Міністерство освіти і науки України

Харківський національний університет імені В.Н. Каразіна

Факультет комп'ютерних наук

Спеціальність 125 «Кібербезпека»

Освітня програма «Безпека інформаційних та комунікаційних систем»

«Допущено до захисту»

Зав. кафедрою БІСТ

Сергій РАССОМАХІН

_____

« » 2022 р.

Пояснювальна записка

до кваліфікаційної роботи магістра

на тему: «Аналіз методів пошуку даних у криптографічно захищеній базі даних»

оцінка « »

Голова ЕК

Доценко С.І. _____

Керівник проф., д.т.н. Єсін В.І.

Рецензент проф., д.т.н. Доля Г.М.

Виконавець : студент групи КБ-61

Махмудов Т.Ф.

Харків – 2022

РЕФЕРАТ

Звіт про виконання дипломної роботи: 74 сторінок, 7 рисунків, 65 формул, 3 таблиці, 1 додаток та 29 джерел.

Мета роботи полягає у аналізі методів пошуку даних у криптографічно захищених базах даних для виявлення ефективних та безпечних представників, а також вироблення рекомендацій щодо вибору певного методу в залежності від потреб проекту.

У роботі було розглянуто та проаналізовано 6 методів шифрування з можливістю пошуку, для кожного з яких були приведені алгоритми, приклади використання цих методів, надані пояснювальні рисунки та таблиці. Проведено аналіз для виявлення складності та рівня безпеки методів, а також розглянута продуктивність практичних реалізацій. На підставі аналізу зроблено висновки про доцільність використання того чи іншого методу шифрування з можливістю пошуку на практиці, а також зроблені рекомендації щодо комбінації описаних методів для отримання очікуваного результату.

Результати виконання цього проекту можуть бути використані як систематизований огляд для ознайомлення з сучасними та перспективними методами пошуку даних у криптографічно захищених БД, а також як рекомендації щодо вибору того чи іншого методу для проектів, в яких зберігання даних покладається на третю сторону.

Можливими напрямками розвитку є пошук та розгляд сучасніших методів, а також асиметричних методів з можливістю пошуку, в яких приділяється більше уваги на виразність запитів, а не на ефективність схеми.

Ключові слова: ШИФРУВАННЯ З МОЖЛИВІСТЮ ПОШУКУ, КЛЮЧОВЕ СЛОВО, ІНДЕКС, ДИНАМІЧНЕ СИМЕТРИЧНЕ ШИФРУВАННЯ З МОЖЛИВІСТЮ ПОШУКУ, ТОКЕН ПОШУКУ, ЛАЗІВКА.

ABSTRACT

Thesis report: 74 pages, 7 figures, 65 formulas, 3 tables, 1 appendix and 29 sources.

The purpose of the paper is to analyze search methods in cryptographically protected databases to identify effective and safe representatives, as well as to make recommendations for choosing certain method depending on the needs of the project.

The paper considered and analyzed 6 searchable encryption methods, for each of them algorithms were given, examples of the use of these methods, explanatory figures and tables were provided. The analysis was conducted to evaluate the complexity and level of security of the methods, and the performance of practical implementations was also considered. On the basis of the analysis, conclusions were made about the feasibility of using one or another searchable encryption method in practice, and recommendations were made regarding the combination of the described methods to obtain expected results.

The results of this paper can be used as a systematic review familiarization with modern and promising methods of data search in cryptographically protected databases, as well as recommendations for choosing one or another method for projects in which data storage relies on a third party.

Possible areas of development are search and consideration of more modern methods, as well as asymmetric searchable methods, which focus more on query expressiveness rather than scheme efficiency.

Keywords: SEARCHABLE ENCRYPTION, KEYWORD, INDEX, DYNAMIC SYMMETRIC SEARCHABLE ENCRYPTION, SEARCH TOKEN, TRAPDOOR.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

BIEX – Boolean inclusion-exclusion encryption

BP – Backward privacy

BRC – Best range cover

CKA – Chosen keyword attack

CPA – Chosen plaintext attack

DIEX – Dynamic inclusion-exclusion encryption

EDB – Encrypted database

FHE – Fully homomorphic encryption

FP – Forward privacy

IEP – Inclusion-exclusion principle

IEX – Inclusion-exclusion encryption

IND – Indistinguishability

MM – Multimap

ORAM – Oblivious random access machine

PDSSE – Parallel and dynamic symmetric searchable encryption

PE – Predicate encryption

PEKS – Public-key encryption with keyword search

PK – Private key

PPE – Property-preserving encryption

RCPRF – Range-constrained pseudorandom function

RO – Random oracle

ROM – Random oracle model

RSA – Rivest, Shamir, Adleman

SE – Searchable encryption

SSE – Symmetric searchable encryption

TDP – Trapdoor permutation

ZMF – Z-IDX matryoshka filter

АШМП – Асиметричне шифрування з можливістю пошуку

БД – База даних

ЗБД – Зашифрована база даних

ЛМВ – Локальне мультивідображення

МВ – Мультивідображення

ОПВФ – Обмежена псевдовипадкова функція

ПВП – Псевдовипадкова перестановка

ПВФ – Псевдовипадкова функція

СШ – Структурне шифрування

СШМП – Симетричне шифрування з можливістю пошуку

ЧЧД – Червоно-чорне дерево

ШМП – Шифрування з можливістю пошуку

ВСТУП

За останні роки хмарні послуги стали важливою частиною життя більшості людей, замінивши та полегшивши виконання їхньої щоденної рутини. Тепер необов'язково мати на своєму персональному комп'ютері набір інструментів від Office, так як у будь-який момент можна скористатися безкоштовними аналогами Google Docs, Google Sheets або Office 365. Так само на ринку з'явилося безліч рішень для зберігання даних у хмарі: Google, Microsoft, Dropbox, Mega та багато інших, але у всіх з них стоїть питання безпеки даних, що зберігаються. Адже дані не лише передаються через Інтернет, але ще й зберігаються у третіх осіб, тому, не дивлячись на запевнення осіб, які надають послуги, необхідно пам'ятати про безліч новин про витоки даних.

Питання дотримання безпеки даних, а саме їх конфіденційності та цілісності, як правило, у теперішний час вирішується за рахунок використання відповідних криптографічних примітивів з урахуванням розвитку обчислювальних потужностей. Але, у зв'язку із специфічним способом зберігання у хмарі, виникає питання ефективності пошуку необхідної інформації. Проблема, яка розглядається у цій роботі, полягає у тому, що шифрування унеможливлює доступ до даних без ключів для зловмисника, але позбавляє власника даних можливості проведення пошуку за цією інформацією.

Одним з очевидних рішень цієї проблеми є завантаження всієї бази даних, з подальшим її локальним розшифруванням і пошуком бажаних результатів отриманих розшифрованих даних. Проте для більшості застосунків такий підхід є недоцільним. Якщо ж покласти на сервер розшифрування даних, щоб він міг виконувати запити і надсилати користувачеві лише результати, то знижується рівень безпеки, оскільки дані, що захищені шифруванням, розкриваються серверу. Тому бажано мати можливість проведення пошуку у повному обсязі на стороні сервера без розшифрування даних і, отже, з найменшою можливою втратою конфіденційності даних.

Об’єктом дослідження є процес пошуку даних у криптографічно захищених хмарних базах даних.

Предметом дослідження є методи та алгоритми пошуку даних у криптографічно захищених хмарних базах даних.

Метою проекту є аналіз методів пошуку даних у криптографічно захищених базах даних для виявлення ефективних та безпечних представників, а також вироблення рекомендацій щодо вибору певного методу в залежності від потреб проекту.

Основними завданнями при виконанні дипломної роботи можуть вважатися наступні:

- розглядання понять безпеки, визначених для шифрування з можливістю пошуку;
- проведення класифікації методів пошуку, розгляд їх алгоритмів та особливостей;
- аналіз методів пошуку з точки зору їхньої ефективності та забезпечуваного рівня безпеки;
- визначення обґрунтованих висновків, вироблення та надання рекомендацій щодо використання проаналізованих методів пошуку.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Шифрування з можливістю пошуку (ШМП, англ. searchable encryption) – це вид шифрування, при якому мається можливість робити пошук за зашифрованими даними з мінімальним витоком інформації.

Ця особливість робить його дуже зручним при використанні хмарних сховищ, для яких зазвичай розглядається так званий напівчесний захист від постійного внутрішнього супротивника. Тобто супротивник / зловмисник сприймається напівчесним (чесним-але-допитливим (honest-but-curious)) – це законний учасник комунікаційного протоколу, який не буде відхилятися від визначеного протоколу, але намагатиметься дізнатися всю можливу інформацію з законно отриманих повідомлень [1].

1.1 Класифікація шифрування з можливістю пошуку

1.1.1 Підходи к шифруванню з можливістю пошуку

Можливість пошуку у ШМП досягається декількома способами:

1) вироблення таким чином шифртексту, щоб за ним можна було виконувати пошук;
2) створення індексів, за якими виконуватиметься пошук.

До представників першого способу можна віднести підхід, запропонований Сонгом, Вагнером та Перігом [2], у якому вони пропонують розбивати відкритий текст на слова фіксованого розміру (наприклад, це можуть бути файли або записи реляційної бази даних). Після чого треба згенерувати випадкові числа для кожного з вихідних слів, та для кожного з згенерованих чисел отримати результат за допомогою псевдовипадкової функції (ПВФ). При цьому треба відмітити, що ключ ПВФ повинен залежати від вихідного слова. Наприкінці треба підсумовувати за модулем 2 вихідні слова з конкатенацією псевдовипадкового числа та результату псевдовипадкової функції від цього числа. Для зручнішого сприйняття наведемо вираз для визначення шифртексту у формулі нижче.

$$C_i = W_i \oplus T_i, \tag{1.1}$$

де C_i – це шифртекст,

W_i – ключове слово,

$T_i = S_i \,||\, F_{k_i}(S_i)$ – конкатенація псевдовипадкового числа S_i та результату псевдовипадкової функції від цього числа $F_{k_i}(S_i)$.

Для пошуку за шифрованими даними певного ключового слова, потрібно знати ключ, за допомогою якого було обчислено псевдовипадкову функцію для даного слова, а також само це слово. Якщо це відомо, то, підсумовуючи з шифртекстом відповідне вихідне слово, очікується отримати конкатенацію псевдовипадкового числа та результату псевдовипадкової функції від цього числа. Але, таким чином, розкривається вихідне слово, яке потрібно знайти. Щоб вирішити цей недолік, треба просто зробити попереднє шифрування вихідних слів. Такий нескладний алгоритм допомагає робити пошук за зашифрованим текстом.

Методи, що базуються на генерації індексів для вихідних слів є найбільш розповсюдженими серед існуючих методів SE.

В загальному випадку мається набір документів та набір ключових слів. У кожному з документів містяться деякі ключові слова. Ключові слова шифруються (наприклад, за допомогою геш-функції) з використанням ключа власника. Зашифровані документи та індекс зберігаються на сервері, а для пошуку створюються так звані лазівки або люки (англ. trapdoor), при чому генерувати такі лазівки може тільки володілець секретного ключа. Після їх генерації, вони надсилаються до сервера, який за їх допомогою може відновити покажчики на відповідні зашифровані документи.

Між цим, самі індекси можуть бути двох видів:

1) прямі індекси – відповідають кожному документу;
2) інвертовані індекси – відповідають кожному ключовому слову.

Припустимо, що мається n документів та k ключових слів, тоді таблиці індексів будуть виглядати як на рисунку 1.1.

ПРЯМИЙ ІНДЕКС

документ	ключові слова
d1	w2, w5, w7
d2	w1, w2
...	...
dn	w6, w9, wk

ІНВЕРТОВАНИЙ ІНДЕКС

ключові слова	документ
w1	d2, d5, d11
w2	d1, d2
...	...
wk	d6, d12, dn

Рисунок 1.1 – Види індексів

У першому випадку отримуємо час пошуку, що прямо пропорційний до кількості документів, тобто $O(n)$, у другому ж залежність йде від кількості ключових слів. Наприклад, якщо проводити пошук за певним ключовим словом, то маємо список документів, в яких воно наявне, відповідно й складність буде зведена до цієї кількості.

1.1.2 Види шифрування з можливістю пошуку

Розглянемо основні методи шифрування з можливістю пошуку, які конструюються на базі різних ідей та технік.

1) Oblivious Random Access Machine (ORAM)

ORAM – метод, що був представлений у [3], це компілятор, який перетворює алгоритми таким чином, що отримані алгоритми зберігають поведінку вводу-виводу оригінального алгоритму, але шаблони доступу до пам’яті перетвореного алгоритму не залежать від шаблонів доступу до пам’яті вихідного алгоритму. Більш детально проблема розкриття шаблону доступу розглядається далі (див. 1.3).

Суть методу полягає в тому, що необхідно знайти деяку інформацію серед зашифрованої. При доступі до деякої ділянки пам'яті, що містить необхідний клієнту елемент, ця ділянка завантажується клієнтом та розшифровується його секретним ключом, проводяться необхідні операції з інформацією, після чого виконується перестановка деяких або всіх елементів бази даних (БД) в обраній ділянці, розшифрований блок знов шифрується та завантажується на сервер.

Серед переваг методу є його висока безпека, яка запобігає витоку шаблонів пошуку та доступу, але він не користується популярністю через його низьку ефективність. Навіть останні роботи, що мають відчутне покращення, все ще не можуть досягти необхідного рівня для практичного застосування.

2) Fully Homomorphic Encryption (FHE)

Гомоморфне шифрування – це вид шифрування, який дозволяє робити певні операції над зашифрованим текстом, результат яких відповідає зашифрованому результату операцій над відповідним відкритим текстом.

Якщо таке шифрування є гомоморфним лише для окремої операції, то воно є частковим. Перше повністю гомоморфне шифрування було запропоновано у [4]. Однак сучасні FHE схеми також є досить вартісними з точки зору обчислень та зберігання.

3) Property-Preserving Encryption (PPE)

В роботі [5] було вперше представлено шифрування зі збереженням властивостей, суть якого полягає в тому, що з шифртексту можна визначити якими властивостями він володіє.

PPE надає доступні методи ефективного шифрування баз даних шляхом збереження їх певних функціональних можливостей. PPE включає в себе детерміноване шифрування, шифрування з розкриттям порядку (order-revealing encryption) і шифрування зі збереженням формату (format-preserving encryption). Хоча простота та сумісність робить його вимогливим у реальних системах, безпека залишається досить слабкою.

4) Predicate Encryption (PE)

Предикатне шифрування було представлено та визначено у [6]. PE дозволяє проводити детальний (fine-grained) контроль доступу, тобто такий, що базується на декількох факторах. Під його визначення підпадають й інші підходи пошуку за зашифрованими даними, такі як шифрування на основі атрибутів (attribute-based encryption), шифрування на основі ідентичності (identity-based encryption), приховане векторне шифрування (hidden vector encryption).

В предикатному шифруванні секретні ключі асоціюються з предикатами, а шифртексти – з атрибутами. Клієнт може розшифрувати деякий шифртекст, якщо при застосуванні предиката на атрибут шифртексту отримує одиницю.

При цьому можливі дві варіації PE: перша використовує відкриті індекси, але цей метод не підходить у нашому випадку через витік множини атрибутів, за якими було зашифровано дані. Другий метод скриває атрибути і підходить для наших цілей, проте він все ще є менш ефективним за симетричне шифрування.

5) Структурне шифрування

У [7] було представлено структурне шифрування, яке, по суті, є генералізацією симетричного шифрування з можливістю пошуку, яке базується на індексах. Структурне шифрування (СШ) – це шифрування структури даних, яке дозволяє робити запити до неї за допомогою токенів, що можуть бути згенеровані тільки з використанням секретного ключа. При цьому, процес запиту не повинен розкривати корисну інформацію про запит чи дані.

Вибір того чи іншого підходу (а, отже, й методу) – це завжди пошук компромісу між ефективністю, безпекою та виразністю схеми. Як визначено у [8], якщо ви віддаєте значну перевагу безпеці та виразності над ефективністю, то вам слід дивитися у бік ORAM та FHE, оскільки вони дозволяють повністю уникнути витоку інформації та деякі з них дозволяють робити випадкові запити, хоча вони й дуже далекі від практично прийнятного рівню ефективності. Якщо ж для вас важливим є ефективність та виразність над безпекою, то слід звернути увагу на схеми, що базуються на PPE, проте з таких схем витікає досить багато інформації. І якщо вас хвилює безпека та ефективність за рахунок слабкої виразності, то ваш вибір повинен бути у бік схем логічного пошуку за зашифрованими даними та схемами, заснованими на структурному шифруванні, оскільки вони є досить ефективними та безпечними, проте запити не дуже варіативні, через що такі рішення більше підходять для NoSQL баз даних.

1.1.3 Симетричне та асиметричне шифрування з можливістю пошуку

Також можна провести класичну для шифрів дихотомію на симетричне на асиметричне SE. Проте для таких систем є певна особливість: для симетричної

системи є тільки один ключ, який є секретним, відповідно є тільки один користувач – власник ключа, що може здійснювати запис та зчитувати дані, але, якщо йде мова про асиметричну систему, то в ній наявний відкритий ключ, за допомогою якого багато користувачів можуть записувати дані на сервер, а здійснювати пошук може тільки власник секретного ключа. Більш того, якщо використовувати розподілений секретний ключ, то це дозволить створити систему, в якій буде більш однієї людини, що може робити пошук за шифртекстом. Отже, по типу письменники/читачі системи можуть бути 1/1, */1, 1/* та */*.

1.1.4 Динамічні та статичні схеми

Динамічні схеми ШМП – це такі, що дозволяють проводити оновлення структури даних. Тобто, якщо є деяка множина документів, то є можливість додавати нові елементи та видаляти вже присутні. При цьому реалізація цього може бути різною, деякі схеми підтримують додавання, але не підтримують видалення.

Для того, щоб такі операції проводити, як правило, генерується відповідний токен оновлення (подібно до токену пошуку), в якому зазначаються ідентифікатор файлу та ключові слова, пов’язані з ним, що треба додати чи видалити.

Але стає зрозумілим, що ці операції також можуть бути небезпечними та уможливлюють витік інформації. З цього приводу у [9] було визначено декілька понять безпеки в контексті ШМП:

1) Пряма секретність або forward privacy (FP) – гарантує, що сервер не дізнається, чи містять додані документи ключові слова, пошук за якими проводився раніше.
2) Зворотна секретність або backward privacy (BP) – гарантує, що сервер не може застосовувати запити к видаленим документам.

Більш того, у роботі [10] автори виділили три рівня зворотної секретності: перший є найпотужнішим, на цьому рівні схема при додаванні нового документу розкриває документи, що містять ключове слово w, що наявне у запиті, момент

часу, коли вони були додані, а також загальну кількість додавань за ключовим словом w; другий рівень розкриває інформацію першого рівня, а також момент часу, коли було виконано будь-яке оновлення за ключовим словом w; третій, найслабкіший, рівень розкриває всю інформацію другого рівня, а також яке саме видалення скасувало певне попереднє додавання ключового слова.

1.2 Шаблони пошуку та доступу

Для кращого розуміння ШМП треба визначити декілька нескладних, проте надважливих понять.

Шаблон пошуку – це інформація про те, чи створено будь-які два запити за одним ключовим словом чи ні.

Шаблон доступу – це інформація про те, які документи містять ключове слово для кожного запиту користувача.

Як зазначається у роботі [11], дуже багато ШМП страждають від того, що з зазначених вище шаблонів витікає інформація, серед цих представників є і розглянутий нами шифр Сонга, Вагнера, Періг. У симетричних схемах ця проблема ґрунтується на тому, що для генерування лазівки зазвичай використовуються детерміністичні алгоритми, тобто для певного ключового слова завжди буде генеруватись однаковий запит. Тут стає очевидним, що супротивник може без зусиль встановити, чи мають два різні запити одне й те ж саме ключове слово.

З іншого боку, є шаблон доступу, який, в свою чергу, може розкрити шаблон пошуку. Якщо шаблон доступу однаковий, то, ймовірно, два пошукових запити містять однакове ключове слово. У [12] зазначається, що майже у всіх симетричних ШМП витікає шаблон доступу. Також в цій роботі наводиться приклад, де супротивник перехвачує пошукові запити та відповідні до них шаблони доступу, після чого, маючи деяку попередню інформацію про документи, що зберігаються на сервері, він може визначити пару ключових слів та відповідних до них ключових запитів за допомогою статистичного аналізу. Встановивши ці два запити, стає легшим співвіднести інші ключові слова до відповідних пошукових запитів.

1.3 Моделі безпеки схем шифрування з можливістю пошуку

Перед розглядом конкретних методів необхідно зрозуміти, як саме визначається безпека тієї чи іншої схеми ШМП. Відразу треба відмітити, що існуючі для шифрів моделі безпеки (як, наприклад, нерозрізненість шифртексту при атаках на основі підібраного відкритого тексту або IND-CPA) не підходять для шифрування з можливістю пошуку, бо в SE витік інформації зазвичай стосується кількості документів та ключових слів, індексів, а також шаблонів пошуку та доступу, з яких вже можна отримати інформацію про самі зашифровані дані.

Першим, хто надав визначення безпеки в рамках ШМП, був Гох у 2003 році [13], він визначив, що симетрична схема ШМП є безпечною, якщо документи є нерозрізненими при атаках з адаптивно підібраними ключовими словами (indistinguishability of keywords against adaptive chosen keyword attack або IND1-CKA). Тобто це визначення гарантує, що супротивник не може визначити зміст документів з відповідного йому індексу, а також індексів інших документів, про які він міг дізнатись з попередніх запитів. Ця схема також зобов’язує створювати індекси з однаковою кількістю ключових слів для документів, що мають однаковий розмір. Це необхідно, щоб супротивник не зміг визначити, який з документів закодовано в індексі. Однак ця модель зовсім не враховує безпеку лазівок, так як запропонована Гохом схема має широке застосування, де лазівки не потрібні, а симетричне ШМП (СШМП або SSE) – лише одне з цих застосувань.

Пізніше Ченг та Міценмахер запропонували свою, більш строгу, модель [14], в якій супротивник не міг відрізнити два індекси для документів навіть різного розміру, проте індекси для цих документів все одно повинні були вміщувати однакову кількість ключових слів. Також вони намагалися визначити поняття безпечності для лазівок, однак їх визначення вийшло невдалим, що написав у своїй роботі Куртмола [15], зазначивши, що навіть небезпечні схеми можуть виконати зазначені в моделі умови.

Куртмола у 2006 не тільки вказав на невдале визначення Ченга та Міценмахера, він зі своїми співавторами визначили власні моделі IND1-CKA та IND2-CKA, зазначивши, що врахування лазівок є обов'язковим, так як вони нерозривно пов'язані з безпекою індексів. Перша модель передбачає нерозрізненість проти атак з неадаптивно підібраними ключовими словами, друга – з адаптивно. Обидві моделі гарантують, що ні індекси, ні лазівки не розкривають жодну інформацію про вміст документу та ключові слова, асоційовані з документом (за виключенням тієї, яку можна вивести з шаблонів пошуку та доступу), проте перша передбачає одночасну генерацію всіх запитів супротивником, а друга дозволяє генерувати запити на базі вже визначених лазівок та результатів запитів. Ці моделі були прийняті та використовуються нині.

Проте все зазначене вище стосується тільки симетричних схем шифрування з можливістю пошуку. Що стосується асиметричних ШМП (АШМП), в таких схемах з'являється ще один ключ – відкритий, за допомогою якого генеруються лазівки, отже, безпека лазівок не враховується. Першою роботою стосовно моделі безпеки АШМП стала стаття Дена Боне та інших [16], у якій вони запропонували визначати безпеку асиметричних схем нерозрізненістю двох зашифрованих ключових слів, доки супротивник не має лазівки до цих слів, тобто цю модель також можна назвати IND-CKA або PK-CKA. Аналогічно 1 версія буде для неадаптивно підібраних ключових слів, а друга – для адаптивно.

Додатково можна зазначити таку широко застосовану модель, як модель випадкового оракула або ROM (Random Oracle Model). Ця модель є ідеалізованою, вона дещо середнє між строгим доказом безпеки схем та відсутністю цього доказу зовсім. В моделі використовується випадково обрана функція H, що може бути розрахована тільки за допомогою запита до так званого оракула, який приймає на вхід деяке значення x та повертає $H(x)$, при цьому для кожного x є своя унікальна відповідь, і при кожному запиті x завжди повертається одне й те ж саме значення $H(x)$. Слід зазначити, що з оракулом

можуть взаємодіяти й супротивники та отримувати відповіді на свої запити, а небезпека визначається тим, чи зможе супротивник, маючи відповідь оракула $H(x)$, знайти таке x', що $H(x') = H(x)$[17]. Якщо зможе, то схема є небезпечною. Проте в реальному світі не існує ідеалізованих функцій, тому використовуються достатньо стійкі геш-функції, отже, й безпека схем визначається цими криптопримітивами.

2 ХАРАКТЕРИСТИКА МЕТОДІВ ШИФРУВАННЯ З МОЖЛИВІСТЮ ПОШУКУ

У цьому розділі розглядаються методи шифрування з можливістю пошуку. Всі зазначені методи відносяться до симетричного шифрування з можливістю пошуку, а також всі, окрім одного, є динамічними. При цьому увага не приділяється методам, що вже були розглянуті та проаналізовані у інших роботах, в яких доведена їх неспроможність чи то з погляду безпеки, чи то з боку ефективності. Наприклад, у роботі [18] було проведено серйозний аналіз великої кількості методів, однак більшість з них виявилися недостатньо ефективними для їх практичного використання. Деякі з зазначених схем є досить повільними при прийнятному рівні безпеки, інші ж працюють досить швидко, проте мають неприйнятний витік інформації.

2.1 Паралельне та динамічне симетричне шифрування з можливістю пошуку

Цей метод (позначимо як PDSSE) описано у роботі [19]. Він є динамічним у тому сенсі, що метод дозволяє змінювати структуру документів, що зберігається на сервері, шляхом додавання нового чи видалення одного з існуючих документів. Паралельним він є завдяки тому, що операції пошуку можна розбивати на окремі ітерації, що можуть виконуватися одночасно декількома процесорами. При цьому метод також є симетричним, тобто у схемі передбачається, що існує тільки секретний ключ, а значить є тільки один письменник та читач.

Даний метод засновано на добре відомій структурі даних – червоно-чорних деревах (ЧЧД) – це бінарні дерева пошуку з одним додатковим бітом пам’яті на вузол: його кольором, який може бути червоним або чорним. Обмежуючи кольори вузлів на будь-якому простому шляху від кореня до листа, червоно-чорні дерева гарантують, що жоден такий шлях не буде більш ніж вдвічі довший за будь-який інший, так що дерево буде приблизно збалансованим [20].

Розглянемо властивості ЧЧД, які роблять його збалансованим:

1) Кожен вузол червоний або чорний.
2) Корінь – чорний.
3) Кожен лист – чорний.
4) Якщо вузол червоний, то обидва його дочірні елементи чорні.
5) Для кожного вузла всі прості шляхи від вузла до нащадків містять стільки ж чорних вузлів.

Проте в методі використовуються деяка модифікація ЧЧД: листя дерева зберігають покажчики на документи, а інші вузли – ідентифікатори документів. Також кожен вузол u зберігає бітовий вектор $data_u$ довжиною m біт, де i-ий біт відображає чи наявне ключове слово w_i у його нащадках. Тут встає питання того, що вузли мають по два нащадки (припустимо, що лівий нащадок – це v, а правий – це z), отже, бітовий вектор кожного вузла u визначається як «побітове або» векторів його нащадків, тобто за формулою (2.1).

$$data_u = data_v \ \& \ data_z \tag{2.1}$$

Тоді якщо i-ий біт дорівнює одиниці, то є хоча б один шлях, який приводить до документу d_j, що містить ключове слово w_i. А для знаходження всіх документів, які містять ключове слово, просто необхідно перевіряти вектори нащадків, доки не зустрінемо 0 в i-ій позиції вектора, або не доберемося до листа дерева.

Тепер перейдемо до головного – алгоритмів, за якими працює даний метод. Для роботи цих алгоритмів знадобляться дві псевдовипадкові функції G, P, а також випадковий оракул H. Маємо вісім алгоритмів:

1) Генерація ключу

Генерується три випадкові строки довжиною k біт – K_1, K_2, r. Перші дві строки – це секретні ключі, що необхідні для ПВФ, а на базі r генерується ще один секретний ключ для схеми шифрування даних – K_3.

2) Шифрування

По-перше, для кожного ключового слова генерується секретний ключ на основі ПВФ та ключа K_2:

$$SK_i = Gen\left(1^k; G_{K_2}\left(w_i\right)\right) \tag{2.2}$$

Таким чином, отримуємо m ключів SK_i.

По-друге, з використанням ключа K_3, шифруються документи з множини DB, що містять ключове слово w_i, визначимо це як $DB\left(w_i\right)$, для кожного документа отримуємо відповідний шифртекст c_{i_j}. Шифртекст зберігається на диску, а документ у відкритому вигляді видаляється, при цьому вони мають однакові ідентифікатори.

Для кожного вузла u створюються дві геш-таблиці розміром m – $\lambda_{0u}, \lambda_{1u}$ – та зберігаються у вузлі. Причому ключ у цій геш-таблиці – це деякий бітовий рядок довжиною k, а значення – це зашифроване логічне значення. Розглянемо обчислення детальніше:

$$\lambda_{bu}\left[P_{K_1}\left(w_i\right)\right] \leftarrow Enc\left(SK_i,\ data_u\left[i\right]\right), \tag{2.3}$$

де b – це деяке випадкове значення, що отримується за допомогою випадкового оракула (див. (2.4)),

$data_u\left[i\right]$ – це біт, який визначає, чи ведуть нащадки вузла u до документа, який містить ключове слово w_i.

$$b = H\left(P_{K_1}\left(w_i\right),\ id\left(u\right)\right), \tag{2.4}$$

де $id\left(u\right)$ – ідентифікатор деякого вузла u.

В іншу геш-таблицю записується випадковий бітовий рядок. Після цього вектор $data_u$ видаляється. Розглянемо цей крок ще раз докладно: мається дві геш-таблиці для кожного вузла, але тільки одна з них буде містити справжню

інформацію про ключові слова у нащадках, причому ця інформація буде у зашифрованому вигляді. Інший рядок лише збиває, щоб сервер не міг зрозуміти який з них містить справжню інформацію.

В підсумку алгоритму отримується зашифрований індекс – це наше ЧЧД, а також шифртекст, тобто наші зашифровані документи – $c = \left(c_{i_1}, \ldots, c_{i_n} \right)$.

3) Генерація токену пошуку

Для генерації токену пошуку, за допомогою якого сервер буде виконувати пошук, треба згенерувати ключ SK_i для ключового слова, як у алгоритмі шифрування, тобто за формулою (2.2).

Після за допомогою ключа генерується сам токен за формулою (2.5):

$$T_s = \left(P_{K_1}\left(w_i \right), SK_i \right) = \left(T_1, T_2 \right) \tag{2.5}$$

4) Пошук

На базі токену, що створений у третьому алгоритмі, сервер робить пошук за шифртекстом. Починаючи з кореня дерева, рекурсивно виконуються наступні кроки:

- розраховуємо b за формулою (2.4), щоб визначити коректну геш-таблицю, після чого розшифровуємо її для отримання бітового вектору, що містить інформацію про наявність ключових слів у документі:

$$a = Dec\left(T_2,\ \lambda_{bu}\left[T_1 \right] \right) = Dec\left(SK_i,\ \lambda_{bu}\left[P_{K_1}\left(w_i \right) \right] \right) \tag{2.6}$$

- якщо a дорівнює 0, то виконання завершується;
- якщо даний вузол є листом, в деяку змінну c_{w_i} дописується шифртекст, на який посилається даний вузол, тобто

$$c_{w_i} = c_{w_i} \,||\, c_u \tag{2.7}$$

- інакше продовжуємо пошук спочатку у лівому нащадку, потім у правому.

Наприкінці отримуємо шифртекст c_{w_i}, що містить всі документи у зашифрованому вигляді, в яких наявне ключове слово w_i.

5) Оновлення структури дерева

Використовуючи цей алгоритм, є можливість додати або видалити якийсь документ до/з дерева. Для цього треба надати індекс документу та вказати операцію (додавання чи видалення), яку треба зробити. Ця операція проводиться як для червоно-чорного дерева, а її детальний опис вставки чи видалення елементу можна знайти у [20]. Проте цей алгоритм не є самостійним, він використовується як частина, повертаючи ту частину ЧЧД, яку треба змінити, щоб додати/видалити документ, – $T\left(upd\right)$.

6) Оновлення токену

Якщо проводиться додавання документа, то необхідно його зашифрувати, отримуємо шифртекст c_i. Після цього треба вставити документ, отримуємо модифіковане ЧЧД – $T'\left(upd\right)$. У цьому дереві треба провести перерозрахунки у кожному вузлі, а саме:

- створити дві нові геш-таблиці;
- заповнити їх згідно алгоритму шифрування, а саме за формулою (2.3).
- отримуємо змінену частину дерева $T_{upd} = \left(T'\left(upd\right), c_i\right)$.

7) Оновлення

Враховуючи 2 зазначених вище алгоритми, на підставі T_{upd}, залишається внести зміни до ЧЧД та перерахувати пошуковий індекс і шифртекст.

8) Розшифрування

Розшифрування є аналогічним шифруванню та виконується за допомогою ключа K_3, отримуємо відкритий текст, тобто наш документ d_i.

Розглянемо приклад реалізації методу, наведений у [19].

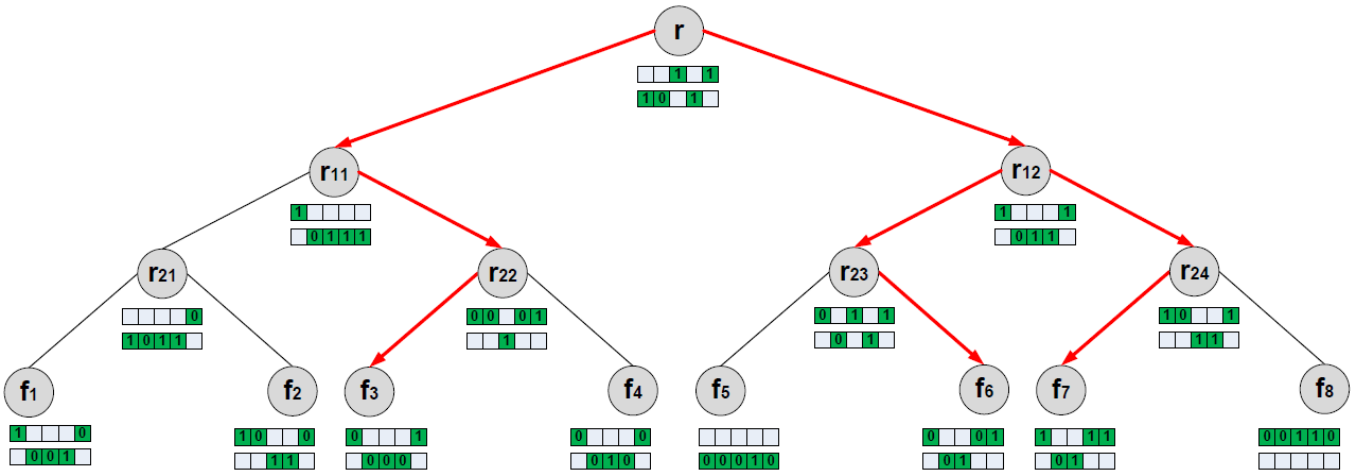

Рисунок 2.1 – Побудова динамічного СШМП з використанням ЧЧД

На рисунку 2.1 зображено реалізацію методу, в якій є 8 документів та 5 ключових слів. Виходячи з кількості ключових слів, кожен вузол зберігає по два 5-ти бітних вектори, один з яких містить справжню інформацію о ключових словах у своїх нащадках, проте у зашифрованому вигляді. Червоними стрілочками показано пошук п’ятого ключового слова, в результаті отримуємо, що п’яте ключове слово знаходиться у 3, 6 та 7 документах.

2.2 Метод Dyn2Lev

У[21] автори запропонували цілих три схеми, які, проте, мають однаковий базис. Перша є базовою, друга має деякі покращення в продуктивності, а третя є динамічною версією другої. Назва методу Dyn2Lev йде саме від динамічної версії (dynamic), а також відсилає на конструктивну особливість: використовуються два рівня (2 level) блоків покажчиків на індекси.

2.2.1 Базова схема

На вхід подається деяка БД, яку будемо позначати як DB, причому вона складається з d кортежів, що містять ідентифікатор та множину ключових слів, тобто:

$$DB = \left(id_i,\ W_i\right)^d, \tag{2.8}$$

де ідентифікатор id – це бітовий рядок довжиною λ,

W_i – це деяка множина, що складається з ключових слів, причому ця множина є підмножиною W.

Треба відзначити, що λ є загальним параметром безпеки у схемі, який також визначає довжину ключа.

Отже, розглянемо конструкцію методу. Маємо деякий словник $Dict$, псевдовипадкову функцію F та симетричну схему шифрування $\sum$. На першому етапі генерується загальний секретний ключ K. На базі цього ключа, подібно до попереднього методу, генеруються ключі для кожного ключового слова за допомогою ПВФ – F, а також шифруються ідентифікатори.

Далі переходимо до ключових слів. Для кожного ключового слова існує деякий набір записів, що містять це ключове слово – $DB(w)$, а ці записи мають певні ідентифікатори. Для ідентифікатору кожного запису в $DB(w)$ розраховується псевдовипадкова позначка шляхом застосування ПВФ до лічильника. Паралельно з цим ідентифікатор шифрується, після чого отримуємо пару позначка-шифртекст, яка додається до списку L. Більш формальний опис наведено у формулах (2.7) та (2.8).

$$\forall\ w \in W : K_1 \,||\, K_2 \leftarrow F(K,\ w) \tag{2.9}$$

$$\forall\ id \in DB(w) : l \leftarrow F(K_1,\ c); d \leftarrow Enc(K_2, id); c = c + 1; L = L \,||\, (l, d), \tag{2.10}$$

де c – це просто деякий лічильник, що зростає після кожної ітерації,

L – це список, що містить пари позначка-шифртекст.

Важливим є те, що всі записи списку сортуються лексикографічно за позначками, тому що без цієї умови з схеми буде витікати інформація про порядок обробки записів. З цього списку будується словник D, що виступає в ролі індекса в цьому методі.

Для пошуку за деяким ключовим словом w, клієнт повинен згенерувати ключ для цього слова, як наведено у формулі (2.9), та відправити його серверу. Сервер в свою чергу обчислює позначку, послідовно перебираючи значення лічильника та обчислюючи ПВФ від нього. Після сервер питається вилучити значення за розрахованим ключом з словника D. Якщо такого ключа в словнику немає, то повертається деяке значення $\perp$, що вказує на необхідність

продовження пошуку. Пошук продовжується поки отримане з словника значення не буде відмінним від $\perp$.

2.2.2 Адаптована схема 2Lev

Проте базова версія, описана вище, має певні недоліки з точки зору ефективності. Першим та самим очевидним рішенням буде згрупувати ідентифікатори для кожного ключового слова, щоб не робити витягнення даних $DB(w)$ разів. Тоді припустимо, що для ключового слова w є $\left|DB(w)\right|$ записів, що потрібно витягнути з БД. Нехай B буде відповідати розміру блока, в якому будуть зберігатися згруповані ідентифікатори, отже, таких блоків буде $\frac{\left|DB(w)\right|}{B}$. А останній блок доповнюється до розміру B, якщо це потрібно.

Таким чином можна значно скоротити кількість витягнень: чим більше розмір блоку, тим ефективніше буде працювати схема. Але тут встає інше питання: для кожного ключового слова w є різна кількість відповідних записів $DB(w)$, таким чином, для деяких слів, котрим відповідають значні обсяги записів, витягнення все ще може бути неефективним. З іншого боку, якщо B обирається надто великим, то будуть страждати ключові слова, котрим відповідає маленька кількість записів, тому як доповнення до розміру блоку B буде створювати багато надлишкової інформації.

Першим рішенням цього недоліку було використання покажчиків, додаючи ще один шар між ключовими словами та ідентифікаторами. В цьому випадку зберігаються блоки ідентифікаторів не у словнику, а у деякому масиві A, причому записуються вони у випадковому порядку. А замість цих блоків у словнику зберігаються покажчики, причому теж у блоках (для меншої кількості витягнень), нехай розмір блоку буде b. Цей підхід дозволяє не створювати надто великі блоки з ідентифікаторами, при цьому значно скорочує кількість витягнень, що необхідно зробити для певного ключового слова.

Проте це все ще містить певні недоліки, тому як для одної БД можуть бути як ключові слова з великою кількістю документів, так і з незначною. Автори пропонують модифікацію схеми під назвою 2lev (в честь якої й йде частина назви

всього методу). Пропонується ввести ще один рівень: обробляти різні множини ідентифікаторів в залежності від їх розміру.

Розрізняти множини будемо на маленькі, середні та великі. Визначається все через розміри блоків: блоки розміром b у словнику та блоки розміром B у деякому зовнішньому масиві. Отже, маленькі множини мають кількість записів менше або дорівнює b, середні множини мають кількість записів, що лежить у межах

$b < \left|DB(w)\right| \le Bb$, великі – $Bb < \left|DB(w)\right| \le B^2 b$. Верхня границя для великих множин дорівнює $B^2 b$, з чого випливає, що необхідно обирати B та b таким чином, щоб в не існувало множини потужністю більше $B^2 b$.

Для маленьких множин ідентифікатори будуть зберігатися напряму у словнику, таким чином, не буде потреби у використанні покажчиків. Для середніх будуть створюватися блоки покажчиків, що вказують на відповідні блоки ідентифікаторів. Для великих множин є 2 рівні (2lev): у словнику зберігаються покажчики, що вказують на блоки покажчиків, які, в свою чергу, вказують на блоки ідентифікаторів. Отже, є три способи. Розглянемо приклад, щоб зрозуміти як змінюється кількість витягнень в залежності від підходу для підвищення ефективності. Візьмемо $B = 500$ та $b = 100$.

Таблиця 2.1 – Кількість витягнень в залежності від підходу

$\left\|D(w)\right\|$	*Блоки ідентифікаторів*	*Блоки покажчиків*	*Два рівня блоків покажчиків*
100	1	1	1
200	2	1	1
1100	11	1	1
5100	51	1	1
10100	101	1	1
50100	501	2	1
500100	5001	11	1
5000100	10001	101	1

Таким чином, з таблиці 2.1 видно, як росте кількість необхідних витягнень для множин різного розміру. Коли кількість записів менше або дорівнює b, то при всіх підходах буде лише одне витягнення, тому не має сенсу використовувати більш ресурсномісткі методи, достатньо групувати ідентифікатори по блоках та зберігати їх у словнику. Коли ж ця межа

перетинається (тобто записів більше 100), то переходимо до другого підходу, який буде ефективним до межи в $B*b$, у нашому випадку – це 50 тисяч записів. І якщо ця межа теж перетинається, то вже використовується третій підхід з двома проміжними рівнями, завдяки чому для множин потужністю до B^2*b (в нашому випадку це 25 мільйонів записів) необхідно буде зробити тільки одне витягнення.

Враховуючи описані вище модифікації схеми з підходом 2Lev, розглянемо зміну алгоритмів побудови та пошуку. Спочатку визначається розмір множини: якщо вона маленька, то створюємо блок довжиною b, в якому будуть зберігатись ідентифікатори, за потребою доповнюємо блок до b елементів, після чого додаємо пару позначка-блок до словнику. Якщо множина середня, то розбиваємо ідентифікатори на блоки довжиною B (останній блок за необхідності доповнюємо), які випадковим шляхом записуємо до зовнішнього масиву A. Позначки на ці блоки зберігаються у іншому блоці розміром b, який у парі з позначкою записується до словнику. А якщо множина є великою, то ідентифікатори розбиваються на блоки довжиною B (останній блок за необхідності доповнюємо), які випадковим шляхом записуються до зовнішнього масиву A. Далі з покажчиків на блоки ідентифікаторів теж формуємо блоки розміром B (останній також доповнюємо за потребою), які теж додаються до масиву A. Нарешті з покажчиків на блоки покажчиків формується блок довжиною b (якщо необхідно, то також доповнюємо), а пару позначка-блок додаємо до словника. Після перебору всіх множин отримуємо словник та масив, що складають зашифровану базу даних (ЗБД або EDB).

Для пошуку сервер робить дії, аналогічні з пошуком у базовій конструкції, але є й відмінність: тепер у словнику для кожного ключового слова завжди є тільки один запис, відповідно, лічильник завжди буде дорівнювати нулю. Таким чином, позначка розраховується завжди від 0. Якщо клієнт надіслав коректний ключ, то сервер успішно витягує блок, що відповідає позначці. Якщо цей блок містить ідентифікатори, то він відправляє їх клієнту, інакше він використовує покажчики для отримання блоків з масиву A. Якщо ці блоки містять ідентифікатори, то він їх відправляє клієнту, інакше знову використовує

позначки, але тепер вже точно отримує блоки ідентифікаторів, які відправляє клієнту.

2.2.3 Динамічна схема Dyn2Lev

Для динамічної конструкції передбачається використання двох ЗБД, перша представляє собою одну зі схем, описаних вище, а друга – самий базовий варіант методу. Отже, якщо треба додати або оновити запис, то клієнт відправляє відповідний запит до серверу, надсилаючи також ідентифікатор запису та відповідне цьому запису ключове слово або множину ключових слів. Для кожного з ключових слів, що відповідають цьому запису, треба згенерувати відповідний ключ, далі генерується позначка за значенням лічильника, а також шифрується ідентифікатор запису. Пара позначка-шифртекст вноситься до другої ЗБД – D^+, тобто словника, що буде зберігати додані пари позначка-індекс запису, внесення також виконується у лексикографічному порядку для уникнення витоку інформації. Треба відмітити, що, для проведення таких операцій клієнт повинен зберігати поточне значення лічильника для кожного ключового слова, або ж це можна покласти на сервер та витягувати значення при необхідності. З цією метою створюється ще один словник D^{count}, який містить пари ключове слово-значення лічильника. Крім цього, для цієї ЗБД генерується окремий ключ K^+.

Пошук за певним ключовим словом тепер проводиться в два етапи: спочатку в статичній EDB, після в словнику з доданими записами D^+. Для цього клієнт передає обидва ключі – K та K^+, а сервер, поступово збільшуючи лічильник, отримує відповідні ідентифікатори та передає клієнту.

Проте схема не є повністю динамічною, якщо в ній відсутня можливість видалення записів. Для цієї мети створюється нова структура – множина S_{rev}. Для операцій видалення генерується ще один ключ – K^-. Подібно до операції додавання, маємо ідентифікатор деякого запису, а також множину ключових слів. Для кожного ключового слова генерується специфічний ключ, за допомогою якого знаходиться ПВФ від ідентифікатора запису, назвемо це

значення ідентифікатор відкликання (англ. revocation id). Всі ці записи додаються до множини у лексикографічному порядку.

У зв'язку з появою можливості видалення, процес додавання теж дещо змінюється, оскільки, як можна зрозуміти з опису етапу видалення, буквально нічого не видаляється. При додаванні є деякий ідентифікатор запису та множина ключових слів. Сервер перебирає ключові слова, генеруючи специфічні для кожного з них ключі, шляхом знаходження ПВФ з ключом K^+. Для кожного з ключових слів він отримує значення лічильника з D^{count}, після чого генерує позначку. Також він шифрує ідентифікатор запису за допомогою ключа K_2^+, а тепер ще й знаходить ПВФ з ключем K_1^- від цього ідентифікатора. Отриману трійку (позначка, шифртекст, ідентифікатор відкликання) додаємо до списку L у лексикографічному порядку, після чого відправляємо серверу. Сервер, в свою чергу, перевіряє ці трійки, і генерує вектор-відповідь r для клієнта наступним чином: якщо в i-ій трійці ідентифікатор відкликання наявний у множині S_{rev}, то в векторі i-ий біт встановлюється в 1, а сам ідентифікатор видаляється з множини S_{rev}. Інакше сервер встановлює біт в 0, після чого додає пару позначка-шифртекст до D^+. Наприкінці сервер відправляє вектор r клієнту. Клієнт перевіряє отриманий вектор та в позиціях, в який біт встановлено в 0, він обирає відповідне ключове слово та збільшує відповідний лічильник.

Для пошуку клієнт відправляє всі ключі для певного ключового слова: $\left(K_1, K_2, K_1^+, K_2^+, K_1^-\right)$. Сервер робить пошук в точності, як у алгоритму вище, але наприкінці перевіряє кожен знайдений ідентифікатор: він знаходить ПВФ від нього за ключом K_1^- та дізнається, чи наявне розраховане значення у множині S_{rev}. Якщо так, то він відзиває цей ідентифікатор, якщо ні – відправляє його користувачу.

2.3 Логічне симетричне шифрування з можливістю пошуку

2.3.1 Криптопримітиви методу

Наступний метод визначено у роботі [22], в якій зосереджується увага на виразності методу. Два попередньо розглянуті методи дозволяють по одному

ключовому слову отримати певний перелік ідентифікаторів документів або записів БД, що містять це ключове слово. Проте іноді зручним є пошук тих документів чи записів, що містять в собі деяку множину ключових слів. Перед тим, як перейти до безпосереднього розгляду методу, визначимо декілька понять, важливих для розуміння реалізації методу.

Онлайн шифр – це блочний шифр, визначений у роботі [23]. Його особливістю є те, що шифрування може проводитись в онлайн або у потоковому режимі. Такі шифри мають наступну вимогу: якщо є деяке повідомлення $M = M_1 \| M_2 \| ... \| M_l$ та відповідний йому шифртекст $C = C_1 \| C_2 \| ... \| C_l$, то для розрахування деякого блоку шифртексту C_i достатньо знати блоки відкритого тексту з 1 до i. Таким чином, блок шифртексту C_i не залежить від блоків відкритого тексту $M_{i+1},\ldots,M_l$.

Мультивідображення (MB) або multimap (MM) – це відображення, яке для кожного ключа зіставляє більше одного значення. Зокрема, у схемі використовується кортеж значень.

Фільтр Блума – це ймовірнісна структура даних, яку запропонував Бартон Блум у 1970 році [24]. Ця структура даних має дві операції: додавання елемента до множини та перевірка наявності елементу у множині. Ймовірнісність структури полягає у тому, що можливі відповіді на питання наявності елементу – це «можливо» або «ні». Тобто фільтр Блума припускає хибно позитивні відповіді, але хибно негативні неможливі. Ідея полягає у тому, що існує бітовий масив розміру m та k геш-функцій, що видають значення від 0 до $m-1$. Якщо треба додати деякий елемент, то від нього розраховуються всі геш-функції, та відповідні розрахованим значенням біти масиву встановлюються в одиницю. А при перевірці також розраховуються значення геш-функцій та витягуються відповідні значення з масиву, і, якщо всі значення дорівнюють одиниці, то елемент може бути присутнім, а якщо маємо хоча б один нуль, то елементу точно немає у структурі даних.

Inclusion-exclusion principle (IEP) або принцип включень-виключень – це техніка, що дозволяє визначати потужність об’єднань кінцевого числа кінцевих множин, завдяки ж цьому можна отримати множину унікальних елементів цього об’єднання. Принцип, як випливає з назви, полягає у почерговому застосуванні включення та виключення: спочатку включаються всі множини; після цього виключаються попарні перетини; далі включаються потрійні перетини; після виключаються четверні перетини і так далі до n-го порядку, де n – кількість множин.

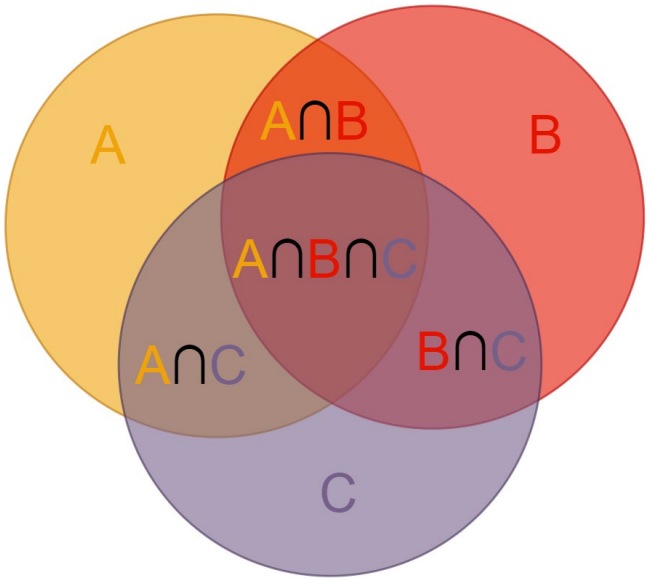

Рисунок 2.2 – Перетин трьох множин

На рисунку 2.2 можна бачити перетин множин A, B і C, отже, розглянемо приклад IEP для цього випадку. При підсумовуванні трьох множин, двічі враховуються елементи, що наявні в двох множинах, та тричі враховуються елементи, що наявні в усіх трьох множинах. Тому, після підсумовування всіх множин, необхідно виключити подвійні перетини. Таких перетини три, і після їх віднімання елементи, що наявні рівно у двох множинах, будуть представлені у єдиному екземплярі. Але, як зазначалось вище, елементи, що наявні у всіх трьох множинах, були враховані тричі, проте на минулому кроці їх тричі відняли, тобто тепер їх не залишилось. Отже, необхідно знову включити їх до загального результату. Таким чином, було продемонстровано застосування IEP для отримання унікальних елементів об'єднання множин A, B і C.

2.3.2 Базова схема IEX

Перейдемо до визначення головної конструкції методу – IEX (inclusion-exclusion encryption), назва якої відсилає на принцип включень-виключень (IEP). Для її побудови знадобиться схема шифрування словнику Σ_{DX}, схема шифрування мультивідображення Σ_{MM}, псведовипадкова функція F та схема шифрування з секретним ключем SKE. Розглянемо нижче три алгоритми, визначені для цієї схеми.

1) Налаштування

На вхід приймається параметр безпеки k, який визначає довжину ключів, та деяка база даних DB. На цьому етапі ініціалізується словник DX та глобальне мультивідображення MM_g. Це глобальне МВ встановлює відповідність між деяким ключовим словом w та зашифрованими ідентифікаторами документів, в яких міститься w, – $DB(w)$. Зашифровані ідентифікатори у методі називаються тегами та отримуються через алгоритм шифрування схеми SKE наступним чином:

$$tag_{id} = SKE.Enc_{K_1}\left(id; F_{K_2}\left(id \,\|\, w\right)\right) \tag{2.11}$$

У формулі (2.11) наведено отримання тегу для деякого ідентифікатора запису $id \in DB(w)$, що містить ключове слово w. При цьому для шифрування використовується ключ, що генерується за допомогою ПВФ від конкатенації ідентифікатора документу та ключового слова.

Далі у глобальне МВ за ключем w встановлюється відображення на розрахований тег. Після цього ініціалізується локальне мультивідображення (ЛМВ) для деякого ключового слова w – MM_w. Розмір цього МВ будемо визначити наступним чином: нехай маємо ключові слова, що наявні у документах разом з w, позначати їх будемо як $co(w)$, отже, й розмір МВ буде співпадати з потужністю $co(w)$. Тобто можна сказати, що це ключові слова, множини ідентифікаторів документів котрих мають перетин з множиною

$DB(w)$. Далі для кожного ключового слова $v \in co(w)$ перебираються всі ідентифікатори документів конкретного перетину $DB(w) \cap DB(v)$, та для кожного з них розраховується тег за формулою (2.11). Розраховані значення тегів додаються до локального мультивідображення MM_w за ключем v. Отримане локальне мультивідображення використовується для швидкого визначення документів, що містять обидва ключових слова – w та v. Наприкінці, локальне МВ шифрується за допомогою схеми Σ_{MM}, отримуємо зашифроване МВ – EMM_w, яке записуємо у словник DX за ключем w.

Перелічені вище кроки робляться для кожного ключового слова, після чого глобальне відображення MM_g шифрується за схемою Σ_{MM} та отримується EMM_g, а також шифрується словник DX за схемою Σ_{DX} та отримується EDX. Отже, отримана зашифрована БД – $EDB = \left(EMM_g, EDX\right)$.

Для більшої наочності розглянемо приклад: по-перше, визначимо глобальне мультивідображення та побудуємо відповідну структуру у вигляді перетину множин елементів, результат представимо на рисунку 2.3.

MMg

w_1	tag_1,tag_4, tag_{10}
w_2	tag_3,tag_5
w_3	tag_1,tag_9
w_4	tag_6
w_5	tag_4, tag_5, tag_7
w_6	tag_1,tag_4, tag_8

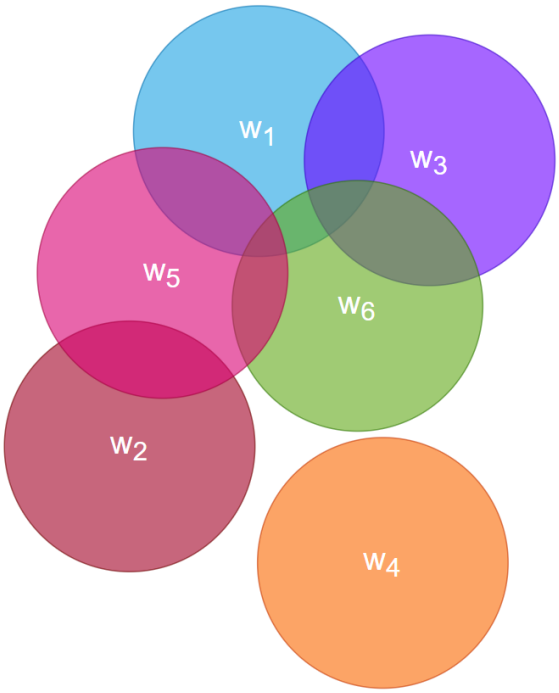

Рисунок 2.3 – Приклад побудови глобального мультивідображення

Насправді, теги з однаковим індексом відрізняються самі по собі, оскільки для кожного ключового слова використовується спеціальний для нього

секретний ключ, але індекс вказує на те, що теги згенеровані для одного й того ж документу чи запису. Після цього переходимо до побудови локального МВ, відобразимо його на рисунку нижче.

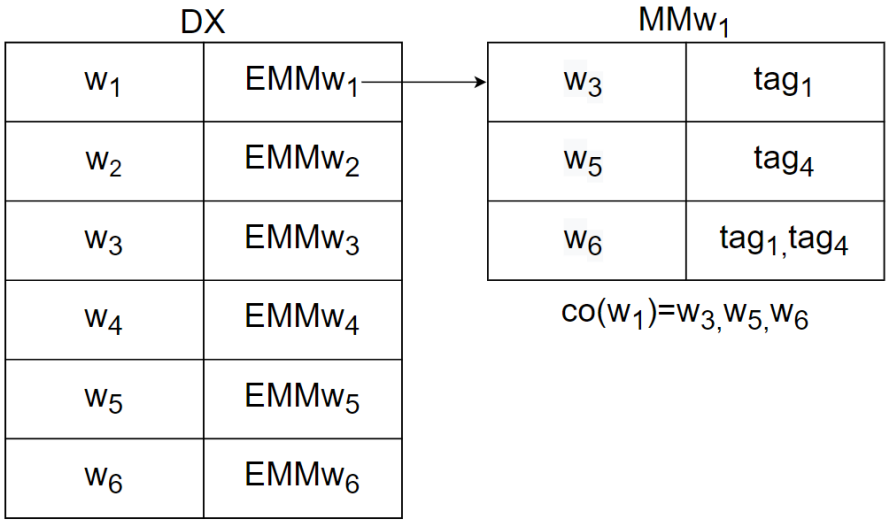

Рисунок 2.4 – Приклад побудови словника та локального мультивідображення

З рисунку 2.4 видно, що у словнику записані всі ключові слова, кожне з котрих посилається на відповідне йому зашифроване локальне мультивідображення. Також приклад локального МВ наведено для першого ключового слова: у ньому записані всі ключові слова з множини $co(w_1)$, котрим зіставлені теги тих документів, в яких наявні обидва ключових слова.

2) Генерація токену

Для того, щоб сервер міг здійснювати пошук за зашифрованими даними, клієнт повинен згенерувати та передати відповідний токен. Отже, нехай є деякий запит до серверу, що містить множину ключових слів $\mathrm{w} = (w_1, \ldots, w_q)$.

Розглянемо ідею, за якою планується витягнення необхідних документів, після чого опишемо алгоритм генерації токену. Кожне з ключових слів має перелік $DB(w)$ ідентифікаторів документів, в яких воно наявне. Щоб отримати всі елементи без дублювання виконуємо наступний алгоритм: беремо перше ключове слово w_1 та отримуємо всі ідентифікатори документів, яких воно наявне – $DB(w_1)$. Після цього видаляються всі ідентифікатори документів, в яких наявне хоча б одне з ключових слів від w_2 до w_q. Таким чином, отримуємо

унікальні елементи для ключового слова w_1. Далі додаємо всі ідентифікатори w_2, після чого аналогічно видаляємо ті ідентифікатори, що містять хоча б одне з ключових слів від w_3 до w_q. Цей крок повторюється для всіх ключових слів до w_{q-1}. Ідентифікатори останнього ключового слова просто додаються до загального списку.

Тепер детальніше до алгоритму. Мається зашифрована БД, в якій є глобальне МВ та словник, що містить локальні МВ, всі вони також зашифровані. Цьому для кожного з ключових слів $w_i \in \left(w_1,\ldots,w_{q-1}\right)$ генерується так званий підтокен:

$$tk_i = \left(dtk_i, gtk_i, ltk_{i+1},\ldots,ltk_q\right), \tag{2.12}$$

де dtk_i – це токен, що використовується для витягнення зашифрованого локального МВ EMM_{w_i} з зашифрованого словника EDX, генерується цей токен з використанням схеми Σ_{MM},

gtk_i – це токен, що використовується для витягнення множини тегів T_{w_i} документів $DB\left(w_i\right)$ з зашифрованого глобального МВ – EMM_g, генерується цей токен з використанням схеми Σ_{DX}.

Локальні токени $ltk_j, j \in \left(i+1,\ldots,q\right)$ використовуються для витягнення з локального МВ списків тегів $T_i^{'}$ тих документів, що містять одночасно w_i та w_j. Для генерування токенів використовується схема Σ_{MM}. Як результат алгоритму отримуємо $tk = \left(tk_1,\ldots,tk_{q-1},gtk_q\right)$.

3) Пошук

На вхід сервер приймає токен tk. Як і було описано у попередньому алгоритмі, сервер використовує токени для поступового знаходження списку ідентифікаторів документів. Для цього він поступове виконує наступне: для $w_i, i \in [1, q-1]$ витягує з глобального мультивідображення теги документів, що

містять це ключове слово. Після цього від цієї множини він віднімає теги документів, що містять ключові слова, які йдуть у запиту після нього:

$$DB(w_i) \setminus \bigcup_{j=i+1}^{q} \left(DB(w_i) \bigcap DB(w_j) \right) \tag{2.13}$$

Теги останнього ключового слова він просто знаходить без якогось віднімання. У результаті множини об'єднуються та клієнту відправляється множина тегів $T = \bigcup_{i=1}^{q} T_i$.

2.3.3 Логічна схема BIEX

Проте все це відноситься до операції диз'юнкції: описаний метод повертає всі ідентифікатори документів, в яких наявне хоча б одне ключове слово з запиту. А схема позиціонує себе як логічну, тобто таку, яка як запит приймає логічне твердження. Отже, розглянемо логічну або Boolean модифікацію IEX (BIEX) нижче, вона має такі ж три алгоритми, проте алгоритм налаштування повністю співпадає, тому перейдемо до алгоритмів пошуку та генерації токену.

Спочатку визначимо, що любий логічний запит можна записати у кон'юнктивній нормальній формі, тобто:

$$\Delta_1 \wedge \ldots \wedge \Delta_l, \ \Delta_i = w_{i,1} \vee \ldots \vee w_{i,q} \tag{2.14}$$

А набір кон'юнкцій – це знаходження перетину деякої кількості множин, кожна з котрих може бути представлена деякою диз'юнкцією Δ_i. Причому треба відмітити, що перетин цих множин завжди буде підмножиною Δ_1. По суті, напочатку маємо множину елементів Δ_1, після цього видаляємо всі елементи, котрих немає у Δ_2, далі всі елементи, котрих нема у Δ_3 і так до Δ_l.

1) Генерація токену

Отже, є l виразів, які поєднуються кон'юнкцією, причому результат всього запиту завжди буде підмножиною першого виразу Δ_1. При цьому у кожному виразі Δ_i є якесь число ключових слів q_i. Таким чином, для першого виразу

виконуємо алгоритм генерації токену зі схеми IEX. На виході маємо токен виду $tk_1 = \left(tk_{1,1},\ldots,tk_{1,q_1-1},gtk_{1,q_1}\right)$, а кожен підтокен токену tk_1 має наступний вигляд: $tk_{1,1} = \left(dtk_{1,1},gtk_{1,1},ltk_{1,1,2},\ldots,ltk_{1,1,q_1}\right)$.

Далі потрібно згенерувати токени для всіх інших виразів $\Delta_2 - \Delta_l$. Нехай є деяке $j \in [q_i], i \in (2,l)$, тоді:

$$tk_i = \left(tk_{i,1},\ldots,tk_{i,q_i}\right); tk_{i,j} = \left(ltk_{i,j,1},\ldots,ltk\left(i,j,q_i\right)\right) \quad (2.15)$$

2) Пошук

Як зазначалось вище, спочатку необхідно отримати теги, що задовольняють виразу Δ_1. У алгоритмі генерації токену було згенеровано токен для цього виразу, використовуючи алгоритм IEX.Token, отже, можна отримати теги виконавши алгоритм IEX.Search, передавши туди отриманий токен tk_1:

$$I_1 \leftarrow IEX.Search\left(EDB,tk_1\right) \quad (2.16)$$

Отримуємо маємо множину тегів I_1, що відповідає результату виразу Δ_1. Далі знаходимо перетин отриманої множини з $DB(\Delta_2)$, при цьому пам'ятаємо, що Δ_2 – це диз'юнкція, що складається з q_2 ключових слів, тобто необхідно знайти наступний вираз:

$$\Delta_1 \wedge \Delta_2 = \Delta_1 \wedge \left(w_{2,1} \vee \ldots \vee w_{2,q_2}\right) = \left(\Delta_1 \wedge w_{2,1}\right) \vee \ldots \vee \left(\Delta_1 \wedge w_{2,q_2}\right) \quad (2.17)$$

Ми перебираємо всі ключові слова, що наявні у Δ_2: по порядку та виявляємо, чи є спільні документи у I_1 та $w_{2,j}, j \in \left[q_1\right]$. Для цього витягуємо токен словнику dtk_j з відповідного токену $tk_{1,j}$, який знаходиться у токені tk_1. За допомогою цього токену можна витягнути зі словника локальне мультивідображення для ключового слова $w_{1,j}$ з виразу Δ_1. Використовуючи

ЛМВ, перебираємо кожне ключове слово з Δ_2, знаходячи спільні документи. Якщо такі документи наявні, то додаємо їх до множини I_2.

Описані вище кроки повторюються й для інших виразів $\Delta_3 - \Delta_l$. На виході отримуємо множину I_l, яка й буде містити відповідь на логічний вираз.

Розглянемо приклад для пошуку по приведеному алгоритму, використовуючи систему наведену на рисунках 2.3 та 2.4. Нехай є наступний запит:

$$w_1 \vee w_2 \wedge w_3 \vee w_4 \wedge w_6 \rightarrow \Delta_1 = w_1 \vee w_2; \Delta_2 = w_3 \vee w_4; \Delta_3 = w_6 \qquad (2.18)$$

Генерація токенів досить очевидна, тому перейдемо відразу до пошуку. У відповідність до алгоритму, спочатку розраховуємо теги для виразу $\Delta_1 = w_1 \vee w_2$ шляхом обчислення алгоритму IEX.Search. На першому кроці для всіх ключових слів, окрім останнього (згідно нашого прикладу, це лише одне ключове слово – w_1), з глобального МВ витягуємо кортеж тегів $T_1 = (tag_1, tag_4, tag_{10})$. З цього кортежу прибираємо теги тих документів, які містять інші ключові слова з виразу Δ_1, для цього звертаємось до локального мультивідображення EMM_{w_1}, але в ньому немає значення за ключем w_2. Після цього з глобального МВ витягуємо теги для останнього ключового слова $T_2 = (tag_3, tag_5)$. Наприкінці знаходимо об'єднання множин тегів:

$$\bigcup_{i=1}^{2} T_i = (tag_1, tag_3, tag_4, tag_5, tag_{10}) = I_1 \qquad (2.19)$$

Отриманий результат і буде відповідати першому запиту Δ_1 та запишеться в I_1. Далі кроки від 2 до l виконуються однаково:

- перебираємо всі ключові слова з Δ_1, для кожного з них витягуємо локальне МВ, у прикладі це EMM_{w_1} та EMM_{w_2};
- для кожного витягнутого ЛМВ перебираємо кожне ключове слово в Δ_i, тобто спочатку це $\Delta_2 : w_3, w_4$, потім перебираємо $\Delta_l = \Delta_3 : w_6$:

- за допомогою ЛМВ знаходимо теги документів, що містять одночасно ключове слово з Δ_1 та Δ_i, отримуємо набір тегів I'; після цього знаходимо перетин множин:

$$I' = EMM_{w_1}\left[w_3\right] = \left(tag_1\right) \tag{2.20}$$

$$I_2 = I_2 \bigcup \left(I_{2-1} \bigcap I'\right) = \left(tag_1, tag_3, tag_4, tag_5, tag_{10}\right) \bigcap \left(tag_1\right) = \left(tag_1\right) \tag{2.21}$$

$$I' = EMM_{w_1}\left[w_4\right] = () \tag{2.22}$$

$$I' = EMM_{w_2}\left[w_3\right] = () \tag{2.23}$$

$$I' = EMM_{w_2}\left[w_4\right] = () \tag{2.24}$$

$$I' = EMM_{w_1}\left[w_6\right] = \left(tag_1,\ tag_4\right) \tag{2.25}$$

$$I_3 = I_3 \bigcup \left(I_{3-1} \bigcap I'\right) = \left(tag_1\right) \bigcap \left(tag_1, tag_4\right) = \left(tag_1\right) \tag{2.26}$$

$$I' = EMM_{w_2}\left[w_6\right] = () \tag{2.27}$$

Таким чином, отримуємо, що відповіддю на логічний запит буде множина $I_l = I_3 = \left(tag_1\right)$, ця множина передається клієнту.

2.3.4 ZMF

Розглянута вище схема має явний недолік – просторову складність:

$$O\left(\sum_{i=1}^{|W|}\left|DB\left(w_i\right)\right| + \sum_{i=1}^{|W|}\sum_{j=1}^{|co(w_i)|}\left|DB\left(v_j\right) \bigcap DB\left(w_i\right)\right|\right) \tag{2.28}$$

Головна неефективність полягає у другому доданку, який представляє собою локальні мультивідображення. Тоді постає питання в тому, яку структуру даних використати, щоб зменшити просторову складність та збалансувати схему. Проблема у тому, що всі сублінійні структури теж мають значну просторову складність. З цього приводу звертаємось до лінійних структур, які можна використовувати зважаючи на те, що локальні МВ містять відносно невелику

кількість елементів, тобто це не позначиться на продуктивності серйозним чином.

Автори методу посилаються на роботу Гоха [13], в якій він запропонував схему, засновану на фільтрах Блума, яка проте не є адаптивно стійкою. Камара та Моатаз наполягають на тому, що схема не є адаптивно стійкою через використовувану у трансформації схему шифрування множин, яка теж не є такою. Якщо ж використовувати відповідну стійку, то й весь метод буде адаптивно стійким. Згідно [7], в схемах структурного шифрування адаптивність може бути досягнута досить ефективно шляхом використання операцій XOR та ПВФ.

Отже, для досягнення адаптивної стійкості, пропонується маскування бітів фільтра Блума шляхом застосування XOR з доповненням, згенерованим псевдовипадковою функцією G. Але ці доповнення треба генерувати таким чином, щоб їх значення залежали тільки від номеру біта фільтру Блума. Необхідність цього можна обґрунтувати наступним чином: якщо є два різні елементи, які необхідно додати до фільтру, і геш-значення від цих елементів співпадають (тобто вказують на один і той же біт фільтру), то існує ймовірність, що для них згенерується різна маска, що порушить коректність перевірки наявності елементу. Тому доповнення буде визначатись як ПВФ від позиції біту. Враховуючи все описане вище, токен для перевірки наявності деякого елементу a буде виглядати наступним чином:

$$tk = \left(F_{K_1}(a), G_{K_2}\left(H_1\left(F_{K_1}(a)\right)\right), \ldots, G_{K_2}\left(H_\lambda\left(F_{K_1}(a)\right)\right)\right) \qquad (2.29)$$

Отже, схема є компактною і адаптивно надійною, але є ще одна вимога: вона повинна бути мультиструктурною: якщо мається декілька структур даних, що зашифровані під одним ключем K, то існує єдиний токен tk, за допомогою якого можна витягнути дані з усіх зазначених структур. Ця вимога випливає з того, що структура даних повинна імітувати локальні МВ, за якими сервер здатен проводити пошук, використовуючи єдиний токен, що був наданий клієнтом. При

цьому кожна структура даних може бути різного розміру, отже, і потребує різні фільтри Блума з різною кількістю геш-функцій та діапазоном геш-значень, що, в свою чергу, потребує зберігання декількох доповнень для кожного фільтру. Через це розмір токену знову сильно зростає, що призводить до неефективності.

Тому автори пропонують нову структуру даних, яка базується на фільтрах Блума: фільтр-матрьошка. Ідея полягає у тому, що створюється єдиний фільтр Блума для максимальної за розміром множини елементів розміром $\frac{\lambda}{\ln 2} * \max\left(\left|S_i\right|\right)$, який буде містити в собі вкладені фільтри Блума для менших множин. Також обирається λ геш-функцій. Тепер для кожної множини S_i будується фільтр Блума розміром $\frac{\lambda}{\ln 2} * \left|S_i\right|$ з геш-функціями $\left(H_1^i, \ldots, H_h^i\right)$, де для всіх $j \in [\lambda]$:

$$H_j^i(a) = H_j(a)_{|p_i},\ p_i = \log\left(\frac{\lambda}{\ln 2} * \left|S_i\right|\right), \tag{2.30}$$

де нижній індекс $|\, p_i$ означає, що деякий рядок усікається до p_i біт.

Такі геш-функції, як і відповідні ним фільтри Блума, будемо називати виведеними. Також зазначимо відразу, що, якщо до деякого рядка s довжиною k біт ставиться верхній індекс $s^{|n}$, то це означає, що рядок доповнюється нулями до n біт. А якщо ставиться індекс $s^{||n}$, то кожен біт рядка доповнюється нулями до довжини n: $s^{||n} = \left(s_1^{|n}, \ldots, s_k^{|n}\right)$.

Також важливим є те, що є окремий фільтр для кожної множини у нашій мультиструктурній схемі, тому доповнення-маска генерується з залежністю від певного фільтра. Для цього подаємо випадковому оракулу номер біта та ідентифікатор фільтра, та отримуємо 0 чи 1. Перейдемо безпосередньо до алгоритмів методу.

1) Генерація ключів

На цьому етапі генеруються два ключи: перший для ПВФ, другий для онлайн шифру.

2) Шифрування

Маємо деяку множину елементів S. Генерується відповідний бітовий масив розміру $m = \frac{\lambda}{\ln 2} * |S|$ та ініціалізується нулями. Далі перебираються всі елементи множини $a \in S$, знаходиться значення ПВФ від елементу:

$$T = F_{K_1}(a) \tag{2.31}$$

Далі для цього ж елементу перебираються всі геш-функції $i \in [\lambda]$, обчислюється геш $l = H_i(T)$, що визначає номер біту, який потрібно встановити. Але до встановлення біту потрібно зашифрувати це значення за допомогою онлайн шифру:

$$s = OC_{K_2}\left(l^{\|B}_{|\log(m)*B}\right) \tag{2.32}$$

Тобто l розділяється на біти, кожен біт доповнюється нулями до довжини B. Отримане значення усікається до довжини $\log(m) * B$. Тоді в масив записується:

$$A\left[l_{|\log m}\right] = 1 \oplus R\left(s, id(S)\right) \tag{2.33}$$

Усікання до $\log m$ необхідно з того приводу, що використовуються геш-функції, визначені для найбільшої множини, які можуть згенерувати на виході велике значення, тому потрібно провести усічення, щоб можливе значення індексу не виходило за межі виведеного фільтру.

Після цього треба також розрахувати маску для всіх інших позицій у масиві, щоб під час тесту не виникало помилок, та змінити відповідні значення. Наприкінці отримуємо зашифровану множину $ESET = A$.

3) Генерація токену

Для генерації токену клієнт розраховує ПВФ як у формулі (2.31), після чого обчислює значення геш-функцій від нього – l. Отримане геш-значення подаємо на вхід онлайн шифру та отримуємо $s_i = OC_{K_2}\left(l^{\|B}\right)$. В результаті отримуємо токен виду:

$$tk = \left(T, s_1, \ldots, s_\lambda\right) \tag{2.34}$$

4) Перевірка

Сервер отримує токен від клієнта та робить пошук за зашифрованою множиною $ESET$. Для цього він перебирає всі геш-функції та обчислює суму за модулем 2 певного біту та визначеної маски:

$$b_i = A\left[H_i(T)_{|\log m}\right] \oplus R\left(\left(s_i\right)_{|\log(m)*B}, id\left(ESET\right)\right);\ i \in [\lambda] \tag{2.35}$$

Якщо всі значення, обчислені за формулою (2.35), дорівнюють одиниці, то елемент з великою ймовірністю наявний, інакше – ні.

2.3.5 Динамічна схема DIEX

Також автори роботи пропонують динамічний (dynamic) варіант схеми IEX (DIEX). При цьому алгоритм налаштування, генерації токену пошуку та сам пошук майже повністю співпадають з статичною схемою IEX. У якості відмінності є те, що в динамічному варіанті застосовується динамічне глобальне МВ EMM_g^+ та динамічний словник EDX^+, локальні ж МВ залишаються статичними. Також на етапі налаштування додатково генерується змінна стану st, що використовується для генерації токенів. Розглянемо нові алгоритми.

1) Генерація токену оновлення

Маємо три вхідних параметри – операція $op = \left\{edit^+, edit^-\right\}$, ідентифікатор документу чи запису id, множина ключових слів W_{id}. Якщо потрібно виконати операцію додавання, то токен генерується наступним чином:

а) перебираємо кожне ключове слово $w \in W_{id}$;

b) розраховується тег, як у формулі (2.11);

c) генерується токен оновлення за схемою мультивідображення:

$$\left(utk_g^w, st_g\right) \leftarrow \Sigma_{MM}^{+}.Token^{update}\left(K, st_g, \left(op, w, tag_{id}\right)\right), \tag{2.36}$$

де st_g – це змінна, в якій зберігається інформація про стан глобального мультивідображення.

d) створюється локальне мультивідображення, куди додається кожне ключове слово з множини W_{id}, окрім w. У відповідність доданим словам ставиться тег tag_{id}, розрахований у пункті b;

e) у відповідність до алгоритму схеми МВ $\Sigma_{MM}.Setup$ проводиться налаштування локального МВ, на виході отримуємо зашифроване МВ EMM_w;

f) генерується токен оновлення за схемою словника:

$$\left(utk_d^w, st_d\right) \leftarrow \Sigma_{DX}^{+}.Token^{update}\left(K, st_d, \left(op, w, EMM_w\right)\right) \tag{2.37}$$

g) отримуємо токен для оновлення:

$$utk = \left(op, \left(utk_d^w\right)_{w \in W_{id}}, \left(utk_g^w\right)_{w \in W_{id}}\right) \tag{2.38}$$

Якщо ж необхідно провести видалення, то токен генерується наступним чином:

a) перебираємо кожне ключове слово $w \in W_{id}$;

b) розраховується тег, як у формулі (2.11);

c) розраховується токен оновлення за формулою (2.36);

d) отримуємо токен для оновлення:

$$utk = \left(op, \left(utk_g^w\right)_{w \in W_{id}}\right) \tag{2.39}$$

Також в обох випадках наприкінці отримуємо оновлений стан:

$$st = \left(st_g, st_d \right) \tag{2.40}$$

2) Оновлення

При оновленні також можливі 2 ситуації: додавання та видалення. Якщо у токені передається операція видалення, то перебираються всі токени оновлення МВ, для кожного з них виконується алгоритм оновлення зі схеми Σ_{MM}^{+}:

$$EMM_g \leftarrow \Sigma_{MM}^{+}.Update\left(EMM_g, utk_g^{w_i}, op \right) \tag{2.41}$$

Якщо ж необхідно додати елементи, то виконується операція оновлення не тільки для глобального мультивідображення, але й ще для словника:

$$EDX \leftarrow \Sigma_{DX}^{+}.Update\left(EDX, utk_d^{w_i}, op \right) \tag{2.42}$$

На виході отримуємо оновлену базу даних:

$$EDB = \left(EDX, EMM_g \right) \tag{2.43}$$

2.4 Метод Σοφος

Σοφος – це симетрична схема шифрування з можливістю пошуку, що відповідає поняттю прямої секретності, запропонована у [25]. Перед початком розгляду схеми визначимо основний її криптопримітив – перестановку лазівки або trapdoor permutation (TDP).

TDP – це така перестановка π над деякою множиною D, що за допомогою відкритого ключа PK, π може бути обчислена без будь-яких проблем за прийнятний час, але інверсія π^{-1} може бути ефективно обчислена лише за допомогою секретного ключа SK.

Розглянемо основну ідею методу: маємо деяку множину ключових слів W, для кожного $w \in W$ існує відповідний список індексів документів, що містять це ключове слово – $\left(ind_0, \ldots, ind_{n_w - 1} \right)$, де $n_w = \left| DB(w) \right| - d_w$ – це кількість документів, що містять ключове слово w, за винятком видалених. Кожний

елемент цього списку шифрується та записується у логічному місці, яке виводиться від ключового слова w та номеру ідентифікатора документа – c. Це логічне місце позначається як $UT_c(w)$ та, по суті, є токеном оновлення, відповідно, якщо клієнт захоче додати новий документ до БД, який містить ключове слово w, то йому треба зашифрувати індекс та розрахувати логічну позицію для нього, тобто токен оновлення.

Для пошуку клієнт повинен згенерувати токен пошуку $ST(w)$, який дозволяє серверу згенерувати відповідний токен оновлення за допомогою геш-функції, цей токен дозволяє витягнути зашифрований індекс документу. При цьому, мається n_w індексів ($c = n_w - 1$), що відповідають ключовому слову w, які необхідно отримати при пошуку, але таким чином, щоб сервер не зміг отримати індекси оновлення $UT_i, i > c$, тобто логічні місця, де будуть зберігатися додані у майбутньому ідентифікатори.

Тут вступає у гру саме TDP, що зазначалась вище: за допомогою токену пошуку $ST_i(w)$ сервер зможе розрахувати $ST_{i-1}(w)$, використовуючи відкритий ключ PK, але для розрахування $ST_{i+1}(w)$ знадобиться секретний ключ. Розглянемо наочну схему, представлену авторами на рисунку 2.5.

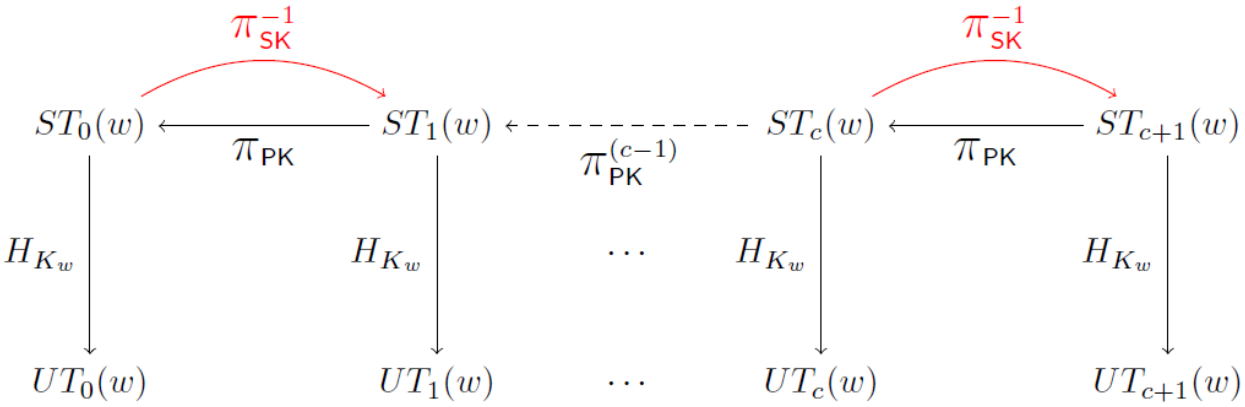

Рисунок 2.5 – Відношення між токенами пошуку та токенами оновлення

Розглянемо детальніше алгоритми, з яких складається метод, всього їх три: налаштування, пошук та оновлення.

1) Налаштування

На цьому етапі генерується ключ K_S, що буде використовуватись при генеруванні унікальних ключів для ключових слів. Також генеруються секретний та відкритий ключі для TDP. Ініціалізуються два відображення: W – для кожного ключового слова w містить відповідну пару значень (ST_c, c), тобто поточну кількість індексів для ключового слова w та відповідний токен пошуку; T – за токеном оновлення UT_i містить відповідний зашифрований індекс e.

2) Пошук

Для пошуку клієнт генерує ключ для ключового слова w за допомогою ПВФ:

$$K_w \leftarrow F_{K_S}(w) \tag{2.44}$$

Далі з відображення W отримуємо токен пошуку та кількість індексів. Якщо за ключовим словом витягнули деяке значення $\perp$, то робимо висновок, що за цим ключовим словом індекси відсутні. Інакше передаємо ці значення з згенерованим ключем до серверу.

Сервер, в свою чергу, перебирає значення від c до 0, для кожного i він розраховує токен оновлення за допомогою геш-функції H_1:

$$UT_i \leftarrow H_1(K_w, ST_i) \tag{2.45}$$

Маючи токен оновлення, тобто логічну позицію, отримуємо зашифрований індекс:

$$e \leftarrow T[UT_i] \tag{2.46}$$

$$ind \leftarrow e \oplus H_2(K_w, ST_i) \tag{2.47}$$

Після цього переходимо до наступного токену пошуку, генеруючи його з перестановки лазівки:

$$ST_{i-1} \leftarrow \pi_{PK}(ST_i) \tag{2.48}$$

Таким самим чином вилучаються всі індекси, що відповідають ключовому слову w, та відправляються клієнту.

3) Оновлення

Оновлення є дещо зворотним процесом від пошуку. Спочатку також генерується ключ для ключового слова, використовуючи формулу (2.44), та витягується пара значень – пошуковий токен та кількість індексів.

Якщо витягнуте значення $\perp$, то за цим ключовим словом індекси відсутні, тому ініціалізується токен пошуку та змінна кількості індексів:

$$ST_0 \xleftarrow{\$} \mathrm{M}; c \leftarrow -1 \tag{2.49}$$

Знак $\$$ позначає, що ST було рівномірно відібрано з M.

Інакше отримується новий пошуковий токен за допомогою інверсії перестановки лазівки та секретного ключа:

$$ST_{c+1} \leftarrow \pi_{SK}^{-1}\left(ST_c\right) \tag{2.50}$$

Після цього оновлюється значення у відображенні W, встановлюється нова пара значень $\left(ST_{c+1}, c+1\right)$.

З токену пошуку отримуємо токен оновлення, використовуючи геш-функцію H_1. Також шифруємо індекс шляхом додавання за модулем 2 індексу та геш-функції H_2 від токену пошуку. Пару токен оновлення/зашифрований індекс відправляємо серверу. Сервер додає шифртекст у відображення T:

$$T\left[UT_{c+1}\right] = e \tag{2.51}$$

Описана схема дозволяє лише додавати документи, проте автори пропонують досить просте рішення для підтримки видалення – реалізувати описану схему двічі: перший екземпляр буде для додавання, другий для видалення, а пошук буде різницею алгоритмів пошуку цих двох екземплярів.

Також треба відмітити, що клієнт повинен зберігати досить багато інформації, що може ускладнити використання методу на обмежених пристроях. З цього приводу пропонується знизити просторову складність за рахунок додаткових обчислень. Для цього пропонується не зберігати токени пошуку, а лише ідентифікатор, з якого буде генеруватись ST_0 за допомогою ПВФ, а з ST_0 можна отримати інші токени. Також можна зробити генерування не від ідентифікатору, а навіть від самого ключового слова w. При цьому, не потрібно обчислювати TDP n_w разів: необхідний по номеру токен можна обчислити прямо з ST_0. Більш детальний опис можна знайти в оригінальній роботі [25].

2.5 Метод Fides

У цьому підрозділі розглядається підхід та схема на його основі, що були запропоновані у роботі [10]. Відмінність цієї схеми від розглянутих вище полягає у тому, що вона відповідає прямій секретності, та в той же час підтримується зворотна секретність.

Підхід описує, як з звичайної SSE схеми Σ отримати схему, що відповідає поняттю зворотної секретності. Ця схема є деякою надбудовою над іншими схемами, вона пропонує зберігання не індексів, а комбінації індексу та операції, після чого ця комбінація шифрується ключем відповідного ключового слова w:

$$E_{K_w}(ind, op) \tag{2.52}$$

В іншому, базова схема Σ працює як зазвичай. Проте є декілька важливих моментів щодо зазначеної схеми: при пошуку сервер не зможе ідентифікувати необхідні записи за ключовим словом, оскільки він не бачить індексів (тому що вони зашифровані). З цього приводу вводиться ще один раунд протоколу: після того, як клієнт отримає (2.52), він розшифровує цю структуру та отримує індекси, які відсилає серверу, після чого вже проводиться безпосередній витяг документів.

Другою особливістю є те, що документи не видаляються з структури даних на серверній стороні, більш того, вони будуть відправлятися клієнту в результаті

першого раунду пошуку. Тут рішення досить просте: отримуючи відповідь, клієнт прибирає всі індекси, що були видалені, та відправляє решту серверу у відкритому вигляді, але, поряд з цим, він відправляє ці ж самі індекси, зашифровані новим ключем. Таким чином, сервер зможе видалити всі старі індекси, а на заміну додасть нові, утримуючу базу даних в актуальному стані.

Застосовуючи описані вище операції до різних SSE схем, можна отримати різні екземпляри BP-надійної схеми. Fides базується на схемі Σοφος.

2.6 Метод Diana

2.6.1 Фреймворк FS-RCPRF

В цьому фреймворку, визначеному у роботі [10], використовується обмежена псевдовипадкова функція (ОПВФ), яку було паралельно розроблено та представлено у роботах [26] та [27]. Ця ПВФ особлива тим, що асоціюється з сімейством логічних схем $\mathcal{C}$. Для кожної з схем цього сімейства $C \in \mathcal{C}$ мається обмежений ключ K_C, який можна обчислити за допомогою майстер-ключа ПВФ. Цей обмежений ключ дозволяє проводити обчислення тільки за такими значеннями x, для яких $C(x) = 1$.

ОПВФ має два алгоритми: перший – *Constrain*, який приймає ключ K та схему C, а повертає обмежений ключ K_C. Другий алгоритм – *Eval*, приймає обмежений ключ K_C та значення x, повертає деяке значення $y \in Y$.

Розглянемо фреймворк FS-RCPRF (forward secure scheme based on the range-constrained pseudorandom function), тобто заснований на ПВФ з обмеженим діапазоном, відповідаючий поняттю прямої секретності. За допомогою описаного нижче алгоритму та деякої ОПВФ можна створити SSE-схему, що буде відповідати поняттю прямої секретності. А ідея полягає у тому, щоб генерувати токени оновлення для ключового слова w за допомогою ОПВФ у режимі лічильника, який збільшується після кожного додавання ідентифікатора, що містить задане ключове слово. Для пошуку клієнту потрібно відправити ключ для ОПВФ, який дозволяє проводити обчислення значень від 0 до $n_w - 1$.

1) Налаштування

Генерується загальний секретний ключ, з якого будуть генеруватися специфічні ключі для ключових слів, – K_Σ. Ініціалізується відображення W, в якому за ключовим словом w зберігається поточна кількість індексів документів c, що містять ключове слово w. Також ініціалізується пусте відображення EDB.

2) Пошук

Генерується новий ключ для ключового слова w та витягується кількість індексів документів, що містять це ключове слово, – c:

$$K_w \| K_w^{'} \leftarrow F_{K_\Sigma}(w); c \leftarrow W[w] \tag{2.53}$$

Якщо c дорівнює $\perp$, то повертаємо пусту множину (індексів немає). Інакше обчислюємо токен пошуку за допомогою ОПВФ:

$$ST \leftarrow \tilde{F}.Constrain(K_w, C_c) \tag{2.54}$$

У формулі (2.53) визначена деяка схема C_c, обчислення цієї схеми буде 1, якщо на вхід подається $\{0,\ldots,c\}$. Після цього пошуковий токен, ключ та кількість індексів відправляються серверу.

Сервер перебирає $i \in (c;0)$, для кожного з них він обчислює ОПВФ за допомогою токену пошуку:

$$T_i \leftarrow \tilde{F}.Eval(ST, i) \tag{2.55}$$

За обчисленим значенням знаходиться токен оновлення, що за сумісництвом є логічною позицією, в якій зберігається зашифрований індекс:

$$UT_i \leftarrow H_1(K_w^{'}, T_i) \tag{2.56}$$

Далі, як і у Σοφος, витягується зашифрований індекс, після чого розшифровується шляхом додавання за модулем 2:

$$ind \leftarrow e \oplus H_2\left(K_w^{'}, T_i\right) \tag{2.57}$$

Таким ж чином знаходимо всі інші ідентифікатори.

3) Оновлення

Розраховується ключ та витягується кількість індексів за формулою (2.53). Якщо $c = \perp$, то ініціалізуємо його нулем.

Після цього потрібно додати необхідні значення в відображення T та W:

$$T_w^{c+1} \leftarrow \tilde{F}.Eval\left(K_w, c+1\right); W\left[w\right] \leftarrow c+1 \tag{2.58}$$

Також розраховується токен оновлення та шифрується індекс:

$$UT_{c+1} \leftarrow H_1\left(K_w^{'}, T_w^{c+1}\right); e \leftarrow ind \oplus H_2\left(K_w^{'}, T_w^{c+1}\right) \tag{2.59}$$

Токен оновлення та зашифрований індекс відправляються серверу, він додає у EDB зашифрований індекс.

2.6.2 Обмежена псевдовипадкова функція BRC

Для реалізації схеми Diana використовується ОПВФ, яка будується з псевдовипадкової функції GGM, що базується на бінарному дереві [28]. Ця ОПВФ була побудована у [27] та отримала назву найкраще покриття діапазону або best range cover (BRC).

Нехай мається деякий генератор псевдовипадкових чисел $G: \{0,1\}^{\lambda} \to \{0,1\}^{2\lambda}$, тоді позначимо результат від деякого випадкового початкового значення k як $G(k)$, а першу та другу його частину як $G_0(k)$ та $G_1(k)$ відповідно. Тоді ПВФ GGM від деякого цілого числа x довжиною n біт буде визначатись як:

$$F_K\left(x\right) = G_{x_0}\left(\ldots\left(G_{x_n-1}\left(K\right)\right)\right) \tag{2.60}$$

Таким чином, листя дерева у GGM – це результат функції F, вона може бути зіставлена з вихідним значенням x, а також частковим обчисленням цієї

функції (тобто з $l < n$ біт числа x). Наведемо простий приклад дерева ПВФ з [28]:

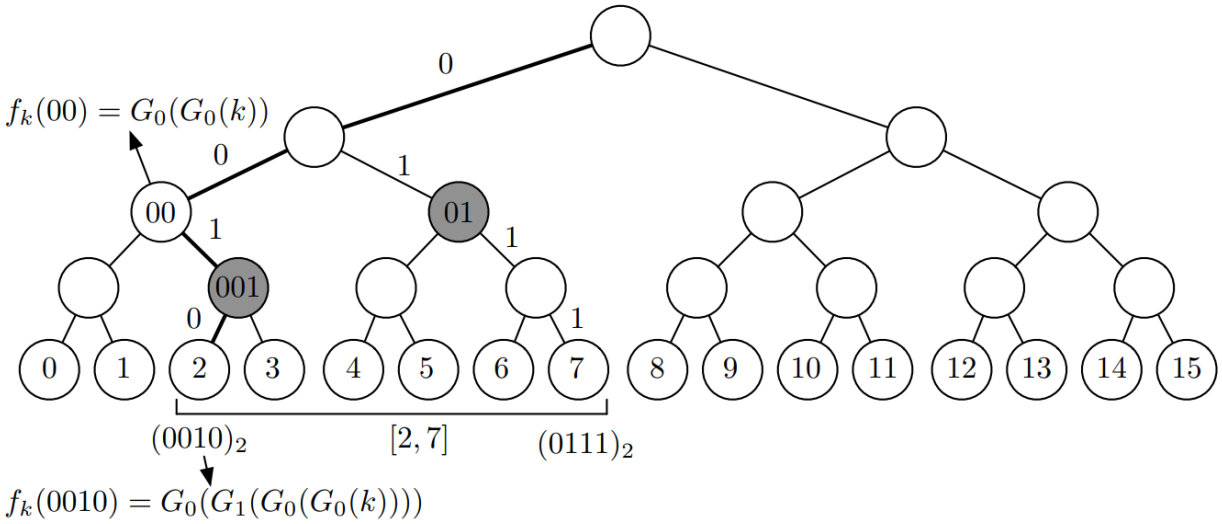

Рисунок 2.6 – Приклад деревовидної ПВФ GGM

З рисунку 2.6 видно, як отримуються значення у листях та взагалі у вузлах: записуються значення від кореня до вузла. Якщо потрібно пройти до листа з позначкою 3, то обчислюємо $G_1\left(G_1\left(G_0\left(G_0\left(k\right)\right)\right)\right)$. Також можемо проводити асоціювання з частковими значеннями ПВФ: $G_1\left(G_1\left(f_k\left(00\right)\right)\right)$ або $G_1\left(f_k\left(001\right)\right)$. При цьому для деякого діапазону, наприклад, [2-7], як на рисунку 2.6, існує множина таких часткових значень, які являють собою піддерева. Для обраного діапазону це $f_k\left(001\right)$ з глибиною 1 та $f_k\left(01\right)$ з глибиною 2 (глибина до листя). Задача BRC як раз і полягає в тому, щоб знайти мінімальну кількість таких часткових значень.

У методі Diana треба обмежити ПВФ на проміжку $\left[0,c\right]$, для цього будуть генеруватися такі вузли, що покривають цей діапазон, причому не містять шляху до листя, що в заданий діапазон не входять. В іншому, все йде згідно алгоритму FS-RCPRF.

2.6.3 Модифікація $\text{Diana}_{\text{del}}$ з можливістю видалення

В описаному вище методі можливе лише додавання. Уможливити видалення можна подібно до Σοφος, тобто створити два екземпляри SE-схеми з підтримкою прямої секретності, одна з яких буде для додавання, а інша – для

видалення. Результат отримується як різниця результатів двох схем, але, якщо ці операції буде проводити сервер, то він буде отримувати інформацію про видалені записи. Пропонується інший підхід: при додаванні додаються записи виду (w,ind) до першого екземпляру. Поряд з цим передається пара значень $\left(F'\left(K_w,(w,ind)\right),Enc_{K'}(c)\right)$, де F' це деяка ПВФ, а Enc – деяка CPA-надійна схема шифрування. Сервер приймає цю пару та заносить до спеціального відображення. При видаленні додається до другого екземпляру пара значень $\left(w,F'\left(K_w,(w,ind)\right)\right)$.

При пошуку даних спочатку робиться пошук у другому екземплярі, сервер витягує відповідні ключовому слову теги $F'\left(K_w,(w,ind)\right)$, що відповідають видаленим записам. Далі він застосовує ці теги для витягнення $Enc_{K'}(c)$ з відображення, та відправляє ці зашифровані ідентифікатори клієнту. Клієнт розшифровує ідентифікатори та дізнається про те, які записи були видалені. Маючи цю інформацію, клієнт з початкового діапазону $[0,c]$ генерує декілька діапазонів, які не включають видалені записи. Якщо маємо видалені ідентифікатори $x_1,\ldots,x_n$, то діапазони мають вигляд $[0,x_1-1],[x_1+1,x_2-1],\ldots,[x_n+1,c]$.

Варто зауважити, що цей метод припускає, що один і той же індекс не додається двічі. Якщо його було видалено, то його не можна додати знов.

3 АНАЛІЗ МЕТОДІВ ШИФРУВАННЯ З МОЖЛИВІСТЮ ПОШУКУ

Проведемо аналіз схем, розглянутих у розділі 2, визначивши часову, комунікаційну та просторову складності, а також виявивши рівень безпеки схеми.

3.1 Аналіз методу паралельного та динамічного симетричного шифрування з можливістю пошуку

Спочатку розглянемо часову складність цього методу. Метод використовує ЧЧД, висота якого не перевищує $\log n$, отже, час пошуку є логарифмічним по кількості документів n. При цьому потрібно зробити $\left|DB(w)\right|$ таких пошуків, але кожен з них може проводитись окремо, тобто й паралельно. Тоді, якщо використовується розпаралелювання з p процесорами, то складність пошуку w визначається як $O\left(\frac{\left|DB(w)\right|}{p}\log n\right)$, а якщо послідовно, то $O\left(\left|DB(w)\right|\log n\right)$. А якщо є достатня кількість процесорів, а саме $\log n$, то складність зводиться до $O\left(\left|DB(w)\right|\right)$.

Вставка та видалення у ЧЧД мають часову складність $O(\log n)$. Якщо оновлюється деякий документ d, що містить q ключових слів, то таке оновлення буде вимагати q незалежних операцій у векторах, що зберігають інформацію про ключові слова у нащадках. Значить послідовна складність оновлення досягає $O(q\log n)$, а при розпаралелюванні з використанням p процесорів – $O\left(\frac{q}{p}\log n\right)$. Але при такому оновленні сервер зможе отримати інформацію щодо змінених позицій у векторах, тоді зможе зрозуміти до яких ключових слів ці зміни

відносяться. Тому в алгоритмі зачіпаються всі біти, отже складність становить $O(m\log n)$ та $O(\frac{m}{p}\log n)$ відповідно для послідовного та паралельного варіантів.

Комунікаційна складність пошуку залежить від токену, який має постійну величину, а також від кількості документів, що містять ключове слово – $O\left(\left|DB(w)\right|\right)$. Оновлення ж потребує передачі множини ключових слів, за якими оновлюється документ, а також передачі самого ідентифікатору: $O\left(\left|W_{id}\right| + m * \log n\right)$.

Не менш важливим чинником успішності методу є його просторова складність. У нашому випадку для індексу, тобто ЧЧД, у кожному вузлі генерується m-бітовий вектор, отже, витрачається $O(m * n)$. А якщо йде мова про великі бази даних, де відповідно мається велика кількість ключових слів, то реальні розміри індексу будуть надто великі.

Якщо розглядати метод з точки зору безпеки, то схема є CKA2-надійною, тобто її безпеку доведено у моделі IND2-CKA, але доведення безпеки виконано у моделі RO. Ґрунтується її безпека на шифруванні, а значить й на ключах, що використовуються для цього шифрування. Для їх генерації використовувались ПВФ та випадковий оракул, отже, вони можуть вважатись безпечними.

Проте токени, що генеруються у методі, є детерміністичними, тобто для певних ключових слів токени завжди будуть однаковими, отже, з них витікає шаблон пошуку. Також сервер розуміє, що ключовому слову, документи з яким треба знайти, відповідає певна сукупність шифртекстів. Крім цього, витікає кількість ключових слів, кількість документів, ідентифікатори документу та розмір кожного документу.

3.2 Аналіз методу Dyn2Lev

Як і у попередньому підрозділі, почнемо з часової складності методу. Для пошуку серверу передаються ключі, завдяки яким він розраховує позначку та отримує зашифровані значення ідентифікаторів, що відповідають шуканому ключовому слову. Отже швидкість напряму залежить від кількості слів, що

містять ключове слово w, при цьому в динамічній версії є множина доданих записів, яку теж треба враховувати, нехай її потужність буде a_w. Тоді складність пошуку можна визначити як $O\left(\left|DB\left(w\right)\right|+a_w\right)$. Але це складність при послідовному виконанні, алгоритм можна розбити на окремі пошукові ітерації, тоді час виконання можна скоротити до $O\left(\frac{\left|DB\left(w\right)\right|+a_w}{p}\right)$. При цьому треба відмітити, що, при пошуку у динамічній схемі, додатково перевіряється, чи наявний ідентифікатор у списку відкликання, і якщо він наявний, то його відправка скасується. Отже, якщо буде дуже багато видалених записів, то ефективність методу значно знижується, тому автори зазначають, що схема призначена для БД, в яких дуже рідко відбувається видалення, а також рекомендується періодично робити перешифрування всієї БД, яке також підвищує стійкість схеми проти витоку.

Стосовно часу оновлення, при додаванні, як і при видаленні, потрібно відправити лише ідентифікатор запису та множину ключових слів. І, після цього, для кожного ключового слова буде розраховуватись ПВФ, потім буде робитись додавання до словнику D^+ або множини S_{rev}. Отже, часову складність алгоритму оновлення можна оцінити як $O\left(W_{id}\right)$, тобто вона залежить від потужності множини ключових слів, що передається серверу. Але якщо говорити про оновлення певного запису з певним ключовим словом, то воно займає постійний час – $O\left(1\right)$.

Комунікаційна складність пошуку залежить від кількості документів, що містять ключове слово, тобто складність становить $O\left(\left|DB\left(w\right)\right|\right)$. Оновлення приймає запит, що складається з множини ключових слів, та одного ідентифікатору файлу, отже, становить $O\left(\left|W_{id}\right|+m\log n\right)$.

Просторова складність індексу залежить від кількості записів у БД, тобто визначається як $O\left(n\right)$. Але треба відзначити, що додатково зберігається надлишкова інформація, така як покажчики, що теж впливає на займаний

простір. При цьому для алгоритмів існують допоміжні структури даних, що зберігають значення лічильників, додані або вилучені записи.

Безпека схеми доведена у неадаптивній моделі IND1-CKA, а також в адаптивній моделі IND2-CKA, але з використанням ROM. При цьому з схеми витікає інформація про розмір множин та їх перетин. Також витікають шаблони пошуку та оновлення, оскільки використовується детерміністичне шифрування. Отже, при оновленні сервер може встановити, чи додавалися або видалялися певні ключові слова до певного ідентифікатора, а також те, чи робився пошук за цими ключовими словами за певним ідентифікатором. Це можна встановити завдяки ідентифікатору відкликання, що генерується при додаванні або видаленні.

Також є практичні результати застосування методу [21]: експеримент проводився на машині з двома Intel Xeon 2.4GHz E5620, 96 ГБ RAM, шости 1TB SAS дисками, налаштованими згідно RAID-0. З бази даних з кількістю пар ключове слово/ідентифікатор 114,5 мільйонів було отримано ЗБД розміром 15 ГБ, при кількості пар 1145 мільйонів – 52 ГБ. Також проводився пошук у БД з мільярдом пар ключове слово/ідентифікатор, при пошуку ключового слова, що має 10 відповідних йому записів, витратилося близько 120 мс, а на пошук ключового слова, що має 10 тисяч відповідних записів, було витрачено близько 170 мс.

3.3 Аналіз логічного симетричного шифрування з можливістю пошуку

Часова складність алгоритму пошуку методу IEX досягає:

$$O\left(q^2 * M\right), \tag{3.1}$$

де q – це кількість ключових слів у запиті,

$M = \max\limits_{i \in [q]} \left| DB\left(w_i\right) \right|$, тобто це потужність максимальної множини документів для деякого ключового слова w_i.

Така складність отримується, оскільки для кожного ключового слова отримується список тегів, і при цьому для кожного ключового слова, що йде у

запиті після поточного, визначається набір тегів тих документів, що містять відразу 2 ключових слова.

Складність логічного варіанту BIEX, що підтримує довільні логічні запити, дещо відрізняється та дорівнює:

$$O\left(q^2 * \left(\max_{w \in \Delta_1} \left| DB(w) \right| + l * \left| DB(\Delta_1) \right| \right) \right), \tag{3.2}$$

де Δ_1 позначає собою першу кон'юнкцію, тобто перший вираз, що складається з диз'юнкцій.

Причому перший доданок позначає час пошуку тегів першої диз'юнкції (тобто це застосування формули (3.1) для першої диз'юнкції), а другий доданок позначає загальну кількість локальних мультивідображень.

Динамічний варіант також залежить від використовуваних у ньому реалізацій словника та мультивідображення. Якщо вони є оптимальними, то час пошуку дорівнює часу пошуку IEX, тобто (3.1), або (3.2), якщо підтримує логічні запити. Час оновлення ґрунтується на SSE схемі, що використовується як мультивідображення.

Комунікаційна складність пошуку залежить від токену: у IEX це $O(q)$, у BIEX це $O(lq)$. А також передаються всі теги, що задовольняють запиту – це $O\left(\left|DB(query)\right|\right)$, де $query$ – це запит клієнта, для IEX – $w_1 \vee \ldots \vee w_q$, для BIEX – це $\Delta_1 \wedge \ldots \wedge \Delta_l$, де $\Delta_i = w_1 \vee \ldots \vee w_{q_i}$. Тоді комунікаційна складність для IEX та BIEX дорівнює відповідно $O\left(\left|DB(query)\right| + q\right)$ та $O\left(\left|DB(query)\right| + lq\right)$.

Просторова складність IEX (як і BIEX) залежить від реалізації мультивідображення (тобто схеми SSE) та при використанні оптимальних сублінійних схем у цій якості визначається як:

$$O\left(\sum_w \left(\left| DB(w) \right| + \sum_{v \in co(w)} \left| DB(w) \bigcap DB(v) \right| \right) \right) \tag{3.3}$$

Проте її можна значно знизити при використанні більш компактних схем, як-от ZMF, що представлена у роботі. В цьому методі створюється лише один бітовий масив – фільтр-матрьошка, що заснований на фільтрах Блума, довжина його відповідає потужності найбільшої множини. Якщо ця схема використовується у поєднанні з IEX, то множини будуть представляти собою перетини $DB(w)$ та $DB(v)$, тому просторова складність буде визначатися як:

$$O\left(\sum_w\left(\left|DB(w)\right|+\max_{v\in co(w)}\left|DB(w)\bigcap DB(v)\right|\right)\right) \tag{3.4}$$

Тут страждає часова складність, оскільки час пошуку буде лінійним за кількістю фільтрів-матрьошок $O(\lambda)$, але це компенсується невеликою потужністю ЛМВ.

Безпека схеми IEX та її модифікацій здебільшого ґрунтується на схемі, що використовується як МВ. Відповідна річ йде про пряму та зворотну секретність: якщо підлегла схема задовольняє вимогам, то й IEX теж. Але можна визначити, що точно буде витікати, спираючись на базові схеми SSE, що можуть бути використані як МВ, наприклад, схеми описані в цій роботі. Отже, витікає наступна інформація: розмір БД; загальний обсяг всіх ЛМВ у словнику; при запитах витікають шаблони пошуку та доступу; витікає про перетини $DB(w_i)$ та $DB(w_j)$, де $j>i$.

Для логічного варіанту схеми наявні ті ж самі недоліки, вони відносяться до кожного диз'юнктивного виразу запиту. Так само й для динамічної модифікації, але ще додається витік з операцій оновлення, який залежить від підлеглої схеми мультивідображення.

Для отримання практичних результатів проводились порівняння схем IEX-ZMF та IEX-2Lev на базі Intel Xeon E5-2680 v2 (Ivy Bridge), 60 ГБ RAM [22]. В результаті було отримано, що IEX-ZMF працює значно повільніше, ніж IEX-2Lev: для одних й тих самих запитів час пошуку повільніший у 30 разів для кон'юнктивних запитів та в 10-20 разів для диз'юнктивних запитів. Якщо

говорити про логічні запити, то для IEX-ZMF вони повільніші в середньому у 50 разів.

Також час налаштування системи значно повільніший у ZMF: якщо припустити, що БД має мільярд пар ключове слово/запис, то для налаштування IEX-ZMF потребує 217 годин, коли IEX-2Lev впорається за 22.

Токени, що генеруються для IEX-ZMF, також більше в 300 разів токенів для IEX-2Lev, що напряму впливає на швидкість комунікації між сервером так клієнтом.

Але все це компенсується компактністю ZMF: для БД з кількістю пар ключове слово/запис в 34 мільйони, розмір якої приблизно 34 МБ, IEX-ZMF створює структуру, що займає 0.9 ГБ, в той час як конструкція IEX-2Lev займає 9.8 ГБ.

3.4 Аналіз методу Σοφος

Почнемо з часової складності: вона є оптимальною та визначається тим, скільки разів було додано ключове слово у парі з деяким ідентифікатором – $O\left(\left|DB\left(w\right)\right|\right)$. При цьому треба зауважити, що це включає обидва екземпляри схеми: для додавань та для видалень, оскільки для отримання коректного результату потрібно провести пошук у обох з них. Проте на практиці схема працює відносно повільно через використання криптопримітивів RSA (Rivest, Shamir, Adleman).

Для оновлення треба обчислити новий токен пошуку, з якого виводиться новий токен оновлення, а також зашифрувати індекс. Це завжди займає однаковий час, отже, час оновлення – $O\left(1\right)$.

Комунікаційна складність пошуку становить $O\left(\left|DB\left(w\right)\right|\right)$, оскільки сервер передає ідентифікатори лише тих записів, що не було видалено. Оновлення має постійну комунікаційну складність $O\left(1\right)$, тому що завжди передаються зашифрований індекс та токен оновлення, що мають постійний розмір.

Просторова складність для клієнта з урахуванням модифікацій, що покращують розмір за рахунок додаткових обчислень, складає $O\left(m*\log n\right)$, тому

що для кожного ключового слова зберігається лічильник, який може мати максимальне значення n.

Сервер зберігає всі записи, що були додані у БД, при цьому видалені записи дублюються (запис присутній у двох екземплярах схеми). Тому складність дорівнює $O\left(\sum_w \left(\left|DB(w)\right| + d_w\right)\right)$.

Схема є адаптивно стійкою, проте доказ виконано у моделі RO. Також це перша з розглянутих схем, що досягає прямої секретності. З схеми витікає шаблон пошуку, а також вся історія взаємодій з ключовим словом w, а саме всі модифікації $DB(w)$, витікає операція, яку було проведено (додавання/видалення), час, коли це було зроблено, а також ідентифікатор. А знаючи ідентифікатор, можна зрозуміти коли це слово було додано, коли видалено.

Розглянемо реальну продуктивність методу, експеримент було проведено на базі Intel Core i7 4790K 4.00GHz CPU, 16 ГБ RAM, 250 GB Samsung 850 EVO SSD [25]. У оцінці не зазначається кількість пар, а лише окрема кількість ключових слів та документів, тому оберемо варіант приблизно схожий на експеримент для методу Dyn2Lev: для 1.4 мільйонів документів та 23 тисячі ключових слів. Тоді клієнт витрачає 572 КБ на зберігання лічильників, а сама ЗБД важить 64 МБ. Для БД цього розміру пошук для ключового слова з 10 відповідними записами займає лише 0.2 мс, пошук для ключового слова з 10 тисячами записів – приблизно 70 мс. Якщо обрати БД з 14 мільйонами документів та 200 тисячами ключових слів, то отримаємо 5 МБ відображення ключових слів та лічильників, а сама ЗБД буде важити приблизно 500 МБ. При цьому пошук для ключових слів з 10 та 10 тисячами записів буде займати 0.2 мс та 70 мс відповідно, як і для меншої БД.

3.5 Аналіз методу Fides

Fides заснований на Σοφος, тому вони дуже схожі по показникам ефективності. Часова складність пошуку також дорівнює $O\left(\left|DB(w)\right|\right)$, але на пошук витрачається два раунди комунікації між сервером та клієнтом. Хоча

здається, що схема працює дуже швидко, але на практиці це не так через використання TDP RSA.

Оновлення займає постійний час – $O(1)$, як і у Σοφος.

Комунікаційна складність пошуку дещо відрізняється від Σοφος, оскільки передаються індекси видалених записів, після чого вже клієнт відправляє решту індексів, які містять ключове слово та не були видалені. Таким чином, комунікаційна складність становить $O\left(\left|DB(w)\right| + d_w\right)$.

Комунікаційна складність оновлення відповідає Σοφος – $O(1)$.

Просторова складність як і у Σοφος – $O(m * \log n)$ для клієнта.

Сервер зберігає замість індексів зашифровані комбінації індекс/операція, а їх кількість завжди підтримується актуальною, тому що видалені елементи прибираються з БД, тому просторова складність $O\left(\sum_w \left(\left|DB(w)\right|\right)\right)$.

Якщо говорити про безпеку, то вона як і у Σοφος, але витікає не вся інформація про модифікації, а лише кількість оновлень з ключовим словом та час, коли ці модифікації було виконано. Таким чином, схема має II рівень зворотної секретності.

3.6 Аналіз методу $\text{Diana}_{\text{del}}$

Часова складність пошуку є оптимальною та становить $O\left(\left|DB(w)\right|\right)$, оскільки при пошуку потрібно зробити звернення до псевдовипадкового генератора для кожного з ідентифікаторів від 0 до c, щоб розрахувати всі листя (лазівки) для заданого діапазону.

Для оновлення також треба розрахувати листя, але в цьому разі від кореня для нового значення $c+1$. Отже, складність цього дорівнює $O\left(\log\left|DB(w)\right|\right)$, оскільки обчислюється ПВФ для кожного з елементів у шляху до $c+1$, а глибина дерева є логарифмічною.

Комунікаційна складність пошуку становить $O\left(\left|DB(w)\right| + d_w \log\left|DB(w)\right|\right)$, оскільки передаються не тільки необхідні індекси за ключовим словом, але й

лазівки, що отримуються від ОПВФ, для всіх видалених записів, кожна з лазівок має просторову складність $O\left(\log\left|DB(w)\right|\right)$. Алгоритм оновлення має постійну комунікаційну складність $O(1)$.

Просторова складність становить $O(m*\log n)$ на стороні клієнта, оскільки він зберігає лічильники для кожного ключового слова, при цьому значення лічильника не може перевищувати кількість документів.

Сервер зберігає індекси для ключових слів, а також видалені записи, отже складність дорівнює $O\left(\sum_w\left(\left|DB(w)\right|+d_w\right)\right)$.

Безпека методу відповідає поняттю IND2-CKA, доведення виконано у ROM. Також схема відповідає поняттю прямої та зворотної секретності, при цьому рівень BP – III. Це означає, що схема розкриває документи, що містять ключове слово w, час, коли вони були додані, а також час всіх оновлень, що стосувалися цих документів. Більш того, третій рівень зворотної секретності розкриває також яке саме видалення скасувало певне попереднє додавання ключового слова. Також розкривається й шаблон пошуку.

Розглянемо реальну продуктивність методу, експеримент було проведено на базі Intel Core i7 4790K 4.00GHz CPU, 16 ГБ RAM, 250 GB Samsung 850 EVO SSD [10]. Для БД з 19 мільйонами документів для ключових слів з відповідною кількістю записів 10 та 10 тисяч, час пошуку дорівнює 0.15 мс та 2 мс відповідно.

3.7 Порівняння розглянутих методів

Для зручності сприйняття проведеного аналізу зведемо всі дані про методи до єдиної таблиці 3.1 за показниками часової (обчислювальної), комунікаційної та просторової складностей, а загальну характеристику схем зведемо до таблиці 3.2.

Позначки у таблиці мають наступні значення:

- m – загальна кількість ключових слів;
- n – загальна кількість записів/документів;
- p –кількість процесорів;

- $DB(w)$ – кількість записів, що містять ключове слово w;
- d_w – кількість видалених записів, що містять ключове слово w;
- q – це кількість ключових слів у логічному запиті;
- $M = \max_{i \in [q]} \left| DB(w_i) \right|$ – це потужність максимальної множини документів для деякого ключового слова w_i;
- l – кількість диз'юнкцій у запиті;
- W_{id} – множина ключових слів, що передається при оновленні у методах PDSSE та Dyn2Lev;
- $query$ – логічний запит;
- $co(w)$ – множина ключових слів, що наявні у документах разом з w;
- $ex_1 = \max_{v \in co(w)} \left| DB(w) \bigcap DB(v) \right|$ – потужність найбільшої множини у локальному МВ;
- $ex_2 = \sum_{v \in co(w)} \left| DB(w) \bigcap DB(v) \right|$ – загальна кількість локальних МВ.

Таблиця 3.1 – Порівняння розглянутих методів шифрування з можливістю пошуку за складністю

Назва	*Часова складність*		*Комунікаційна складність*		*Просторова складність*	
	Пошук	*Оновлення*	*Пошук*	*Оновлення*	*Клієнт*	*Сервер*
PD SSE	$O\left(\frac{\lvert DB(w)\rvert}{p}\log n\right)$	$O(\frac{m}{p}\log n)$	$O\left(\lvert DB(w)\rvert\right)$	$O\left(\lvert W_{id}\rvert + m\log n\right)$	$O(1)$	$O(m*n)$
Dyn 2Lev	$O\left(\frac{\lvert DB(w)\rvert + d_w}{p}\right)$	$O(1)$	$O\left(\lvert DB(w)\rvert\right)$	$O\left(\lvert W_{id}\rvert + m\log n\right)$	$O(1)$	$O(n)$
BIEX -ZMF	$O\left(q^2\left(M + l\lvert DB(\Delta_1)\rvert\right)\right)$	–	$O\left(\lvert DB(query)\rvert + lq\right)$	–	$O(1)$	$O(\sum_w(\lvert DB(w)\rvert + ex_1))$
BIEX -2Lev	$O\left(q^2\left(M + l\lvert DB(\Delta_1)\rvert\right)\right)$	–	$O\left(\lvert DB(query)\rvert + lq\right)$	–	$O(1)$	$O(\sum_w(\lvert DB(w)\rvert + ex_2))$
Σοφος	$O\left(\lvert DB(w)\rvert\right)$	$O(1)$	$O\left(\lvert DB(w)\rvert\right)$	$O(1)$	$O(m*\log n)$	$O\left(\sum_w\left(\lvert DB(w)\rvert + d_w\right)\right)$
Fides	$O\left(\lvert DB(w)\rvert\right)$	$O(1)$	$O\left(\lvert DB(w)\rvert + d_w\right)$	$O(1)$	$O(m*\log n)$	$O\left(\sum_w\left(\lvert DB(w)\rvert\right)\right)$
Diana	$O\left(\lvert DB(w)\rvert\right)$	$O\left(\log\lvert DB(w)\rvert\right)$	$O\left(\lvert DB(w)\rvert + d_w\log\lvert DB(w)\rvert\right)$	$O(1)$	$O(m*\log n)$	$O\left(\sum_w\left(\lvert DB(w)\rvert + d_w\right)\right)$

Таблиця 3.2 – Порівняння розглянутих методів шифрування з можливістю пошуку

Назва	*Базис*	*Криптопримітиви*	*Безпека*	*FP*	*BP*	*Витік*
PDSSE	Червоно-чорне дерево	ПВФ, геш-таблиця, геш-функція, CPA-надійна схема шифрування	IND2-CKA^{RO}	–	–	Шаблон пошуку, шаблон доступу, кількість ключових слів, розмір БД, ідентифікатори документу, розмір документів
Dyn2Lev	Словник, кортеж	ПВФ, CPA-надійна схема шифрування	IND2-CKA^{RO}	–	–	Шаблон пошуку, шаблон доступу, шаблон оновлення, розмір БД, розмір зовнішнього масиву, розмір блоків
BIEX-ZMF	Базується на ZMF, фільтр-матрьошка, мультивідображення, словник	ПВФ, ПВП, фільтр Блума, онлайн-шифр, CPA-надійна схема шифрування	IND2-CKA^{RO}	–	–	Шаблон пошуку, шаблон доступу, розмір БД, загальний обсяг всіх ЛМВ у словнику, перетини множин
BIEX-2Lev	Базується на 2Lev, мультивідображення, словник, кортеж	ПВФ, ПВП, CPA-надійна схема шифрування	IND2-CKA^{RO}	–	–	Шаблон пошуку, шаблон доступу, розмір БД, загальний обсяг всіх ЛМВ у словнику, перетини множин + виток з 2Lev
Σοφος	Відображення	ПВФ, RSA TDP, геш-функція, CPA-надійна схема шифрування	IND2-CKA^{RO}	+	–	Шаблон пошуку, вся історія взаємодій з ключовим словом: тип операції, час та ідентифікатор документу
Fides	Базується на Σοφος, відображення	ПВФ, RSA TDP, геш-функція, CPA-надійна схема шифрування	IND2-CKA^{RO}	+	II	Шаблон пошуку, кількість оновлень за ключовим словом та час, коли вони були виконані
Diana$_{del}$	Відображення	ПВФ, ОПВФ BRC, геш-функція, CPA-надійна схема шифрування	IND2-CKA^{RO}	+	III	Шаблон пошуку, кількість оновлень, їх час, ідентифікатор документу, історія оновлень цього документу, яке саме видалення скасувало додавання запису

4 ПРАКТИЧНІ РЕКОМЕНДАЦІЇ

Базуючись на проведеному у 3 розділі аналізі, виробимо практичні рекомендації щодо застосування кожного з розглянутих методів, а також щодо вибору певного методу в залежності від умов проекту [29].

На підставі проведеного аналізу методу PDSSE була визначена його відносно висока складність серед більш сучасних методів ШМП, яка підвищує складність пошуку для великих баз даних, навіть при незмінної кількості записів, що містять шукане ключове слово. Проте цей недолік дещо компенсується можливістю паралельного обчислення. Другим та більш серйозним недоліком є його просторова складність: хоча клієнт зберігає тільки секретні ключи, сервер на кожний елемент дерева зберігає m-бітний вектор, тобто по біту на кожне ключове слово. Навіть середні за розміром бази даних можуть мати досить багато ключових слів, через що PDSSE займе дуже багато місця на диску. Цих двох недоліків достатньо, щоб зробити висновок про неефективність практичного застосування методу.

Метод Dyn2Lev, завдяки використанню словника, здатен проводити операції оновлення за постійний час. Складність пошуку та займаний простір теж мають досить ефективні значення. Серед недоліків схеми можна відмітити зростаючу кількість видалених записів, які не видаляються насправді, з чого випливає необхідність в періодичному перешифруванні всієї БД. При цьому, метод використовує досить багато покажчиків, тому для отримання ідентифікаторів також витрачається зайвий час, тому реальна швидкість методу хоча й краща за PDSSE та є непоганою на практиці, все ще поступається більш сучасним методам. Тому і його використання на практиці на даний момент більш не є доцільним.

Схема IEX, як і її логічна та динамічна модифікації, відрізняється від всіх інших розглянутих схем, оскільки являє собою надбудову над іншими схемами для досягнення більшої виразності запитів. Її ефективність та безпека

здебільшого базуються на підлеглій схемі, що використовується у якості мультивідображення, тобто нічого не заважає реалізувати вказані схеми над якоюсь більш ефективнішою та безпечнішою схемою. Серед запропонованих авторами схем BIEX-ZMF та BIEX-2Lev вибір йде або убік компактності отриманої ЗБД (ZMF), або убік швидкості роботи та комунікаційної складності (2Lev). Отже, рішення про використання IEX залежить від необхідності виконувати логічні запити до БД у конкретному проекті, зважаючи на збільшення часу пошуку, комунікаційної та просторової складності.

Метод Σοφος, порівняно з раніше розглянутими методами, характеризується кращою часовою та комунікаційною складністю. При цьому серед недоліків можна виділити необхідність клієнта зберігати значення лічильників для кожного ключового слова. При цьому, незважаючи на часову складність, реальний час пошуку схеми страждає через використання перестановки лазівки RSA, але він все ще значно кращий за той же Dyn2Lev.

При цьому, за допомогою фреймворка FS-RCPRF на базі Σοφος було визначено метод Fides, який відповідає не тільки поняттю прямої секретності, але й зворотної другого рівня, при цьому має майже однакові показники складності (комунікація має 2 раунди замість одного). Тому використання Σοφος не є доцільним, оскільки Fides має кращі показники безпеки.

Модифікація схеми Diana, що дозволяє проводити видалення, базується повністю на симетричних криптопримітивах, що у поєднанні з оптимальною часовою складністю робить її найшвидшою з розглянутих схем. При цьому вона відповідає поняттям прямої секретності та зворотної секретності третього рівня, а що головне – працює значно швидше всіх інших розглянутих методів. Хоч вона і розкриває більше інформації, ніж Fides, але швидкість пошуку розрізняється в десятки разів.

Отже, серед розглянутих методів ШМП, кращим та ефективнішим є метод Diana. Якщо безпека для проекту є надто важливою, то можна звернути увагу на Fides, поступившись швидкістю роботи.

Розглянуті методи перш за все підходять для NoSQL баз даних, оскільки мають дуже слабку виразність запитів. Виразність можна покращити реалізацією IEX або BIEX поверх основної схеми для можливості створювання логічних запитів, але це погіршить всі інші характеристики методу.

ВИСНОВКИ

1) Широке поширення конфіденційних даних у відкритих інформаційно-телекомунікаційних системах стимулювало дослідження в галузі безпечного управління даними, у тому числі досліджень, пов'язаних з можливістю здійснення пошуку даних у криптографічно захищених базах даних. Сьогодні пошук у криптографічно захищених БД досяг поворотного моменту у своєму розвитку. При цьому, незважаючи на велику кількість існуючих підходів, в даний час немає домінуючого рішення для всіх різних випадків використання, не існує найбільш захищеної системи з можливістю пошуку даних або набору методів. Тому в даній роботі було проаналізовано деякі методи, що дають змогу наблизити вирішення проблеми безпечного зберігання конфіденційних даних на ненадійному віддаленому сервері.

2) З метою кращого розуміння особливостей аналізованої предметної галузі у роботі визначено основні поняття термінологічного базису досліджень, наведено класифікацію методів та схем шифрування з можливістю пошуку, описано моделі безпеки, що застосовуються для оцінки захищеності схем SE.

3) При проведенні порівняльного аналізу аналізованих методів пошуку даних у захищених БД враховувалися ефективність, безпека та виразність схем шифрування з можливістю пошуку. Хоча більшою мірою оцінка аналізованих методів пошуку даних здійснювалася на підставі таких характеристик, як часова, комунікаційна та просторова складності.

Надано асимптотичні оцінки складності алгоритмів з точки зору часу обчислення, кількості інформації, що передається між клієнтом та сервером, та кількості інформації, що зберігається і клієнтом, і сервером. Але цього недостатньо для визначення швидкості та тим більше безпеки схеми. Тому було наведено конструкції та криптопримітиви, на яких базується схема, а також детально описано інформацію, що витікає при роботі схеми.

4) На основі проведеного аналізу методів, були вироблені рекомендації щодо застосування аналізованих методів на практиці у різних проектах.

Згідно рекомендаціям, серед розглянутих доцільно застосовувати методи Fides та Diana. Використання надбудови IEX можливе, якщо дуже важливою є виразність запитів. Однак, слід розуміти, що при її використанні значно погіршується продуктивність.

Розглянуті методи доцільно використовувати переважно для NoSQL баз даних через їхній посередній рівень виразності, у тому числі навіть при використанні IEX.

5) У роботі отримані значення певних показників якості (часова, комунікаційна та просторова складності) досліджуваних методів, які можуть бути використані для порівняння з іншими методами та схемами БМП, у тому числі нових, для виявлення більш прийнятних з них у конкретних умовах застосування.

6) Наведені у роботі висновки та рекомендації дозволяють оцінити можливість практичного використання у конкретних проектах проаналізованих методів пошуку даних у зашифрованих БД.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Paverd A., Martin A., Brown I. Modelling and automatically analysing privacy properties for honest-but-curious adversaries. *Tech. Rep*. 2014.
2) Song D. X., Wagner D., Perrig A. Practical techniques for searches on encrypted data. *Proceeding 2000 IEEE symposium on security and privacy*. 2000. P. 44-55.
3) Goldreich O., Ostrovsky R. Software protection and simulation on oblivious RAMs. *Journal of the ACM (JACM)*. 1996. Т. 43. №. 3. P. 431-473.
4) Gentry C. Fully homomorphic encryption using ideal lattices. *Proceedings of the forty-first annual ACM symposium on Theory of computing*. 2009. P. 169-178.
5) Pandey O., Rouselakis Y. Property preserving symmetric encryption. *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, Berlin, Heidelberg, 2012. P. 375-391.
6) Katz J., Sahai A., Waters B. Predicate encryption supporting disjunctions, polynomial equations, and inner products. *Annual international conference on the theory and applications of cryptographic techniques*. Springer, Berlin, Heidelberg, 2008. P. 146-162.
7) Chase M., Kamara S. Structured encryption and controlled disclosure. *International conference on the theory and application of cryptology and information security*. Springer, Berlin, Heidelberg, 2010. P. 577-594.
8) Kamara S. Encrypted search. *XRDS: Crossroads, The ACM Magazine for Students*. 2015. Т. 21. №. 3. P. 30-34.
9) Stefanov E., Papamanthou C., Shi E. Practical dynamic searchable encryption with small leakage. *Cryptology ePrint Archive*. 2013.

10) Bost R., Minaud B., Ohrimenko O. Forward and backward private searchable encryption from constrained cryptographic primitives. *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 2017. P. 1465-1482.
11) Search pattern leakage in searchable encryption: Attacks and new construction / Liu C. et al. *Information Sciences*. 2014. T. 265. P. 176-188.
12) Islam M. S., Kuzu M., Kantarcioglu M. Access pattern disclosure on searchable encryption: ramification, attack and mitigation. *Ndss*. 2012. T. 20. P. 12.
13) Goh E. J. Secure indexes. *Cryptology ePrint Archive*. 2003.
14) Chang Y. C., Mitzenmacher M. Privacy preserving keyword searches on remote encrypted data. *International conference on applied cryptography and network security*. Springer, Berlin, Heidelberg, 2005. P. 442-455.
15) Searchable symmetric encryption: improved definitions and efficient constructions / Curtmola R. et al. *Proceedings of the 13th ACM conference on Computer and communications security*. 2006. P. 79-88.
16) Public key encryption with keyword search / Boneh D. et al. *International conference on the theory and applications of cryptographic techniques*. Springer, Berlin, Heidelberg, 2004. P. 506-522.
17) Kaltz J., Lindell Y. Introduction to modern cryptography: principles and protocols. *Chapman and Hall*. *2008*.
18) A survey of provably secure searchable encryption / Bösch C. et al. *ACM Computing Surveys (CSUR)*. 2014. T. 47. №. 2. P. 1-51.
19) Kamara S., Papamanthou C. Parallel and dynamic searchable symmetric encryption. *International conference on financial cryptography and data security*. Springer, Berlin, Heidelberg, 2013. P. 258-274.
20) Introduction to algorithms / Cormen T. H. et al. *MIT press*. 2022.
21) Dynamic searchable encryption in very-large databases: Data structures and implementation / Cash D. et al. *Cryptology ePrint Archive*. 2014.

22) Kamara S., Moataz T. Boolean searchable symmetric encryption with worst-case sub-linear complexity. *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, Cham, 2017. P. 94-124.

23) Online ciphers and the hash-CBC construction / Bellare M., et al. *Annual International Cryptology Conference*. Springer, Berlin, Heidelberg, 2001. P. 292-309.

24) Bloom B. H. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*. 1970. Т. 13. №. 7. P. 422-426.

25) Bost R. Σοφος: Forward secure searchable encryption. *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 2016. P. 1143-1154.

26) Boneh D., Waters B. Constrained pseudorandom functions and their applications. *International conference on the theory and application of cryptology and information security*. Springer, Berlin, Heidelberg, 2013. P. 280-300.

27) Kiayias A. et al. Delegatable pseudorandom functions and applications. *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. 2013. P. 669-684.

28) Goldreich O., Goldwasser S., Micali S. How to construct random functions. *Journal of the ACM (JACM)*. 1986. Т. 33. №. 4. P. 792-807.

29) Махмудов Т., Єсін В. Аналіз методів пошуку даних у криптографічно захищеній базі даних. *Комп'ютерні науки та кібербезпека*. 2022. №. 2. С. 4-19. doi: 10.26565/2519-2310-2022-2-01

ДОДАТОК А

СПИСОК ПУБЛІКАЦІЙ МАГІСТРА

ISSN 2519-2310 CS&CS, Issue 2(22) 2022

УДК 681.04

АНАЛІЗ МЕТОДІВ ПОШУКУ ДАНИХ У КРИПТОГРАФІЧНО ЗАХИЩЕНІЙ БАЗІ ДАНИХ

Теймур Махмудов, Віталій Єсін

Харківський національний університет імені В.Н. Каразіна, майдан Свободи, 4, Харків, 61022, Україна
timkin552364964@gmail.com, v.i.yesin@karazin.ua

Рецензент: В'ячеслав Калашников, д. ф.-м.н., проф., Технологічний університет Монтеррея, 64849 Монтеррей, Нуево-Леон, Мексика
kalash@itesm.mx

Надійшло: Серпень 2022.

Анотація: *Питання забезпечення безпеки даних, а саме їх конфіденційності та цілісності, як правило, у теперішній час вирішується за рахунок використання відповідних криптографічних примітивів з урахуванням розвитку обчислювальних потужностей. Але, у зв'язку із специфічним способом зберігання (у хмарі), виникає питання ефективності пошуку необхідної інформації. Проблема, яка розглядається у цій роботі, полягає у тому, що шифрування унеможливлює доступ до даних без ключів для зловмисника, позбавляючи при цьому, власника даних, можливості проведення пошуку за цією інформацією. В роботі розглянуто декілька методів шифрування з можливістю пошуку. Для кожного з них приведені алгоритми, приклади використання цих методів і надані пояснювальні рисунки та таблиці. Розглянуті методи є симетричними та динамічними, завдяки чому вони ефективні та мають відносно високий рівень безпеки, але низьку виразність запитів, через що знаходять найбільше використання у NoSQL базах даних. Проведено аналіз для виявлення складності та рівня безпеки методів, а також розглянута продуктивність практичних реалізацій. Зроблено висновки про доцільність використання того чи іншого методу шифрування з можливістю пошуку на практиці. Запропоновані рекомендації щодо комбінації описаних методів для отримання очікуваного результату.*

Ключові слова: *шифрування з можливістю пошуку, ключове слово, індекс, динамічне симетричне шифрування з можливістю пошуку, токен пошуку, лазівка.*

1. Вступ

За останні роки хмарні послуги стали важливою частиною життя більшості людей, замінивши та полегшивши виконання їхньої щоденної рутини. Тепер необов'язково мати на своєму персональному комп'ютері набір інструментів від Office, так як у будь-який момент можна скористатися безкоштовними аналогами Google Docs, Google Sheets або Office 365. Так само на ринку з'явилося безліч рішень для зберігання даних у хмарі: Google, Microsoft, Dropbox, Mega та багато інших, але у всіх з них стоїть питання безпеки даних, що зберігаються. Адже дані не лише передаються через Інтернет, але ще й зберігаються у третіх осіб, тому, не дивлячись на запевнення осіб, які надають послуги, необхідно пам'ятати про безліч новин про витоки даних.

Для здійснення пошуку можна завантажити всю базу даних (БД) і виконати розшифрування. Проте для більшості застосунків цій підхід є недоцільним. Якщо ж покласти на сервер розшифрування даних, щоб він міг виконувати запити і надсилати користувачеві лише результати, то знижується рівень безпеки, оскільки дані, що захищені шифруванням, розкриваються серверу. Тому слід мати можливість проведення пошуку у повному обсязі на стороні сервера без розшифрування даних, що зменшує ризик втрати конфіденційності даних.

У роботі запропоновано огляд і аналіз предметної області, що розглядається, саме, шифрування з можливістю пошуку (ШМП), проводиться його класифікація та наводяться моделі безпеки, які характеризують рівень захищеності методів ШМП. Крім того проведено стислий розгляд конструкції і аналіз відповідних методів за часовою, комунікаційною та просторовою складностями, а також за рівнем забезпечуваної ними безпеки.

