослідження

Розробка способу зручного використання та налаштування параметрів алгоритму гомоморфного шифрування даних на основі бібліотеки HElib з застосуванням коефіцієнту пріоритетності, який дозволяє оптимізувати процес шифрування і надає можливість користувачу вибирати між швидкодією та крипостійкістю.

Об'єкт дослідження – процеси шифрування, параметри та алгоритми шифрування на основі бібліотеки HElib.

Предмет дослідження – способи підвищення ефективності гомоморфного шифрування за рахунок оптимізації параметрів алгоритмів шифрування даних на основі бібліотеки HElib .

Методи дослідження

Математичне та програмне моделювання, емпіричний та порівняльний аналіз.

Наукова новизна

1. Обґрунтовано використання гомоморфного шифрування, алгоритмів та параметрів бібліотеки HElib для забезпечувати надійність обробки, зберігання та передачі даних.

2. Проведено аналіз впливу значень параметрів схеми гомоморфного шифрування на основні характеристики, такі як швидкодія та крипостійкість.

3. Запропоновано коефіцієнт пріоритетності, що дозволяє оптимізувати процес шифрування.

4. Розроблено спосіб зручного використання та налаштування параметрів алгоритму гомоморфного шифрування даних на основі бібліотеки HElib з застосуванням коефіцієнту, який дозволяє надає можливість користувачу вибирати між швидкодією та крипостійкістю.

Практична цінність одержаних результатів полягає в тому, що розроблений спосіб шифрування забезпечує зменшення навантаження на сервери й не витрачає час на шифрування даних, які цього не потребують.

Апробація результатів дисертації.

Основні положення і результати роботи представлені та обговорені на:

- XIII конференція молодих вчених «ПМК-2020» у «Прикладна математика та комп’ютинг»;

Публікації.

За темою досліджень опубліковано 1 наукову працю - тези доповіді на конференція.

Структура та обсяг дисертації.

Магістерська дисертація складається з чотирьох розділів.

Для вирішення поставленої задачі у першому розділі атестаційної роботи проведено аналіз сучасних криптографічних методів та їх можливості. Окреслено переваги та недоліки кожного з методів. Досліджено поняття гомоморфізму та гомоморфного шифрування, проаналізовано розвиток схем гомоморфного шифрування, техніки, що у них використовуються та їх особливості.

У другому розділі роботи визначено основні математичні поняття, на яких базується схема гомоморфного шифрування, що використовується у бібліотеці HElib, визначено основні алгоритми бібліотеки та її структуру.

У третьому розділі з-поміж розглядуваних середовищ для розробки було обрано Python Django для реалізації серверної частини вебдодатку. Описана логіка взаємодії серверу та програми шифрування даних. Для реалізації способу керування параметрами системи був створений коефіцієнт, зміна якого впливає на результати шифрування.

У четвертому розділі наведено порівняння швидкості роботи звичайного серверу з однаковими параметрами в алгоритмі шифрування для кожного повідомлення та системи з розробленим способом керування цими параметрами в залежності від пріоритету користувача.

Ключові слова.

Способи шифрування, гомоморфне шифрування, відкритий ключ, закритий ключ, швидкодія алгоритму, крипостійкість, параметри алгоритму.

Abstract

Actuality of theme

Today, many aspects of everyday life are increasingly related to information and communication systems and services that must ensure the reliability of data processing, storage and transmission. Cryptographic transformation tools have become a mandatory part of the system software. Users of computer systems and networks can use data encryption and decryption without any in-depth knowledge of cryptography.

One of the ways to solve this problem is to use the mechanism of homomorphic encryption.

Today, there are many homomorphic encryption algorithms, but they are not effective enough for practical application.

Optimization of parameter values, in which homomorphic encryption will provide the necessary cryptographic transformations of user data, while maintaining their confidentiality, is the basis of the method of data encryption based on HElib, which allows you to manage parameters for each case. This approach contributes to the further development of the mechanism of homomorphic encryption.

The purpose and objectives of the study

Development of a method for easy use and adjustment of parameters of the homomorphic data encryption algorithm based on the HElib library using a priority factor, which allows to optimize the encryption process and allows the user to choose between speed and cryptographic stability.

The object of research - encryption processes, parameters and encryption algorithms based on the HElib library.

The subject of research is ways to increase the efficiency of homomorphic encryption by optimizing the parameters of data encryption algorithms based on the HElib library.

Research methods

Mathematical and program modeling, empirical and comparative analysis.

Scientific novelty

1. The use of homomorphic encryption, algorithms and parameters of the HElib library to ensure the reliability of data processing, storage and transmission is justified.

2. The analysis of influence of values of parameters of the scheme of homomorphic encryption on the basic characteristics, such as speed and cryptoresistance is carried out.

3. The priority coefficient is offered, which allows to optimize the encryption process.

4. A method for easy use and adjustment of the parameters of the homomorphic data encryption algorithm based on the HElib library with the use of a coefficient that allows the user to choose between speed and crypto-resistance.

The practical value of the obtained results is that the developed method of encryption reduces the load on the servers and does not spend time encrypting data that does not require it.

Approbation of dissertation results.

The main provisions and results of the work are presented and discussed at:

• XIII Conference of Young Scientists "PMK-2020" in "Applied Mathematics and Computing";

Publications.

On the topic of research published 1 scientific work - abstracts for the conference.

The structure and scope of the dissertation.

The master's dissertation consists of four sections.

To solve this problem in the first section of the certification work conducted an analysis of modern cryptographic methods and their capabilities. The advantages and disadvantages of each method are outlined. The concepts of homomorphism and homomorphic encryption are studied, the development of homomorphic encryption schemes, the techniques used in them and their features are analyzed.

The second section of the work defines the basic mathematical concepts on which the homomorphic encryption scheme used in the HElib library is based, defines the main algorithms of the library and its structure.

In the third section, Python Django was selected for development to implement the server part of the web application. The logic of interaction between the server and the data encryption program is described. To implement the method of managing system parameters, a coefficient was created, the change of which affects the results of encryption.

The fourth section compares the speed of a normal server with the same parameters in the encryption algorithm for each message and the system with a developed way to manage these parameters depending on the user's priority.

Keywords.

Encryption methods, homomorphic encryption, public key, private key, algorithm performance, cryptoresistance, algorithm parameters.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

ПЕОМ – personal computer - персональна електронно обчислювана машина

ПГШ – FHE повністю гомоморфне шифрування

ЧГШ – PHE частково гомоморфне шифрування

AES – вдосконалений стандарт шифрування (advanced encryption standard)

DES – стандарт шифрування даних (data encryption standard)

BGV – Бракерскі, Джентрі, Вайкунтанатан (Brakerski, Gentry, Vaikuntanathan)

CRT – китайська теорема про залишки (Chinese remainder theorem)

FFT – швидке перетворення Фур'єра (Fast Furrier transformation)

GMP – багатоточна бібліотека GNU (GNU multi-precision library)

HElib – гомоморфна бібліотека шифрування (homomorphic encryption library)

LWE – навчання з помилкою (learning with error)

MD – хеш-функція (message digest)

NTL – бібліотека теорії чисел (number theory library)

RLWE – кільцеве навчання з помилкою (ring learning with error)

SHA – хеш-функція secure hash algorithm

SIMD – принцип одна інструкція, кілька даних (single instruction, multiple data)

ВСТУП

Сьогодні життя або робочий процес неможливо уявити без використання Інтернету. Все більше сервісів стали доступними у режимі онлайн, та мають на увазі обмін конфіденційними даними та інформацією. Наприклад: інтернет-банкінг, сплата рахунків онлайн, електронна пошта, обмін приватними та службовими повідомленнями тощо. Якщо ці дані будуть отримані сторонніми особами, то це може нанести шкоду не тільки окремому користувачу, а й усій системі онлайн бізнесу.

Щоб уникнути цього, були введені різноманітні заходи безпеки особистих даних у мережі. Головними серед них є процеси шифрування та дешифрування даних, які вивчає криптографія.

Ця наука вже давно зарекомендувала себе як надійний засіб для гарантованого зберігання конфіденційності інформації. Проте, у зв’язку із ростом та розвитком засобів передачі інформації, телекомунікаційних та інформаційних мереж, та, в особливості, Інтернету, перед криптографією стають нові завдання. Одним із них є забезпечення обчислень над зашифрованими даними.

Задача зберігання та обчислення даних все частіше покладається на сторонніх осіб, що викликає потребу у створенні такої схеми шифрування, яка дозволяє працювати не з відкритими даними, а з їх зашифрованим аналогом. Саме це і дозволяє робити гомоморфне шифрування.

На сьогодні вже існує чимало алгоритмів повністю гомоморфного шифрування. Але усі вони не є достатньо ефективними для практичного застосування. Розуміння оптимальних значень параметрів, при яких бібліотеки гомоморфного шифрування будуть забезпечувати швидке оперування із даними користувача, до того ж зберігаючи їх конфіденційність, може сприяти скорішому розвитку сфери гомоморфних шифрів. Тому робота є актуальною. Визначення оптимальних параметрів для бібліотеки HElib (homomorphic encryption library) в реальних умовах для кожного користувача, щоб його час відгуку був мінімальним в залежності від даних якими він оперую є ціллю нашої роботи.

Враховуючи зростання впровадження інформаційних технологій у різноманітні сфери життя суспільства, криптографічний захист є найпопулярнішим та найнадійнішим засобом захисту конфіденційної інформації.

По-перше, криптографія активно використовується для передачі інформації через відкриті канали зв’язку, такі як Інтернет, мобільний зв’язок і таке інше.

По-друге, широко застосовується шифрування у різноманітних банківських системах, обслуговуванні банківських кредитних карт тощо.

По-третє, це підвищує безпечність зберігання інформації на персональному комп’ютері.

Метою роботи є дослідження залежності ефективності схеми гомоморфного шифрування від її основних параметрів та вибір оптимальних значень для кожного користувача з метою зменшення часу затримки повідомлень на стороні сервера в залежності від їх розміру.

1. ОСОБЛИВОСТІ ВИКОРИСТАННЯ АЛГОРИТМІВ ШИФРУВАННЯ

1.1 Сучасні методи та типи шифрування

Одним із способів захистити конфіденційні дані від небажаного розголошення є використання шифрування. Шифрування - це механізм перетворення із чистого текстового формату у складний та розсіяний текстовий формат, який не може зрозуміти будь-хто, хто очікує від уповноважених сторін [5]. За останнє десятиліття криптологи винайшли 11 різних алгоритмів шифрування для забезпечення безпеки даних або інших текстових форматів. Методи шифрування - це алгоритми, засновані на ключах, де текст шифрується та розшифровується за допомогою секретного ключа.

Шифрування - це перетворення інформації з метою приховування її від несанкціонованих осіб, але з наданням доступу до неї авторизованим користувачам. Важлива відмінність (характеристика) будь-якого алгоритму шифрування - використання ключа, що визначає вибір конкретного перетворення із набору можливих перетворень для цього алгоритму.

Шифрування призначене для захисту конфіденційної інформації від зловмисників, які можуть перехопити приватне повідомлення. Зазвичай одна сторона зашифровує повідомлення і надсилає його зашифрованим до приймаючої сторони, яка розшифровує та отримує повідомлення. Щоб запобігти перехопленню повідомлення третьою стороною, необхідно забезпечити надійне та ефективне шифрування інформації. У цьому випадку шифротекст не повинен містити жодної інформації про відкритий текст.

Існує багато алгоритмів шифрування. Але всі можна класифікувати в основному за типом ключів та режимом перетворення.

Алгоритми, що використовують один і той же секретний ключ для шифрування та дешифрування, називаються шифруванням симетричних ключів, а методи, що використовують різні ключі, називаються асиметричними ключами.

Класифікацію методів шифрування наведено на рисунку 1.1.

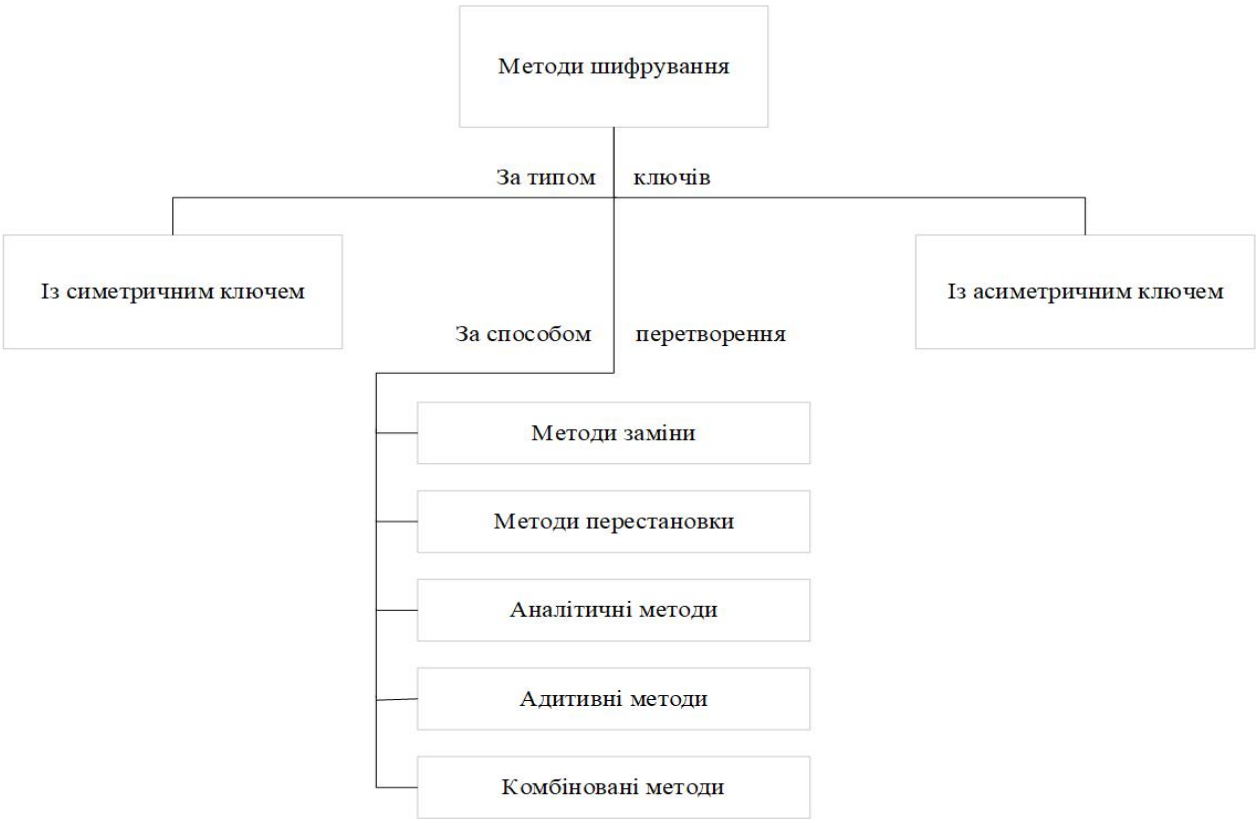

Рисунок 1.1 – Класифікація методів шифрування

1.1.1 Шифрування із симетричним ключем

Симетричне шифрування використовує той самий ключ для шифрування повідомлення та його дешифрування. Ключ до алгоритму обирається сторонами до початку обміну інформацією. Таким чином, стає зрозумілим, що ключ є найважливішою частиною алгоритму і що кожен, хто має до нього доступ, зможе розшифрувати повідомлення. Тому його потрібно зберігати безпечно. Це ставить проблему початкової передачі ключа. Існують методи криптографічних атак, які дозволяють тим чи іншим способом розшифрувати інформацію, не маючи ключа або не перехоплюючи її на етапі обміну. Незважаючи на відсутність стійкості алгоритму до атак, симетричне шифрування має одну велику перевагу: швидкість.

Двома популярними шифруваннями симетричних ключів є Стандарт шифрування даних (DES) та Розширена система шифрування (AES). Ці методи шифрування описані нижче

Стандарт шифрування даних DES був розроблений для уряду США і призначений для використання громадськістю [6]. Алгоритм складний, але

повторюваний, що робить його дуже придатним для реалізації на одноцільових мікросхемах. Багато програмних та апаратних систем мають DES як реалізовану систему шифрування. Алгоритм був розроблений в 70-х роках компанією IBM. Це комбінація алгоритмів транспонування та заміщення, але вона виконує ці алгоритми неодноразово один над одним. Алгоритми застосовуються загалом за 16 циклів. Починається з шифрування блоків відкритого тексту на 64 біти кожен ключем на 64 біти. Фактичне значення ключів - будь-яке 56-бітове число, а інші 8 бітів використовуються для перевірки цифр, що не заважає шифруванню або його реалізації.

Потрійний DES (3DES) використовує три ключі для підвищення міцності шифрування. Ключі мають той самий розмір, що і у звичайного DES (64-бітний розмір із силою 56 біт). Застосування 3DES полягає в тому, що ви спочатку шифруєте першим ключем, потім другим поверх цього і, нарешті, третім. Це надає алгоритму міцність 112-бітового ключа [6]. Отже, 3DES значно сильніший за звичайний DES, але вимагає більших обчислень та більшої кількості ключів. Існує інша версія 3DES, яка використовує лише два ключі. Ця версія використовує принцип шифрування, дешифрування, шифрування. Це означає, що він використовує перший ключ для шифрування, потім розшифровує поверх цього, а потім другий ключ для шифрування поверх цього. Це дає міцність у 80 біт, що все ще сильніше, ніж DES, але слабше, ніж у попередньому 3DES.

Double DES використовує два ключі однакового розміру та складністю DES. Він шифрує відкритий текст першим ключем, а потім другим поверх нього. Однак це лише збільшує теоретичну міцність ключа 1, що призводить до 57-бітового значення ключа. Через це цей метод шифрування не такий популярний, як DES або 3DES, оскільки підвищена міцність на 1 не варта підвищеної складності, яку пропонує алгоритм.

AES: вдосконалена система шифрування AES, також відома як алгоритм Ренделя, - це швидкий і потужний алгоритм шифрування, який може бути реалізований на простих процесорах. Він має надійну математичну основу і використовує безліч елементів, таких як підстановка, транспонування, побітова

функція «зсув виключного АБО» та інші додаткові механізми для досягнення надійного алгоритму шифрування. AES використовує повторювані раунди роботи; 10, 12 або 14 раундів для ключів розмірами 128, 192 або 256 біт. Кожен раунд містить чотири кроки, які замінюють і шифрують біти. Для додавання дифузійних бітів від ключа вони часто поєднуються з бітовими результатами підстанції та транспонування. На відміну від раундів DES, який обмежений лише 16 раундами, раунди AES можна збільшити для підвищення безпеки шифрування. Якщо фахівці з безпеки вважають, що 10, 12 або 14 раундів занадто мало, єдина зміна, яку їм потрібно зробити, - це налаштувати алгоритм у циклі повторення та змінити обмеження необхідних раундів [6].

Існує два типи обробки вхідних даних у симетричних шифрах: блоки та потоки.

Однією з характеристик блочного шифру є обробка блоку з декількох байтів за одну ітерацію (зазвичай 8 або 16). Блочні криптосистеми ділять текст повідомлення на окремі блоки, а потім перетворюють ці блоки за допомогою ключа.

Трансформація має використовувати наступні принципи:

- підстановок - тобто зміна будь-якого знаку відкритого тексту або ключа впливає на велику кількість символів зашифрованого тексту, що приховує статистичні властивості відкритого тексту;
- перетасовка - використання перетворень, що ускладнює отримання статистичних зв’язків між зашифрованим та відкритим текстом.

До переваг блочних шифрів можна віднести подібність процедур шифрування та дешифрування, які, як правило, відрізняються лише порядком дій. Це спрощує створення пристроїв шифрування, оскільки дозволяє використовувати однакові блоки як для шифрування, так і для дешифрування.

Поточне шифрування - це симетричне шифрування, при якому кожен символ відкритого тексту перетворюється на символ зашифрованого тексту на основі не тільки використовуваного ключа, але і його розташування в потоці відкритого тексту.

Основні відмінності між поточним і блоковим шифрами:

- забезпечення шифрування за допомогою поточного алгоритму шифрування в режимі реального часу не залежить від обсягу та бітової глибини перетвореного потоку даних;
- у поточних синхронних шифрах, на відміну від блокових, немає ефекту поширення помилки, тобто кількість спотворених елементів у розшифрованій послідовності дорівнює числу d - спотворені елементи послідовності шифру;
- поточна структура шифрування може мати вразливості, які дозволяють криптоаналітику отримувати додаткову інформацію про ключ;
- поточні шифри, на відміну від блокових шифрів, можна атакувати за допомогою лінійної алгебри. Лінійний та диференціальний аналіз також успішно використовується для розшифровки поточних чисел.

Більшість симетричних шифрів використовують складну комбінацію безлічі підстановок і перестановок. Багато з цих шифрів виконуються за кілька проходів (іноді до вісімдесяти), використовуючи відповідний ключ для кожного проходу. Набір «ключів доступу» для всіх проходів називається «розкладом ключів». Зазвичай він створюється з головного ключа шляхом виконання деяких операцій над ним, включаючи перестановки та заміни.

Типовим способом побудови симетричного алгоритму шифрування є мережа Фейстеля. Алгоритм будує схему шифрування на основі конкретної функції F (D, K), де D - це елемент даних, розмір якого вдвічі менший за блок шифрування, а K - ключ до цього проходу. При використанні мережі Фейстеля алгоритми шифрування та дешифрування майже повністю збігаються (виключаючи зворотний порядок передачі ключів у розкладі), що значно спрощує апаратну реалізацію.

Операція перестановки перемішує біти повідомлення відповідно до певного правила. Саме ці операції дають змогу отримати "ефект лавини". Операція перестановки є лінійною, тобто .

Операції заміни виконуються шляхом заміни значень частини повідомлення (найчастіше в 4, 6 або 8 біті) на стандартне число, вбудоване в алгоритм, шляхом доступу до масиву констант. Операція заміни вносить нелінійність в алгоритм.

Прикладами шифрів із симетричним ключем є: шифр Цезаря, шифр Віженера, DES (data encryption standard), Rijndael (AES – advanced encryption standard).

1.1.2 Шифрування із асиметричним ключем

Асиметричним шифрування - це процес шифрування, при якому ключ, що використовується для шифрування повідомлення, відрізняється від ключа, що розшифровує повідомлення. Існує два ключі: а) закритий та б) відкритий. Будь-хто може мати доступ до відкритого ключа, але закритий ключ тримається в таємниці. Коли один використовується для шифрування, інший використовується для дешифрування. У схемі шифрування, що виконується за допомогою відкритого ключа, забезпечує секретність повідомлення, оскільки ніхто, крім особи, що має приватний ключ, не може його розшифрувати.

Принципова відмінність асиметричної криптосистеми від симетричної полягає у тому, що для шифрування інформації та її подальшого дешифрування використовуються різні ключі:

- відкритий ключ **К** використовується для шифрування інформації, обчислюється із таємного ключа **к**;
- таємний ключ **к** використовується для дешифрування інформації, яка зашифрована за допомогою парного йому відкритого ключа **К**.

Ці ключі розрізняються таким чином, що за допомогою обчислень не можна вивести таємний ключ **к** із відкритого ключа **К**. Тому відкритий ключ **К** може вільно передаватися каналами.

Асиметричні системи також називають криптосистемами із відкритим ключем.

Для того, щоб передати таємне повідомлення отримувачу B, потрібно зашифрувати це повідомлення на відкритому ключі отримувача, після чого він зможе його дешифрувати на своєму таємному ключі.

Таємний та відкритий ключі генеруються попарно. Таємний ключ має залишатись у його власника та бути надійно захищеним від несанкціонованого доступу. Копія відкритого ключа має знаходитись у кожного абонента криптографічної мережи, з котрим буде обмінюватись інформацією власник таємного ключа.

Алгоритм Діффі - Геллмана:

Обмін ключами Діффі – Хеллмана є специфічним методом обміну криптографічними ключами, запропонованим у 1976 р. Він дозволяє двом сторонам, які не мають попередньої інформації одна про одну, встановити спільний секретний ключ через незахищений канал зв'язку. Цей ключ використовується для шифрування подальших комунікацій за допомогою симетричного шифру ключа.

RSA:

Алгоритм RSA розроблений Ронам Рівестом, Аді Шаміром та Леонардом Адлеманом із MIT. Він має два ключі: відкритий та закритий ключі. Обидва ключі використовуються для шифрування та дешифрування. Відправник шифрує повідомлення за допомогою відкритого ключа, і коли повідомлення отримує приймач, тоді він може розшифрувати його за допомогою власного приватного ключа. Він використовує два простих числа та генерує приватний та відкритий ключі. Алгоритм RSA залежить від добутку цих двох чисел, представлених n.

Алгоритм цифрового підпису (DSA):

Алгоритм цифрового підпису - це криптографічний алгоритм із відкритим ключем, призначений для автентифікації цифрових повідомлень. Повідомлення підписується секретним ключем для отримання підпису, а потім це перевіряється відкритим ключем. Будь-яка сторона може перевірити підписи, але лише одна сторона, що має секретний ключ, може підписати повідомлення. Дійсний цифровий підпис дає підставу одержувачу вважати, що повідомлення було

створене відомим відправником. У нього є секретний ключ, і він не був змінений під час транспортування.

Переваги алгоритмів шифрування із асиметричним ключем над алгоритмами із симетричним:

- в асиметричних криптосистемах вирішена складна проблема розподілу ключів між користувачами, кожен користувач може сам згенерувати свою пару ключів, а відкриті ключі користувачів можуть вільно публікуватися та поширюватися мережевими комунікаціями;
- зникає квадратична залежність числа ключів від числа користувачів. В асиметричній криптосистемі число ключів, що використовуються, пов’язане із числом абонентів лінійною залежністю (у системі, що складається з N користувачів, використовуються 2·N ключів), а не квадратичною, як у симетричних системах;
- асиметричні криптосистеми дозволяють реалізовувати протоколи взаємодії сторін, які не довіряють один одному, оскільки при використанні асиметричних криптосистем закритий ключ має бути відомим тільки його власнику.

Недоліки алгоритмів шифрування із асиметричним ключем:

- на сьогодні не існує математичного доведення незворотності використовуваних в асиметричних алгоритмах функцій;
- асиметричне шифрування значно повільніше за симетричне;
- необхідність захисту відкритих ключів від заміни.

Найпоширенішими сучасними алгоритмами із використанням асиметричних ключів є: RSA (Rivest-Shamir-Adleman, Рівест-Шамір-Адлеман), DSA (digital signature algorithm), Elgamal (шифросистема Ель-Гамаля), Diffie-Hellman (обмін ключами Діффі-Хелмана), ECC (elliptic curve cryptography, криптографія на еліптичних кривих).

1.1.3 Методи шифрування за способом перетворення

Існуючі методи шифрування відрізняються не тільки типом ключів, а й використовуваним способом перетворення.

Шифрування методом заміни полягає у заміні відповідно до певного правила символів вихідної інформації, які записані одним алфавітом, символами іншого алфавіту.

Під час шифрування методом перестановки символи тексту переставляються відповідно до певного алгоритму всередині блоку та шифруються.

Методи аналітичного шифрування, як правило, засновані на використанні аналітичних перетворень матричної алгебри.

При використанні адитивних методів шифрування виконується послідовна компіляція цифрових кодів, що відповідають символам вихідної інформації, з певною спеціальною послідовністю кодів, яка називається гамма. Тому адитивні прийоми також називають смиренням.

Поєднання декількох різних методів є ефективним способом підвищення міцності шифрування. Для послідовного шифрування оригінального тексту використовується два або більше методів. На практиці найпоширенішими є такі комбінації методів:

- перестановка + гамування;
- підстановка + гамування;
- гамування + гамування;
- підстановка + перестановка.

1.1.4 Хеш-функції

Криптографічні хеш-функції є ще одним важливим примітивом і можуть бути використані для забезпечення цілісності оброблених даних. У симетричній криптографії вони також часто використовуються як будівельні блоки для інших криптографічних примітивів, таких як потокові шифри, коди автентифікації повідомлень або автентифіковані схеми шифрування. Хеш-функції не

використовують секретний ключ, на відміну від симетричних примітивів, про які вже йшлося раніше, і стискають інформацію у довільний розмір, подану на вхід скінченну кількість даних до фіксованого розміру вихідного повідомлення. Останній можна розглядати як відбиток пальця, тобто «унікальний» ідентифікатор вхідних даних. Ці примітиви належать до сімейства так званих односторонніх функцій, що означає, що їх практично неможливими інвертувати. Сфера їх застосування включає: перевірку цілісності даних, перевірку пароля, генерацію псевдовипадкових чисел та автентифікацію повідомлень. Формальне визначення хеш-функції таке.

Хеш-функція – це така функція, яка дозволяє привести будь-який масив даних до числа заданої довжини.

Колізією для функції h називається пара значень x, y, $x \neq y$, така, що $h(x) = h(y)$.

При побудові функції хешування потрібно, щоб рішення наступних криптоаналітичних задач було складним:

- за заданим $y = h(x)$ визначити x (тобто h має бути односторонньою функцією);
- для заданого *x* знайти інше x^i, таке, що $h(x) = h(x^i)$, (тобто h має бути вільною від колізій функцією);
- знайти пару *x*, x^i, ($x \neq x^i$, таку, що $h(x) = h(x^i)$ (тобто h є точно вільною від колізій функцією).

Існує два важливі типи криптографічних хеш-функцій: ключова та безключова. Хеш-функції ключів також відомі як коди автентифікації повідомлень. Вони дозволяють без додаткових засобів гарантувати як надійність джерела даних, так і цілісність даних у системах з користувачами, а також довіряти один одному.

Хеш-функції без ключа називаються кодами виявлення помилок. Вони дозволяють додаткові засоби (наприклад, шифрування), щоб гарантувати

цілісність даних. Ці хеш-функції можна використовувати в тих системах, де користувачі довіряють один одному, і навпаки.

Найпоширенішими у світі алгоритмами хешування сьогодні є:

- алгоритми MD2 / 4/5/6. MD4 містить три цикли по 16 кроків кожна. У 1993 році був описаний алгоритм злому цієї функції, тому сьогодні він майже не використовується. Найпоширенішою з цього сімейства є функція MD5. Він містить чотири цикли по 16 кроків кожен. Але доведено, що функція створює колізії, тому її використання поки не рекомендується.
- алгоритми SHA (алгоритм безпечного хешування). Відбувається активний перехід від стандартів версії SHA-1 до SHA-2. Обидві версії повільніші за функції сімейства MD. SHA-2 - загальна назва алгоритмів SHA224, SHA256, SHA384 і SHA512. SHA224 та SHA384 є по суті аналогами SHA256 та SHA512, відповідно, лише після обчислення дайджесту частина інформації, яку вони містять, відкидається. Їх слід використовувати лише для забезпечення сумісності зі старим обладнанням.

1.2 Застосування алгоритмів шифрування

Важливо також переконатися, що неможливо перервати передачу даних як від одержувача, так і від відправника.

Використовуючи хеш-функції, вони реалізують:

- перевірка цілісності даних. Ідея полягає в тому, щоб зберегти хеш-код, а потім порівняти його з перерахованим хеш-значенням для тих самих даних. Нерівність цих двох значень означатиме порушення цілісності;
- системи автентифікації з використанням хешування паролів;
- створення та перевірка ЕЦП (електронний цифровий підпис). Механізм хеш-функції використовується для зменшення часу, необхідного для створення та перевірки підпису, а також для зменшення його довжини.

Симетричне, асиметричне шифрування та хешування можуть бути скомпоновані разом, щоб досягти найвищих показників у швидкодії та забезпеченні безпеки, на що окремо кожен метод шифрування не може

забезпечити. Наприклад, реалізація протоколу IPsec (internet protocol security) для забезпечення захисту даних, що передаються за мережевим протоколом IP (internet protocol).

1.3 Гомоморфне шифрування

Гомоморфне шифрування - це тип шифрування, при якому, виконуючи певні дії із зашифрованим текстом, одночасно змінюється і відкритий текст. Більше того, вам не потрібно знати відкритий текст.

Гомоморфні властивості різних криптографічних систем були винайдені в 1978 р. Рівестом, Адлеманом та Дертузо [1]. Вони опублікували статтю, в якій пропонується використовувати "приватний гомоморфізм" не лише для забезпечення безпеки даних, а й для того, щоб дані могли використовувати недовірені особи. Рівест і Адлеман - два з трьох відомих винахідників схеми шифрування RSA, яка була першою схемою шифрування з відкритим ключем з гомоморфними властивостями.

Однак для забезпечення безпеки RSA повинен додати до повідомлення кількість випадкових бітів перед його шифруванням. Але це призводить до втрати гомоморфних властивостей. Щоб уникнути набивання повідомлень, протягом останніх трьох десятиліть було запропоновано багато схем шифрування з відкритими ключами з різними гомоморфними властивостями.

Характер шифрування за своєю суттю приховує деяку функціональність вхідного тексту, ускладнюючи або навіть неможливо використовувати певні операції. Рівест, Адлеман та Дертузо запропонували схему шифрування, яка дозволяла б майже необмежене використання зашифрованого повідомлення. Цю схему назвали абсолютно гомоморфною. Щоб бути повністю гомоморфною, схема повинна допускати як додавання, так і множення. Але при багаторазовому шифруванні тексту його основні властивості втрачаються через так званий шум зашифрованого тексту.

Шум шифрування - це довільно вибране значення, яке навмисно додається до схеми шифрування, щоб ускладнити зловмисникам розшифровку

повідомлення. Основна відмінність класичної криптографії від гомоморфних схем полягає в тому, що повідомлення не перетворюються на гомоморфні схеми, а до них просто додається шум, тоді як цей шум і ці повідомлення важко розрізнити.

У 2009 році здобув ступінь доктора Стенфордського університету. Крейг Джентрі дійшов висновку, що замість надання гомоморфних властивостей лише в схемі шифрування краще періодично оновлювати зашифрований текст, позбавляючи його надмірності, тобто шуму. [2] Після цього схема, яка дозволяла обмежену кількість операцій, тепер могла дозволити виконувати більшу кількість операцій.

У математиці гомоморфізм описує перетворення з одного набору даних на інший, зберігаючи зв'язок між елементами двох наборів даних. Цей термін походить від грецьких слів, що означають "ОМОС" - рівна, однакова і "морфінова" форма. Оскільки дані в гомоморфній схемі шифрування зберігають однакову структуру, однакові математичні операції - незалежно від того, виконуються вони над зашифрованими чи розшифрованими даними - дають еквівалентні результати. В абстрактній алгебрі гомоморфізм – це відображення алгебраїчної системи A, що зберігає основні операції та основні відносини.

Існує твердження, що група G гомоморфна групі H якщо кожному елементу g із групи G поставлено у відповідність певний елемент f(g) групи H (до того ж кожний елемент групи H поставлено у відповідність хоча б одному елементу групи G) так, що для усіх елементів $g, g' \in G$ справедлива рівність $f(gg') = f(g)f(g')$.

Однак, відображення групи G в групу H не є взаємно однозначним: кожному елементу g групи G відповідає один певний елемент f(g) з H, але різним елементам з G може бути поставлено у відповідність той же самий елемент з H.

Група – це набір G, який разом із операцією $\circ$ (назвемо його груповим законом G), поєднує у собі будь-які два елементи a та b, щоб утворити інший елемент, позначений як $a \circ b$.

Щоб бути групою, набір і операція $(G,\circ)$ повинні відповідати чотирьом вимогам, відомим як аксіоми груп:

- для усіх a, b у наборі G результат операції $a \circ b$ так само буде належати набору G;
- для усіх a, b та c у наборі G, $(a \circ b) \circ c = a \circ (b \circ c)$;
- існує такий елемент e у наборі G, що для кожного елементу a у наборі G зберігається рівність $(e \circ b) = a \circ e = a$. Це унікальний елемент, тому можна назвати його елементом ідентичності;
- для кожного a у наборі G існує елемент b у тому ж наборі такий, що $(a \circ b) = b \circ a = e$, де e – це елемент ідентичності.

Гомоморфізм груп наведено на рисунку 1.2.

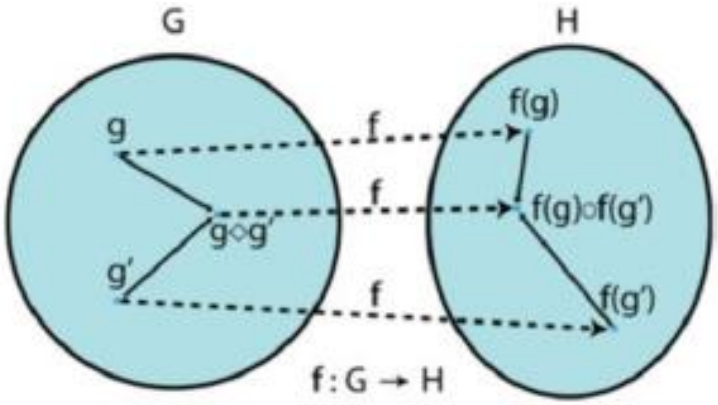

Рисунок 1.2 – Гомоморфізм груп

Розрізняють повністю гомоморфні (ПГ) та частково гомоморфні (ЧГ) криптосистеми.

Схема вважається ЧГ, якщо вона може правильно виконувати обмежену кількість операцій через неможливість правильного розшифрування після введення певного порогового рівня шуму операціями. Ця система може бути гомоморфною, якщо дозволяє виконувати одну з операцій: додавання або множення, без створення шуму шифрування. Коли схема називається ЧГ з додаванням, це означає, що ви можете виконувати необмежену кількість операцій

додавання над зашифрованим текстом, тобто необмеженої кількості вкладень до вихідного тексту.

Система множення або мультиплікативно-гомоморфна схема, повинна дозволити необмежену кількість операцій (що відповідає множенню вихідних текстів), перш ніж правильно розшифрувати і, нарешті, вихідний текст.

На високому рівні суть повністю гомоморфного шифрування проста: дані шифротексти, що шифрують π1,. . . , πt, повністю гомоморфне шифрування повинно дозволити кожному (а не лише власнику ключів) виводити зашифрований текст, який шифрується f(π1, ..., πt) для будь-якої потрібної функції f, доки цю функцію можна ефективно обчислити. Інформації про π1,. . . , πt або f (π1, ..., πt), або будь-які проміжні значення відкритого тексту, не повинні витікати; вхідні, вихідні та проміжні значення завжди шифруються.

Формально існують різні способи визначення того, що означає кінцевий зашифрований текст f (π1, ..., πt). Мінімальна вимога - правильність. Повністю гомоморфна схема шифрування *E* повинна мати ефективний алгоритм $\mathsf{Evaluate}_{\mathcal{E}}$, який для будь-якої дійсної пари ключів *E* (sk, pk), будь-якої схеми *C* та будь-яких текстів шифру $\psi_i \leftarrow \mathsf{Encrypt}_{\mathcal{E}}(\mathrm{pk}, \pi_i)$ виводить

$$\psi \leftarrow \mathsf{Evaluate}_{\mathcal{E}}(\mathrm{pk}, C, \psi_1, \ldots, \psi_t)$$

$$\mathsf{Decrypt}_{\mathcal{E}}(\mathrm{sk}, \psi) = C(\pi_1, \ldots, \pi_t)$$

Якщо схема може виконувати необмежену кількість операцій обох типів, то вона є ПГ.

Стандартна система шифрування може бути описана наступним чином:

- генерація ключів (key generation – KeyGen);
- шифрування (encryption – Encrypt);
- дешифрування (decryption – Decrypt).

Гомоморфна криптосистема окрім трьох перерахованих вище алгоритмів включає у себе ще алгоритм обчислення – evaluation (Eval).

Схематично цей алгоритм обчислень представлено на рисунку 1.3.

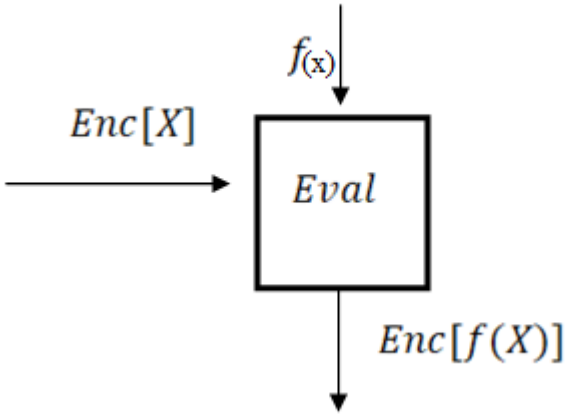

Рисунок 1.3 – Алгоритм обчислень у схемі гомоморфного шифрування

На вхід алгоритму подається зашифроване повідомлення Enc[X] та деяка математична функція f(x). Результатом роботи алгоритму буде інше зашифроване повідомлення Enc[X_2], до того ж $X_2 = f(X)$.

Схема гомоморфного шифрування може вважатися коректною, якщо для будь-якої пари ключів Secret key (SK) та Private key (PK), будь-якої функції f(x), будь-якого набору відкритих текстів $X = \{x_1, x_2, ..., x_n\}$ та набору шифртекстів $C = \{c_1, c_2, ..., c_n\}$ таких, що $C = Enc[X]$, виконується умова – якщо $r = Eval(PK, C, f)$, тоді $Decrypt(SK, r) = f(x_1, x_2, ..., x_n)$.

Схема Джентрі – це асиметрична схема шифрування, яка використовує ідеальні решітки.

Крейг Джентрі запропонував першу повністю гомоморфну схему шифрування, вперше вирішивши центральну відкриту проблему криптографії. Така схема дозволяє обчислювати довільні функції над зашифрованими даними без ключа дешифрування. Він використовував ідеальні решітки, які мають особливі властивості гомоморфізму як щодо додавання, так і для множення. Він розпочав із схеми частково гомоморфного шифрування, яку можна перетворити на повністю гомоморфну схему шифрування за допомогою ключового методу, званого Bootstrapping.

Повністю гомоморфна схема шифрування Джентрі покладається на схему Ideal Lattice для своєї безпеки. Схема Джентрі перетворила довільні функції додавання та множення у логічні схеми решіток NAND. Шлюз NAND був

кращим, оскільки це універсальний шлюз, і будь-яка булева функція може бути виражена суто в термінах роботи NAND.

Схема генерує секретний ключ і кілька відкритих ключів, кожен з яких має шум. Це було створено для того, щоб зловмиснику було важче створити секретний ключ із відомого відкритого ключа. Відкриті ключі використовуються для шифрування даних, і тоді на них можна виконувати обчислення. Однак шум у зашифрованому тексті збільшується з кожним наступним обчисленням. Зрозуміло, що вектор шуму в зашифрованому тексті подвоюватиметься з кожною операцією додавання до цього зашифрованого тексту і буде збільшуватися в квадратичному розмірі з кожною операцією множення. У якийсь момент секретний ключ вже не можна використовувати для дешифрування, оскільки шуму стало занадто багато. Як тільки значення шуму перевищить певний пороговий рівень, розшифровка тексту буде неправильною. Отже, у частково гомоморфній схемі функції, які можуть виконувати гомоморфні обчислення в зашифрованому тексті, повинні працювати з поліномами малого обмеженого ступеня. Схеми, що реалізують це шифрування, були названі частково гомоморфними. Але якщо кожен раз зменшувати рівень шуму в отриманому зашифрованому тексті, ви можете зробити багато гомоморфних арифметичних обчислень за допомогою цього зашифрованого тексту. Для того, щоб зробити схему повністю гомоморфною, Джентрі винайшов техніку, яка називається Bootstrapping.

Концепція Джентрі полягала в тому, щоб зашифрувати дані двічі і таким чином, щоб помилки після першого шифрування могли бути усунені.

У будь-якій даній схемі частково гомоморфного шифрування зашифрований текст можна "оновити", якщо він розшифровується за допомогою секретного ключа гомоморфного шифру, в якому значення шуму дуже мале. Основна ідея процедури початкового завантаження полягає в тому, що вхідне повідомлення буде зашифровано та розраховано на ключ, поки значення шуму не досягне своєї межі. Якщо ця межа досягнута, наступним кроком є повторне зашифрування вже зашифрованого тексту, але цього разу за допомогою зашифрованого секретного

ключа. Це призводить до іншої зашифрованої версії того самого відкритого тексту. Якщо ступінь дешифрувального полінома досить мала, тоді шум у новому зашифрованому тексті стане меншим, і тому його можна буде використовувати в наступній гомоморфній операції (додавання або множення). Після цього виконується процедура дешифрування за допомогою першого відкритого ключа. Важливим моментом є те, що схема, яка використовує процедуру грунтування, повинна підтримувати обчислення з поліномами високого ступеня, але в той же час мати алгоритм дешифрування, який можна розглядати для обробки поліномів низького ступеня. Якщо обчислюваний ступінь полінома перевищує ступінь полінома, який розшифровується, то можна припустити, що схема підтримує процедуру початкового завантаження і може бути перетворена в схему повністю гомоморфного шифрування.

У 2010 році Смартом (Smart) та Веркаунттерном (Vercaunteren) була прийнята перша спроба реалізувати схему Джентрі. Вони реалізували варіант схеми, котра використовувала принципово ідеальні решітки та вимагала від числа, що визначає решітку, бути простим числом. Схема була частково гомоморфною, а також не підтримувала достатньо великі параметри, тому схема дешифрування не підтримувала бутстрепінг. Навіть після того, як було знайдене просте число, що буде визначати решітку, все ще потрібно було обчислити секретний ключ. Це потребувало великих обчислюваних ресурсів [3].

Пізніше того ж року, Джентрі та Халеві (Halevi) презентували іншу реалізацію схеми Джентрі. Ця реалізація була першою, що підтримувала бутсрепінг. Вони зробили це шляхом відкидання умови для числа, що буде визначати решітку, бути простим. Для тестування своєї реалізації вони використовували комп’ютер із 24 Гб оперативної пам’яті та із різними довжинами публічних ключів – 17 Мб, 69 Мб, 284Мб та 2.25 Гб. Генерація ключів зайняла 2.5 секунди, 41 секунду, 8,4 хвилини та 2,2 години відповідно. Час, затрачений на процедуру бутсрепінгу, був 6 секунд, 32 секунди, 2,8 хвилин та 31 хвилину [4].

М. ван Дейк, К. Джентрі, С. Халеві та В. Вайкунтанатан запропонували повністю гомоморфне шифрування над цілими числами, яке зазвичай називають

схемою DGHV. Нехай m - звичайний текст, який є бітом повідомлення, тобто m ∈ {0, 1}. k - секретний ключ, який є великим непарним числом. Відправник вибирає випадкові цілі числа q, r такі, що r <k / 2, де r - деякий шум. Для шифрування ми розраховуємо:

$$c = m + kq + 2r$$

Розшифровка виконується за допомогою простих модульних операцій.

$$m = (c \mod k) \mod 2$$

Тепер для двох таких текстів шифру $c_i = m_i + kq_i + 2r_i \text{ for } i = 1,2$ ми маємо такі результати:

$$c_1 + c_2 = (m_1 + m_2) + k(q_1 + q_2) + 2(r_1 + r_2)$$

$$c_1 * c_2 = (m_1 * m_2) + k(m_1 q_2 + m_2 q_1 + k q_1 q_2 + 2 q_1 r_2 + 2 r_1 q_2) + 2(m_1 r_2 + r_1 m_2 + 4 r_1 r_2)$$

З рівняння, очевидно, що шум зростає експоненціально із збільшенням числа множень, тоді як в попередній вона збільшується лінійно. Оскільки шум додається, після деякого моменту повідомлення не може бути відновлене розшифровкою. Отже, це не схема повністю гомоморфного шифрування, але її можна перетворити на таку за допомогою техніки Bootstrapping Джентрі.

Bootstrapping - це метод, який усуває шум, накопичений під час багатьох таких операцій, без дешифрування до відкритого тексту. Операція видалення шуму та наступне використання також виконується гомоморфно.

У наведеній вище схемі, оскільки один і той же ключ k використовувався як для шифрування, так і для дешифрування, це симетричний шифр ключа. Версія відкритого ключа має незначні зміни на етапі шифрування, які наведено нижче.

p - приватний ключ. Відкритий ключ - це набір багатьох цілих чисел форми $x_i = pq_i + r_i$ для $i = 1, 2, \ldots, n$, де q_i, r_i - випадкові цілі числа. Відправник вибирає випадкову підмножину S з $\{x_1, x_2, \ldots, x_n\}$. Тому зашифрований текст подається як

$$c = m + 2\sum_{i \in S} x_i + 2r$$

Розшифровка знову виконується як m = (c mod p) mod 2.

Пізніше З. Бракерскі та В. Вайкунтанатан розробили іншу схему, твердість якої базується на проблемі "Навчання з помилками" (LWE). LWE - це проблема машинного навчання, яку важко вирішити. Ця проблема є узагальненням проблеми паритетного навчання.

Серед усіх схем повністю гомоморфного шифрування - схема Бракерського-Джентрі-Вайкунтанатана (BGV) є однією з найбільш перспективних.

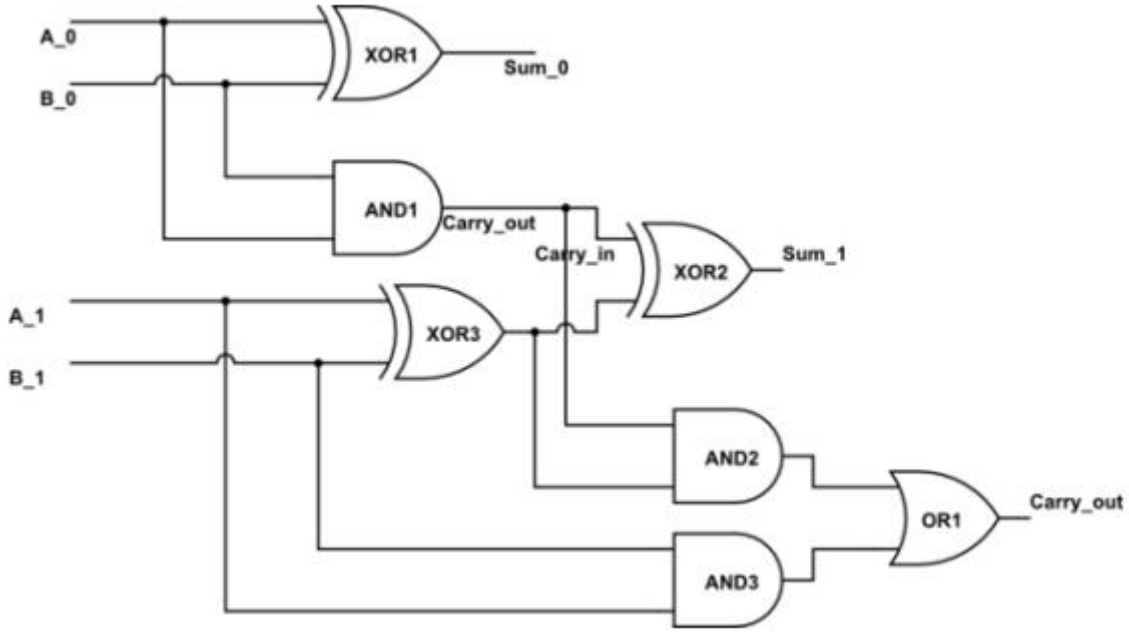

Рисунок 1.4 – Дворозрядний суматор

Оскільки ці схеми виконують побітове шифрування, у тексті шифру вони зберігають одне ціле число на звичайний текстовий біт, що призводить до збільшення вартості сховища. Щоб додати два 8-бітові числа гомоморфно, нам потрібно реалізувати всю булеву схему повного суматора на тексті шифру. Множення ще більш складні та трудомісткі.

Хончао Чжоу та Грегорі Ворнелл запропонували гомоморфну схему шифрування над цілими векторами. Безпека цієї схеми також покладається на проблему машинного навчання «Навчання з помилками» (LWE). Він підтримує обмежений набір програм, таких як вилучення функцій та агрегування даних. Основна перевага цієї схеми полягає в тому, що вона практична і може добре працювати для обмеженого набору програм.

Разом з тим у середині того ж самого 2010 року Мартеном Ван Дійком (Marten Van Dijk) була запропонована ще одна схема, що реалізує ПГШ. Вона забезпечує ПГШ оперуючи із цілими числами, а не з решітками. Головним досягненням став той факт, що ця схема стала концептуально простішою за усі попередні. Безпека параметрів системи забезпечувалась проблемою знаходження найбільшого спільного дільника для таємного та одного із публічних ключів. Не дивлячись на свою простоту, схема усе ще не була досить ефективною [5].

Друге покоління схем повністю гомоморфного шифрування є більш ефективним, оскільки ранні системи були досить повільними. Щоб схема використовувалася у великих масштабах, вона повинна бути простою у використанні. Основними вдосконаленнями попередніх схем, які дозволили запустити нове покоління схем в історії, були:

- нова конструкція алгоритму, яка допомогла застосувати операції SIMD (одна інструкція, кілька даних) із зашифрованого тексту;
- нова технологія, що називається перемиканням модулів, яка замінює процедуру повільного посилення;
- нова діаграма, яка вимагає набагато менше часу для гомоморфного обчислення.

Смарт та Веркаунтерен створили схему, що підтримує SIMD-операції (одиночний потік команд та множинний потік даних), та яка має менший ключ та розмір шифротексту за попередні схеми. Однак, процес генерації ключів у схемі був такий самий, як у Джентрі та Халеві, а отже, повільний. У їх конструкції створюється вектор відритого тексту, який складається із слотів (частин відкритого тексту) за допомогою техніки пакування[3].

На практиці, щоб отримати $c_1 \ldots c_{20,}$ що є результатом множення двадцяти пар чисел від (a_1,b_1) *до* (a_{20},b_{20}) це може бути виконано шляхом обчислення $(a_1 \cdot b_1)$, $(a_2 \cdot b_2)$ і тд. Однак, схема Смарта та Веркаунтерена дозволяє зберігати усі значення $a_1 \ldots a_{20}$ у єдиному шифротексті, скажімо b. У такому разі буде потрібно обчислити тільки $c = a \cdot b$, та після дешифрування c отримаємо доступ до усіх

елементів $c_1 \ldots c_{20}$. Таким чином, виконується єдина операція множення, яка, в дійсності, помножує декілька елементів одночасно[3].

У той самий час два інших вчених Бракерськи (Brakerski) та Вайкутанатан (Vaikuntanathan) винайшли новий алгоритм зменшення шуму у шифротексті, що базується на проблемі навчання із помилками (LWE – learning with errors). Вони вводять новий метод зменшення розмірів зашифрованого тексту та новий метод зміни модулів (modulus switching) для цього скорочення. Також зменшується і складність дешифрування [6].

Останнім проривом у цій галузі стало створення схеми BGV (Бракерскі, Джентрі, Вайкунтанатан), що дозволяє використовувати операції повністю гомоморфного шифрування у схемах з поліномами довільного розміру [7]. У ланцюзі поріг шуму лише збільшується лінійно з мультиплікативним рівнем, а не експоненціально. Це означає, що процедура завантаження більше не потрібна для реалізації схеми. Схема асиметрична і шифрує біти. Існує дві версії криптосистеми BGV: перша стосується векторів чисел і заснована на вирішенні задачі LWE [8], а друга - стосується поліномів цифр і заснована на вирішенні задачі RLWE (навчання кільце з помилками) [9]. Друга версія розглядається у роботі, детальний опис якої наведено у наступному розділі.

Загалом, після прориву Джентрі у 2009 році було створено та впроваджено більше однієї систем з використанням повністю гомоморфоного шифрування. Огляд існуючих систем, їх основних характеристик та завдань, що гарантують їх стійкість до взломів, наведено у Додатку А.

Висновки до розділу

У даному розділі розглянуто найбільш популярні методи шифрування. Одним із перспективних серед них є гомоморфне шифрування.

Проведений аналіз сучасних методів шифрування, як за способом перетворення так і за типом ключа, розглянуто симетричний та асиметричний методи шифрування, а також різновиди хеш-функцій.

Проаналізовано особливості, окреслено переваги та недоліки основних методів шифрування. Показано, що на сьогоднішній день залишається не вирішеним питання забезпечення надійної обробки, зберігання та передачі даних в інформаційно-комунікаційних системах. Одним із шляхів вирішення вказаної проблеми є використання гомоморфного шифрування.

Користувачі інформаційно-комунікаційних систем та сервісів можуть використовувати шифрування та розшифрування даних фактично без глибоких знань в області криптографії.

2 БІБЛІОТЕКА HELIB

Сьогодні у відкритому доступі існує вже декілька варіантів реалізації ПГШ, таких як бібліотека HElib, FHEW бібліотека та Paillier. Для досліджень й подальшого використання була обрана перша.

Бібліотека HElib була написана Шаєм Халеві та Віктором Шупом (Victor Shoup) на мові C++ та використовує математичну бібліотеку NTL (number theory library) [10] наряду із бібліотекою GMP (GNU multi-precision library) [11].

HElib - це бібліотека програмного забезпечення з відкритим кодом, яка реалізує гомоморфне шифрування, запущена в 2013 році. HElib реалізує як схеми BGV, так і CKKS. Її можна розглядати як мову збірки для гомоморфного шифрування, оскільки більша частина бібліотеки має відносно низький рівень, це означає, що вона забезпечує низькорівневі процедури, такі як встановлення, множення, додавання, зсув тощо. Бібліотека теорії чисел (NTL) та Бібліотека Gnu MultiPrecicion (GMP). HElib поширюється з ліцензією Apache 2.0. HElib використовує вирівняні схеми HE, що означає, що параметри повинні бути попередньо визначені користувачем. Через низькорівневий підхід до HElib, його слід розглядати здебільшого як інструмент для дослідників, але вони стверджують, що, сподіваємось, зможуть надавати функції вищого рівня в майбутньому. Спочатку HElib впроваджував лише схему BGV, перш ніж включити підтримку схеми CKKS.

Бібліотека NTL надає структури даних та алгоритми для маніпулювання цілими числами довільної довжини, а також вектори, матриці та поліноми цілих чисел. Бібліотека GMP призначена для виконання обчислень із такими числами, обчислень значень з плаваючою комою, цілих чисел та раціональних чисел з довільною точністю.

HElib використовує схему шифрування BGV. Саме ця схема вперше продемонструвала практичність використання повністю гомоморфного шифрування у реальному житті. Схема використовує деякі вдосконалення, які були додані до основної схеми Джентрі:

- техніка релінеаризації, щоб уникнути квадратичного зростання розміру зашифрованого тексту та ключів;
- техніка зміни модуля, зменшення зростання шуму після процедури множення;
- їх комбінація, що допомагає зменшити мультиплікативну глибину алгоритму дешифрування початкового завантаження;
- RLWE для забезпечення безпеки система розглядає проблему навчання з помилками в кільцях;
- підтримка принципу SIMD;
- використання зашифрованого тексту у форматі Double CRT (китайська теорема залишків), що зменшує витрати на обчислення;
- процедура обтікання зашифрованого тексту, яка за алгоритмом схожа на Double CRT і дозволяє шифрувати багато текстів в єдиний зашифрований текст.

З іншого боку, дві процедури - повторна лінеаризація та модуляризація - вимагають обміну великою кількістю ключів між власником даних та обчислювальним блоком. Крім того, перехід на підмодулі накладає нові обмеження на вибір параметрів.

Халеві та Шуп детально описали численні алгоритми у HElib [12].

2.1 Визначення понять та алгебраїчний простір системи

Кільце - це множина R, для елементів якої задано дві двійкові операції (+ і ×, тобто додавання і множення). Властивості цих операцій подібні до властивостей додавання та множення цілих чисел (комутативна, асоціативність, розподільна, існування нейтральних елементів та обернених відносно додавання). Найпростішим прикладом кільця є набір цілих чисел $Z = \{\ldots -2, -1, 0, 1, 2, \ldots\}$, тому вони замкнені відносно таких операцій: додавання та множення.

Поле - це набір, для елементів якого визначені операції додавання, віднімання, множення та ділення (крім ділення на нуль); більше того, властивості

цих операцій подібні до властивостей звичайних числових операцій над раціональними, дійсними чи комплексними числами. Найпростішим полем є поле раціональних чисел (тобто дробі). Припустимо, що K - кільце, тоді f (x) - багаточлен ступеня n у цьому кільці, який задається формулою:

$$f(x) = \sum_{i=0}^{n} a_n \cdot x^i , \qquad (2.1)$$

де $a_i \in K$ – це числа із множини К.

Тоді K[x] – це множина, що складається з усіх многочленів будь-якого кінцевого степеню у кільці К та називається кільцем многочленів.

Многочлен $\Phi_m(X) = \prod(x - \varepsilon_k)$, де ε1,…, εφ(m) – це первісні корені степеню m із одиниці, називають круговим многочленом. Наприклад, $\Phi_1(X) = X - 1,\ \Phi_2(X) = X + 1,\ \Phi_3(X) = X^2 + X + 1,\ \Phi_4(X) = X^2 + 1.$

Як зазначалося раніше, розмір відкритого тексту є дуже важливим параметром для повністю гомоморфного шифрування. Він має великий вплив на кількість обчислень та витрати пам'яті. Крім того, поки що не існує методу перемикання між двійковим та цілочисельним шифруванням. Отже, якщо схема використовує двійкові операції, повинен бути простір відкритого тексту A_p, де p = 2.

Текст у схемі представляється у вигляді векторів. Простір вхідного тексту – це кільце $A_2 = A/2 \cdot A$, іншими словами двійковий многочлен за модулем $\Phi_m(X)$, де $\Phi_m(X) = X^m + 1$, до того ж m має бути ступенем двійки.

Схема використовує техніку Smart-Vercauteren, засновану на китайській теоремі залишків (CRT), щоб «упакувати» вектор бітів у двійковий поліном. Іншими словами, якщо багаточлен $\Phi_m(X)$ розкласти за модулем 2 на ℓ мінімальні коефіцієнти $\Phi_m(X) = F_1(X) \cdot F_2(X) \cdot \ldots \cdot F_\ell(X) (\mathrm{mod}\ 2)$, тоді поліном відкритого тексту $a(X) \in A_2$ буде $a_i = a \bmod F_\ell$ - шифруванням кожного біта вхідного повідомлення. Як і у випадку з числами, додавання, множення та застосування

CRT в просторі A_2 відповідатиме операціям з векторними елементами (тобто поліномами).

Простір зашифрованого тексту на цій схемі складається з векторів $A_q = A/q \cdot A$, де q - неспарений модуль, який змінюється під час гомоморфних обчислень. Зокрема, зменшується система, параметризована ланцюжком модулів розміру $q_0 < q_1 < ... < q_{L-1}$ і перша процедура шифрування відбувається на найбільшому модулі q_{L-1}. Щоб визначити елементи рядка вибираються L + 1 малих простих чисел однакового розміру $p_0, p_1, ..., p_L$, тоді кожен новий модуль j рядка визначатиметься за формулою:

$$q_j = \prod_{j=0}^{L} p_j . \tag{2.2}$$

Ці модулі використовуються потім задля зменшення шуму у зашифрованому тексті за допомогою техніки зміни модулів. Таким чином, многочлен a – це матриця розмірністю (L+1)×φ(m), що складається з обчислень коренів a_j за модулем p_i за допомогою швидкого перетворення Фур'є (FFT – fast Furrier transformation) для i у межах від 0 до L, а φ(m) – це функція Ейлера для знаходження усіх взаємно простих з m чисел та менших за нього.

2.2 Задача ring learning with errors

У традиційній криптографії, шифротексти мають бути не розрізненими від випадкової величини. Дуже складно досягти того, щоб зашифрований текст відповідав цій умові та одночасно зберігав гомоморфні властивості.

Задача на вирішення цієї проблеми – це задача на навчання із помилками (LWE – learning with errors). Вона була запропонована Регевом (Regev) та зараз широко використовується у криптографії. Багато дослідників використовують задачу LWE при розробці криптографічних алгоритмів для того, щоб досягти високого рівня безпеки та ефективності схеми.

Задача LWE побудована на знаходженні невідомої змінної із набору лінійних рівнянь. Наприклад, умова задачі може бути наступною:

$$\begin{aligned}
&14s_1 + 15s_2 + 5s_3 + 2s_4 \approx 8 \pmod{17} \\
&13s_1 + 14s_2 + 14s_3 + 6s_4 \approx 16 \pmod{17} \\
&6s_1 + 10s_2 + 13s_3 + 1s_4 \approx 3 \pmod{17} \\
&10s_1 + 4s_2 + 12s_3 + 16s_4 \approx 12 \pmod{17} \\
&9s_1 + 5s_2 + 9s_3 + 6s_4 \approx 9 \pmod{17} \\
&\vdots \\
&6s_1 + 7s_2 + 16s_3 + 2s_4 \approx 3 \pmod{17}
\end{aligned} \tag{2.3}$$

Кожне рівняння є вірним для деякої доданої похибки (припустимо, ± 1) та ціллю є визначити s. Відповідно буде s = (0,13,9,11) [8].

Знайти s буде досить простим завданням, коли помилок не буде введено, і його можна визначити за поліноміальний час за допомогою методу Гауса. З помилкою завдання буде набагато складніше.

З часом стало необхідним модифікувати задачу LWE для виконання обчислень з поліномами, використання яких у криптосистемі забезпечує більшу надійність, ніж використання векторів чисел.

Любашевський, Пейкерт та Регєв у своїй роботі [13] запропонували варіант проблеми LWE з кільцевою роботою. Проблема називається RLWE (навчання кільцем помилок). Основна ідея RLWE полягає в тому, що вектор представляється як поліном, отриманий шляхом прийняття значень одного вектора за модулем іншого, за нередукованим многочленом з цілими коефіцієнтами (n-й круговий багаточлен, у якому n - ступінь двох).

Оскільки досліджувана система побудована на кільцях, необхідно більш детально розглянути стан задачі RLWE, яка визначається трьома параметрами:

- ціле додатне число m, яке є ступенем двох і яке визначено на схемі як параметр потужності;

- позитивним цілим числом q, що у схемі визначається як модуль;
- ймовірнісним розподілом χ над кільцем $A = Z[X]/(X^m + 1)$ та який визначається як «розподіл помилок».

Задача RLWE полягає у тому, щоб результати наступних двох розподілів ймовірностей не можна було розрізнити за допомогою обчислень.

Перший розподіл. Виберемо дві випадкових рівномірних величини s та a із кільця $A_q = A/q \cdot A$. Далі, виберемо елемент e із кільця A, що був обраний із розподілу помилок χ. Обчислимо $b = sa + e$. Результатом буде (a,b).

Другий розподіл. Виберемо дві випадкових рівномірних величини a і b із кільця $A_q = A/q \cdot A$. Результатом є (a,b).

2.3 Основні алгоритми HElib

Розглянемо основні алгоритми, на яких базується бібліотека HElib.

Генерація таємного ключа:

Щоб отримати вектор значень $\vec{t}$, який утворює таємний ключ sk необхідно вибрати із розподілу помилок χ для кожного виміру схеми n число $t = \chi^n$. Таким чином:

$$sk = \vec{s} = (1, t[1], ..., t[n]). \quad (2.4)$$

Генерація публічного ключа:

Вхідним параметром алгоритму приймається наступні величини:

- значення таємного ключа $sk = \vec{s} = (1, \vec{t})$, де $s[0] = 1$;
- значення вектору $\vec{t}$;
- значення модуля q;
- результат функції Ейлера $N = \varphi(m)$.

Генерується матриця B розміру N×n із числами із кільця A_q, а також генерується вектор $\vec{e}$, значення якого вибираються із розподілу χ^N. Обчислюється $b = B \cdot \vec{t} + 2 \cdot \vec{e}$. Генерується ще одна матриця A з кількістю стовпців (n + 1), яка

буде складатись із матриці b та наступною за нею матрицею –B (що має n стовбців). Публічний ключ буде дорівнювати pk = A.

Шифрування:

Щоб зашифрувати повідомлення m, яке дорівнює нулю або одиниці, створюється вектор, що складається з двох елементів - самого повідомлення та нуля. Довільно вибирається значення r множини χ та значення множини χ2. Тоді шифрування повідомлення розраховуються наступним чином:

$$\vec{c} := \vec{m} + 2 \cdot \vec{e} + A^T \cdot r . \quad (2.5)$$

Звідси випливає, що вектор $\vec{c}$ буде складатись із суми векторів $\vec{m}$, малого парного вектору $2\vec{e}$ та малого елементу кільця r, помноженого на зашифровані в єдине число нулі з матриці публічного ключа (A^T). Останній добуток і є умовою задачі RLWE.

Ключовою характеристикою схеми BGV є те, що поточний секретний ключ і модуль змінюються, коли операції повторного шифрування застосовуються до зашифрованого тексту. Під час гомоморфного обчислення схема застосовує п’ять різних операцій у зашифрованому тексті. Три з них - це додавання, множення та автоморфізм - семантичні операції, що використовуються для перетворення зашифрованого тексту. Дві інші операції, зміна ключа та зміна модуля, дозволяють змінити складність розрахунку. Документ розглядає лише чотири: додавання, множення, зміна ключа та зміна модуля.

Додавання.

Гомоморфне додавання векторів двох текстів шифру, пов’язаних з одним секретним ключем і одним і тим же модулем, здійснюється простим додаванням векторів із простору A_q. Якщо многочлени відкритого тексту були зашифровані як $a_1, a_2 \in A_2$, тоді їх сума буде шифруванням $a_1 + a_2 \in A_2$.

Множення.

Являється фактом, що коли два аргументи мають розмірність n у просторі A_q, тоді добуток зашифрованого тексту матиме розмірність n^2. Використовуючи поліноміальну арифметику, це значення розмірності замість цього можна збільшити лише $2n-1$ разів. Операція множення двох текстів шифру не змінює поточний модуль, а змінює поточний ключ: якщо два тексти вхідного шифру, які перемножуються, були зашифровані однаковим вектором секретного ключа s розміром n, і вони шифрувати поліноми відкритого тексту, тоді після завершення операції розмір ключа буде n^2, а сам ключ буде помножений на себе. І тоді зашифрований добуток багаточленів буде виглядати так - $a_1 \cdot a_2 \in A_2$.

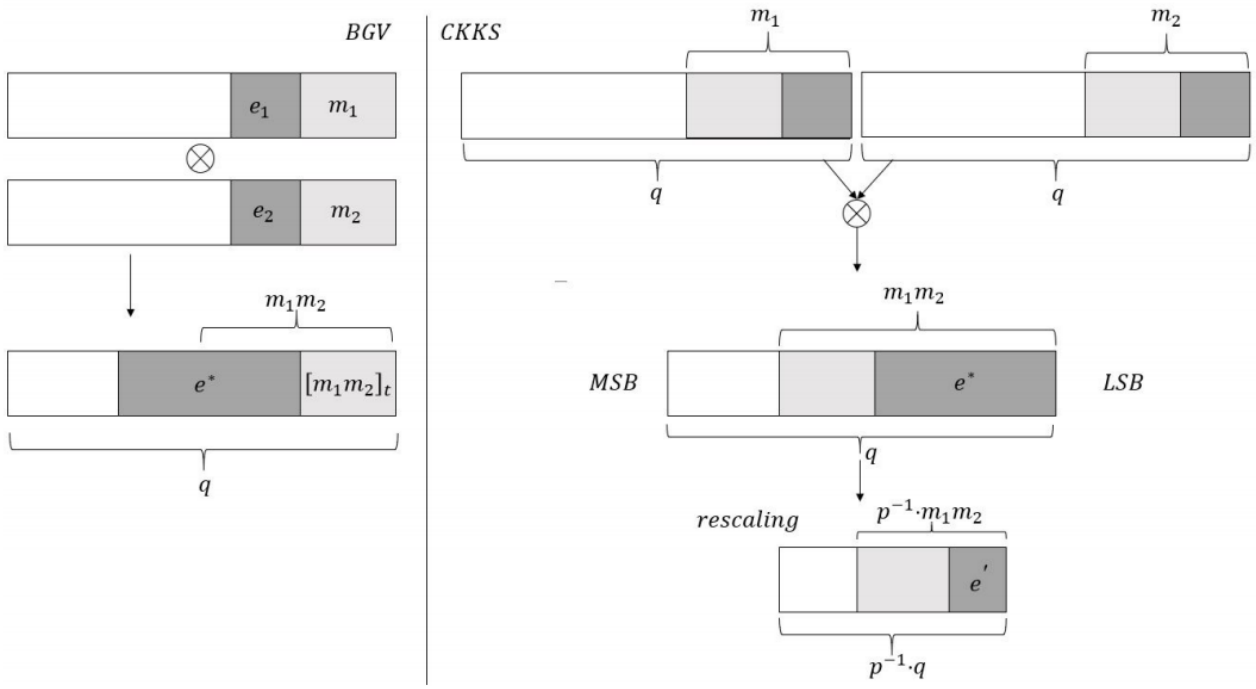

Рисунок 2.1 – Множення та масштабування BGV та CKKS

Зміна ключа.

Для того, щоб забезпечити можливість перетворення зашифрованого тексту на одному ключі в зашифрований текст, який посилається на одне і те ж вхідне повідомлення, але на інший ключ, схема BGV містить додаткові компоненти. Наприклад, це може знадобитися в ситуації з множенням зашифрованого тексту, яка була описана вище. Отже, відкритий ключ для шифрування на конкретному модулі q генерується відповідно до секретного ключа, і при його множенні

значення цього секретного ключа збільшиться, що збільшить значення шуму, яке потім буде додано до зашифрований текст. Отже, розмір секретного ключа після множення двох текстів шифру слід зменшити і отримати добуток цих текстів шифру на інший ключ.

Для того, щоб перетворити ключ розмірності s в ключ s розмірності n (маючи на увазі той самий модуль q), діаграма додає матрицю W з числами з простору A_q, в якому кожен стовпець є шифруванням відповідного відносного ключа s' до ключа s і до модуля q. Крім того, зашифрований текст c', отриманий відносно ключа s', використовується при обчисленні нового зашифрованого тексту, який буде відповідати ключу s наступним чином:

$$c = W \cdot c'. \tag{2.6}$$

Зміна модуля.

Операція зміни модуля необхідна для того, щоб зменшити рівень шуму. Припустимо, що модуль q дорівнює x^{λ}, де λ – це параметр безпеки, а x – значення шуму. Так як система має ланцюг модулів $q_0 < q_1 < \ldots < q_{L-1}$, то для $i < \lambda$ кожний окремий модуль буде знаходитись наступним чином:

$$q_i = q / x^i. \tag{2.7}$$

Якщо перемножити два шифротексти, що відносяться до того ж самого модуля, то значення шуму x зростає до x^2. Але після зміни модуля на менший за нього, рівень шуму знову стане x. Таким чином, кожна операція зміни модуля зменшує рівень шуму в x разів. Отже, для того, щоб шифротекст c, що відноситься до модулю q та ключа s конвертувати у шифротекст c', який шифрує те саме повідомлення відносного такого ж ключа s, але іншого модулю q', шифротекст c просто масштабується на результат від ділення q'/q та подальшого його округлення, щоб отримати шифротекст у вигляді цілого числа.

Ця операція залишає таємний ключ s, відносно якого були отримані шифротексти, незмінним, у той час як модуль змінюється з q на q' та рівень шуму також зменшується. Виходячи з того, що криптосистема має ланцюг модулів, то кожен раз, як рівень шуму буде занадто великим, модуль шифрування буде змінюватись з q_i на q_{i-1}, щоб зменшити цей рівень. Як тільки модуль стане q_0, то далі буде неможливо використовувати цю техніку і у такому разі зі зростанням шуму буде застосовуватись процедура бутстрепінгу.

Для того, щоб отримати повністю гомоморфну властивість, схеми повинні мати можливість зменшувати компонент шуму, який утворюється при кожному додаванні та множенні. Бутстрепінг - це метод, при якому схема запускає свою схему дешифрування, щоб усунути поточний шум, одночасно додаючи новий шум із самої схеми дешифрування. Щоб цей шум був повністю гомоморфним, новий шум повинен бути меншим за шум, який видаляється, щоб зберегти повністю гомоморфні властивості. Джентрі запропонував свою техніку завантаження як рішення цієї проблеми. Беручи алгоритм дешифрування схеми шифрування та перетворюючи його на схему із зашифрованим текстом та шифруванням приватного ключа як вхід, результатом схеми буде повторне шифрування зашифрованого тексту. Якщо цей ланцюг дешифрування досить дешевий для оцінки, тоді наш новий зашифрований текст матиме менше шуму, ніж оригінальний. Отже, дозволяючи робити більше операцій над зашифрованими текстами, надаючи нам повністю гомоморфну властивість. Тоді можна оцінити будь-яку схему, виконуючи процедуру завантаження, коли це необхідно при обчисленні.

2.4 Структура бібліотеки HElib

HElib можна розділити на чотири рівні. Перші два рівні можна кваліфікувати як "математичні", а два інших - криптографічні. Структура бібліотеки наведена на рисунку 2.2.

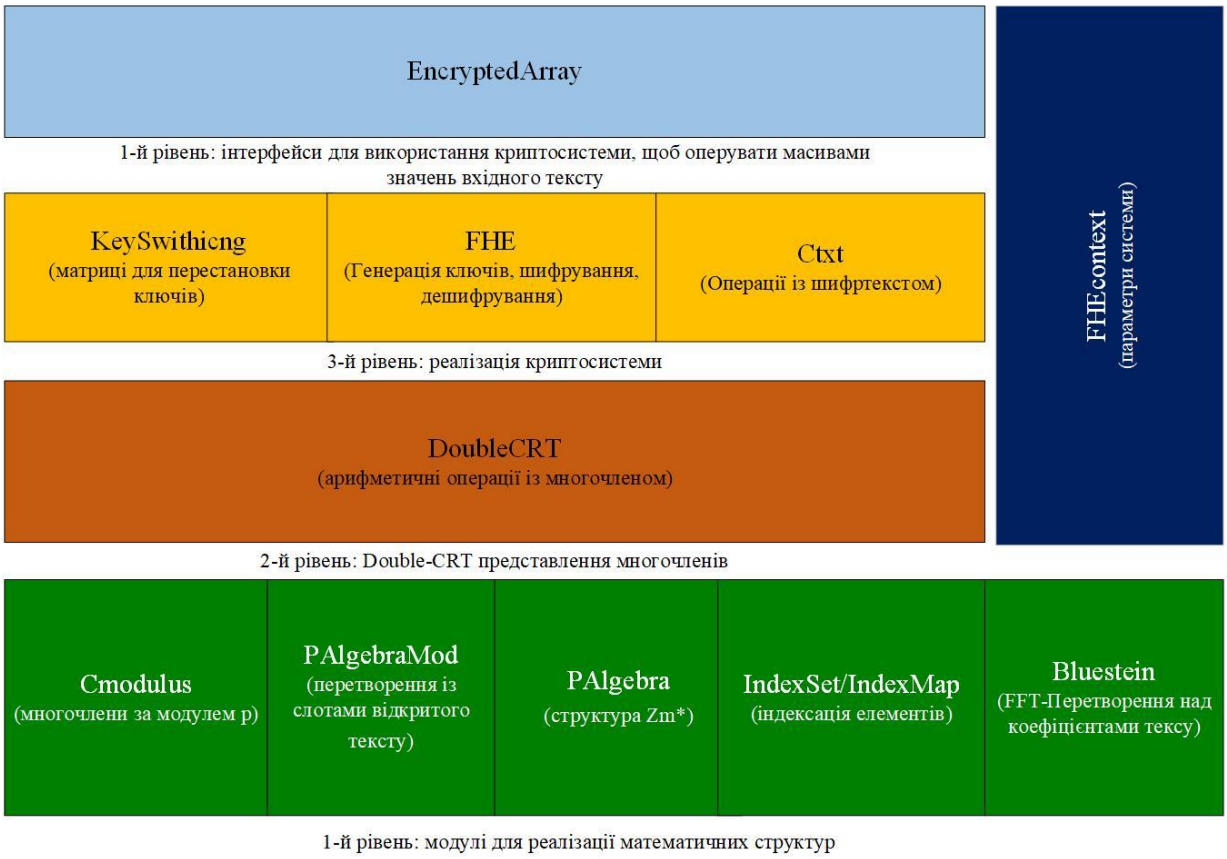

Рисунок 2.2 – структура бібліотеки HElib

Перший рівень реалізує математичні структури, другий рівень реалізує процедуру DoubleCRT для поліномів, третій рівень реалізує саму криптосистему (з двійковим поліномом як вхідним текстом), а четвертий рівень забезпечує інтерфейси для використання криптосистеми.

Bluestein модуль реалізує FFT із ступенем, відмінним від двійки над простими числами у полі Z_p із використанням алгоритму Блюштайна. Використано многочлени, що були отримані за модулем p, щоб закодувати дані, які виходять із функції, де вони обробляються за допомогою FFT. Зокрема, даний модуль побудований на бібліотеці Шаупа NTL [10] та працює як з великими цілими числами, так і з малими.

Класи Cmodulus та CModulus.

Ці класи є рівнем зображення тих даних, які були отримані за допомогою FFT по відношенню до кожного простого числа (Cmodulus для малих простих чисел, а CModulus – для великих). Вони зберігають сам вектор коефіцієнтів,

корінь многочлену, його ступінь, результат перетворення FFT із зворотними ступенями кореню та іншу додаткову інформацію.

Клас PAlgebra – це базовий клас, що реалізує та вміщує у собі структуру Z_m^* для вхідного тексту. Елемент класу визначається двома параметрами – m та p.

Клас PAlgebraMod відповідає за факторизацію шифротексту наступним чином:

$$\Phi_m(X) = \prod_{j=0}^{L} F_j(X) (\mathrm{mod}\ p). \tag{2.8}$$

Крім того, зашифрований текст "загортається" в поліноми, які є інтервалами вхідного тексту. У класі також реалізований алгоритм зворотного перетворення - від розташування вхідного тексту до векторів. Крім того, об'єкт PAlgebraMod зберігає так звані "таблиці масок", які полегшують переміщення слотів із зашифрованими даними (їх потім використовуватиме клас EncryptedArray).

У цій реалізації системи всі поліноми представлені як DoubleCRT, як зазначалося раніше, і вони посилаються на один з підмножин малих простих чисел у системі. Ця сама підмножина може змінюватися під час обчислень, тому може статися так, що один і той же поліном у певний момент посилається на різну кількість простих чисел (наприклад, як в алгоритмі зміни ключа). Щоб забезпечити стійкість до таких перетворень, бібліотека реалізує класи IndexSet, що містять довільну підмножину НЕ-негативних чисел та IndexMap, що містить набір елементів даних, індексованих цією довільною тирсою.

Об’єкти більш високих рівнів бібліотеки так чи інакше відносяться до параметрів, що були визначені на нижчих рівнях, такі як число m (що визначає групи Z_m^* та $Z_m^* / \langle p \rangle$, а також кільце $A = Z[X] / \Phi_m(X))$ та послідовність малих простих чисел, яка визначає ланцюг модулів. Щоб забезпечити зручність доступу до цих параметрів, є клас FHEcontext, який вміщує усі ці параметри.

Перше, що включає FHEcontext – це вектор об’єктів Cmodulus, які вміщують малі прості числа, які визначають ланцюг модулів. Клас також включає різноманітні алгебраїчні структури для арифметичних операцій із вхідним текстом, зокрема:

- PAlgebra zMstar – структура $Z_m^* / \langle p \rangle$;
- PAlgebraMod alMod – структура $Z[X]/(\Phi_m(X), p)$.

У класі FHEcontext міститься таблиця вже попередньо-обчислених значень, що відповідають конкретному m (параметр, що визначає m-ий круговий многочлен). Якщо параметр m не було задано, то йому присвоюється значення 0 і він обирається за таблицею, наведеною на рисунку Б.1 у додатку Б виходячи з інших параметрів.

Якщо значення було задано, то тоді воно перевіряється на вірність за іншим алгоритмом, який наведено на рисунку 2.3.

```
// If m is not set yet, just set it close to N. This may be a lousy
// choice of m for this p, since you will get a small number of slots.

if (m==0) {
  // search only for odd values of m, to make phi(m) a little closer to m
  for (long candidate=N|1; candidate<10*N; candidate+=2) {
    if (GCD(p,candidate)!=1) continue;

    long ordP = multOrd(p,candidate); // the multiplicative order of p mod m
    if (d>1 && ordP%d!=0 ) continue;
    if (ordP > 100) continue;  // order too big, we will get very few slots

    long n = phi_N(candidate); // compute phi(m)
    if (n < N) continue;       // phi(m) too small

    m = candidate;  // all tests passed, return this value of m
    break;
  }
}
```

Рисунок 2.3 – Лістинг коду із алгоритмом знаходження m

Якщо значення не задовольняє умовам системи, то її подальше функціонування не можливе.

У функції, що перевіряє m на правильність можна задати інше значення для існуючих параметрів. Розглянемо ці параметри:

- k – це параметр безпеки;
- L – це кількість простих чисел шифротексту, які користувач хоче підтримувати;
- c – кількість стовбців у матриці, яка використовується для зміни ключів;
- p, d – параметри, що визначають простір відкритого тексту A_p^d;
- s – межа, числа нижче якої – це кількість слотів шифротексту, які можливо обробляти;
- chosen_m – параметр, яким задається конкретне значення m, та яке потім перевіряється на задовільність усім умовам.

Модуль DoubleCRT – це основа бібліотеки (має приставку «double», з англійської «подвійний», тому що може оперувати із шифротекстами як у двійковому представлені, так і у поліноміальному). Клас представляє многочлен $a \in A_q$ як матрицю з кількістю стовбців φ(m) та одним рядком для кожного малого простого числа p_i. Кожен і-ий рядок містить FFT представлення a за модулем p_i.

У класі реалізовані функції для перетворення многочленів із коефіцієнтів у формат DoubleCRT та назад. Клас підтримує множину арифметичних операцій, таких як додавання, множення і т. п. Вони завжди виконуються у просторі A_q для деякого модуля q. Арифметичні операції можуть бути застосовані тільки до об’єктів у представленні DoubleCRT, що відносяться до конкретних екземплярів FHEcontext. Усе інше спричинить помилку.

На третьому рівні бібліотеки реалізована сама гомоморфна криптосистема BGV. На цьому рівні є три головних модуля:

- Ctxt – реалізує виконання арифметичних операцій із шифротекстами;
- FHE – модуль, у якому генеруються таємні та публічні ключі;
- KeySwitching – допоміжний модуль, у якому визначається, яку саме матрицю для зміни ключів генерувати.

На четвертому рівні бібліотеки надаються декілька інтерфейсів, які дозволяють шифрувати многочлени вхідного тексту та гомоморфно їх обробляти. Вхідний текст трансформується у многочлени шляхом використання технік кодування/декодування, які реалізовані у модулі PAlgebraMod.

При ініціалізації бібліотеки HElib нам спочатку потрібно встановити згадані раніше визначені користувачем параметри. Тоді потрібно виконати дії зі списку:

- FHEcontext ініціалізується за допомогою визначених користувачем параметрів, надаючи екземпляру HElib зручний доступ до цих параметрів за допомогою методів доступу та деяких функцій утиліти.
- Ланцюг модулів будується за допомогою buildModChain (context, bits, c) з параметрами *bits* і *c* для суми бітів і стовпців у матрицях перемикання ключів.
- Секретний ключ генерується з відповідним контекстом.
- Обчислюються необхідні матриці перемикання ключів на основі секретного ключа.
- Тоді генерується відкритий ключ із секретного ключа.
- Потім створюється EncryptedArray нашого об’єкту контексту.
- Об'єкт зашифрованого тексту створюється на основі відкритого ключа.
- Нарешті, використовується EncryptedArray та відкритий ключ для шифрування відкритого тексту в об’єкт зашифрованого тексту.

EncryptedArray - це клас, що використовується для зберігання відкритих текстів. Це надає нам методи для обробки об’єктів зашифрованого тексту як звичайних одновимірних масивів.

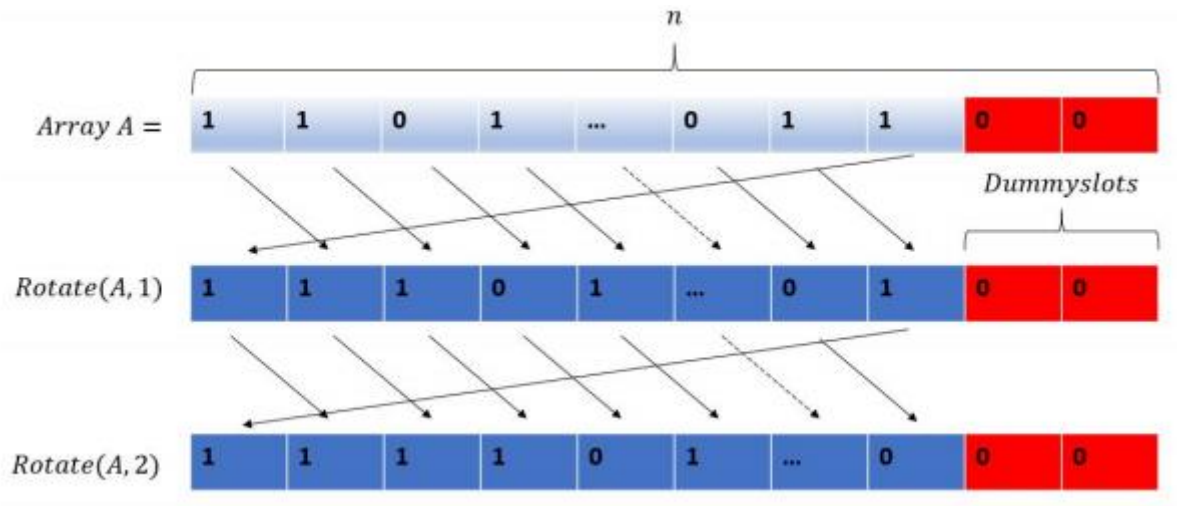

Рисунок 2.4 – Робота функції повороту.

Висновки до розділу

Розглянуто основні математичні поняття, на яких базується схема гомоморфного шифрування, що використовується у бібліотеці HElib.

Окреслена структура даної бібліотеки та розглянуто основні процедури, які використовуються в ній.

Проведено методологічний аналіз існуючих основних алгоритмів, на яких базується бібліотека HElib: генерація таємного ключа, генерація публічного ключа, шифрування, додавання, множення, зміна ключа, зміна модуля.

Сформульовано задачі та особливості визначення основних параметрів для зручного використання алгоритму гомоморфного шифрування даних на основі бібліотеки HElib для вебдодатків.

3. ЗАПРОПОНОВАНИЙ СПОСІБ ВИКОРИСТАННЯ ГОМОМОРФОНОГО ШИФРОВАННЯ У ВЕБДОДАТКУ

3.1. Вибір середовища

При виборі середовища розробки програмного забезпечення важливою є наявність можливості зручної зміни графічного інтерфейсу та якомога більша кількість реалізованих бібліотек функцій, які у свою чергу забезпечують ефективну реалізацію необхідного функціоналу. Вибір середовища й мови програмування дозволив значно скоротити час розробки додатку. Обов'язковою вимогою була можливість інтегрувати функціонал бібліотеки HElib у середовище.

Вибір технології визначається виходячи з потреб та необхідних можливостей, таких як:

- Тип проекту: бізнес-додаток, гра, платіжне програмне забезпечення
- Тип продукту: динамічний месенджер або платформа аналізу даних
- Географія застосування: місцева, загальнодержавна чи в усьому світі
- Бюджет

Python - мова програмування виникла на початку 90-х років і досі є однією з найбільш інноваційних, гнучких та універсальних технологій завдяки бібліотекам, що постійно розвиваються, мають досконалу документації та передові реалізації. Наприклад, Python - це мова переходу до науки про дані, машинного навчання та проектів ШІ [16].

Переваги Python:

- скорочує час виведення продукту на ринок;
- має простий синтаксис;
- має широкий спектр засобів розробки та фреймворків.

Недоліки Python:

- однопоточна мова програмування;
- слабка у мобільних обчисленнях [16].

3.2. Конфігурація сервера

Для роботи, перед тим як писати реалізацію запропонованого способу, необхідно вибрати віртуальну машину (VM), на якій розміщуємо сервер Apache. VM - це ПК з Linux, з підключенням до мережі Інтернет, яка була відокремлена від зовнішнього світу. Щоб отримати доступ до цієї мережі, спочатку потрібно підключити ПК до шлюзу, а потім до потрібного мережевого пристрою, як показано на рисунок 3.1.

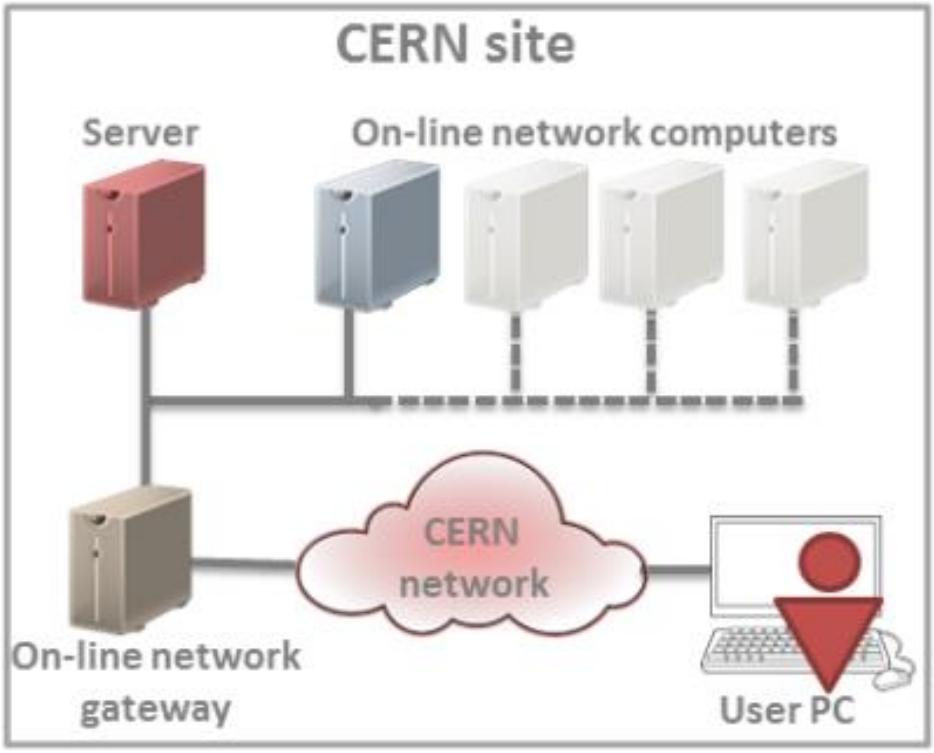

Рисунок 3.1 – Конфігурація сервера

3.2.1 Підготовка VM

У проекті використовується стаціонарний ПК з операційною системою Windows 10 для підключення до віртуальної машини за допомогою програми X-Win32 2012. Ця програма була розроблена для ОС Windows для відображення віддалених робочих столів Linux та окремих програм через підключення до локальної мережі. Покрокова блок-схема підготовки VM показана на рисунок 3.2.

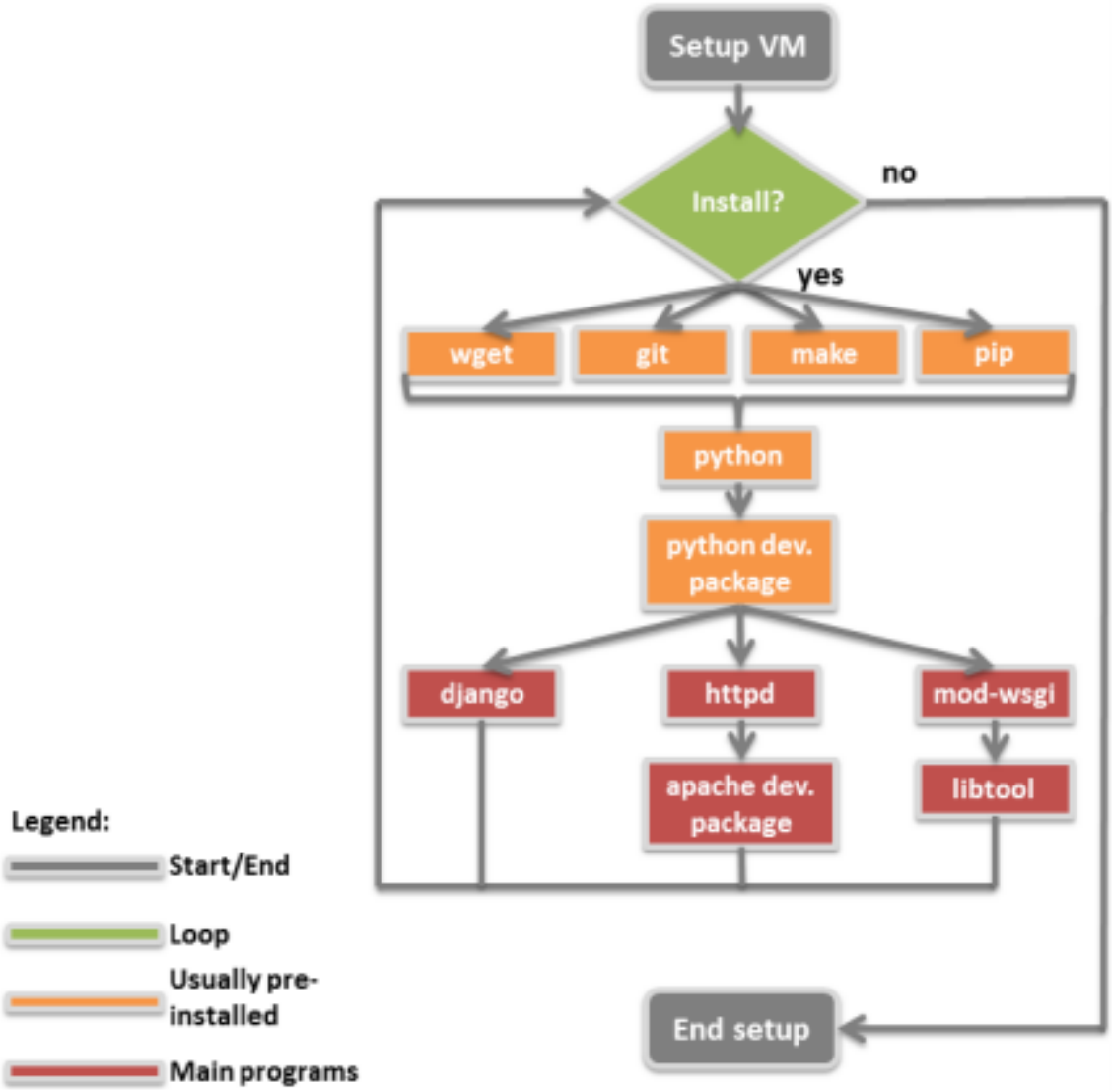

Рисунок 3.2 – Схема підготовки VM.

Помаранчева зона блок-схеми представляє компоненти, які зазвичай попередньо інстальовані на більшості дистрибутивів Linux, таких як make, wget та Python. Червона зона показує основні програми в проекті, які потрібно було встановити, такі як Django Web Framework, mod-wsgi та httpd-сервер.

3.3. Опис ключових класів, методів та властивостей вебдодатку

Робота програма організована через взаємодію з об’єктами.

Вебдодаток складається з трьох модулів, що наведені на рисунок 3.3.

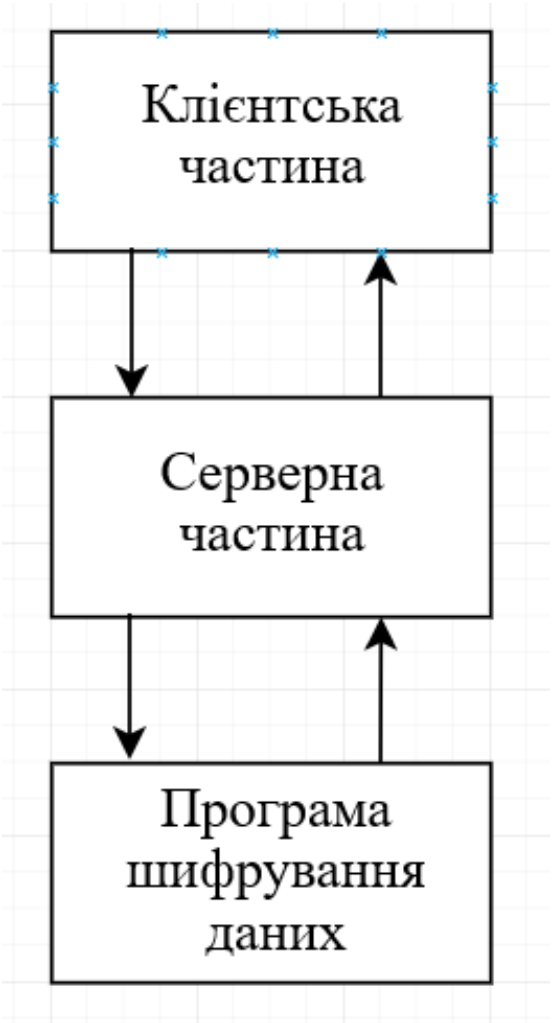

Рисунок 3.3 – Загальна структура вебдодатку.

3.3.1. Серверна частина.

Серверна частина поставляється у вигляді системи, яка складається з декількох контролерів, реалізованих за допомогою фреймворка Django. Вони призначені для обробки запитів з боку клієнта системи. Всі дані, оброблені на стороні сервера, зберігаються в базі даних, і для цього проекту використано спрощену версію SQlite, оскільки її досить для проведення порівняльного аналізу системи. Якщо користувач вводить неправильні дані, сервер під час обробки запиту знаходить їх і повертає повідомлення з помилкою, яку було допущено й варіантами її ліквідації .

Одним з найважливіших сценаріїв, який контролюється головним сервером, є процес входу. Головний сервер взаємодіє з кожним шаром серверної архітектури при вході в систему.

Для керування діями користувачів у роботі створено наступні класи:

Клас Register(generics.GenericAPIView):

Створює обліковий запис користувача й зберігає його в базі даних для подальшого надання користувачеві послуг вебдодатком. Від користувача необхідно ввести інформацію про себе, таку як:

- логін;
- пароль;
- ім’я;
- дата народження ;
- номер телефону.

Отримавши дані сервер перевіряє, чи наявний такий користувач в системі, якщо ні то повертає результат успішної реєстрації.

Повертає:

- файл кукі-сесії та код стану 201 при успішній реєстрації;
- текст помилки та код стану 4xx при відмові.

Параметри:

- permission_classes – визначає рівень відкритого доступу до методів, у даному випадку дозволено всім(AllowAny);
- serializer_class – необхідний для створення об’єкту нового користувача системи;
- allowed_methods – дозволені методи звернення користувачем до серверу. У даному випадку: 'POST', 'OPTIONS', 'HEAD'.

Методи:

- post_register – автоматичний перехід при успішній реєстрації;
- get_response – отримання файлу кукі-сесії при реєстрації;
- get_error_response – отримання помилки при відмові реєстрації;
- post – обробка запиту користувача.

Код даного класу наведено на рисунок 3.4.

```
class Register(generics.GenericAPIView):
    permission_classes = (AllowAny,)
    serializer_class = UserRegisterSerializer
    allowed_methods = ('POST', 'OPTIONS', 'HEAD')

    def post_register(self):
        auth_login(self.request, self.user)

    def get_response(self):
        token = Token.objects.create(user=self.user)
        return Response({"key": token.key}, status=status.HTTP_201_CREATED)

    def get_error_response(self):
        return Response(self.serializer.errors, status=status.HTTP_400_BAD_REQ

    def post(self, request, *args, **kwargs):
        self.serializer = self.get_serializer(data=self.request.data)
        if not self.serializer.is_valid():
            return self.get_error_response()

        self.user = self.serializer.save()
        self.post_register()
        return self.get_response()
```

Рисунок 3.4 – Клас Register.

Клас Login(generics.GenericAPIView):

Перевіряє облікові дані та повертає заголовок файлу кукі-сесії, якщо облікові дані є дійсними та автентифікованим. Викликає метод авторизації Django Auth.

Приймає:

- ім’я користувача;
- пароль.

Повертає:

- ключ-сесії та код стану 200 при успішній авторизації;
- текст помилки та код стану 4xx при відмові.

Параметри:

- permission_classes – визначає рівень відкритого доступу до методів, у даному випадку дозволено всім(AllowAny);
- serializer_class – необхідний для створення об’єкту нового користувача системи;
- allowed_methods – дозволені методи звернення користувачем до серверу. У даному випадку: 'POST', 'OPTIONS', 'HEAD'.

Методи:

- login – перевірка користувача та його рівень доступу;
- get_response – отримання ключа-сесії при успішній авторизації;
- get_error_response – отримання помилки при відмові авторизації;
- post – обробка запиту користувача.

Код даного класу наведено на рисунок 3.5.

```
class Login(generics.GenericAPIView):
    permission_classes = (AllowAny,)
    serializer_class = UserLoginSerializer
    allowed_methods = ('POST', 'OPTIONS', 'HEAD')

    def login(self):
        self.user = self.serializer.user
        auth_login(self.request, self.user)

    def get_response(self):
        token, _ = Token.objects.get_or_create(user=self.user)
        return Response({"key": token.key}, status=status.HTTP_200_OK)

    def get_error_response(self):
        return Response(self.serializer.errors, status=status.HTTP_400_BAD_REQUEST)

    def post(self, request, *args, **kwargs):
        self.serializer = self.get_serializer(data=self.request.data)
        if not self.serializer.is_valid():
            return self.get_error_response()
        self.login()
        return self.get_response()
```

Рисунок 3.5 – Клас Login.

Клас Profile(models.Model):

Перевіряє облікові дані користувача збережені в базі даних.

Приймає:

- нову модель користувача

Повертає:

- код стану 200 при успішній зміні даних користувача
- текст помилки та код стану 4xx при відмові

Код даного класу наведено на рисунок 3.6.

```
class Profile(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    image =models.ImageField(default='default.jpg', upload_to='profile_pics')
    def __str__(self):
        return f'{self.user.username} Profile'

    def save(self):
        super().save()

        img = Image.open(self.image.path)

        if img.height > 300 or img.width > 300:
            output_size = (300, 300)
            img.thumbnail(output_size)
            img.save(self.image.path)
```

Рисунок 3.6 – Клас Profile.

Для отримання повідомлень сервер використовує 2 протоколи зв’язку:

- HTTP;
- WebSocket.

Протокол HTTP є односпрямованим, коли клієнт надсилає запит, а сервер - відповідь. Користувач відправляє запит на сервер, цей запит надходить у формі HTTP або HTTPS, після отримання сервера надішле відповідь клієнту, після відправки відповіді з'єднання закривається, кожен запит HTTP або HTTPS

встановлює нове з'єднання з сервером кожного разу, і після отримання відповіді з'єднання розривається само собою.

HTTP - це протокол без стану, який працює на вершині TCP, що є протоколом, орієнтованим на підключення, який гарантує доставку передачі пакетів даних за допомогою тристоронніх методів рукостискання та повторної передачі втрачених пакетів.

HTTP може працювати поверх будь-якого надійного протоколу, орієнтованого на підключення, такого як TCP, SCTP. Коли клієнт відправляє HTTP-запит на сервер, між клієнтом і сервером відкривається TCP-з'єднання, і після отримання відповіді TCP-з'єднання розривається, кожен HTTP-запит відкриває окреме TCP-з'єднання з сервером, наприклад, якщо клієнт надішле 10 запитів на сервер, буде відкрито 10 окремих з'єднань HTTP. які закриваються після отримання відповіді / резервної копії.

Інформація про повідомлення HTTP, кодована в ASCII, кожне повідомлення HTTP-запиту складається з версії протоколу HTTP (HTTP / 1.1, HTTP / 2), методів HTTP (GET / POST тощо), заголовків HTTP (тип вмісту, довжина вмісту), інформація про хост і тіло, яке містить фактичне повідомлення, яке передається на сервер. Розмір заголовків HTTP варіювався від 200 байт до 2 КБ, загальний розмір заголовка HTTP становить 700-800 байт. Коли веб-програма використовує більше файлів cookie та інших інструментів на стороні клієнта, які витрачають функції зберігання агента, це зменшує корисне навантаження заголовка HTTP.

WebSocket є двонаправленим, повнодуплексним протоколом, який використовується в тому ж сценарії зв'язку клієнт-сервер, на відміну від HTTP, він починається з ws: // або wss: //. Це протокол з підтримкою стану, що означає, що зв’язок між клієнтом та сервером буде продовжувати існувати, доки його не припинить одна із сторін (клієнт чи сервер). Після закриття з'єднання клієнтом чи сервером з'єднання розривається з обох кінців. Метод, що реалізує розрив з’єднання наведено на рисунку 3.8.

Після встановлення зв'язку - обмін повідомленнями відбуватиметься у двонаправленому режимі, поки зв’язок між клієнтом-сервером не розірветься. Метод, що реалізує обмін повідомленнями наведено на рисунку 3.9.

Принцип роботи WebSocket дещо відрізняється від того, як працює HTTP, код стану 101 позначає протокол перемикання.

Вебдодаток в режимі реального часу використовує WebSocket для відображення даних на стороні клієнта, які постійно надсилаються сервером. У WebSocket дані безперервно передаються в те саме з'єднання, яке вже відкрите, тому він швидший і покращує продуктивність програми.

Програма чату: програма чату використовує WebSocket для встановлення з'єднання лише один раз для обміну, публікації та трансляції повідомлення серед абонентів. Він повторно використовує одне і те ж з'єднання WebSocket для надсилання та отримання повідомлення та передачі один до одного повідомлень.

Методи для встановлення WebSocket з’єднання клієнта з сервером наведено на рисунку 3.7. Необхідні для того, щоб клієнт міг оновлювати дані, які були змінені на сервері, без додаткового оновлення всієї сторінки й створення нових з’єднань з сервером.

```
def ws_connect(message):
    """
    Websocket onconnect channel
    """
    room = find_room(message['path'])
    if not room:
        return
    log.debug('chat connect room=%s client=%s:%s',
        room, message['client'][0], message['client'][1])
    Group('chat-' + room, channel_layer=message.channel_layer).add(message.reply_channel)
```

Рисунок 3.7 – Створення WebSocket з’єднання.

```
def ws_disconnect(message):
    """
    Websocket onclose channel
    """
    room = find_room(message['path'])
    if not room:
        return

    Group('chat-' + room, channel_layer=message.channel_layer).discard(message.reply_channel)
```

Рисунок 3.8 – Розрив WebSocket з’єднання.

```
def ws_receive(message):
    """
    Websocket Receive channel
    """
    room = find_room(message['path'])
    if not room:
        return

    # Parse out a chat message from the content text, bailing if it doesn't
    # conform to the expected message format.
    import json
    try:
        data = json.loads(message['text'])
    except ValueError:
        log.debug("ws message isn't json text=%s", text)
        return

    if not data:
        return

    # Save Chat to db, ideally perform this in another channel
    data['room'] = room
    log.debug('chat message room=%s username=%s message=%s',
        room, data['user'], data['content'])
    serializer = ChatAddSerializer(data=data)
    if serializer.is_valid():
        serializer.save()
        log.debug('Saved!')
    else:
        log.debug("ws message unexpected format data=%s", serializer.errors)
        return

    # Relay chat message to users in this room
    Group('chat-'+room, channel_layer=message.channel_layer).send(
        {"text": json.dumps(serializer.data)}
    )
```

Рисунок 3.9 – Обмін повідомленнями через WebSocket з’єднання.

Нижче перелічено помітні зміни в бібліотеці HElib для реалізації схеми SS.

DoubleCRT :: randomizeZeroOne () відбирає кільцевий елемент у формі оцінки таким чином, що кожна оцінка дорівнює 0 або 1 з імовірністю 0,5.

FHESecKey :: GenSecKeyZeroOne () генерує секретний елемент кільцевого ключа, використовуючи

DoubleCRT :: randomizeZeroOne. Ключ імпортовано за допомогою

FHESecKey :: ImportSecKey. Vanilla HElib зберігає вагу хеммінга секретного ключа у вигляді коефіцієнта разом із своїм відкритим ключем, щоб полегшити обчислення шуму зашифрованого тексту, зашифрованого за допомогою відкритого ключа. Секретні ключі в SS все ще мають очевидну вагу хеммінга, але у формі оцінки замість форми коефіцієнта. З додатку A.5 [GHS12] спостерігається, що отримана дисперсія шуму від ключа гаммінгів h у формі коефіцієнта є такою ж, як вага гаммінга h у формі оцінки. Основною перевагою генерації ключа SS є те, що ключ генерується у формі оцінки, і його не потрібно перетворювати з форми коефіцієнта для виконання ефективних обчислень. Vanilla HElib спочатку відбирає ключі із заданою вагою хеммінга у формі коефіцієнта, а потім охоплює їх у представлення DoubleCRT.

Ctxt :: multiplyByWithSquareKey (Ctxt c1, Ctxt c2) обчислює добуток c1 і c2 за умови, що базовий секретний ключ відбирається, як у KeyGenSS. Результат зберігається в ньому. Залежно від прапора компілятора, процедура множення Vanilla HElib замінюється цим методом, так що всі способи множення автоматично перетворюються на EvalMultSS.

3.4. Опис залежності алгоритму від параметрів та їх аналіз

У бібліотеці наступні параметри є основними:

- m, p, r – параметри, що визначають простір відкритого тексту $Z[X]/(\Phi_m(X), p^r)$;
- L – кількість рівнів, або кількість простих чисел у шифртексті;
- c – кількість стовпців у матриці для зміни ключа.

Спочатку бібліотеки GMP та NTL повинні бути встановлені. Обидві бібліотеки працюють краще на платформах, заснованих на Unix, тому віртуальна скринька встановлюється для Ubuntu в системах Windows. Бібліотеки не стискаються і встановлюються одна за одною. У терміналі каталог слід змінити на gmp. Після зміни каталогу потрібно налаштувати бібліотеку за допомогою команди "./configure". Потім команда make використовується для створення файлів у бібліотеці та остаточного встановлення за допомогою команди "sudo make install". Ці кроки встановлюють бібліотеку в локальну папку користувачів, тобто / usr / local.

Бібліотека NTL також встановлюється шляхом стиснення завантаженого файлу zip та зміни каталогу в терміналі на джерело бібліотеки нестисненого файлу. Для налаштування бібліотеки використовується команда "./configure NTL_GMP_LIP = on", за якою слідує команда make і команда sudo make install. Це встановить бібліотеку NTL у папку / usr / local. Тепер HElib можна побудувати, змінивши каталог на src (джерело) HElib, а потім за допомогою команд make і make check, де "make" компілює, та будує бібліотеку "fhe.a", а "make check" компілює та запускає набір тестових програм. Бібліотека HElib містить три рівні, а саме математичний, криптографічний та шар переміщення даних. Вона використовує бібліотеку NTL для математичних операцій. Основними операціями, передбаченими в цій схемі, є генерація ключів, шифрування, дешифрування та кілька гомоморфних процедур шифрування, таких як додавання, множення, "обслуговування зашифрованого тексту" тощо. У файлі Test_general.cpp використовується поєднання операцій над чотирма зашифрованими текстами. Такі функції, як add (), mul (), rotate () та sub () використовує як параметри шифротексти c1, c2, c3, c4, секретний ключ, зашифрований масив тощо.

У бібліотеці надається декілька тестів, які перевіряють функціонування якогось конкретного модуля або операції. Наприклад:

- Test_matmul – тестує функціональність множення зашифрованого вектору на матрицю відкритого тексту;

- Test_PolyEval – тестує функціональність обчислення многочлену вхідного тексту у точці шифрування;
- Test_bootstrapping – тестує функціонування процедури бутстрепінгу;
- Test_binaryArith – тестує реалізацію арифметичних операцій із числами у двійковому представлені.

Ці тести дозволяють зрозуміти, чи працює конкретний модуль належним чином, а також час, витрачений на певну функцію. Наприклад, після запуску тестової програми Test_matmul отримуються результати, деякі з яких показані на рисунку 3.10.

Після вказівки основних параметрів програми (круговий поліноміальний порядок, модуль, текстовий простір шифру тощо) надаються функції бібліотеки, які використовувались під час виконання програми. Ця інформація представлена у формі "назва функції: загальний час у секундах, витрачений на цю функцію загалом / кількість викликів функції під час обробки вхідного тексту = час, витрачений в секундах на одну операцію функції ця функція [місце в програмі, де функція] ".

Наприклад, другий рядок із подібною інформацією на рисунку 3.10 показує, що функція BluesteinFFT зайняла 22.1228 секунд. Це було викликано 4653 рази під час обробки тексту введення. Поділивши перше значення на друге, отримуємо результат, що на одноразову обробку цієї функції було витрачено близько 0,005 секунди. Ця функція викликається у файлі bluestein.cpp у рядку 65. Час виконання та кількість викликів можуть різнитися залежно від основних налаштувань бібліотеки.

Наприклад, візьмемо найважливіші функції бібліотеки – Encrypt, Decrypt, BluesteinFFT, DoubleCRT та функцію, що відповідає за генерацію матриці таємних ключів GenKeySWmatrix. Прослідкуємо за зміною кількісті їх викликів відповідно до зміни параметру m. Результати занесемо у таблицю 3.1.

```
HElib requires NTL version 10.0.0 or higher, see http://shoup.net/ntl
If you get compilation errors, try to add/remove -std=c++11 in Makefile

./Test matmul x m=18631 L=6
*** Test MatMul: m=18631, p=2, r=1, L=6, dim=0, nt=1, full=0, block=0, force bsgs=0

q = 2459293
m = 18631, p = 2, phi(m) = 18000
  ord(p)=25
  generator 17 has order (== Z m^*) of 120
  generator 1177 has order (== Z m^*) of 6

  |pubKey|=110807540 bytes
numOfPrimes = 5

bitsize of prime = 179.991
, security=610.034
  #threads=1, vector-dimension=120, 6 products in parallel, Nice!!

  BasicAutomorphPrecon: 0.327313 / 5 = 0.0654626   [matmul.cpp:60]
  BluesteinFFT: 22.1228 / 4653 = 0.00475453   [bluestein.cpp:65]
  CRT reconstruct: 0.498879 / 605 = 0.000824593   [PAlgebra.cpp:554]
  CompMod: 0.118649 / 605 = 0.000196114   [PAlgebra.cpp:518]
  Decrypt: 0.216728 / 5 = 0.0433456   [FHE.cpp:667]
  DoubleCRT: 1.57922 / 105 = 0.0150402   [DoubleCRT.cpp:378]
  DoubleCRT: 14.6936 / 600 = 0.0244893   [DoubleCRT.cpp:436]
  EncodeMartix MatMul: 15.6555 / 5 = 3.1311   [Test matmul.cpp:50]
  Encrypt: 0.351768 / 5 = 0.0703536   [FHE.cpp:313]
  FFT: 6.52782 / 1263 = 0.0051685   [CModulus.cpp:248]
  FFT: 6.53193 / 334 = 0.0195567   [DoubleCRT.cpp:48]
  FFT: 14.676 / 3000 = 0.00489201   [CModulus.cpp:262]
  FFT: 14.6852 / 600 = 0.0244754   [DoubleCRT.cpp:67]
  FFT remainder: 0.172852 / 3000 = 5.76173e-05   [CModulus.cpp:267]
  FFT remainder: 0.452115 / 1263 = 0.000357969   [CModulus.cpp:253]
  GenKeySWmatrix: 3.65831 / 51 = 0.0717316   [FHE.cpp:588]
```

Рисунок 3.10 – Частина результатів роботи програми Test_matmul

Таблиця 3.1 – Залежність кількості виклику функції від параметру m

	Кількість викликів функцій			
1	2	3	4	5
m	BluesteinFFT	Encrypt/Decrypt	DoubleCRT	GenKeySWmatrix
4368	3014	5	114	42
4858	3898	5	134	55
10262	4004	5	184	65
18632	1644	5	104	50
21846	3024	5	114	46

Проведений аналіз показує, що кількість викликів функції бібліотеки залежить від параметра m, оскільки це основний параметр, на якому базуються всі криптосистемні операції, що використовуються в бібліотеці. Але з наведених тестів помітно, що кількість викликів до процедур шифрування та дешифрування не змінюється. Оскільки кількість викликів їх залежить лише від кількості місць у вхідному тексті, тобто кількості зашифрованих бітів.

Керуючи основними параметри алгоритму, встановлено, що зміна параметра m, який задає круговий поліном, впливає на розмір відкритого ключа. Але через те, що випадкові величини в основному присутні в генерації відкритого ключа, залежність зростання ключа від параметра m не є лінійною, як це видно на рисунку 3.11.

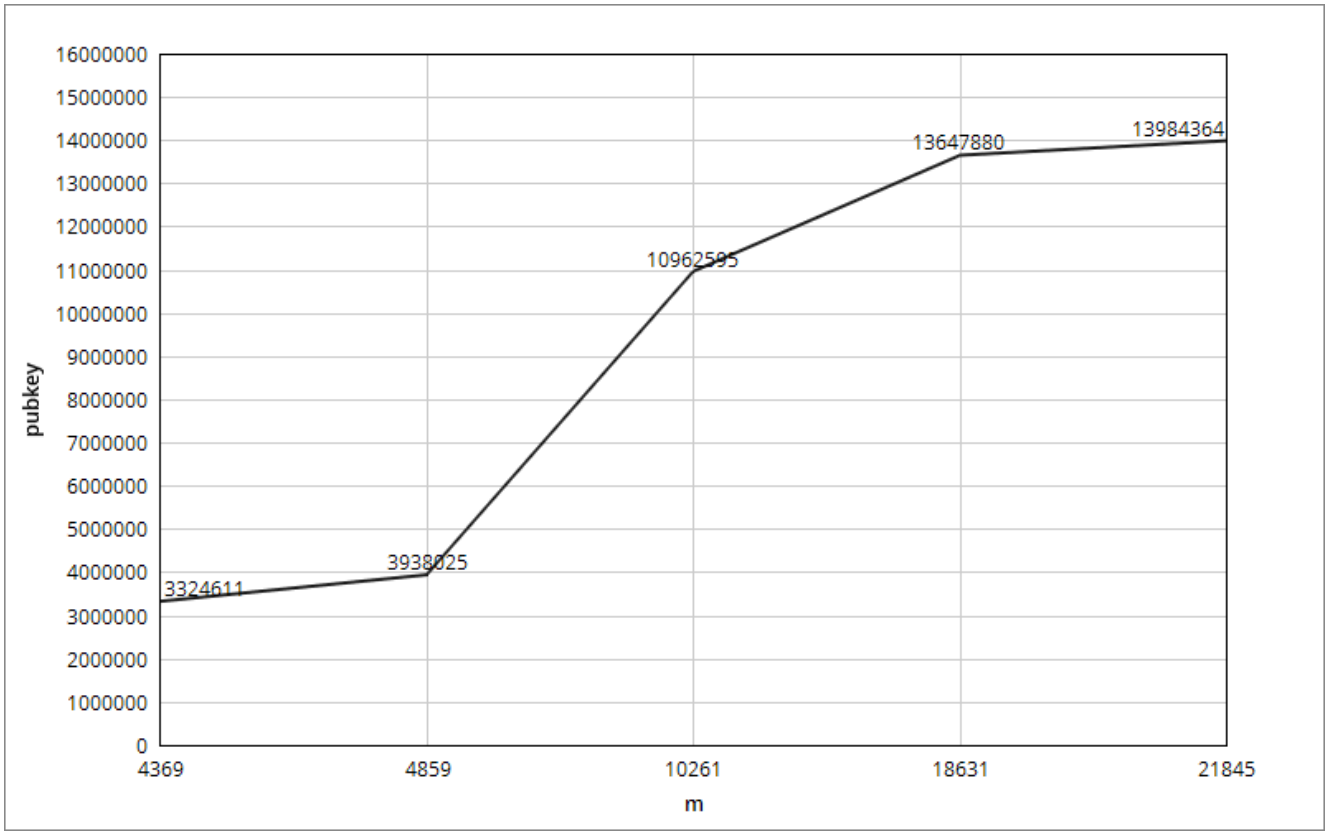

Рисунок 3.11 – Залежність розміру публічного ключа від m

Публічний ключ є головним параметром для функції шифрування, тому чим більший він є, а, відповідно, чим більше m, тим більше потребується часу на шифрування повідомлення. Але залежність є вже лінійною, бо алгоритм

шифрування побудований на обчисленнях декількох випадкових величин, а не однієї, що забезпечує деяку сталість. Графік залежності наведено на рисунку 3.12.

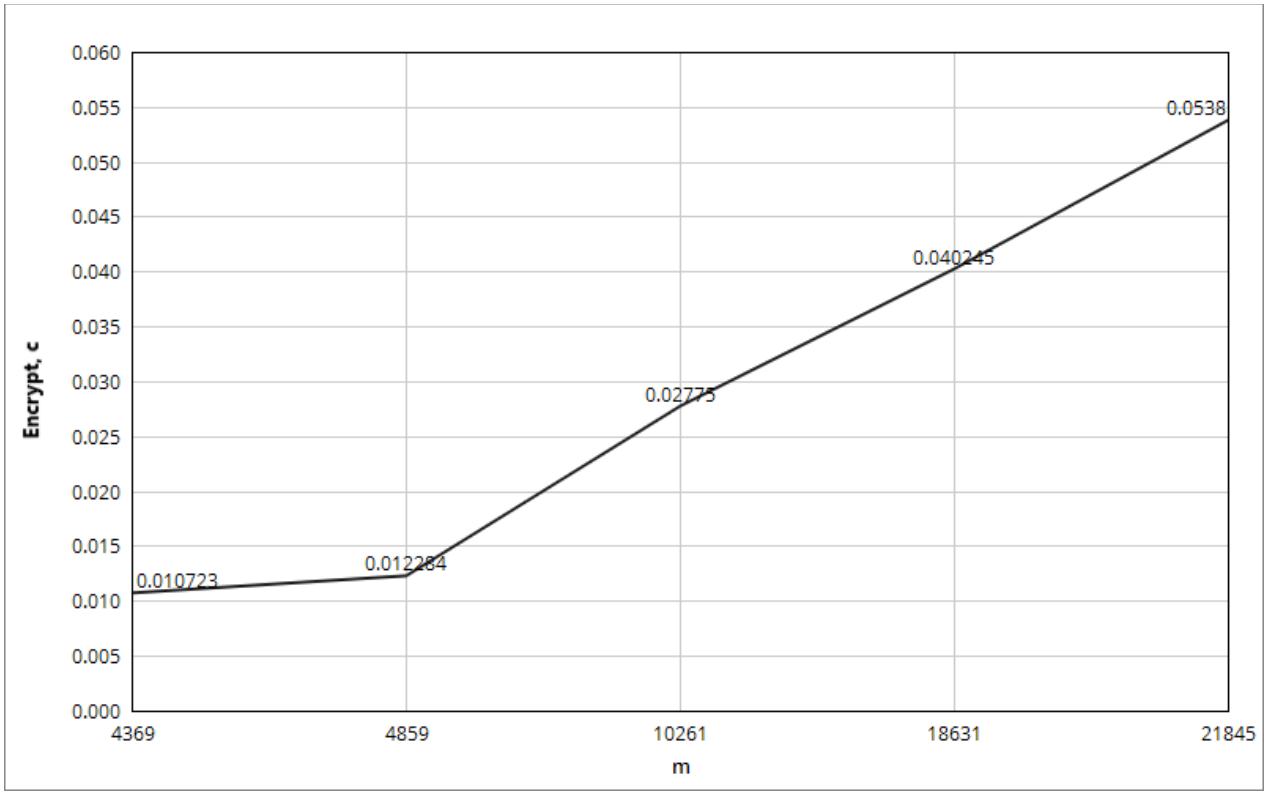

Рисунок 3.12 – Графік залежності часу шифрування тексту від параметру m

3.4.1 Дослідження безпеки способу шифрування

У криптографії рівень безпеки – це міра стійкості, якої досягає криптографічний примітив, такий як шифр або хеш-функція. Рівень безпеки зазвичай вказується у бітах, де вираз «n-бітна безпека» означає, що атакуючий повинен виконати 2^n операцій, щоб зламати шифр, але вже запропоновано і точніші моделі, які визначають затрати для зловмисника. Така властивість шифру дозволяє зручно порівнювати ефективність алгоритмів шифрування та вона є корисною при об’єднанні декількох криптографічних примітивів у гібридній криптосистемі.

Для деяких сфер застосування гомоморфного шифрування безпечні обчислення будуть надмірно ускладнювати процес обробки даних. У таких випадках можна застосовувати простіші за побудовою схеми. Схема шифрування є слабкою, якщо гомоморфно сформовані шифртексти можуть бути дешифровані

належним чином та утворюють вхідне повідомлення, а також якщо їх довжина залежить тільки від параметрів безпеки та довжини повідомлення (а не від кількості вхідних шифртекстів).

У бібліотеці існує окрема функція, яка обчислює рівень безпеки для різних наборів параметрів. Частину коду із цією функцією наведено на рисунку 3.13.

```
//! @brief An estimate for the security-level
double securityLevel() const {
  long phim = zMStar.getPhiM();
  IndexSet allPrimes(0,numPrimes()-1);
  double bitsize = logOfProduct(allPrimes)/log(2.0);
  return (7.2*phim/bitsize -110);
}
```

Рисунок 3.13 – Лістинг функції з оцінки рівня безпеки

У функції наявні наступні змінні:

- phim – це значення функції Ейлера для параметру m;
- numPrimes відповідає кількості малих простих чисел у ланцюзі модулів;
- екземпляр allPrimes класу IndexSet містить усі ці прості числа
- bitsize зберігає в собі величини кожного простого числа у бітах.

Змінюючи параметр m та підбираючи для нього значення із таблиці 3.1 можна зробити висновок, що рівень безпеки системи зростає із збільшенням значення m. Графік залежності рівня безпеки від параметру m наведено на рис 3.14.

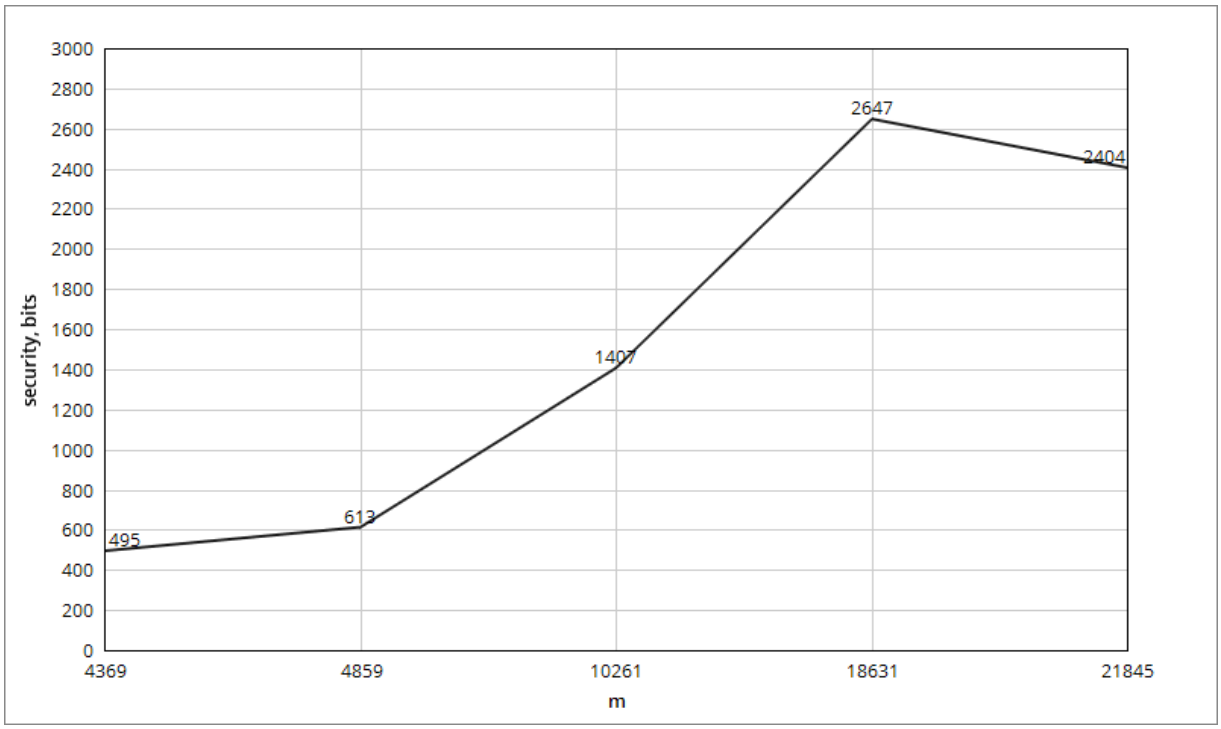

Рисунок 3.14 – Графік залежності рівня безпеки від параметру m

Найбільшого свого значення, а саме 2647 біт, рівень безпеки досягає при m = 18631. Тому у подальших дослідженнях буде використано саме це значення. Тепер можливо з’ясувати, чи залежить рівень безпеки від зміни кількості рівнів схеми. Графік залежності рівня безпеки від кількості рівнів схеми L наведено на рисунку 3.15.

Дивлячись на цей графік стає зрозумілим, що рівень безпеки залежить від параметру L, адже він впливає на значення змінної numPrimes, що розглядається на рисунку. 3.15 бо фактично і є кількістю простих чисел у шифртексті.

У ході досліджень також виявлено, що розмір простих чисел у бітах також впливає на рівень безпеки. Залежність рівня безпеки від кількості бітів у простому числі наведено у таблиці 3.2.

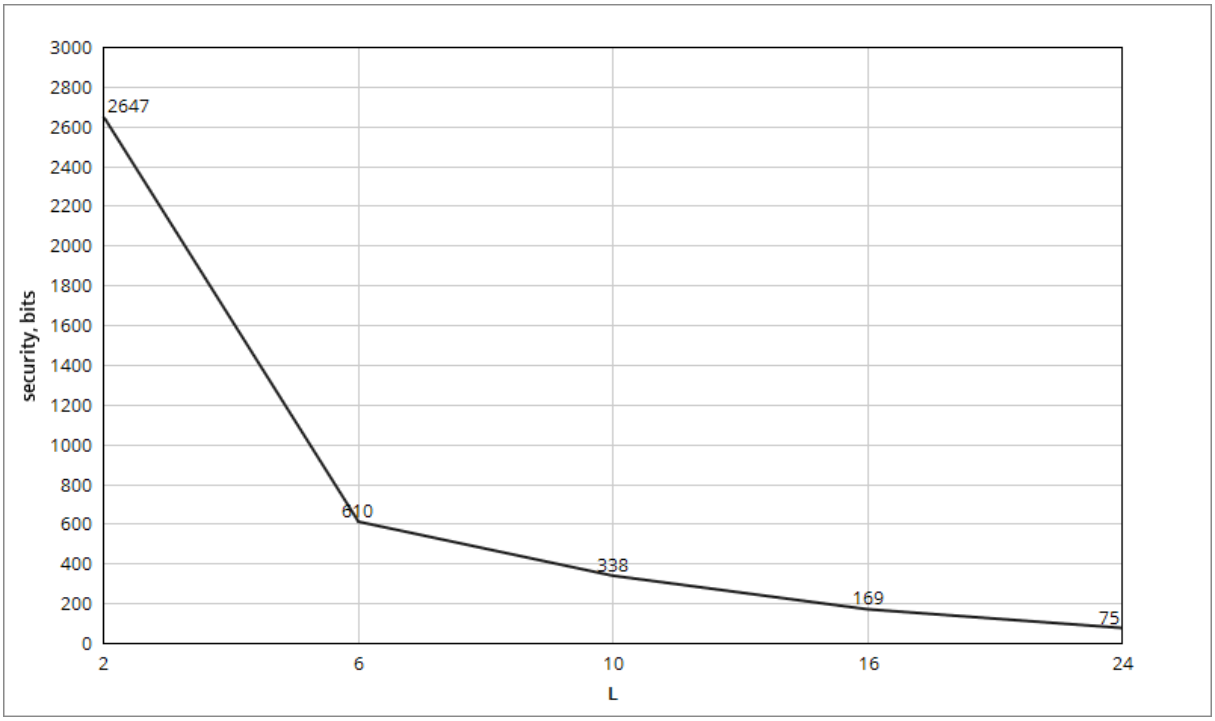

Рисунок 3.15 – Графік залежності рівня безпеки від кількості рівнів схеми L

Таблиця 3.2 – Залежність рівня безпеки від кількості бітів у простому числі

L	Рівень безпеки, біт	Бітовий розмір числа
2	2646	47
6	611	179
10	339	289
16	168	464
24	74	699
32	70	753
40	68	801
44	63	875
48	61	912

Виходячи з таблиці 3.2, бачимо, що ця залежність рівня безпеки від кількості бітів у простому числі прямопропорційна залежності рівня безпеки від кількості рівнів. Інакше кажучи, чим більше бітовий розмір простого числа, тим менший рівень безпеки схеми.

Таким чином, можна зробити висновок, що із зростом параметру m рівень безпеки зростає, але із зростом інших параметрів – він значно падає, що вказує на те, що система стає більш вразливою.

З огляду на те, що параметри бібліотеки визначаються автоматично, у випадку, коли вони не були попередньо зазначені, то HElib можна вважати простою у використанні для користувачів різного рівня обізнаності про функціонування алгоритмів бібліотеки.

Підсумовуючи дані дослідження ми дійшли висновку, що оптимальними значеннями для основних параметрів системи є такі:

- m = 18631, бо саме при цьому значенні система має найбільший рівень безпеки, що наведено на рисунку 3.18;
- p = 2, бо цей параметр визначає простір шифртексту, і найбільш ефективним за швидкодією є двійковий простір;
- r = 1, бо цей параметр є степенем простору шифртексту та для зберігання швидкості обчислень, степінь має дорівнювати одиниці;
- L = 2, бо кількість рівнів відповідає кількості простих чисел у ланцюзі модулів шифрування, і чим менше цих чисел, тим менша вірогідність знайти значення, що їх генерує та обчислити таємні ключі кожного рівня.

3.5 Створення коефіцієнта пріоритетності шифрування

У профілі кожного користувача виставляється коефіцієнт-пріоритету шифрування, який у свою чергу представлений вибіркою значень від 1 до 10.

1 – це варіант, який має найбільшу швидкодію та має найменший рівень безпеки

10 - це варіант, який має найменшу швидкодію, але найбільший рівень безпеки.

При створені нового повідомлення користувачем, воно передається на сервер разом з коефіцієнтом пріоритетності. На сервері отримане повідомлення шифрується за допомогою додатку написаного з використанням бібліотеки HElib та керує параметрами m, p, r, L бібліотеки в залежності від переданого

коефіцієнту. Отримане зашифроване повідомлення зберігається у базі даних для подальшого використання.

Висновки до розділу

Обґрунтовано вибір середовища для реалізації програмного продукту з використанням способу керування параметрами алгоритму гомоморфного шифрування за допомогою значення коефіцієнта пріоритетності.

Описано процес підготовки VM й властивості серверу запущеного на ній для обробки та передачі даних з використанням механізму гомоморфного шифрування з урахуванням коефіцієнта пріоритетності.

Проведено аналіз бібліотеки й пошук оптимальних параметрів, що впивають на швидкодію та рівень безпеки передачі даних для створення коефіцієнта пріоритетності.

Використання даного підходу дозволяє зменшити навантаження на сервер, через зменшення кількості операцій з даними, які не потребують високого рівня безпеки.

Розроблений на основі способу гомоморфного шифрування з використанням коефіцієнта пріоритетності програмний продукт дозволяє оптимізувати роботу сервера й надати необхідні ресурси для обробки даних, які потребують максимального рівня безпеки.

4 АНАЛІЗ ЕФЕКТИВНОСТІ ЗАПРОПОНОВАНОГО СПОСОБУ

4.1 Порівняння швидкодії способу шифрування з коефіцієнтом пріоритетності та без нього

Для порівняння способу з використанням коефіцієнту пріоритетності створена база даних з повідомлень. Кожне повідомлення має висталений заздалегідь коефіцієнт. Спочатку сервер запрограмований ігнорувати коефіцієнт й шифрувати дані з визначеними, оптимальними параметрами, спрямованими лише на надання найбільшої крипостійкості. Отримавши ці дані, увімкнули на сервері додатковий контролер, який відповідає за аналіз коефіцієнту пріоритетності й подальше керуванням параметрами програми шифрування даних.

Кожне повідомлення у даному порівняні розглядається, як окрема задача й має свій час виконання.

В таблиці 4.1 наведено розмір кожної задачі й виставлений коефіцієнт пріоритетності для подальшого порівняння. Коефіцієнти обрано таким чином, щоб сервер мав, як прості задачі так і складні.

В таблицях 4.1 та 4.2 наведено результати тесту з ввімкненим контролером, що відповідає за розроблений спосіб та без нього.

Таблиця 4.1 – Розмір кожної задачі та присвоєний довільний коефіцієнт

Номер задачі	Бітовий розмір повідомлення	Коефіцієнт пріоритетності
1	8	10
2	256	5
3	512	1
4	1024	10
5	2048	5
6	4096	1

7	8192	10
8	16384	5
9	32768	1
10	65536	10
11	131072	5
12	262144	1
13	524288	10
14	1048576	5
15	2097152	1
16	4194304	10
17	8388608	5
18	16777216	1
19	33554432	5
20	67108864	1

Таблиця 4.2 – Час, затрачений на задачу без використання нового способу

Номер задачі	Час
1	0.00006
2	0.00086
3	0.00095
4	0.00127
5	0.00138
6	0.00158
7	0.00263
8	0.00400
9	0.00411
10	0.00426
11	0.00459
12	0.00482
13	0.00611
14	0.00618
15	0.00732
16	0.00870
17	0.00873
18	0.00908
19	0.00996
20	0.01284

Таблиця 4.3 – Час, затрачений на задачу з використання нового способу

Номер задачі	Час
1	0.00022
2	0.00118
3	0.00059
4	0.00139
5	0.00161
6	0.00108
7	0.00293
8	0.00293
9	0.00111
10	0.00511
11	0.00275
12	0.00090
13	0.00676
14	0.00357
15	0.00122
16	0.00902
17	0.00531
18	0.00147
19	0.01066
20	0.00712

Рисунок 4.1 – Порівняння алгоритму з використанням коефіцієнту пріоритетності та без нього

Оскільки даний спосіб спрямований на те, щоб користувачі могли обирати необхідний для них варіант шифрування й подальшого зберігання та обробки даних на сервері в залежності від виставленого ними коефіцієнту, розглянемо усі можливі варіанти навантаження на сервер. Для порівняння візьмемо дані з минулої таблиці, де коефіцієнти були виставлені хаотично й нові дані наведені з виставленими коефіцієнтами для забезпечення максимального рівня безпеки повідомлень та даних при виборі максимальної швидкодії. Щоб обрати перший варіант – спрямований на рівень безпеки, необхідно виставити коефіцієнт пріоритетності у позицію «10», а для швидкодії - у позицію «1».

Таблиця 4.3 – Час, затрачений на задачу з коефіцієнтом пріоритетності спрямованим на забезпечення високого рівня безпеки

Номер задачі	Час
1	0.00006
2	0.00086
3	0.00095
4	0.00127
5	0.00138
6	0.00158
7	0.00263
8	0.00400
9	0.00411
10	0.00426
11	0.00459
12	0.00482
13	0.00611
14	0.00618
15	0.00732
16	0.00870
17	0.00873
18	0.00908
19	0.00996
20	0.01284

Таблиця 4.3 – Час, затрачений на задачу з коефіцієнтом пріоритетності спрямованим на забезпечення швидкодії системи

Номер задачі	Час
1	0.00022
2	0.00118
3	0.00059
4	0.00139
5	0.00161
6	0.00108
7	0.00293
8	0.00293
9	0.00111
10	0.00511
11	0.00275
12	0.00090
13	0.00676
14	0.00357
15	0.00122
16	0.00902
17	0.00531
18	0.00147
19	0.01066
20	0.00712

Рисунок 4.2 – Порівняння алгоритму з використанням різних коефіцієнтів пріоритетності

Як видно з графіків, спосіб шифрування з коефіцієнтом пріоритетності має переваги при використані різних коефіцієнтів. Якщо користувачі ставитимуть пріоритет лише на безпеку, то навантаження на сервер тільки збільшиться. Але при правильному розподілі ресурсів системи, використання даного способу суттєво зменшить час реакції серверу.

Висновки до розділу

Подано характеристику вхідних повідомлень й результати роботи серверу з використанням способу шифрування з коефіцієнтом пріоритетності й без нього. Проведено аналіз отриманих даних та побудовано графік, для того щоб побачити візуальну залежність результатів від способу шифрування.

Розглянуто варіант роботи системи з виставленими коефіцієнтами лише на пріоритет рівня безпеки – «10» та з пріоритетом на швидкодії системи «1». Результати наведені у таблицях й побудовано графік їх залежності.

Проаналізувавши результати, зроблено висновок, що даний спосіб шифрування може себе лише при сумлінному користуванні. Якщо ж користувачі обиратимуть тільки, щось одне, система не зможе розкрите себе у повній мірі.

ВИСНОВКИ

В атестаційній роботі досліджено параметри системи гомоморфного шифрування на основі бібліотеки HElib. На основі значень цих параметрів створено коефіцієнт пріоритетності, який необхідний для запропонованого способу реалізації гомоморфного шифрування у вебдодатку.

З цією метою проведений аналіз сучасних методів шифрування, як за способом перетворення так і за типом ключа, розглянуто симетричний та асиметричний методи шифрування, а також різновиди хеш-функцій.

Проаналізовано особливості, окреслено переваги та недоліки основних методів шифрування. Показано, що на сьогоднішній день залишається не вирішеним питання забезпечення надійної обробки, зберігання та передачі даних в інформаційно-комунікаційних системах. Одним із шляхів вирішення вказаної проблеми є використання гомоморфного шифрування.

Проведено методологічний аналіз існуючих основних алгоритмів, на яких базується бібліотека HElib. Сформульовано задачі та особливості визначення основних параметрів для зручного використання алгоритму гомоморфного шифрування даних на основі бібліотеки HElib.

Проведено пошук оптимальних значень параметрів, що впивають на швидкодію та рівень безпеки передачі даних. На основі отриманих даних створено коефіцієнта пріоритетності, необхідний для реалізації запропонованого в роботі способу шифрування даних у вебдодатку.

Використання даного підходу дозволяє зменшити навантаження на сервер, через зменшення кількості операцій з даними, які не потребують високого рівня безпеки.

Запропонований спосіб відповідає визначеним вимогам та містить необхідний функціонал для надання рекомендацій користувачам, які можуть використовувати шифрування та розшифрування даних фактично без глибоких знань в області криптографії.

Оптимізація значень параметрів, при яких гомоморфне шифрування забезпечить необхідні криптографічні перетворення даних користувача, зберігаючи їх конфіденційність, є основою способу шифрування даних на базі HElib.

Результати, наведені в роботі, свідчать про ефективність даного способу за умови вибору коефіцієнту пріоритетності користувачем, що забезпечує подальше оптимальне використання параметрів програми шифрування даних.

Даний підхід сприяє подальшому розвитку механізму гомоморфного шифрування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Rivest R. Data banks and privacy homomorphisms / R. Rivest, L. Adleman, M. Dertouzos. // Foundations of Secure Computation. – 1978. – С. 159–180.

2. Craig G. A fully-homomorphic encryption scheme : / Craig Gentry – Stanford University, 2009.

3. Smart N. Fully-homomorphic encryption with small key and ciphertext sizes / N. Smart, F. Vercauteren. // Springer. – 2010. – С. 380–443.

4. Gentry C. Implementing gentrys fully-homomorphic Encryption scheme / C. Gentry, S. Halevi. // Springer. – 2011. – С. 79–148.

5. Fully homomorphic encryption over the integers / M.Van Dijk, C. Gentry, S. Halevi, V. Vaikuntanathan. // Springer. – 2010. – С. 14–43.

6. Brakerski Z. Leveled fully homomorphic encryption without bootstrapping / Z. Brakerski, C. Gentry, V. Vaikuntanathan. // ACM. – 2012. – С. 309–325

7. Brakerski Z. Ecient fully homomorphic encryption from (standard) LWE / Z. Brakerski, V. Vaikuntanathan. // SIAM Journal on Computing. – 2014. – №43. – С. 831–871.

8. Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In Advances in Cryptology–CRYPTO 2011, pages 505–524. Springer, 2011.

9. Zvika Brakerski and Vinod Vaikuntanathan. Ecient fully homomorphic encryption from (standard) lwe. SIAM Journal on Computing, 43(2):831–871, 2014.

10. L´eo Ducas and Daniele Micciancio. Fhe bootstrapping in less than a second. Technical report, Cryptology ePrint Archive, Report 2014/816, 2014.http://eprint.iacr.org, 2014.

11. Craig Gentry. A fully homomorphic encryption scheme. PhD thesis, Stanford University, 2009.

12. Craig Gentry and Shai Halevi. Implementing gentrys fully-homomorphic encryption scheme. In Advances in Cryptology–EUROCRYPT 2011, pages 129–148. Springer, 2011.

13. Craig Gentry, Shai Halevi, and Nigel P Smart. Better bootstrapping in fully homomorphic encryption. In Public Key Cryptography–PKC 2012, pages 1–16. Springer, 2012

14. Lyubashevsky V. On Ideal Lattices and Learning with Errors Over Rings / V. Lyubashevsky, C. Peikert, O. Regev. // EUROCRYPT. – 2013. – С. 220–258.

15. Comparing Python and Node.Js: Which Is Best for Your Project? [Електронний ресурс]. –Режим доступу до ресурсу: https:// djangostars.com/blog/comparing-python-node-js-best-project/