КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет комп'ютерних наук та кібернетики

Кафедра теоретичної кібернетики

Кваліфікаційна робота

На здобуття ступеня бакалавра

за спеціальністю 122 Комп'ютерні науки

на тему:

Гомоморфне шифрування для захисту даних в хмарних та туманних технологіях

Виконав студент 4-го курсу
Дмитро МАЛЬОВАНИЙ _____
(підпис)

Науковий керівник:
професор, доктор фіз-мат. наук
Анатолій ПАШКО _____
(підпис)

Засвідчую, що в цій роботі немає запозичень праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Роботу розглянуто й допущено до захисту
на засіданні кафедри
теоретичної кібернетики
"____"_____ 2023р
протокол №____
Завідувач кафедри
Юрій КРАК _____
(підпис)

Київ - 2023

РЕФЕРАТ

Обсяг роботи 53 сторінки, 8 зображень, 5 лістингів, 33 джерел посилань. ШИФРУВАННЯ, ГОМОМОРФНЕ ШИФРУВАННЯ, ХМАРНІ ТА ТУМАННІ КОМУНІКАЦІЇ, БЕЗПЕКА ПЕРЕДАЧІ ДАНИХ, ЗАХИСТ ДАНИХ, БЕЗПЕКА, БЕЗПЕКА ДАНИХ В БАНКІВСЬКІЙ СИСТЕМІ.

Об'єктом роботи є дослідження можливостей використання гомоморфного шифрування в хмарних та туманних обчисленнях. Предметом роботи, є реалізація спрощеної банківської системи, для демонстрації можливостей гомоморфного шифрування.

Метою роботи є дослідження технології повного гомоморфного шифрування в хмарних та туманних технологій.

Методи розроблення: дослідження гомоморфних схем, аналітичне дослідження алгоритмів над схемою. Інструменти розроблення: Мова C++, Бібліотека HeLib з вільно поширюваною ліцензією Apache 2.0, додаткові бібліотеки для зручної роботи з Json, обробки та ініціації TCP з'єднань, та інш.

Результати роботи: Описана логіка криптографічних схем гомоморфного шифрування, проведений аналітичний огляд існуючих реалізацій схем, наведені переваги та недоліки використання технології гомоморфного шифрування, реалізований та продемонстрований в роботі застосунок спрощеної банківської системи з використанням FHE.

Технологія гомоморфного шифрування, може використовуватись в будь-якій сфері де потрібна конфіденційність даних, зокрема вона дозволяє тримати їх приватними для сторони яка їх обробляє, що забезпечує ще вищий рівень безпеки.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

FHE – Fully homomorphic encryption (Повне гомоморфне шифрування).

SHE – Somewhat homomorphic encryption scheme

RSA – Криптографічний алгоритм з відкритим ключем, який базується, розрахунковій складності великих полупростих чисел.

PKI – Public key infrastructure - набір інструментів які використовують пару (приватний, публічний) ключ, та в якій між користувачами передається тільки публічні ключі, залишаючи приватні анонімними.

Boolean Circuit – Булева схема - це математична модель, що використовується для представлення та обробки булевих функцій. Вона складається з логічних елементів, які з'єднані між собою для виконання логічних операцій над двійковими входами $\{0, 1\}$ та формування двійкових виходів. Схема складається з взаємопов'язаних логічних елементів, таких як (AND, NOT, OR).

FLT – Мала теорема ферма [1].

NAND gate – NOT, AND операції за допомогою який може бути представлена булева схема.

ВСТУП

FHE або повне гомоморфне шифрування, це тип шифрування яке дозволяє виконувати розрахунки на зашифрованих даних, не вимагаючи, щоб вони були розшифровані для цього. Результатом розрахунків або ж гомоморфної операції над даними є зашифровані дані, які можуть бути розшифровані ключем з тої ж самої пари, з якої вони були зашифровані.

Завдяки особливості виконувати операції над зашифрованими даними без попереднього дешифрування, FHE стає гарним рішенням в задачах передачі даних в незахищених середовищах, в не авторизованих середовищах, або при передачі над чутливих даних, які не повинні бути видимі для сторони яка займається їх обробкою.

Оцінка сучасного стану об'єкта дослідження або розробки

Вперше технологія FHE була запропонована в 1978 в році, але майже 30 років не було авторитетних досліджень на цю тему і на той момент вже існувала система RSA, яка була краща за багатьма параметрам. Починаючи з 2009 року дослідження та розробки на тему гомоморфного шифрування дуже актуальні й розвиток цієї технології відбувається надзвичайно швидко, покращуючи швидкість виконання операцій, швидкість дешифрування, та розширюючи область застосування шляхом додавання більш комплексних гомоморфних операцій. На цей час існує багато рішень на типові проблеми з використовуванням FHE, які конкурують між собою в різних аспектах, та постійно розвиваються.

Актуальність роботи та підстави для її виконання

Безпечність передачі даних в незахищених середовищах та авторизація отримувача були завжди дуже важливими, рідко хто нехтує цим, оскільки не хоче, щоб їх данні були скомпрометовані або перехоплені. Окрім цього все частіше, за потребою складних обчислень, користувачі звертаються до віддалених машин, також відомі як хмари. Звісно кожен користувач хоче, щоб їх данні були захищенні під час передачі, та хоче бути впевнений, що він передає данні саме туди, куди планував.

Для забезпечення вище описаних вимог, користувач використовує чинні технології, такі як PKI. Єдина не вирішена проблема PKI або інших технологій, це вимога повного дешифрування даних, це означає що при отриманні зловмисником доступу до хмари або віддаленого сервера, у нього буде доступ до не зашифрованих даних. Хоча сучасні хмари дуже добре захищенні, розраховувати на те що зловмисник не зможе отримати до них доступ - не варто.

Для розв'язання проблеми, яку не вирішує PKI, чудово підходить FHE, оскільки сервер, зберігає і виконує операції над даними в зашифрованому вигляді, тому навіть якщо зловмисник отримає доступ до сервера або хмари, отримати дані в нього не вдасться.

Звісно є і деякі обмеження у використанні FHE: по-перше, операції над даними обов'язково повинні бути гомоморфні, по-друге, алгоритм застосування операції над зашифрованими даними дуже повільний. Якщо задача вимагає обробку великої кількості даних, або операція повинна бути не гомоморфна, то можливо краще подумати в сторону застосування інших криптосистем.

Мета й завдання роботи

Мета роботи дослідити інсуючі гомоморфні схеми, визначити їх криптографічну схему, обмеження та можливі області застосування. Також, необхідно показати імплементації існуючих повних та частково гомоморфних схем, описати їх вразливості, аналітично порівняти описані схеми.

Також метою роботи є засвідчення того що гомоморфне шифрування застосоване до задач безпечної передачі даних у хмарних та туманних технологіях. Завдання полягає в тому, щоб показати теоретично та практично, що

дані користувача можуть бути безпечно передані та оброблені хмарою, без розкриття цих даних для хмари.

Також необхідно перевірити результати практичного використання FHE на коректність, та порівняти накладні витрати, по часу та пам'яті, виконання операції над зашифрованими даними, та над не зашифрованими.

Об'єкт і методи дослідження

Для дослідження коректності та застосованості FHE і практичної реалізації системи з використанням технології FHE, було вибрано клієнт-серверний застосунок, де сервер буде виконувати роль хмари, та з'єднання клієнта з сервером відбувається в незахищеному середовищі.

Областю реалізації буде спрощена банківська система, де хмара буде виконувати роль банку, який дозволяє користувачу додавати, знімати, та передивлятись свій баланс віртуальних грошей. При цьому серверний застосунок повинен бути реалізований таким чином, що він не буде знати нічого, ні про користувача, а ні про то скільки умовного баланса у певного клієнта. Для цього він буде зберігати данні зашифровані FHE у внутрішній базі даних, та публічний ключ клієнта для виконання гомоморфних операцій над даними.

Ця система повинна чудово показати всю силу гомоморфного шифрування: тільки клієнт, який створив баланс за допомогою свого приватного ключа, буде мати можливість мати доступ до свого балансу, як переглядати його, так і виконувати над ним певні операції. Всі інші учасники та користувачі системи не матимуть доступу до даних, що забезпечує їх повну безпеку.

Більш детально про об'єкти та методи дослідження буде описано в другому розділі роботи, фрагменти реалізації будуть наведені в додатках до роботи.

Можливі сфери застосування

Гомоморфне шифрування може бути застосоване в будь-якій сфері де потрібна обробка даних, та для виконання цієї задачі використовується віддалений сервер, або хмара. Використання FHE, гарантує безпечну передачу та обробку без попереднього дешифрування даних, але при цьому накладає обмеження на операцію обробки, яка повинна бути гомоморфна, та значно

знижує час обробки.

Більш детально ця тема буде розкрита в відповідному розділі, де будуть описані як повноцінні області застосування FHE, так і використання FHE як інструмент для створення більш комплексних криптосистем, та інструментів.

Розділ 1
Огляд технології FHE

1.1 Означення

В цій секції описана термінологія, яка використовується в дослідженнях FHE. Деякі з визначень були взяті напряму з документів FHE, інші були перефразовані для того, щоб спростити формальність і зробити їх більш застосованими до обраної задачі.

Нехай є простір вхідного (чистого) тексту $\mathcal{P} = \{0, 1\}$, та сімейства функцій $F = f_1, f_2, ..., f_n$ де $f_n(x) = f(x_1, x_2, x_3, ..., x_k)$ це Булеві функції k аргументів: $f : P^n \to P$. Ми будемо називати F, сімейством Булевих схем (Boolean circuit) C, і використовувати звичайний запис функції $C(m_1, m_2, ..., m_n)$, для позначення оцінки Булевої схеми на кортежі $(m_1, m_2, ..., m_n)$.

Означення 1.1.1 ($\mathcal{C}$-схема розрахунків, або ж просто $\mathcal{C}$-схема [8])**.** Нехай $\mathcal{C}$ це множина Булевих схем, тоді $\mathcal{C}$-схема розрахунків, для $\mathcal{C}$ це набір функцій (GEN, ENC, EVAL, DEC) які задовільняють наступним твердженням:

GEN$(1^\lambda, \alpha)$ - алгоритм генерації ключів, на вхід він приймає, параметр шифрування λ, та допоміжний параметр α. Результат виконання алгоритму це триплет ключів (pk, sk, evk), де ключ pk використовується для шифрування, sk для дешифрування, та evk для виконування розрахунків.

ENC(pk, m) - алгоритм шифрування, на вхід він приймає ключ шифрування pk та фрагмент не зашифрованого (чистого) тексту m. Результат виконання алгоритму це шифр c.

EVAL$(evk, C, c_1, c_2..., c_n)$ - алгоритм розрахунків. На вхід він отримує, ключ розрахунків evl та Булеву схему $C \in \mathcal{C}$, та вхідні аргументи, які можуть бути як шифром, так і результатом виконання минулих розрахунків. Результат виконання алгоритму це результат виконання розрахунків.

DEC(sk, c) - алгоритм дешифрування. На вхід приймає, ключ дешифрування sk, та шифр, або результат виконання розрахунків. Результат виконання алгоритму це не зашифрований (чистий) текст m.

Для подальшого опису властивостей, треба визначити простори даних, які є результатами, або вхідними параметрами описаних алгоритмів:

Нехай $\mathcal{X}$ буде описувати простір *чистого шифру*, $\mathcal{Y}$ - простір результатів виконання розрахунків, і $\mathcal{Z} = \mathcal{X} \cup \mathcal{Y}$. $\mathcal{Z}^*$ - містить кортежі довільної довжини, які складаються з елементів $\mathcal{Z}$. Простори ключів згенерованих **GEN**, позначимо як $\mathcal{K}_p, \mathcal{K}_s, \mathcal{K}_e$ для pk, sk, evk відповідно.Алгоритм **GEN** приймає на вхід параметр в унарній нотації 1^λ та опціональний допоміжний параметр λ з простору $\mathcal{A}$. Також, $\mathcal{C}$ містить простір *дозволених* булевих схем, а $\mathcal{P}$, як було зазначено раніше, область вхідного *чистого (незашифрованого)тексту.*

Тепер можна описати область роботи наведених вище алгоритмів:

$$\begin{aligned}
&\textbf{GEN} : \mathbb{N} \times \mathcal{A} \to \mathcal{K}_p \times \mathcal{K}_s \times \mathcal{K}_e \\
&\textbf{ENC} : \mathcal{K}_p \times \mathcal{P} \to \mathcal{X} \\
&\textbf{EVAL}: \mathcal{K}_e \times \mathcal{C} \times \mathcal{Z}^* \to \mathcal{Y} \\
&\textbf{DEC} : \mathcal{K}_s \times \mathcal{Z} \to \mathcal{P}
\end{aligned}$$

Тоді $\mathcal{X}$ та $\mathcal{Y}$ можна визначити наступним чином:

$$\begin{aligned}
\mathcal{X} &= \{c \mid \textbf{ENC}(pk, m) = c,\ m \in \mathcal{P}\} \\
\mathcal{Y} &= \{z \mid \textbf{EVAL}(evk, C, c_1, c_2, ..., c_n) = z,\ c_i \in \mathcal{Z},\ C \in \mathcal{C}\}
\end{aligned}$$

В деяких схемах, ключі розрахунків та шифрування однакові, але часто це і не так, тому в визначеннях було наведено більш спільний випадок.

В оригінальних документах FHE [8] не було зазначено, що алгоритм розшифровування **DEC** повинен мати можливість працювати з результатом виконання алгоритму шифрування **ENC** - $\mathcal{X}$, і було зазначено, що данні можуть бути розшифровані після виконання розрахунків над ними **EVAL** - $\mathcal{Y}$. Для можливості розшифровування, зразу після зашифровування було запропоновано мати *чисту Булеву схему* або ж по суті функцію $f(x) = x$, для виконання розрахунків і отримання даних які вже можна буде розшифровувати. Більшість сучасних FHE схем, дозволяють проводити операції дешифрування даних, над якими не було проведено розрахунків, тому я не буду заглиблюватись в цю тему.

1.1.1 Атрибути та властивості

Тут представлені характеристики методів гомоморфного шифрування. Ми встановлюємо такі властивості, як компактність і конфіденційність схеми, які забороняють спрощені рішення задачі гомоморфного шифрування, з одного боку, і вимагають таких властивостей, як коректність, для того, щоб навіть називати це схемою шифрування.

Означення 1.1.2 (Коректне розшифровування [3])**.** $\mathcal{C}$-схама має атрибут коректного розшифрування якщо виконується наступне твердження:

$$\mathbf{DEC}(sk, \mathbf{ENC}(pk, m)) = m,$$
$$\text{де } pk, sk, evk \leftarrow \mathbf{GEN}(1^\lambda, \alpha),\ \alpha \in \mathcal{A},\ m \in \mathcal{P}.$$

Це означає, що ми повинні мати можливість безпомилково розшифровувати зашифрований текст.

Означення 1.1.3 (Коректні розрахунки [3])**.** $\mathcal{C}$-схема коректно розраховує всі Булеві схеми $C \in \mathcal{C}$, якщо виконується наступне твердження:

$$\mathbf{DEC}(sk, \mathbf{EVAL}(evk, C, c_1, c_2, ..., c_n)) \ = \ C(m_1, m_2, ...m_n),$$
$$pk, sk, evk \leftarrow \mathbf{GEN}(1^\lambda, \alpha),\ \alpha \in \mathcal{A},\ c_i \in \mathcal{X} \text{ та } m_i \leftarrow \mathbf{DEC}(sk, c_i)$$

Це визначення означає, що розрахунки над зашифрованими даними з подальшим розшифровуванням повинні бути однакові з результатом розрахунків над не зашифрованими даними.

Будемо називати $\mathcal{C}$-схему *коректною* якщо для неї будуть виконуватись (Озн. 1.1.2) та (Озн. 1.1.3) твердження.

Означення 1.1.4 (Компактність $\mathcal{C}$-схеми)**.** $\mathcal{C}$-схема вважається компактною якщо існує поліном p, такий що, для будь-якого кортежу $(pk, sk, evk) \leftarrow \mathbf{GEN}(1^\lambda, \alpha)$, $\alpha \in \mathcal{A}$, будь-якої Булевої схеми $C \in \mathcal{C}$ та шифру $c_i \in \mathcal{X}$, розмір результату виконання $\mathbf{EVAL}(evk, C, c_1, c_2, ..., c_n)$ не більше від $p(\lambda)$ бітів, в не залежності від Булевої схеми.

Озн.чення 1.1.4, показує що під час гомоморфних операцій розмір результату не повинен збільшуватись, і залежить тільки від параметра безпеки λ.

Означення 1.1.5 (Компактно розрахункова $\mathcal{C}$-схема [16])**.** $\mathcal{C}$-схема компактно розраховує всі Булеві схема $C \in \mathcal{C}$, якщо вона компактна (Озн. 1.1.4) та *коректна.*

Безпека схеми

Далі, важливо зупинитись на безпеці та конфіденційності схеми. Безпеку схеми можна розділити на дві компоненти: семантична безпека, та обфускація схеми. Якщо обфускація використовується коли алгоритм шифрування секретний, і вразливий, то семантична безпека описує розподіл вихідних даних з **EVAL** та **ENC**.

Означення 1.1.6 (Конфіденційне гомоморфне шифрування схеми [16](2.16))**.** $\mathcal{C}$-схема вважається безпечною, якщо для будь-якого кортежу $(pk, sk, evk) \leftarrow \mathbf{GEN}(1^\lambda, \alpha)$, $\alpha \in \mathcal{A}$, будь-якої Булевої схеми $C \in \mathcal{C}$ та шифру $c_i \in \mathcal{X}$, такого що $m_i \leftarrow \mathbf{DEC}(sk, c_i)$ існує два розподіли:

$$Dist_1 = \mathbf{EVAL}(evk, C, c_1, c_2, ..., c_n)$$
$$Dist_2 = \mathbf{ENC}(pk, C(c_1, c_2, ..., c_n)$$

які повинні бути статистично або обчислювально нерозрізнені. Ці вимоги показують, що розподіл виконання обчислень Булевої схеми над шифром $Dist_1$ повинен бути однаковий (статистично, обчислювально) з розподілом, отриманим шляхом зашифровування *чистого* тексту, який насамперед являється результатом виконання Булевої операції над незашифрованими даними $Dist_2$.

Часто термін безпечної системи можна зустріти як *Сильно гомоморфна система*[12].

1.1.2 Класифікація

Оскільки не всі схеми FHE мають однакові властивості, цей розділ показує як схеми класифікуються, в залежності від того, які схеми вони можуть обчислювати.

Означення 1.1.7 (Частково Гомоморфна схема або $\mathcal{C}$-Гомоморфізм [8])**.** $\mathcal{C}$-схема називається, частково гомоморфною (SHE), якщо вона має коректне шифрування (Озн. 1.1.2), та коректне обчислення (Озн. 1.1.3).

Для частково гомоморфних $\mathcal{C}$-схем нема вимог до компактності, тому з кожним гомоморфним розрахунком розмір вихідного шифру може збільшуватись. Також нема ніяких вимог до множини Булевих операцій які можуть бути використовувані для розрахунків.

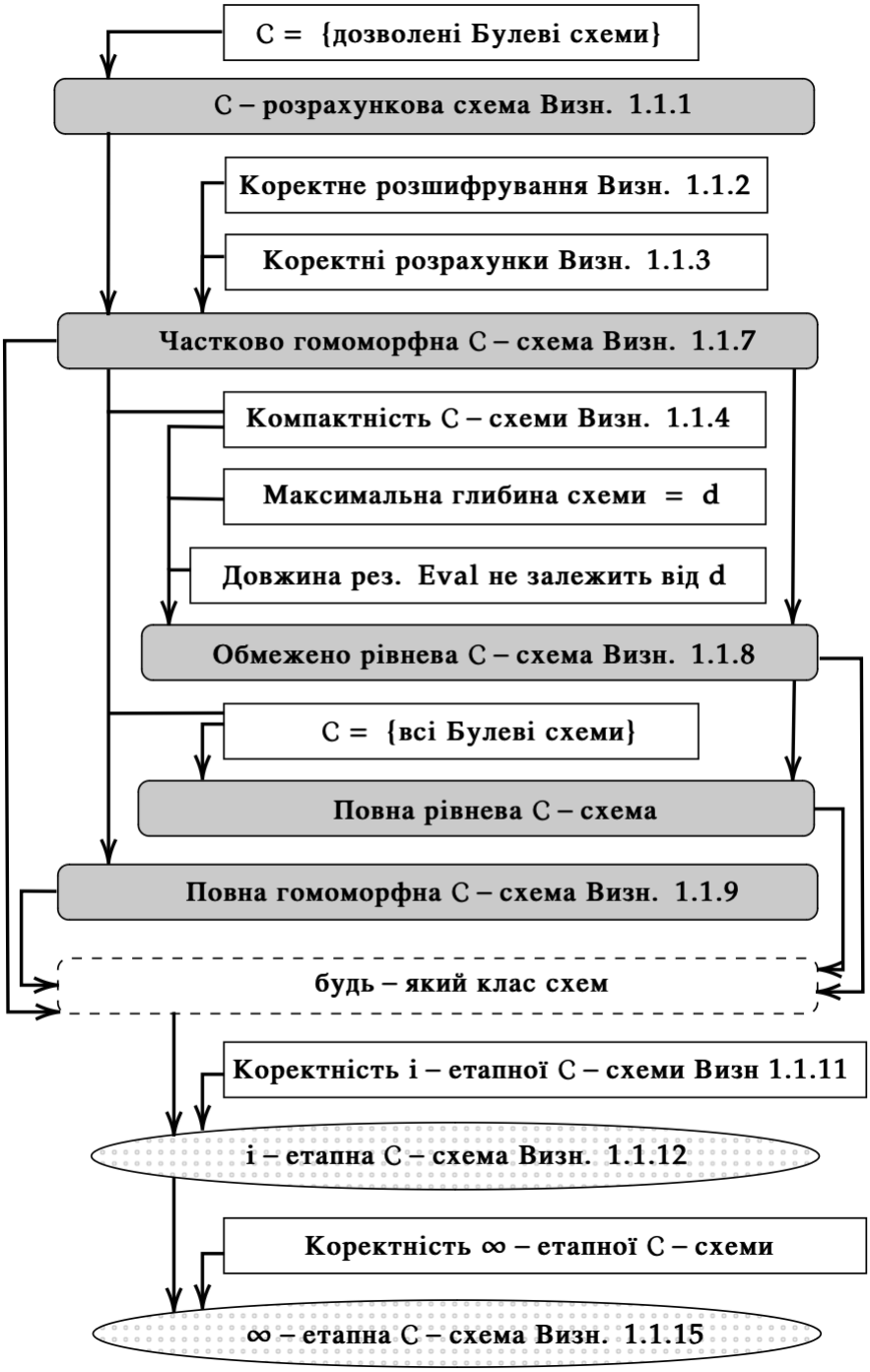

Рис. 1.1: Дерево класифікацій $\mathcal{C}$-схем. Прямокутниками позначені визначення, закруглені затемнені прямокутники, позначають класи $\mathcal{C}$-схеми, а еліпси позначають розширення для етапних розрахунків. Стрілки показуть залежність одного твердження від іншого.

Означення 1.1.8 (Обмежено-рівнева Гомоморфна схема)**.** $\mathcal{C}$-схема називається Обмежено-рівневою, якщо алгоритм генерації ключів **GEN** приймає додатковий параметр $\alpha = d$, який означає максимальну глибину Булевої схеми, яка може бути обчислена. Також застосовані вимоги до компактності, коре-

ктності, і те що розмір вихідних даних розрахунків не повинен залежати від d.

Означення 1.1.9 (Повна Гомоморфна схема)**.** Повною гомоморфною схемою, називають $\mathcal{C}$-схему, до якої застосовані вимоги, коректності, компактності, та вона може обчислювати Булеву схему з множини усіх схем, або ж будь-яку схему.

1.1.3 Композиція розрахунків

Часто, задача потребує декілька послідовних розрахунків, тобто результат певної Булевої схеми повинен слугувати вхідними даними для наступної схеми, або ж простими словами можна це назвати - композиція. Кожну операцію розрахунків над шифром **EVAL** будемо називати *етапом розрахунків.*

З визначення коректних розрахунків 1.1.3 видно що вхідні дані для алгоритму обчислення **EVAL** повинні належати множині $\mathcal{X}$ - або ж множині *чистого шифру*, який є результатом алгоритму **ENC**. Цей розділ описує вимоги, виконуючи які алгоритм розрахунку схеми **EVAL**, може приймати на вхід як результат виконання інших розрахунків $\mathcal{Z}$, так і *чистий шифр* $\mathcal{X}$:

$\mathbf{EVAL}(evk, C, c_1, c_2, ..., c_n)$, де $(pk, sk, evk) \leftarrow \mathbf{GEN}(1^\lambda, \alpha)$, $\alpha \in \mathcal{A}$, $C \in \mathcal{C}$ та $c_i \in \mathcal{X} \cup \mathcal{Z}$

В літературі *розрахунки з етапами* називають ***гомоморфним шифруванням з і-етапами***(i-hop homomorphic encryption [29], [18])

Означення 1.1.10 (Розрахунки з етапами)**.** Обчислення $\mathbf{C}_{i,n}$ в i етапів, та шириною n, визначається множиною Булевих схем $\{C_{kl}\}$, де $1 \leq k \leq i, 1 \leq l \leq$ n, та C_{kl} має kn вхідних даних. За вхідними даними $m_{01}, m_{02}, ..., m_{0n}$ ми обчислюємо:

$$m_{kl} = C_{kl}(m_{01}, m_{02}, ..., m_{0n}, ..., m_{k-1,1}, ..., m_{k-1,n}), \text{ де } 1 \leq k \leq i, 1 \leq l \leq n.$$

Результат розрахунків з етапом після **EVAL** та **DEC** буде *чистий текст* $m_{i1}, m_{i2}, ..., m_{in}$. Озн.чимо початковий *чистий текст* як $\overrightarrow{m_0}$, та вихідний *чистий текст* як $\overrightarrow{m_i}$, тоді можна записати співвідношення $\overrightarrow{m_i} = C_{i,n}(\overrightarrow{m_0})$.

Нехай $(pk, sk, evk) \leftarrow \mathbf{GEN}(1^\lambda, \alpha)$, $\alpha \in \mathcal{A}$, та $c_{i1}, c_{i2}, ..., c_{in} \in \mathcal{X}$, тоді шифр $\{c_{kl}\}$, $1 \leq k \leq i, 1 \leq l \leq n$ обчислюється рекурсивно наступним чином:

$$c_{kl} = \mathbf{EVAL}(evk, C_{kl}, c_{01}, ..., c_{0n}, ..., c_{k-1,1}, ..., c_{k-1,n})$$

Результат розрахунків з етапом над зашифрованими даними, буде шифр $c_{i1}, c_{i2}, ..., c_{in}$. Позначивши початковий (вхідний) шифр як $\overrightarrow{c_0}$ та результівний шифр як $\overrightarrow{c_i}$ можна описати співвідношення яке описує нотацію алгоритму **EVAL** з декількома виходами: $\overrightarrow{c_i} = \textbf{EVAL}(evk, C_{1,n}, \overrightarrow{c_0})$

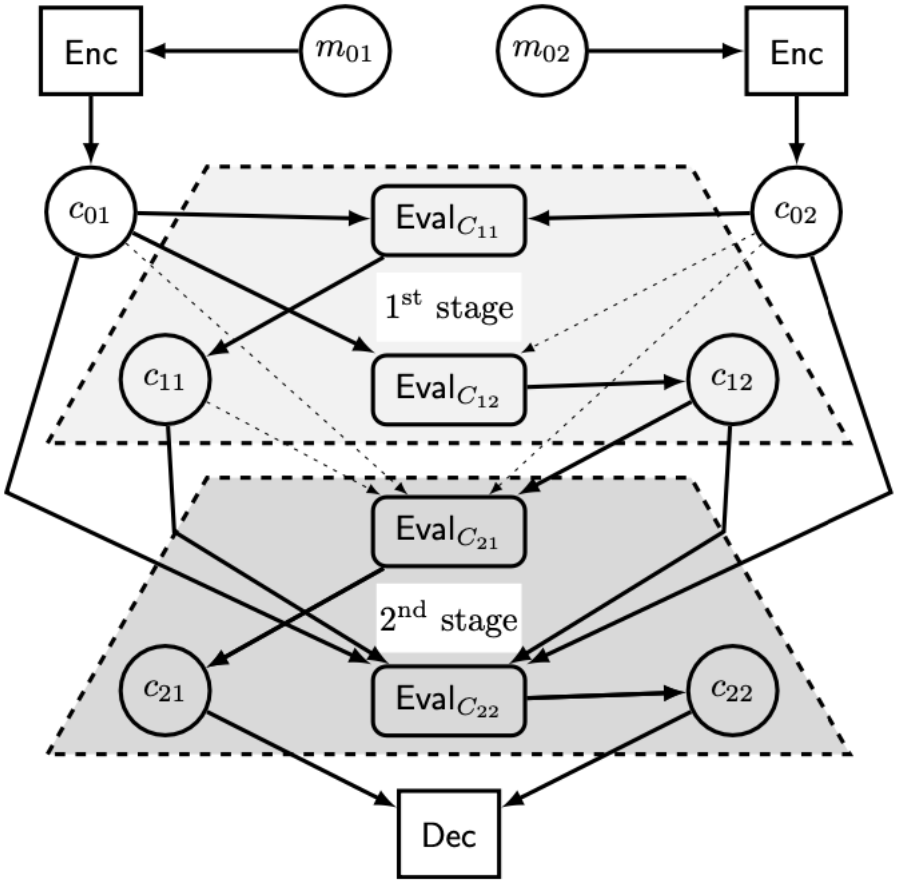

Рис. 1.2: Приклад [3] гомоморфного шифрування з і-етапами, де $i = 2, n = 2$

З вище описаного визначення 1.1.10 можна зробити висновок що вхідними даними для будь-якого етапу, окрім першого, може бути тільки результат попереднього етапу.

На перший погляд, може здатись, що якщо у нас є можливість обчислити довільну Булеву схему, не використовуючи i-етапне шифрування, то повинна бути можливість обчислювати багато схем послідовно. Проте це не так. Нема гарантій того, що результат виконання **EVAL**, буде валідний для використання як вхідні дані для наступного **EVAL**. Наведемо приклад [3]: Нехай у нас є схема C яка приймає на вхід $c_1, c_0, ..., c_n$, та результатом якої є $c'_0, c'_1, ..., c'_v$, та схема C', яка приймає на вхід $c_0, c_1, ..., c_v$ та результат якої $c'_0, c'_1, ..., c'_w$. Тоді існує 2 можливих сценарії: 1) Якщо ми візьмемо композицію C та C' як

схему для розрахунків **EVAL**(evk, $C \circ C'$, $c_1, c_2, ..., c_n$) то ці розрахунки будуть коректними, оскільки виконується одна Булева схема, 2) проте, якщо ми спочатку розрахуємо **EVAL**(evk, C, $c_1, c_2, ..., c_n$) = $c'_1, c'_2, ..., c'_v$, а потім **EVAL**(evk, C', $c'_1, c'_2, ..., c'_v$) то це не спрацює зі звичайною схемою повного гомоморфного шифрування, оскільки вона не гарантує коректність даних після розрахунків, для наступних операцій. Тому якщо стоїть задача виконання послідовних, незалежних обчислень, то варто використовувати схему з i-етапами.

Означення 1.1.11 (Коректність гомоморфного шифрування з i-етапами)**.** Нехай $(pk, sk, evk) \leftarrow$ **GEN**$(1^\lambda, \alpha)$, $\alpha \in \mathcal{A}$, та $\mathbf{C}_{i,n} = \{C_{k,l}\}$ - довільне поетапне обчислення, де n це розмір полінома від λ та $\overrightarrow{c_0} = (c_{01}, c_{02}, ..., c_{0n}) \in \mathcal{X}^n$. Тоді $\mathcal{C}$-схему можна вважати коректною з i-етапами, якщо виконується наступне твердження:

$$\textbf{DEC}(sk, \textbf{EVAL}(evk, \mathbf{C}_{i,n}, \overrightarrow{c_0})) = \mathbf{C}_{i,n}(\textbf{DEC}(sk, \overrightarrow{c_0}))$$

Хоча це визначення і дуже схоже на визначення коректності розрахунків 1.1.3, проте важливо розуміти, що наведене вище визначення застосоване до розрахунків з багатьма етапами, про що свідчить $\mathbf{C}_{i,n} = \{C_{k,l}\}$.

На Рис. 1.2 зображений приклад розрахунків з етапами, де $i = 2, n = 2$.

Тепер, маючи загальне визначення коректності гомоморфного шифрування, можна описати більш часткові випадки шифрування з i-етапами, а саме: i-етапне, мульті-етапне, полі-етапне та ∞-етапне.

Означення 1.1.12 (i-етапна $\mathcal{C}$-схема [18])**.** Нехай $i \in \mathbb{N}$, тоді $\mathcal{C}$-схема i-етапна, якщо вона коректна для всіх j-етапних схем, де $1 \leq j \leq i$.

Замість того щоб параметризувати етапи числом, як в визначенні i-етапної схеми: $i \in \mathbb{N}$, етапи можуть залежати від полінома параметризовані λ.

Означення 1.1.13 (мульті-етапна $\mathcal{C}$-схема [18])**.** Нехай p - деякий поліном, тоді $\mathcal{C}$-схема називається мульті-етапною, якщо вона коректна для всіх j-етапних схем, таких що: $1 \leq j \leq p(\lambda)$.

Означення 1.1.14 (полі-етапна $\mathcal{C}$-схема [3])**.** Нехай p - деякий поліном, та $\alpha \in \mathcal{A}$, тоді $\mathcal{C}$-схема називається полі-етапною, якщо вона коректна для всіх j-етапних схема, таких що: $1 \leq j \leq p(\lambda, \alpha)$.

Означення 1.1.15 (∞-етапна $\mathcal{C}$-схема)**.** $\mathcal{C}$-схема називається ∞-етапною, якщо вона коректна для всіх j-етапних схем для всіх j.

1.1.4 Наслідки та об'єднання визначень

Тепер ми детально розглянемо наслідки визначень, наведених у попередньому розділі. Спочатку ми повернемося до питання компактності та її двох, здавалося б, окремих визначень.

Існує різниця між визначенням компактності яке було запропоноване в роботі Gentry [16], та між визначенням 1.1.4, цей розділ узгоджує ці два визначення.

Для більшості результатів вимагається, щоб допоміжний параметр генерації ключа α був поліноміально обмежений λ, хоча для всіх реалізованих схем це і так, формальної гарантії цього нема, тому в далі описаних визначеннях це буде вимагатись явно.

Означення 1.1.16 (Gentry-компактність [16])**.** $\mathcal{C}$-розрахункова схема вважається компактною за Gentry, якщо існує поліном f, такий що, для кожного значення параметра безпеки λ, алгоритм дешифрування може бути виражений у вигляді Булевої схеми C_{Dec} розміром максимум $f(\lambda)$.

Означення 1.1.17 (Gentry-компактно розрахункова схема [16])**.** $\mathcal{C}$-розрахункова схема вважається компактно разрахунковою за Gentry, якщо для всіх Булевих схем $C \in \mathcal{C}$ виконуються твердження коректності розрахунків (Озн. 1.1.3), та коректності дешифрування (Озн. 1.1.2), та вона Gentry-компактна.

Розмір Булевої схеми, це число логічних вентилів, або ж якщо представляти схему як граф, то це число вершин.

На перший погляд не зрозуміло як два визначення компактності можуть бути узгоджені:

Твердження 1.1.1. Нехай λ поліноміально обмежена λ. $\mathcal{C}$-розрахункова схема Gentry-компактно розрахуновує $\mathcal{C}$, тоді і тільки тоді, коли схема компактно обчислює $\mathcal{C}$.

Доведення цієї теореми можна знайти в додатках до роботи Armknecht та інші. [3].

Твердження 1.1.2. $\mathcal{C}$-розрахункова схема з ідеальною конфіденційністю схеми, передбачає компактність, коли α поліноміально обмежена α.

Термін *ідеальна конфіденційна схема* відноситься до визначення 1.1.6 де розподіл $\mathcal{X} = \mathcal{Y}$, або ж абсолютно нерозрізнений. Доведення цієї теореми можна знайти в додатках до роботи Armknecht та інші. [3].

1.1.5 Зв'язок FHE та і-етапних схем

Нехай α поліноміально обмежена λ, в цій секції будуть представлені результати, що стосуються зв'язку FHE схем та і-стрибкових схем, припускаючи що α поліноміально обмежена λ.

Твердження 1.1.3. Повна гомоморфна (Озн. 1.1.9) $\mathcal{C}$-розрахункова схема, зі статистично конфіденційною схемою - мульті-етапна (Озн. 1.1.13).

Доведення теореми можна знайти в роботі Armknecht [3].

Тепер ми дослідимо зв'язок між повністю гомоморфною та і-етапою схемою, зазначивши, які властивості повинна мати частково гомоморфна схема шифрування, щоб бути повністю гомоморфною. По-перше, ми дослідимо, за яких умов повністю гомоморфна схема допускає нескінченну кількість етапів обчислень:

Твердження 1.1.4. Частково гомоморфна (Озн. 1.1.7) $\mathcal{C}$-розрахункова схема з ідеальною конфіденційністю схеми - ∞-етапна (Озн. 1.1.15).

Доведення: Оскільки схема має ідеальну конфіденційність (Озн. 1.1.6), то розподіл результатів виконання **EVAL** ідентичний до розподілу виконання схеми над *чистим текстом*, або ж: $(\mathcal{X} = \mathcal{Y} = \mathcal{Z})$. Отже результат виконання **EVAL** знову стає валідними вхідними даними, і не залежить від того скільки разів були виконані розрахунки.

Теорема 1.1.4 наводить на іншу теорему в оберненій формі:

Твердження 1.1.5. Частково гомоморфна схема з Булевими схемами $C \in \mathcal{C}$, які містять тільки NAND вентилі, ідеально конфіденційна та ∞-етапна (Озн. 1.1.15) - повна гомоморфна (Озн. 1.1.9).

Доведення: Оскільки схема ідеально конфіденційна, то за Теоремою 1.1.2, вона компактна, тоді для того щоб показати що вона повна комоморфна за визначенням (Озн. 1.1.9) треба показати що вона може обчислювати будь-яку булеву схему. Припустимо обернене, нехай існує Булева схема C, яку схема

не може коректно обчислювати. Представимо що схема C складається лише з NAND вентилів. Оскільки NAND $\in C$ та схема ∞-етапна, ми можемо коректно обчислити кожен NAND вентиль на відповідному вході, незалежно від того який рівень ітерації обчислення має цей вхід. Таким чином, ми знайшли спосіб коректно розрахувати Булеву схему використовуючи $\mathcal{C}$-схему, або ж $C \in \mathcal{C}$, що суперечить припущенню і показує що схема повністю гомоморфна.

Таким чином можна побудувати наступну діаграму, яка показує необхідні вимоги для виконання теореми, та її результат:

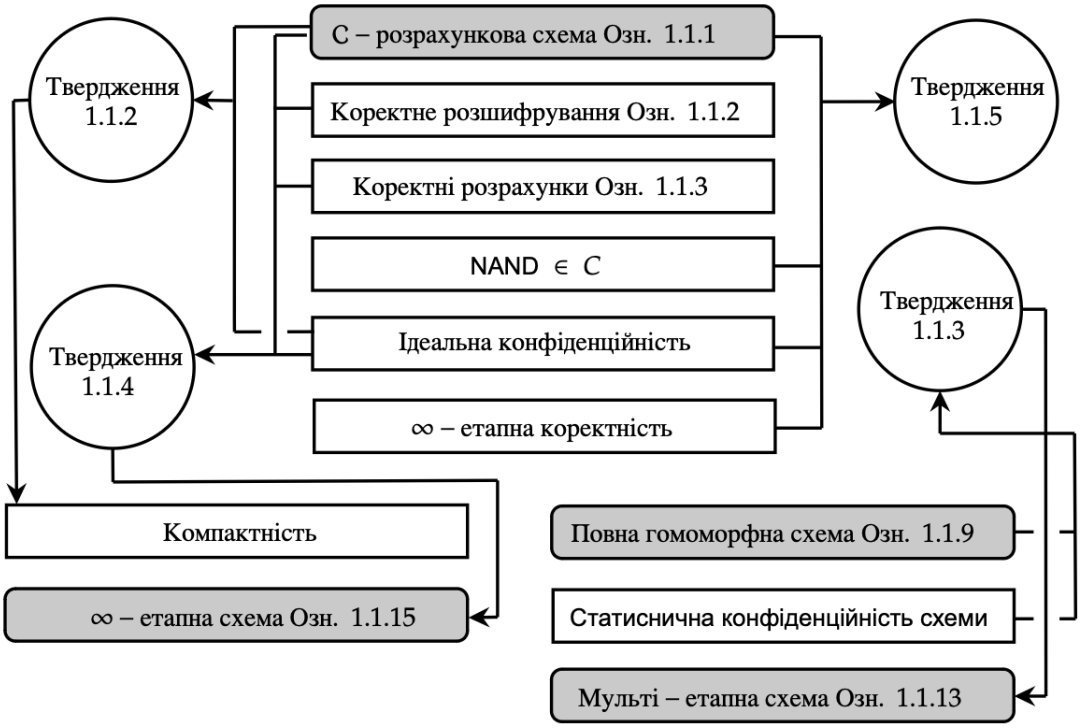

Рис. 1.3: Зв'язок необхідних вимог для теорем, та їх результатів

1.2 Обмеження

Існує велика кількість програм які використовують FHE для вирішування поставленої задачі. Проте наразі існую обмеження у використання цієї технології, далі в цьому розділі буде розглянуто декілька з них.

- Перше обмеження FHE, це жорстка прив'язка пар ключів, що унеможливлює багатьом користувачам використовувати спільні дані. Уявимо

ситуацію де багато користувачів використовують систему яка, в свою чергу покладається на внутрішню базу даних для обчислень. Дані, які були додані в базу даних, можуть бути використані в обчисленнях, тільки користувачем який їх туди додав. Тобто в ситуації коли багато користувачів повинні працювати над одними даними, щоб досягти спільної цілі - FHE обмежений. Проте існує гарний претендент на вирішення цього обмеження [26] Multikey Fully Homomorphic Encryption.

- Друге обмеження це те що FHE потребує дуже великих витрат на обчислення. Розрахунки та проведення операцій над зашифрованими даними виконуються набагато довше ніж на чистих незашифрованих даних. Хоча сучасні алгоритми FHE показують достойні покращення в часі на розрахунки, однак ця проблема все ще залишається одним із головних аргументів не використовувати FHE. Запропоноване часткове розв'язання цієї проблеми, це використовувати Тьюрінг Машини замість Булевих схем [19].

- Третє обмеження полягає в тому що алгоритм розрахунків над зашифрованими, даними не може бути зашифрований сам по собі. Тому, наприклад, маючи складний алгоритм розрахунків акцій, який не повинен бути оприлюднений, складно використати в контексті FHE. Часткове рішення цієї проблеми було запропоноване в роботі Michael Naehrig [27], де він запропонував передавати функцію у зашифрованому вигляді. Проте шифрування алгоритму це не зовсім область відповідальності FHE, і повинна досягатись шляхом обфускації алгоритмів.

1.3 Відомі області застосування

В цьому розділі описані можливі області де може бути ефективно застосовані FHE. Будуть описані як і повноцінні області де може бути застосована технологія, тек і використання як допоміжного інструменту, для побудови більш комплексних систем.

Конфіденційність користувача у рекламних пропозиціях

В сучасному світі реклама може бути не тільки набридливою для користувача, а напроти дуже корисною, якщо алгоритми для підбору цієї реклами базуються на персональних даних, користувача, таких як: його вподобання, рік народження, перегляд певних ресурсів та джерел, локація користувача тощо. Більшість людей відносяться до персональної безпеки дуже відповідально, і не хочуть її розголошувати задля отримання персоналізованої реклами.

Для вирішення цієї проблеми чудово підходить FHE, оскільки він може виконувати алгоритми над зашифрованими даними користувача.

В документі [28] була описана одна з таких систем, де рекомендації для користувача основані на рекомендаціях його друзів. Система застосовує гомоморфне шифрування, щоб була можливість отримувати рекомендації друзів буз розголошення їх особистостей.

Інша реалізація задачі була описана в документі [4]. В реалізації користувач отримує рекомендації від системи, якій не важливо який контент їй був переданий, та від якого користувача. Для побудови такого алгоритму, була зроблена проста, але дуже ефективна FHE схема, яка дозволяє отримувати рекомендації для користувача, який залишається невидимим для системи.

Ще одна робота, яка варта згадки [27], забезпечує рекламні рекомендації на базі локації користувача, виконуючи алгоритми над зашифрованими даними, що не дозволяє зловмисникам отримати, де знаходиться користувач системи.

Конфіденційність даних пацієнта у медичних застосунках

В роботі [27], описане практичне використання FHE в медичних застосунках, де важлива конфіденційність даних пацієнта. В описаній системі пацієнт робить запит до системи зі своїми даними в зашифрованій формі, оскільки користувач системи це власник даних, то тільки він може їх розшифрувати. Застосунок який реалізує сервіс, в свою чергу, може рахувати, чи отримувати з баз даних, інформацію, таку як: група крові, тиск, серцебиття, хвороби та інше. за зашифрованими даними, результат роботи сервісу, також буде зашифровані дані.

В роботі [25] була реалізована подібна система, яка рахує вірогідність сер-

цевого нападу, основуючись на зашифрованих даних пацієнта.

Інтелектуальний аналіз даних

Аналіз даних на великих обсягах інформації дає гарний результат, проте ціна цьому результату приватність даних користувачів.

Була зроблена чудова робота [32] яка реалізує логіку зашифрованого аналізу даних, без втрати точності, і забезпечує безпеку за допомогою FHE схеми.

Конфіденційність фінансових операцій

Хоча як було описано в розділі про обмеження, FHE і не забезпечує шифрування самого алгоритму, його можна використовувати в іншому сценарії:

Представимо, що існує дві компанії X та Y, у компанії X є приватні акції, а у компанії Y є секретний алгоритм який рахує прогноз по динаміці змін акцій. Тоді компанії X достатньо застосувати FHE для своїх даних, що дозволить рахувати над ними алгоритм компанії Y, без дешифрування.

Криміналістичне розпізнавання зображень

В роботі [6] була представлена ще одна чудова сфера застосування FHE. Поліція та інші правоохоронні органи використовують подібні інструменти для пошуку нелегальних фотографій на жорстких дисках, у мережевих потоках даних та інших наборах даних. Поліція використовує базу даних "поганих"хеш-значень зображень. Можливість того, що злочинці можуть отримати доступ до цієї бази даних, перевірити, чи будуть їхні фотографії розпізнані, і, якщо так, змінити їх, викликає серйозне занепокоєння.

Ця схема реалізує сценарій, коли база даних поліції зашифрована, але водночас законний мережевий трафік компанії залишається приватним завдяки використанню дещо гомоморфної стратегії шифрування, запропонованої в наступному документі [9]. Компанія протиставляє хешований і зашифрований потік фотографій зашифрованій базі даних поліції. Тимчасова змінна надається поліції через заздалегідь визначений проміжок часу або поріг, при цьому постачальник послуг нічого не дізнається про саму зашифровану базу даних.

Хмарні обчислення з захищеними даними

Гомоморфне шифрування дозволяє використовувати хмарні ресурси для обробки даних, не розкриваючи їх змісту. Користувач може зашифрувати дані та передати їх в хмару для виконання обчислень. Хмарний провайдер, використовуючи FHE, може виконати різні операції над зашифрованими даними, такі як пошук, фільтрація або обчислення агрегатних функцій, не розкриваючи вмісту даних. Результати обчислень повертаються користувачу, який може розшифрувати їх і отримати оброблені дані.

Туманні обчислення

FHE може бути використана в туманних обчисленнях для забезпечення безпечного виконання обчислень на краю мережі (edge computing). Це особливо важливо в сферах, де важливо зберегти конфіденційність даних, наприклад, в медичних додатках, додатках Інтернету речей (IoT) або системах безпеки.

Використання FHE як інструмент для більш комплексних криптосистем

Далі наведені приклади, як FHE може бути використаний як інструмент, для створення більш складних криптографічних інструментів, таких як: цифрові підписи, MACs, доведень з нульовим розголошенням, та інш.

Делегування розрахунків

Делегування обчислень - це другий великий стовп хмарних обчислень, окрім делегування даних. Користувач може захотіти делегувати обчислення функції f серверу. Однак сервер може бути зловмисним або просто схильним до збоїв, тобто користувач може не довіряти результатам обчислень. Користувач хоче мати доказ того, що обчислення було виконано правильно, і перевірка цього доказу також повинна бути значно ефективнішою, ніж виконання обчислень користувачем.

Одним із прикладів делегування обчислень є автентифікатори повідомлень. Користувач, який делегував обчислення для набору даних, може за-

хотіти перевірити, що результат є дійсно правильним. Тег має бути незалежним від розміру вихідного набору даних, і його може перевірити лише власник приватного ключа. Одна з таких схем була запропонована в роботі [15], яку можна розглядати як симетрично-ключову версію повністю гомоморфних підписів.

Цифрові підписи

В документі [20], вперше була запропонована конструкція підпису, на базі рівнево-повної гомоморфної схеми. Схема може розраховувати довільні Булеві схеми з максимальною глибиною d над підписаними даними і гомоморфно генерувати короткий підпис, який може бути перевірений будь-ким за допомогою відкритого ключа перевірки.

Багатопарне обчислення

Багатосторонні обчислення вимагають взаємодії між учасниками. Робота [13] надає опис того, як частково гомоморфна схема може бути використана для побудови автономного множення під час обчислень. Користувачі використовують дещо гомоморфну схему на етапі попередньої обробки, але повертаються до набагато ефективніших методів багатосторонніх обчислень на етапі обчислень. Користувач завантажує підписані дані x, потім сервер виконує над ними деяку функцію g, яка дає $y = g(x)$. Крім того, сервер публікує підпис $\tau_{g,y}$ для перевірки обчислень.

Ця робота також вводить поняття гомоморфних функцій лазівки (HTDF), одного з будівельних блоків для побудови підпису. Самі HTDF базуються на проблемі малого цілочисельного розв'язку (SIS).

1.4 Існуючі FHE схеми

В цьому розділі будуть поверхнево описані існуючі FHE схеми. Будемо вважати початковою точкою розвитку гомоморфних схем, роботу Gentry [16], до цієї роботи не відбувалось значного впливу на розвиток технологій.

Деякі схеми, які не підпадають під визначення компактності не будуть представлені, тому що вони малозначущі в сучасному світі, такими схемами

можна назвати: Fellows та Koblitz [14]. Деякі схеми підпадають під визначення компактності, але обмежені по операції, наприклад схема Bohen-Goh [5]. Такі схеми не будуть розглядатись оскільки вони не вважаються повними гомоморфними за визначенням 1.1.9.

- **Gentry FHE [16]**
- **FHE через цілі [31]**
- **Пакетне повністю гомоморфне шифрування над цілими числами [11]**
- **Ефективна форма FHE через LWE [8]**
- **Повністю гомоморфне шифрування над LWE-кільцями та захистом повідомлень, що залежать від ключа [9]**
- **Повністю гомоморфне шифрування без перезавантаження (Озн. 1.4.1) [7]**

1.4.1 Перезавантаження схеми та альтернативи

Ключовими моментами в створенні FHE схем є техніка перезавантаження схеми, яка була представлена Gentry [16]. Схема представлена в його документі вважалась зашумленою, тобто, *чистий текст* був схований за шумом, який пізніше знімався операцією розшифровування. Проте цей шум збільшувався з кожною операцією над зашифрованим текстом, і коли цей шум досягав критичної точки, то розшифровування тексту унеможливлювалось.

Щоб вирішити цю проблему, в документі була представлена техніка *перезашифровування*, логіка якої було зашифровування, вже зашифрованого тексту, а при дешифруванні, спочатку знімається верхній шифр, поки тест не стане *чистим*. Тобто, якщо алгоритм розрахунків, може впоратись з процесом дешифрування + NAND вентилем, можна досягти прогресу в розрахунках схеми, які необхідні.

Означення 1.4.1 (Схильна до перезавантаження $\mathcal{C}$-схема)**.** Якщо $\mathcal{C}$-схема, може гомоморфно обчислювати свою власну Булеву схему дешифрування, плюс ще одну операцію NAND, то її можна вважати схильну до перезавантаження.

Основне питання яке виникає до вище наведеного визначення: Чи не порушує безпеку публікація зашифрованого приватного ключа (необхідного для дешифрування) під його власним публічним ключем?

Якщо ми припустимо, що безпечно публікувати шифрування секретного ключа під відповідним йому відкритим ключем, ми досягнемо повністю гомоморфного шифрування і навіть і-рівневого. Таке припущення називається циклічною безпекою. Однак, якщо циклічна безпека не працює, то однією з можливостей є використання ланцюжка пар відкритий ключ/секретний ключ, де секретний ключ завжди зашифрований під наступним відкритим ключем. Це дозволяє відповідним дещо гомоморфним схемам стати гомоморфними за рівнем, де рівень залежить від кількості пар ключів.

Альтернативний спосіб досягнення гомоморфного шифрування належить Brakerski [8]. Проблема все ще полягає в тому, як керувати шумом, але цього разу це досягається за рахунок зменшення модуля простору зашифрованого тексту разом із шумом.

Параметр безпеки, який диктує, наскільки малим може бути модуль, дає обмеження на кількість рівнів. Цей напрямок роботи призводить до нативних рівневих гомоморфних схем. Однак автори зазвичай зазначають, що можна застосувати перезавантаження як оптимізацію, а також як засіб для отримання повністю гомоморфної схеми і-рівнів, знову ж таки, припускаючи кругову безпеку.

Розділ 2
Використання FHE в хмарних технологіях

2.1 Бібліотека HeLib

Для реалізації поставленої задачі буде використовуватись бібліотека гомоморфного шифрування HeLib. HeLib була написана на C++ та реалізовує функціонал Brakerski-Gentry-Vaikuntanathan (BGV), та Cheon-Kim-Kim-Song (CKKS) схем.

З середини 2018 року HElib знаходиться на стадії інтенсивного рефакторінгу для підвищення надійності, розширяємості, продуктивності та, найголовніше, легкості у використанні для дослідників та розробників, які працюють з HE та його застосуванням.

Нижче представлені деякі ключові аспекти того, як HElib реалізує схему FHE:

- **Шифрування на основі поліномів**: HElib використовує схему шифрування на основі поліномів, де відкриті текстові повідомлення (*чистий текст*) подаються у вигляді поліномів над скінченним полем. Процес шифрування полягає у перетворенні текстового повідомлення у поліном, а потім у застосуванні до нього деяких математичних операцій.

- **Генерація ключів**: HElib генерує набір ключів, необхідних для шифрування, розшифрування та гомоморфних операцій. Ці ключі включають секретний ключ, відкритий ключ та розрахункові ключі. Секретний ключ зберігається у таємниці, тоді як відкритий ключ надається стороні, яка виконує обчислення. Розрахункові ключі використовуються для виконання гомоморфних операцій над зашифрованими даними.

- **Шифрування**: Щоб зашифрувати відкрите повідомлення, HElib перетворює його у поліном і застосовує операції шифрування з викори-

станням відкритого ключа. Цей процес включає додавання шумового члена до полінома для забезпечення безпеки.

- **Гомоморфні операції**: HElib підтримує різні гомоморфні операції, такі як додавання, множення та обертання. Ці операції дозволяють виконувати обчислення над зашифрованими даними без їх розшифрування. Для виконання цих операцій HElib використовує математичні методи, такі як поліноміальне множення та модулярна арифметика.

- **Розшифрування**: Щоб розшифрувати результат обчислень, HElib використовує секретний ключ і застосовує операції дешифрування. Шумовий член, доданий під час шифрування, зменшується під час розшифрування, щоб отримати вихідне повідомлення у відкритому вигляді.

- **Оптимізація продуктивності**: HElib включає декілька оптимізацій продуктивності для підвищення ефективності обчислень FHE. Ці оптимізації включають такі методи, як перемикання модулів, завантаження та пакування декількох значень відкритого тексту в один зашифрований текст.

2.1.1 Алгоритми над схемою

- **Гомоморфне додавання:** Гомоморфне додавання дозволяє додавати два поліноми зашифрованого тексту, в результаті чого виходить новий поліном зашифрованого тексту, що представляє собою суму відповідних поліномів відкритого тексту.

  Процес гомоморфного додавання полягає у додаванні коефіцієнтів поліномів зашифрованого тексту за модулем скінченного поля. Математично, для двох поліномів шифрованого тексту $C_1(x)$ і $C_2(x)$, що представляють поліноми відкритого тексту $P_1(x)$ і $P_2(x)$ відповідно: $C_1(x) + C_2(x) = (P_1(x) + P_2(x)) \ mod \ q$, де q - модуль скінченного поля.

  Оскільки додавання є простою операцією, то додавання поліномів зашифрованого тексту можна виконувати безпосередньо, член за членом, без необхідності розшифровувати поліноми. Ця властивість дозволяє проводити обчислення над зашифрованими даними без розкриття оригінального відкритого тексту.

- **Гомоморфне множення:** Гомоморфне множення дозволяє перемножити два поліноми зашифрованого тексту, в результаті чого новий поліном зашифрованого тексту є добутком відповідних поліномів відкритого тексту.

  Процес гомоморфного множення є більш складним і вимагає додаткових кроків:

  1. **Перемноження**: Для множення двох поліномів зашифрованого тексту виконується операція множення поліномів. Ця операція передбачає множення коефіцієнтів поліномів разом, член за членом. Результуючий поліном являє собою поточковий добуток поліномів відкритого тексту.
  2. **Зведення до спільного степеню за модулем**: Після множення многочленів, результуючий многочлен може мати більш високі степені через процес множення. Щоб зберегти цілісність схеми шифрування, отриманий поліном потрібно зменшити по модулю на певний коефіцієнт. Цей множник часто пов'язаний з модулем q скінченного поля.
  3. **Обмін ключами**: Для продовження виконання гомоморфних операцій над отриманим поліномом застосовується процес, який називається *обмін ключів*. Обмін ключів дозволяє перетворити ключ шифрування, який використовувався під час шифрування, на ключ розрахунків, що дає змогу продовжити гомоморфні обчислення.

- **Гомоморфне обертання:** Гомоморфне обертання дозволяє зміщувати коефіцієнти полінома зашифрованого тексту, тим самим змінюючи порядок даних у поліноміальному поданні. Ця операція особливо корисна для виконання операцій над підмножинами даних або перестановки полінома відповідно до бажаних обчислень.

  Для досягнення гомоморфного обертання можна використовувати різні методи, такі як операції модулярної арифметики, матриці перестановок або інші специфічні для шифрування методи. Ці методи дозволяють ефективно зміщувати коефіцієнти, зберігаючи при цьому властивості

шифрування.

Важливо відзначити, що гомоморфні операції вносять шум і можуть впливати на безпеку і точність обчислень. У міру виконання операцій шумовий член накопичується і може вимагати додаткових кроків, наприклад застосування описаної раніше техніки повторного шифрування, щоб зберегти безпеку і правильність обчислень.

Використовуючи гомоморфне додавання, множення і обертання, обчислення можна виконувати над зашифрованими даними без їх розшифровки, що дозволяє проводити обчислення з дотриманням конфіденційності, зберігаючи при цьому конфіденційність відкритих текстових повідомлень.

2.2 Реалізація застосунку

Для демонстрації коректності роботи, та можливості застосування FHE схеми у хмарних технологіях, був реалізований клієнт-серверний застосунок. Застосунок повинен реалізовувати логіку банківської системи, тобто клієнт повинен мати можливість створити рахунок, додати баланс, та зняти баланс з рахунку. Окрім цього, система бути розроблена з розрахунком на те що вона буде знаходитись на хмарі, та між клієнтом і хмарою буде незахищене з'єднання. Також, користувач банківського сервісу хоче бути впевнений в повному захисті даних, в тому числі від сервера. Для реалізації вимог для безпеки даних буде використана FHE з реалізацією BGV схеми в бібліотеці HeLib. Див. Рис 2.1.

2.2.1 Вимоги

Ця секція описує вимоги до розробки банківської системи, яка повинна забезпечувати коректність, та безпеку даних.

Клієнт

Основна задача клієнта, це створити необхідний контекст для FHE шифрування, ініціювати з'єднання з сервером, робити необхідні запити, та обробляти відповіді від сервера.

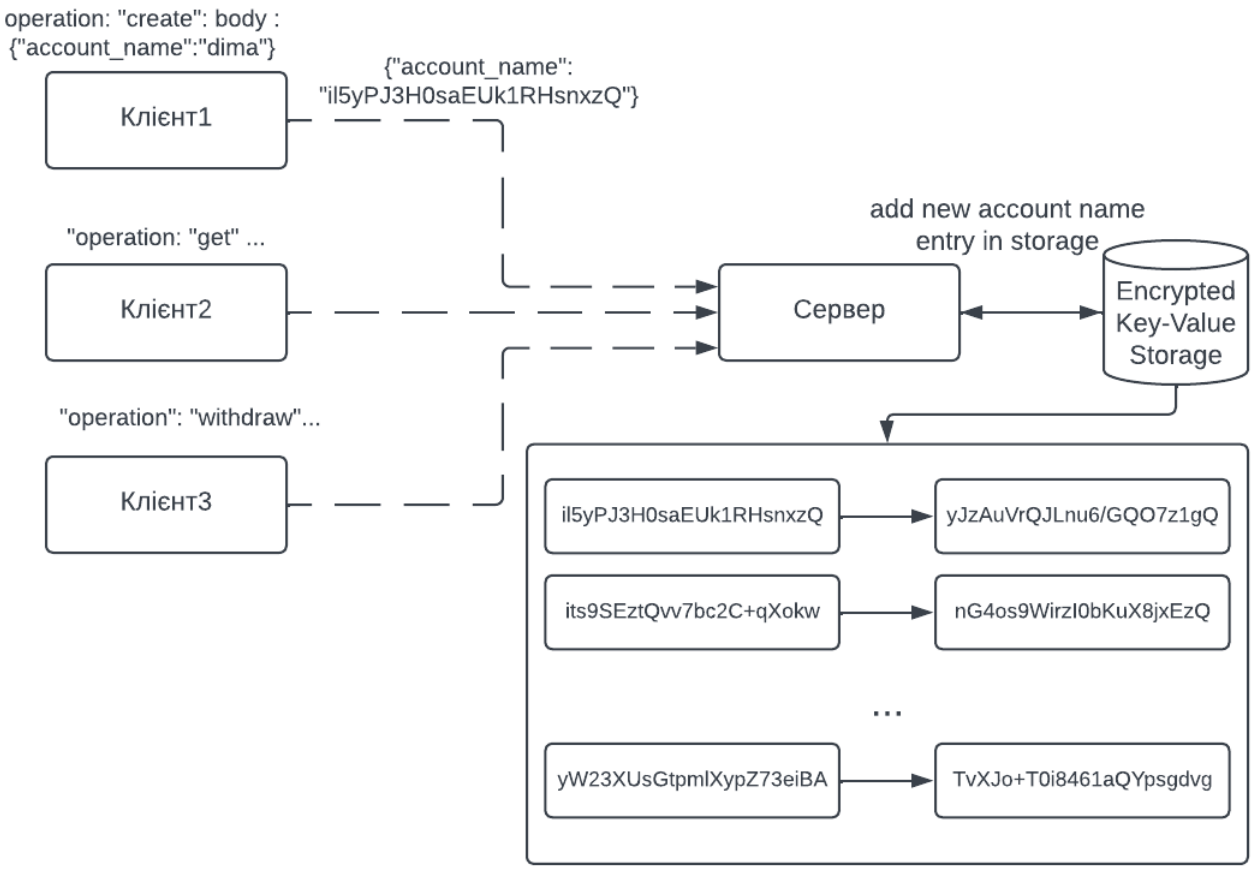

Рис. 2.1: Демонстрація запитів клієнтів до серверу, де перервана стрілочка, означає незахищене середовище, тому данні в ньому будуть зашифровані

Як було написано вище, клієнт повинен бути відповідальний за створення контексту шифрування. Під контекстом, будемо вважати необхідні дані для правильного працювання FHE, зашифровування та розшифровування даних. Деяку частину контекста необхідно буде надати серверу для того, щоб він мав можливість виконувати розрахунки над зашифрованими даними, проте частина контекста повинна бути нерозголошуєма, така як приватний ключ.

Контекст буде зберігатись на машині клієнта у файлі Json формату, і при ініціюванні з'єднання з сервером, у клієнта повинна бути можливість вибору, чи створювати новий контекст, чи використати присутній.

Нехай контекст був згенерований з деякими тестовими параметрами тоді його Json, та приватний ключ буде виглядати наступним чином:

```
{"HElibVersion":"2.2.0","content":{"alsoThick":false,"
    build_cache":false,"digits":[[6,"..."],[11,"..."]],"
    ePrime_param":4,"e_param":12,"gens":[2341,3277,911],"
```

```
hwt_param":120,"m":4095,"mvec":[7,"..."],"ords
":[6,4,6],"p":2,"qs":[249047285761,"..."],"r":1,"scale
":10.0,"smallPrimes":[0,"..."],"specialPrimes
":[15,"..."],"stdev":{"exponent":0,"mantissa":3.2}},"
serializationVersion":"0.0.1","type":"Context"}
```


Лістинг 2.1: Json репрезентація, контексу, деякі довгі послідовносиі чисел були замінені на трикрапку, щоб зменшити кількість тексту


```
{"HElibVersion":"2.2.0","content":{"b":[{"map":[[ large
amount of 64-bits integers]],"set":[6,7,...]}]},"
serializationVersion":"0.0.1","type":"SecKey"}
```


Лістинг 2.2: Json репрезентація приватного ключа

Клієнт повинен правильно серіалізувати свій контекст, для подальшого відправлення серверу, та приватний ключ, для зберігання.


```
{
    "public_context": "Listing 2.1",
    "private_key": "Listing 2.2"
}
```


Лістинг 2.3: Json формат зберігання приватного контексту на клієнті, для можливості його подальшого використання

Для того щоб клієнт мав можливість перевикористовувати створений ним контекст, був розроблений додатковий скрипт, який зберігає публічний та приватний контекст в файл. Клієнт при ініціюванні з'єднання з сервером моє можливість вибрати файл з контекстами, або ж використати новий (згенерований).

Сервер

Задача серверної частини застосунку, це приймати TCP/IP з'єднання, та коректно обробляти запити від клієнта. Окрім глобальної задачі сервера, він повинен виконувати якусь логіку.

Також для зручного використання на хмарі, сервер повинен мати інструмент автоматичної розгортки, для цього буде використана технологія контейнеризації Docker.

Фактично, логіку серверу можна поділити на 2 частини:

- **Пошук сутності в зашифрованій базі даних**: Хоча задача пошуку даних в базі даних, може здатись тривіальною задачею, коли сервер працює повністю над зашифрованими даними, задача стає дещо складнішою, оскільки шифрування одних і тих самих даних, може давати різний результат кожен раз, тому просте зіставлення з даними в базі не дасть коректного результату.

  Для коректного знаходження даних в базі, за ключем буде використовуватись наступний алгоритм:

  Нехай у нас є сховище ключ-значення Рис. 2.2.

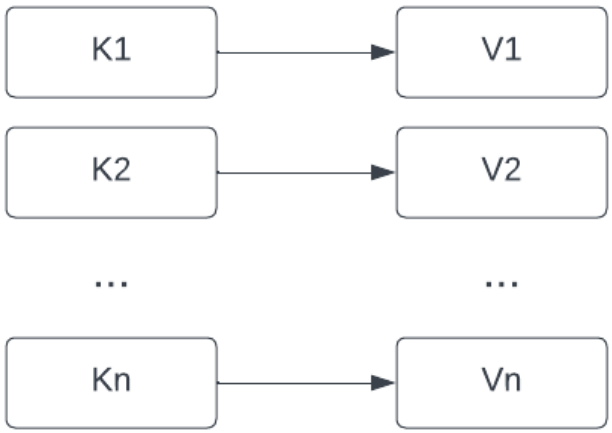

Рис. 2.2: Де K- це множина ключів, та $k_1, k_2, ..., k_n \in K$, $v_1, v_2, ..., v_n \in V$, де V- це множина значень, а n-кількість сутностей в базі. Тобто відображення $K \to V$ є бієктивним.

Високорівнево, алгоритм пошуку в базі, над зашифрованими даними описаний на заображенні Рис. 2.3.

Далі буде описано більш детально алгоритм пошуку за ключем в зашифрованій базі даних.

Першим кроком алгоритму є обчислення операції різниці між запитом і ключами бази даних. Це проста операція віднімання, яка поелементно виконує віднімання у структурі, схожій на масив. В результаті буде отримано різницевий шифротекст, який ми позначимо як Δ_i де $\Delta_1, \Delta_2, ..., \Delta_n \in \delta$. Наразі $\Delta = 0$ якщо $k_q \in K$, і ненульовому значенню в іншому випадку, див. Рис. 2.4.

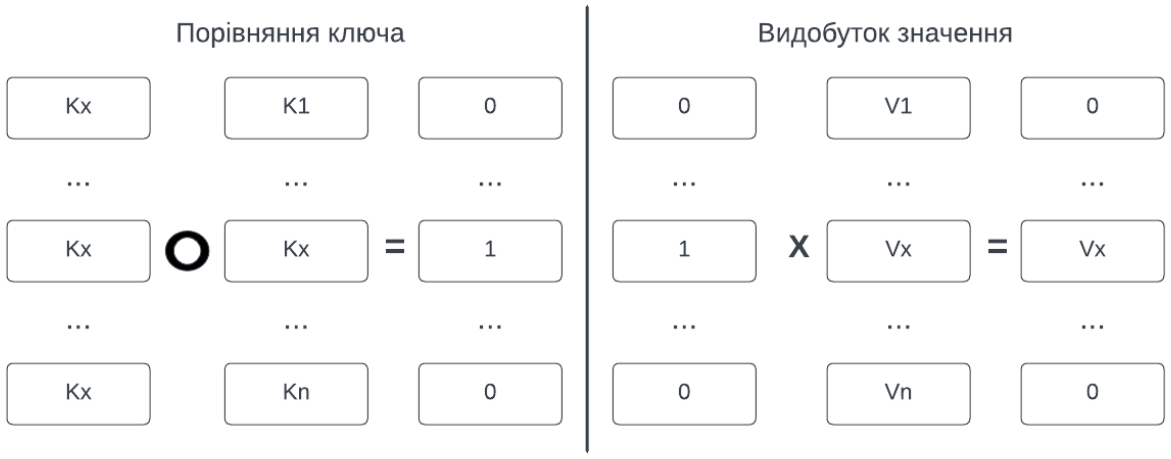

Рис. 2.3: Спрощений алгоритм пошука значення $V_x \in V$, за ключем $K_x \in K$

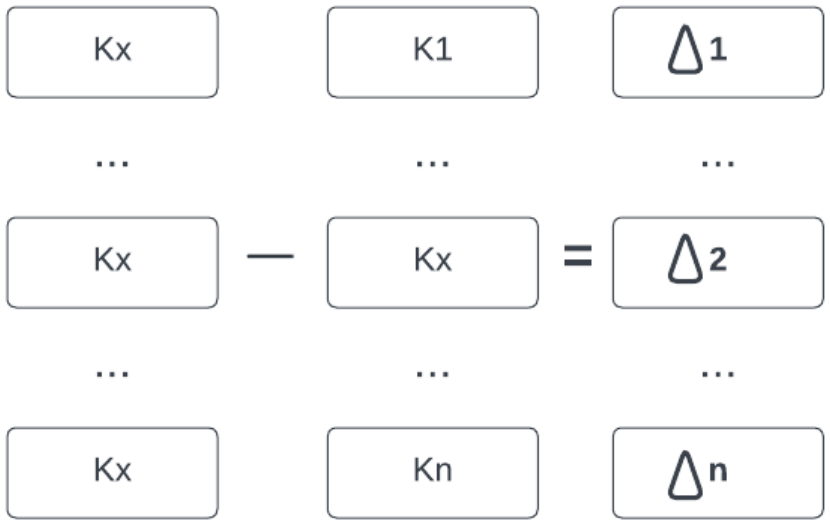

Рис. 2.4: Віднімання ключів, щоб отримати $\Delta_1, \Delta_2, ..., \Delta_n \in \delta$. Де у випадку $K_q \in K\ \Delta_q = 0, \Delta_i \neq 0, i \neq q$

.

Це не зовсім та маска, яка нам потрібна, тому ми повинні виконати іншу операцію, описану далі. Для отримання правильної маски, треба буде застосувати малу теорему Ферма [1] до $\Delta_i \in \delta$. Застосувавши теорему, ми отримаємо наступний результат:

$$\mathrm{LTF}(\Delta_i) = \begin{cases} 1, & \Delta_i \neq 0 \\ 0, & \Delta_i = 0 \end{cases}$$

Проте, ця маска дає обернений результат, щоб отримати коректні значення, треба застосувати операцію інверсії:

$$1 - \mathrm{LTF}(\Delta_i) = \begin{cases} 0, & \Delta_i \neq 0 \\ 1, & \Delta_i = 0 \end{cases}$$

Якщо описати процес знаходження над зашифрованими даними $\mathbf{ENC}(x)$, то це буде виглядати так, як зображено на Рис. 2.5.

Спочатку ми застосовуємо операцію малої теореми Ферма (FLT) [1] до кожного різницевого зашифрованого тексту. Це призводить до шифрування нуля, $E(0)$, якщо різниця дорівнює нулю, тобто є збіг, і шифрування одиниці, $E(1)$, в іншому випадку.

Далі ми використовуємо попередньо обчислені результати операції FLT і віднімаємо це значення від 1. Це значення 1 може бути чистим, оскільки будь-яка операція між зашифрованим текстом і відкритим текстом призводить до зашифрованого тексту. Це призводить до відображення будь-якого ненульового значення в нуль і нуля в одиницю. Таким чином, ми отримуємо маски, які нам потрібні для алгоритму порівняння.

Однак є ще один аспект, який слід взяти до уваги, а саме: як ми повинні діяти з частковими збігами?

Розглянемо наступний сценарій, показаний на зображенні нижче Рис 2.6. Уявіть, що ключ збігається лише з другою літерою запиту і, можливо, з деякими значеннями пропусків, тоді він створить масив, як показано нижче Рис. 2.6.

Оскільки в нашому прикладі нас цікавлять лише точні збіги, цей результат слід вважати таким, що не збігається. Щоб усунути часткові збіги, ми просто копіюємо зашифрований текст, виконуємо обертання

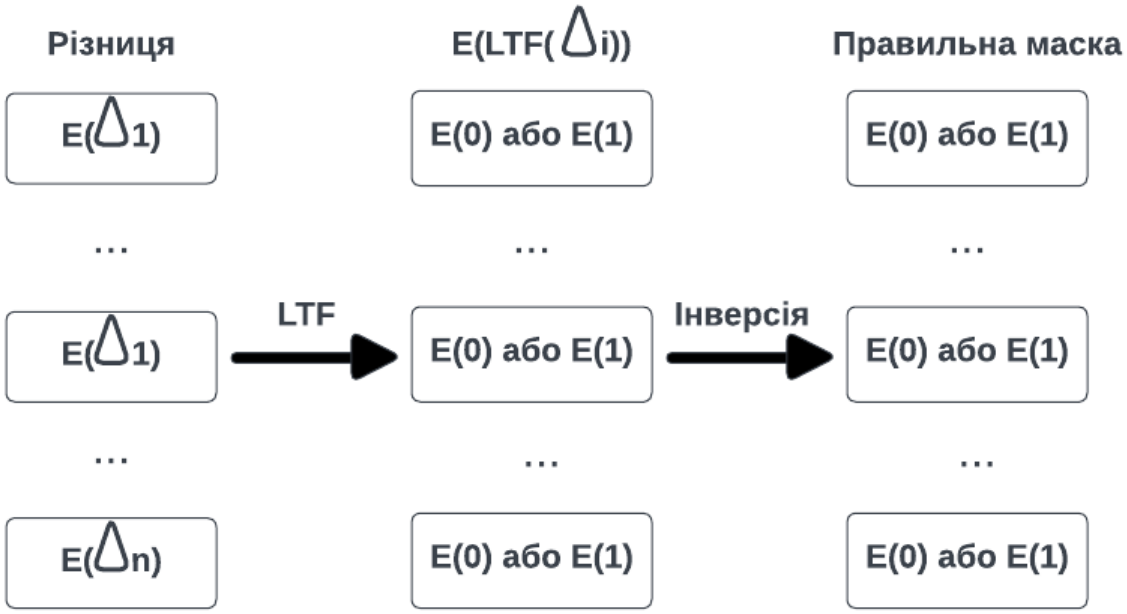

Рис. 2.5: Процес створення правильної маски над зашифрованими даними

структури масиву в копії і перемножуємо його за входом з оригінальною копією. Це означає, що якщо в одному з осередків зашифрованого тексту є хоча б один 0, то цей осередок ефективно обнулить всі інші осередки масиву.

Зауважте, що оскільки ми не можемо знати, який саме шифротекст містить результат, що збігається, частково збігається або не збігається, ця операція також виконується над результатом, що збігається. Однак, оскільки у зашифрованому тексті у кожному слоті має бути 1, то ця операція не повинна мати ніякого ефекту.

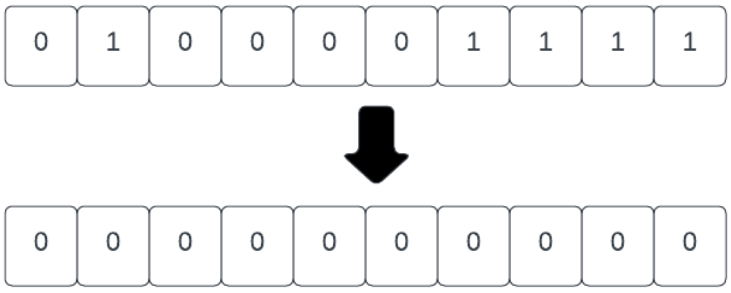

Рис. 2.6: Частковий збіг, який не повинен вважатись коректним

Тепер, коли ми маємо остаточні маски, ми можемо виконати вилучення

даних з бази даних. Цей крок передбачає множення маски на відповідний запис у базі даних. Оскільки наша маска є шифруванням 0, якщо немає збігу, множення її на відповідний запис обнулить цей запис. Крім того, оскільки маска є шифруванням 1, якщо є збіг, множення її на запис поверне сам запис.

Оскільки ключі в нашому прикладі бази даних є унікальними, можна бути впевненим, що на кожен запит буде отримано максимум один унікальний збіг. Використовуючи ці знання, можна об'єднати всі результати кроку вилучення значень в один зашифрований текст. Це пов'язано з тим, що додавання шифрів 0 до значення не змінює саме значення. Це дозволяє економити на зв'язку, оскільки серверу потрібно надсилати клієнту лише один зашифрований текст, а не по одному зашифрованому тексту для кожного запису в базі даних.

Виникає питання: Чому просто не використовувати побайтне порівняння зашифрованого тексту, з ключем? На те є 2 причини: перша і головна, це те що з цим алгоритмом, сервер не може знати чи існує такий ключ в його базі даних чи ні, він просто виконує алгоритм. Друге, це те що контекст може змінитись, наприклад в результаті перешифрування (Озн. 1.4.1), в такому випадку побайтове порівняння не спрацює.

- **Виконання розрахунків над зашифрованими даними**: Кожен біт двійкового числа кодується в один шифрований текст. Таким чином, для 16-бітового двійкового числа ми представимо його у вигляді масиву з 16 унікальних шифротекстів.

$$b_0 = [0]\ [0]\ [0]\ ...\ [0]\ [0]\ [0] \leftarrow \text{шифр для біту 0}$$
$$b_1 = [1]\ [1]\ [1]\ ...\ [1]\ [1]\ [1] \leftarrow \text{шифр для біту 1}$$
$$b_2 = [1]\ [1]\ [1]\ ...\ [1]\ [1]\ [1] \leftarrow \text{шифр для біту 2}$$

цей приклад показує шифр для 3-бітного числа 110b = 6.

Клієнт повинен зашифровувати число (його бінарну репрезентацію) та відправляти на сервер, сервер повинен виконувати операцію над цим зашифрованим числом згідно з вимогами клієнта та перевіряти можливість виконання операції (наприклад що баланс не менше 0).

Комунікація клієнту з сервером (API)

Клієнт повинен мати можливість комунікувати з сервером і робити запит на операції: створити новий рахунок, зняти баланс з рахунку, додати баланс на рахунок, та отримати інформацію про кількість грошей на рахунку. Окрім цього, для деяких операцій, клієнт повинен надати серверу деякі дані, наприклад під час створення рахунку, клієнт повинен відправити FHE контекст для того, щоб сервер міг правильно працювати з цими даними в майбутньому.

Для спрощення імплементації, клієнт буде відправляти та отримувати дані в форматі Json, в якому буде міститись поле про тип операції (create, get, add, withdraw) в полі request (запит), та необхідну інформацію в полі body (тіло).

Також як було описано в Ліст. 2.1. клієнт відповідальний за відправку FHE контексту шифрування, щоб забезпечити можливість серверу коректно працювати з даними клієнта. Важливо зазначити, що цей контекст не дає можливість розшифровувати данні.

В кожному запиті клієнт повинен відправляти зашифровану назву акаунту, окрім цього в деяких операціях необхідна додаткова інформація, наприклад для операцій add, та withdraw необхідна сума балансу для додавання-/зняття.

```
{
    "operation: "add",
    "public_context": "Listing 2.1",
    "private_key": "Listing 2.2"
    "account_name": "FHE Encrypted data",
    "amount": "FHE Encrypted data"
}
```

Лістинг 2.4: Json формат запиту клієнта до сервера

Хоча запит виглядає і досить компактним, на практиці FHE потребує дуже багато ресурсів, тому таке повідомлення буде мати розмір в середньому 6 мегабайт.

Для отримання відповіді сервер збирає повідомлення в залежності від того чи сталась помилка, або все вібдбулось коректно. Наприклад повідомлення яке показує що акаунту не існує, або те що баланс від'ємний, буде виглядати

наступним чином:

```
{
    "operation: "get",
    "status": "error",
    "error_msg": "No access to account",
}
{
    "operation: "withdraw",
    "status": "error",
    "error_msg": "Impossible to withdraw: balance < 0",
}
```

Лістинг 2.5: Json формат помилки клієнту від сервера

2.2.2 Стек технологій для розробки

Для реалізації банківської системи був вибраний наступний стек технологій:

- Мова розробки була вибрана: C++, так як реалізація FHE схем краще всього написана на ній і показують достойну швидкість обчислення
- Для реалізації FHE буде використана бібліотека з відкритим кодом HeLib, з якої буде взята логіка BGV схеми
- Для TCP/IP з'єднання між клієнтом та сервером (хмарою) буде використана бібліотека з колекції Boost: Asio, яка дозволяє виконувати асинхронні операції з IO девайсами, в тому числі в ній знаходиться функціонал для реалізації TCP/IP з'єднань.
- Для більш зручної розробки будуть використані сторонні бібліотеки та інструменти: логування, збірка проєкту, тестування, засоби вимірювання швидкодії.

2.2.3 Результати роботи застосунку

Для коректної роботи з сервером, перш за все треба створити контекст шифрування, для цього необхідно використати скрипт для генерації цього

контексту:

```
[breaklines, caption={Команда генерації нового контексту
шифрування},
captionpos=b]
$> ./code/diploma_code_utils --help
Usage: ./program p m r bits c mvec_size gen_size \
      ords_size output_file [mvec_elements] \
      [gens_elements] [ords_elements]

$> ./code/diploma_code_utils 131 4095 1 1000 2 4 3 3\
     out.context 7 5 9 13 2341 3277 911 6 4 6
```

Згенерує файл Json формата з контекстом та приватним ключем (Лістинг 2.3).

Сервер стандартно запускається на порті 7623

```
> ./code/diploma_code_server
[2023-06-06 14:52:56.427] [debug] Сервер сконфігурований на порті:
7623
[2023-06-06 14:52:56.428] [debug] Сервер запущений
```

Для демонстрації покажемо створення акаунту *test*, та додавання до нього балансу. Клієнтський застосунок виглядає наступним чином:

```
$> ./code/diploma_code_client
Використання: client [COMMAND] --context-path [FILE_PATH]
[ARGUMENTS...]
Команди:
  get           Отримати інформацію про аккаунт
      Параметри: <Назва аккаунту>

  create        Створює новий аккаунт
      Параметри: <Назва аккаунту>

  add           Додати баланс на аккаунт
      Параметри: <Назва аккаунту> <Кількість балансу>
```

```
  withdraw      Зняти баланс з аккаунту
      Параметри: <Назва аккаунту> <Кількість балансу>
$> ./code/diploma_code_client create --context-path out.context
test
[2023-06-06 15:02:00.028] [debug] Зчитування файлу конекста:
out.context
[2023-06-06 15:02:00.726] [debug] Шифрування назви аккаунту...
[2023-06-06 15:02:00.730] [debug] Викликана команда create("test")
[2023-06-06 15:02:00.730] [client] [info] Обробка запиту для
створення акаунту...
[2023-06-06 15:02:00.730] [client] [debug] З'єднання з сервером
успішно встановлено. Відпрака запита на створення аккаунту...
[2023-06-06 15:02:00.820] [client] [debug] Успішно відправлено
4332480 байтів на сервер
[2023-06-06 15:02:01.576] [client] [debug] Успішно отримана
відповідь від сервер: {"body":null,"status":"success"}
```

В той час на серері відбувається обробка запиту:

```
...
[2023-06-06 15:02:00.730] [info] Нове з'єднання: 127.0.0.1:50661
[2023-06-06 15:02:00.837] [Client 127.0.0.1:50661] [debug]
Отриманий запит від клієнта. Прочитано 4332480 байтів.
[2023-06-06 15:02:00.865] [info] Обробка запиту від клієнта.
Запит: create
[2023-06-06 15:02:01.574] [Database] [debug] Контент успішно
доданий в базу даних
[2023-06-06 15:02:01.575] [Client 127.0.0.1:50661] [info] Запит
від клієнта успішно обролений
[2023-06-06 15:02:01.578] [Client 127.0.0.1:50661] [debug] Успішно
відправлено 33 байтів клієнту
...
```

В цей час сервер оброблив запит, та зберіг результати в базі даних, а саме записав їх в пам'ять, та файл який обслуговується сервером щоб зберігати інформацію. Окрім зашифрованої назви акаунту, сервер також збергіає публічний контекст клієнта, щоб у нього була можливість виконувати операції

над даними, після перезапуску сервера.

Тепер покажемо операцію додавання балансу до існуючого акаунту. Звісно, щоб мати можливість виконувати операції з існуючим акаунтом, та мати можливість розшифровувати результат, нам необхідно використовувати той самий контекст:

```
> ./code/diploma_code_client add --context-path out.context test
100
...
[2023-06-06 15:19:39.623] [debug] Викликана команда add("test1",
100)
...
[2023-06-06 15:20:18.478] [client] [debug] Успішно отримана
відповідь від сервер: {"body":null,"status":"success"}
```

З отриманої відповіді від сервера можна побачити, що виконання гомоморфних операції, займає дуже багато часу (майже 30 секунд).

Тепер щоб перевірити що баланс дійсно був доданий можна виконати операцію get, яка повинна повернути інформацію про аккаунт, також цей приклад показує, що клієнт має змогу розшифровувати дані, змінені сервером:

```
> ./code/diploma_code_client get --context-path out.context test
[2023-06-06 15:25:27.589] [debug] Зчитування файлу конекста:
out.context
[2023-06-06 15:25:28.270] [debug] Шифрування назви аккаунту...
[2023-06-06 15:25:28.274] [debug] Викликана команда get("test1")
...
[2023-06-06 15:25:38.367] [client] [debug] Успішно відправлено
5132600 байтів на сервер
[2023-06-06 15:25:39.120] [client] [debug] Успішно отримана
відповідь від сервер: {"body":
"довге зашифроване повідомлення","status":"success"}
[2023-06-06 15:25:42.410] [client] [info] Успішно розшифроване
повідомлення від сервера.
> {"balance": 100}
```

Як ми бачимо, результат лише пошуку в базі, значно кращий аніж додаткова операція додавання, це тому що, наш алгоритм шифрує кожен байт

значення балансу, і додавання повинно відбуватись також побайтно. Також тут видно, що операція розшифрування забирає також якийсь час (3 секунди).

Якщо знайти оптимальні для нашої задачі параметри для створення контексту, то час роботи системи може покращитись, не втрачаючи рівень безпеки.

ВИСНОВКИ

В роботі "Гомоморфне шифрування для захисту даних в хмарних та туманних технологіях"досліджується гомоморфне шифрування, зокрема BGV (Brakerski-Gentry-Vaikuntanathan) схема. Гомоморфне шифрування є принципово новим підходом до захисту конфіденційних даних, який дозволяє виконувати обчислення над зашифрованими даними, зберігаючи їх у зашифрованому вигляді. Це забезпечує високий рівень конфіденційності та захисту інформації, що є особливо важливим у сферах, де зберігаються чутливі дані, наприклад, в банківській сфері.

У рамках дослідження проведена теоретична експертиза гомоморфного шифрування. Були вивчені математичні принципи, на яких ґрунтується гомоморфне шифрування, включаючи алгебраїчні структури та протоколи шифрування.

Для зрозуміння основних концепцій та технічних деталей гомоморфного шифрування було проаналізовано різні підходи, методи та алгоритми, які лежать в основі BGV схеми. Вивчення властивостей гомоморфного шифрування, таких як гомоморфність додавання та множення, а також операцій перетину та об'єднання, було проведено для оцінки його потенціалу в застосуванні до захисту даних.

Додатково, були досліджені сучасні протоколи та алгоритми, які дозволяють оптимізувати та поліпшити ефективність гомоморфного шифрування, зокрема у контексті обробки великих обсягів даних. Це включало аналіз методів оптимізації, таких як гомоморфна оцінка, техніки упаковки та інші методи зменшення обчислювальної складності.

В результаті теоретичної експертизи було отримано глибоке розуміння принципів гомоморфного шифрування, його потенціалу та обмежень. Це дозволило розробити клієнт-серверний застосунок банківської системи з використанням гомоморфного шифрування та бібліотеки HeLib. Теоретична експертиза була важливим етапом для успішної реалізації системи та її викори-

стання в практичних сценаріях.

У роботі було реалізовано клієнт-серверний застосунок банківської системи з використанням бібліотеки HeLib. У цій системі сервер зберігає інформацію про рахунки користувачів у зашифрованому форматі в базі даних. Клієнт може виконувати такі операції, як створення рахунків, додавання балансу, зняття балансу та отримання інформації про рахунок. Всі ці операції відбуваються над зашифрованими даними без необхідності розшифрування їх на сервері, що забезпечує високий рівень безпеки.

Використання гомоморфного шифрування має свої обмеження та недоліки. Основним обмеженням є обчислювальна складність таких систем. Гомоморфне шифрування вимагає значних обчислювальних ресурсів, що може призвести до затримок у виконанні операцій та збільшення обсягу обробки даних. Крім того, розмір зашифрованих даних може бути більшим, ніж у випадку звичайного шифрування, що може вплинути на продуктивність системи.

Недоліком гомоморфного шифрування є також вразливість до атак, зокрема до криптоаналітичних методів, які можуть використовувати математичні властивості схеми для отримання доступу до зашифрованих даних. Пошук ефективних захистів та протоколів залишається активною областю дослідження.

Крім обмежень і недоліків, варто відзначити, що гомоморфне шифрування також вимагає спеціального розуміння та експертизи для його впровадження та використання. Розробка та підтримка систем, які використовують гомоморфне шифрування, можуть вимагати високо кваліфікованого персоналу, який розуміє принципи шифрування та математичні основи, на яких воно ґрунтується.

Усупереч обмеженням та недолікам, гомоморфне шифрування має значний потенціал для захисту даних у хмарних та туманних технологіях, де конфіденційні дані можуть бути оброблені без необхідності розкриття їх змісту. Подальше дослідження та розробка ефективних алгоритмів гомоморфного шифрування можуть сприяти розширенню його застосування та підвищенню безпеки обробки конфіденційної інформації.

ПЕРЕЛІК ДЖЕРЕЛ

1. Fermat's little theorem - Wikipedia. `https://en.wikipedia.org/wiki/Fermat%27s_little_theorem`.

2. Helib c++ library that implements homomorphic encryption - Source. `https://github.com/homenc/HElib`.

3. Frederik Armknecht, Colin Boyd, Christopher Carr, Kristian Gjøsteen, Angela Jäschke, Christian A. Reuter, and Martin Strand. A guide to fully homomorphic encryption. Cryptology ePrint Archive, Paper 2015/1192, 2015. https://eprint.iacr.org/2015/1192.

4. Frederik Armknecht and Thorsten Strufe. An efficient distributed privacy-preserving recommendation system. pages 65 – 70, 07 2011.

5. Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-dnf formulas on ciphertexts. In Joe Kilian, editor, *Theory of Cryptography*, pages 325–341, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

6. Christoph Bosch, Andreas Peter, Pieter Hartel, and Willem Jonker. Sofir: Securely outsourced forensic image recognition. pages 2694–2698, 05 2014.

7. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. Fully homomorphic encryption without bootstrapping. *Electron. Colloquium Comput. Complex.*, 18:111, 2011.

8. Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. Cryptology ePrint Archive, Paper 2011/344, 2011. https://eprint.iacr.org/2011/344.

9. Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In Phillip Rogaway,

editor, *Advances in Cryptology – CRYPTO 2011*, pages 505–524, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

10. Andrei Bulatov. *Boolean Circuits*. duke.edu, https://users.cs.duke.edu/ reif/courses/complectures/Bulatov/32.pdf, Feb 2021.

11. Jung Hee Cheon, Jean-Sébastien Coron, Jinsu Kim, Moon Sung Lee, Tancrède Lepoint, Mehdi Tibouchi, and Aaram Yun. Batch fully homomorphic encryption over the integers. In *Advances in Cryptology - EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 315–335. Springer, 2013.

12. Michael Clear, Arthur Hughes, and Hitesh Tewari. Homomorphic encryption with access policies: Characterization and new constructions. In *Progress in Cryptology – AFRICACRYPT 2013*, pages 61–87. Springer Berlin Heidelberg, 2013.

13. I. Damgard, V. Pastro, N. P. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. Cryptology ePrint Archive, Paper 2011/535, 2011. https://eprint.iacr.org/2011/535.

14. Michael Fellows and Neal Koblitz. Combinatorial cryptosystems galore! 1. 01 1994.

15. Rosario Gennaro and Daniel Wichs. Fully homomorphic message authenticators. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology - ASIACRYPT 2013*, pages 301–320, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

16. Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. crypto.stanford.edu/craig.

17. Craig Gentry, Shai Halevi, Chris Peikert, and Nigel P. Smart. Field switching in bgv-style homomorphic encryption. Cryptology ePrint Archive, Paper 2012/240, 2012. https://eprint.iacr.org/2012/240.

18. Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. i-hop homomorphic encryption and rerandomizable yao circuits. Cryptology ePrint Archive, Paper 2010/145, 2010. https://eprint.iacr.org/2010/145.

19. Shafi Goldwasser, Yael Tauman Kalai, Raluca Ada Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. How to run turing machines on encrypted data. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013*, pages 536–553, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

20. Sergey Gorbunov, Vinod Vaikuntanathan, and Daniel Wichs. Leveled fully homomorphic signatures from standard lattices. Cryptology ePrint Archive, Paper 2014/897, 2014. https://eprint.iacr.org/2014/897.

21. Shai Halevi and Victor Shoup. Algorithms in helib. Cryptology ePrint Archive, Paper 2014/106, 2014. https://eprint.iacr.org/2014/106.

22. Shai Halevi and Victor Shoup. Bootstrapping for helib. Cryptology ePrint Archive, Paper 2014/873, 2014. https://eprint.iacr.org/2014/873.

23. Shai Halevi and Victor Shoup. Design and implementation of helib: a homomorphic encryption library. Cryptology ePrint Archive, Paper 2020/1481, 2020. https://eprint.iacr.org/2020/1481.

24. Marc Joye. Sok: Fully homomorphic encryption over the [discretized] torus. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(4):661–692, Aug. 2022.

25. Kristin Lauter, Michael Naehrig, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? Cryptology ePrint Archive, Paper 2011/405, 2011. https://eprint.iacr.org/2011/405.

26. Adriana Lopez-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. Cryptology ePrint Archive, Paper 2013/094, 2013. https://eprint.iacr.org/2013/094.

27. Michael Naehrig, Kristin Lauter, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? In *Proceedings of the 3rd ACM*

Workshop on Cloud Computing Security Workshop, CCSW '11, page 113–124, New York, NY, USA, 2011. Association for Computing Machinery.

28. Arjan Jeckmans Andreas Peter and Pieter Hartel. Efficient privacy-enhanced familiarity-based recommender system. Embedded Security Group, University of Twente, 2005. https://ofmas.ir/dlpaper/y1091.pdf.

29. Ron Rothblum. Homomorphic encryption: From private-key to public-key. In Yuval Ishai, editor, *Theory of Cryptography*, pages 219–234, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

30. Harsha Tirumala Rutgers University, Swastik Kopparty. *Boolean Circuits and Formulas*. rutgers.edu, https://sites.math.rutgers.edu/ sk1233/courses/topics-S20/lec1.1.pdf, 2020.

31. Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. Cryptology ePrint Archive, Paper 2009/616, 2009. https://eprint.iacr.org/2009/616.

32. Zhiqiang Yang Sheng Zhong Rebecca N. Wright. Privacy-preserving classification of customer data without loss of accuracy. 1Computer Science Department, Stevens Institute of Technology, 2005. https://www.cs.columbia.edu/ rwright/Publications/sdm05.pdf.

33. Amit Sahai Yuval Ishai and David Wagner. Private circuits: Securing hardware against probing attacks, 2006. https://people.eecs.berkeley.edu/ daw/papers/privcirc-crypto03.pdf.

Додаток А
Імплементація алгоритму пошуку в зашифрованій базі даних

```
std::optional<EncryptedDatabase::EncryptedEntry>
    EncryptedDatabase::Lookup(const helib::Ctxt& key) const {
    std::vector<helib::Ctxt> ciphertextMask;
    ciphertextMask.reserve(encryptedKeyValueDb_.size());

    for(const auto& [encryptedKey, encryptedEntry]:
        encryptedKeyValueDb_) {
        helib::Ctxt ciphertextMaskEntry = encryptedKey.key_;
        ciphertextMaskEntry -= key;
        ciphertextMaskEntry.power(encryptedKey.plaintextPrimeModulus_ - 1
        ciphertextMaskEntry.negate();
        ciphertextMaskEntry.addConstant(NTL::ZZX(1));
        std::vector<helib::Ctxt> rotatedCiphertextMasks(
            encryptedKey.encryptedArray.size(), ciphertextMaskEntry);
        for(int i = 1; i < rotatedCiphertextMasks.size(); i++) {
            encryptedArray.rotate(rotatedCiphertextMasks[i], i);
        }
        totalProduct(mask_entry, rotatedCiphertextMasks);
        mask_entry.multiplyBy(encryptedEntry);
        ciphertextMask.push_back(mask_entry);
    }

    helib::Ctxt value = ciphertextMask[0];
    for(int i = 1; i < ciphertextMask.size(); i++) {
        value += ciphertextMask[i];
    }

    return std::make_optional<EncryptedEntry>{value};
}
```

Додаток Б
Створення контексту та приватного ключа

```
// Plaintext prime modulus.
long p = 2;
// Cyclotomic polynomial - defines phi(m).
long m = 4095;
// Hensel lifting (default = 1).
long r = 1;
// Number of bits of the modulus chain.
long bits = 500;
// Number of columns of Key-Switching matrix (typically 2 or 3).
long c = 2;
// Factorisation of m required for bootstrapping.
std::vector<long> mvec = {7, 5, 9, 13};
// Generating set of Zm* group.
std::vector<long> gens = {2341, 3277, 911};
// Orders of the previous generators.
std::vector<long> ords = {6, 4, 6};

helib::Context context = helib::ContextBuilder<helib::BGV>()
                            .m(m)
                            .p(p)
                            .r(r)
                            .gens(gens)
                            .ords(ords)
                            .bits(bits)
                            .c(c)
                            .bootstrappable(true)
                            .mvec(mvec)
                            .build();
// Create a secret key associated with the context.
helib::SecKey secret_key(context);
// Generate the secret key.
```

```
secret_key.GenSecKey();
// Generate bootstrapping data.
secret_key.genRecryptData();
// Public key management.
// Set the secret key (upcast: SecKey is a subclass of PubKey).
const helib::PubKey& public_key = secret_key;
// Get the EncryptedArray of the context.
const helib::EncryptedArray& ea = context.getEA();
// Build the unpack slot encoding.
std::vector<helib::zzX> unpackSlotEncoding;
buildUnpackSlotEncoding(unpackSlotEncoding, ea);
```