УДК 004.056: 004.056.5 **DOI:10.30837/rt.2022.3.210.04**

В.І. ЄСІН, д-р техн. наук, В.В. ВІЛІГУРА

ДОСЛІДЖЕННЯ ОСНОВНИХ СХЕМ ШИФРУВАННЯ З МОЖЛИВІСТЮ ПОШУКУ У БАЗАХ ДАНИХ, ЯКІ ПІДТРИМУЮТЬ SQL

Вступ

Аутсорсинг зберігання та обробки даних на сторонніх серверах, таких як сервери хмар, широко використовується і демонструє вибухове зростання [1]. Однак у міру збільшення масштабу, цінності та централізації даних зростають проблеми безпеки та приватності. Існує виявлений ризик того, що дані, які зберігаються в базах даних, будуть скомпрометовані [2]. А цього, відповідно до різних міжнародних законів і стандартів таких, як: Загальний регламент захисту персональних даних Європейського Союзу (General Data Protection Regulation – GDPR) [3], Стандарт безпеки даних індустрії платіжних карток (Payment Card Industry Data Security Standard – PCI DSS) [4], Закон про переносимість та підзвітність медичного страхування (Health Insurance Portability and Accountability Act – HIPAA) [5, 6] та деякими іншими, не можна допустити. Це стимулювало дослідження у сфері безпечного управління даними та підвищило їх актуальність.

Шифрування – це стандартний підхід до забезпечення конфіденційності даних, що передаються на аутсорсинг так званим чесним, але допитливим серверам хмар. Шифрування унеможливлює доступ до даних без ключів як для інсайдерів (insider), так і для сторонніх осіб (outsider). Однак традиційні схеми шифрування позбавляють користувачів певних функціональних можливостей над даними, таких, наприклад, як пошук [7]. У цьому випадку для пошуку слів у зашифрованих зовнішніх даних користувачам необхідно отримати весь набір даних із відповідного сховища, розшифрувати його та здійснити пошук локально. Такий підхід створює серйозні проблеми з продуктивністю, які зводять нанівець переваги аутсорсингу, внаслідок чого для більшості застосунків він стає неприйнятним.

Проблема пошуку за зашифрованими даними викликала великий інтерес як у наукових колах, так і в індустрії. Однак, як можна помітити [8 – 10], дослідження з шифрування з можливістю пошуку (searchable encryption – SE) більшою мірою зосереджені на сценарії користувача, який передає на аутсорсинг зашифрований набір документів (таких як електронна пошта, медичні записи тощо) і хотів би продовжити пошук за ключовими словами у цьому зашифрованому наборі даних. Хоча на практиці багато компаній, організацій, установ різних форм власності зберігають дані в базах, що використовують реляційну модель даних. Широко поширена мова SQL дозволяє користувачам зберігати, запитувати та оновлювати свої дані у зручній для них формі. Бази даних SQL (до того ж NewSQL та деякі NoSQL бази даних також дозволяють працювати в парадигмі SQL-запитів) забезпечують швидкий пошук та вилучення записів за умови, що сервер SQL може зчитувати вміст даних. Однак шифрування зазвичай заважає серверу зчитувати необхідні дані, а отже, ускладнює пошук у зашифрованих базах даних. Зокрема, механізм криптографічного захисту даних для пошуку за зашифрованими даними, що зберігаються в базі даних, повинен дозволяти серверу ефективно обробляти пошукові запити, не маючи доступу до відкритих даних. Крім того, запити до зашифрованої бази даних мають бути зручними для користувача. Запити зазвичай виражаються стандартною мовою, такою, наприклад, як SQL. Запити повинні імітувати функції пошуку, так само, якби дані не були зашифровані. Безпосереднє застосування рішень для пошуку необхідної інформації в зашифрованих БД даних, що підтримують SQL, не є простим завданням.

Відомі дві основні проблеми конфіденційності [2, 11]. По-перше, власник даних повинен бути впевнений, що дані, які зберігаються на сайті постачальника послуг, захищені від крадіжки даних сторонніми особами. По-друге, дані мають бути захищені навіть від постачаль-

ників послуг (допустимого (valid) користувача, відомого як інсайдер), якщо самим постачальникам не можна довіряти. При цьому зазвичай розрізняють:

– противників / зловмисників, які є напівчесними (semi-hones; або чесними, але допитливими – honest-but-curious), тобто вони наслідують запропоновані протоколи, але можуть пасивно намагатися отримати додаткову інформацію з повідомлень, які вони спостерігають;

– противників, які є шкідливими (malicious), що означає, що вони активно бажають виконувати будь-які дії, необхідні для отримання додаткової інформації чи впливу на роботу системи.

Крім того, розрізняють зловмисників, які зберігаються протягом усього терміну служби бази даних, та тих, які отримують моментальний знімок в один момент часу. Більшість активних досліджень у галузі технології захищеного пошуку розглядає напівчесний захист від постійного внутрішнього противника [2]. У цьому роботі ми також зосередимося на цьому аспекті.

Проєктування захищеної пошукової системи – це баланс між безпекою, функціональністю, продуктивністю та зручністю використання. Описи безпеки зосереджені на інформації, що розкривається або просочується зловмиснику, який має доступ до сервера бази даних. Функціональність насамперед характеризується типами запитів, на які може відповісти захищена база даних. На продуктивність та зручність використання впливають структури даних бази даних та механізми індексування, а також необхідні обчислювальні та мережеві витрати.

Основне завдання в аналізованому аспекті захищеної системи пошуку на віддаленому сервері, якому не довіряють, спрямована на те, щоб сервер нічого не дізнався про дані, що зберігаються в захищеній базі даних, або про запити, а запитувач нічого не дізнався, крім результатів запиту, при цьому залишити можливість використовувати запити SQL типу на зашифрованих даних. Для цього застосовуються різні підходи, системи та методи. Однак, незважаючи на велику різноманітність запропонованих варіантів, немає домінуючого рішення для всіх випадків використання. Не існує найбільш захищеної пошукової системи або набору методів. Користувачі повинні розуміти характеристики системи та компроміси для свого варіанта використання. Тому розглянемо далі деякі з основних існуючих рішень.

1. CryptDB

CryptDB – це система, яка забезпечує безпечне зберігання конфіденційних даних у віддалених БД, у тому числі, що обслуговуються у хмарних сервісах. Вона працює, виконуючи SQL-запити до зашифрованих даних, використовуючи набір ефективних схем шифрування, які підтримують SQL. CryptDB може пов'язувати ключі шифрування з паролями користувачів, так що елемент даних може бути розшифрований лише за допомогою пароля одного з користувачів, які мають доступ до цих даних. В результаті адміністратор бази даних ніколи не отримає доступу до розшифрованих даних, і навіть якщо всі сервери будуть скомпрометовані, зловмисник не зможе розшифрувати дані жодного користувача, що не увійшов до системи [12].

В основі CryptDB лежить використання можливостей СУБД MySQL або PostgreSQL. Основна перевага CryptDB полягає у виконанні запитів над зашифрованими даними, що уможливлює застосування CryptDB на практиці – використання чітко визначеного набору SQL-операторів, кожен з яких може ефективно обробляти зашифровані дані.

У CryptDB розглядаються дві основні загрози [12]:

– Загроза 1 – допитливий адміністратор бази даних – пасивний противник, який намагається дізнатися про конфіденційні дані (шляхом відстеження на сервері СУБД), але CryptDB перешкоджає цьому;

– Загроза 2 – зловмисник, який може отримати повний контроль над застосунками та серверами СУБД. У цьому випадку CryptDB не може надати будь-яких гарантій користува-

чам, які вже працювали з застосунком під час атаки, але все ж таки може забезпечити конфіденційність даних інших користувачів, що вийшли з системи.

Архітектура CryptDB складається з двох частин: проксі-сервера бази даних та немодифікованої СУБД (рис. 1).

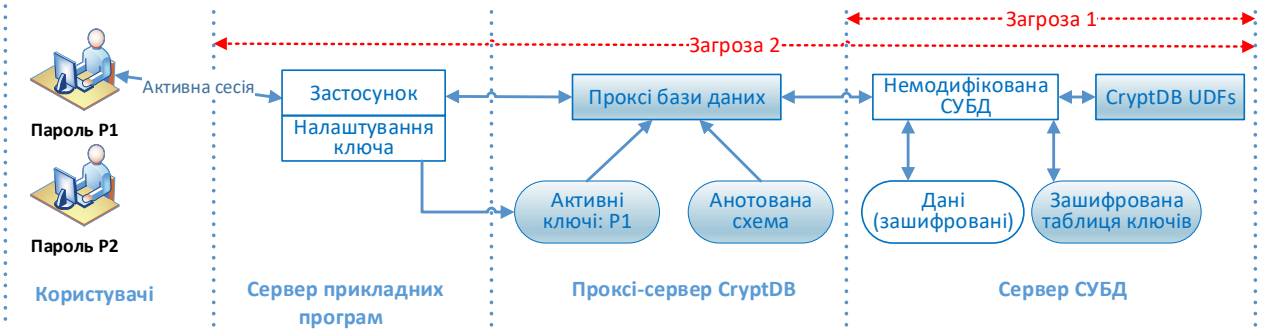

Рис. 1. Архітектура CryptDB

CryptDB використовує функції, що визначаються користувачем (User Defined Functions – UDFs) для виконання криптографічних операцій в СУБД. Прямокутники та закруглені прямокутники представляють процеси та дані відповідно. Затінення вказує на компоненти, додані CryptDB. Пунктирні лінії вказують на поділ між комп'ютерами користувачів, сервером прикладних програм, сервером, на якому працює проксі-сервер бази даних CryptDB (який зазвичай збігається з сервером прикладних програм), і сервером СУБД. CryptDB усуває два типи загроз, показаних пунктирними лініями. При загрозі один допитливий адміністратор бази даних з повним доступом до сервера СУБД відстежує особисті (private) дані, і в цьому випадку CryptDB запобігає доступу адміністратора баз даних до будь-якої приватної інформації. При загрозі 2 зловмисник отримує повний контроль як над програмним, так і над апаратним забезпеченням програми, проксі та серверами СУБД, і в цьому випадку CryptDB гарантує, що зловмисник не зможе отримати дані, що належать користувачам, які не увійшли до системи (наприклад, користувач 2).

Проксі-сервер використовує секретні ключі для шифрування всіх даних, що вставлені. Основна ідея обчислень над зашифрованими даними полягає у тому, щоб дозволити серверу СУБД виконувати обробку запитів до зашифрованих даних, як це було б із незашифрованою базою даних, тобто дозволити йому обчислювати певні функції над елементами даних на основі зашифрованих даних. Наприклад, якщо СУБД необхідно виконати команду угруповання GROUP BY за деяким стовпцем, то сервер СУБД повинен вміти визначати, які елементи в цьому стовпці рівні один одному, але не фактичне значення кожного.

Противник може отримати доступ до ключів, які використовуються для шифрування всієї бази даних. Рішення полягає в тому, щоб зашифрувати різні елементи даних (наприклад, дані, що належать різним користувачам) із різними ключами. Противник, який атакує сервер прикладних програм або проксі-сервер, тепер може розшифрувати лише дані користувачів, що увійшли до системи в даний момент (дані, що зберігаються на проксі-сервері). Дані неактивних користувачів зашифровані ключами, які недоступні зловмиснику і залишаться конфіденційними. Для успішної роботи CryptDB використовує три основні рішення.

1. До зашифрованих баз даних звернення здійснюється за допомогою SQL-запитів, що дозволяють виконувати такі операції, як перевірка на рівність, порівняння порядку, агрегація / підсумовування та з'єднання (це уможливлює здійснення на практиці обробку зашифрованих даних). CryptDB шляхом адаптації відомих схем шифрування (для перевірки на рівність, підсумовування, порівняння порядку) та використання нового криптографічного методу із збереженням конфіденційності для з'єднань, CryptDB шифрує кожен елемент даних таким чином, щоб СУБД могла працювати з перетвореними даними. В основному використовується шифрування із симетричним ключем.

2. Шифрування, що настроюється, на основі запитів. Деякі схеми шифрування пропускають на сервер СУБД більше інформації про дані, ніж інші, але вони необхідні для обробки певних запитів. Щоб уникнути цього, CryptDB ретельно (залежно від запитів) налаштовує схему шифрування для будь-якого заданого елемента даних, використовуючи так звані «цибулини» (onions) шифрування. Onions – це новий спосіб компактного зберігання кількох зашифрованих текстів один в одному в базі даних та запобігання дорогому повторному шифруванню.

CryptDB має можливість перемикатися «на льоту» між різними криптографічними схемами залежно від типу операції, що виконується. Це реалізовано за рахунок «цибулинного» багатоступінчастого шифрування, коли дані зашифровані в кілька шарів різними алгоритмами. У кожного шару свій ключ і свій список операцій, що підтримуються. На нижньому шарі використовуються найнадійніші алгоритми, а операції у верхніх шарах можливі без розшифровування нижніх шарів.

3. Зв'язування ключів шифрування з паролями користувачів, щоб кожен елемент даних у БД можна було розшифрувати тільки за допомогою ланцюжка ключів, що базується на паролі одного з користувачів, що мають доступ до цих даних. В результаті, якщо користувач не авторизований у застосунку, і якщо зловмисник не знає пароль користувача, то зловмисник не може розшифрувати дані користувача, навіть якщо СУБД та сервер прикладних програм повністю скомпрометовані.

CryptDB використовує для різних запитів до бази даних наступні типи шифрування.

Випадковий (Random – RND). RND забезпечує максимальну безпеку в CryptDB: IND-CPA (indistinguishability under chosen plaintext attack – нерозрізненість при атаці за вибраним відкритим текстом), схема є ймовірнісною, що означає, що два рівні значення відображаються в різні шифртексти з ймовірністю близькою до одиниці. З іншого боку, RND не дозволяє ефективно виконувати будь-які обчислення із зашифрованим текстом. Ефективна конструкція RND полягає у використанні блокового шифру, такого як AES або Blowfish, у режимі зчеплення блоків шифртексту (Cipher Block Chaining – CBC) разом із випадковим вектором ініціалізації (IV).

Детермінований (Deterministic – DET). DET має трохи більш слабку гарантію, але, як і раніше, забезпечує надійний захист: можливий витік тільки тих зашифрованих значень, які відповідають одному й тому ж значенню даних, детерміновано генеруючи один і той же зашифрований текст для того самого відкритого тексту. Цей рівень шифрування дозволяє серверу виконувати перевірки на рівність, що означає, що він може виконувати вибірку з предикатами рівності, з'єднання за еквівалентністю, GROUP BY, COUNT, DISTINCT і т. д. З погляду криптографії DET має бути псевдовипадковою перестановкою (pseudo-random permutation – PRP).

Шифрування зі збереженням порядку (Order-preserving encryption – OPE) дозволяє встановлювати відносини порядку між елементами даних на основі їх зашифрованих значень, не розкриваючи самих даних. Якщо $x < y$, то $\mathrm{OPE}_K(x) < \mathrm{OPE}_K(y)$ для будь-якого секретного ключа K. Отже, якщо стовпець зашифрований за допомогою OPE, сервер може виконувати запити діапазону за наявності зашифрованих констант $\mathrm{OPE}_K(c_1)$ і $\mathrm{OPE}_K(c_2)$, відповідних діапазону $[c_1, c_2]$. Сервер також може виконувати SQL запити з конструкціями, функціями, що агрегують, ORDER BY, MIN, MAX, SORT тощо. OPE є слабкішою схемою шифрування, ніж DET, оскільки вона розкриває порядок. Таким чином, проксі-сервер CryptDB показуватиме серверу стовпці, зашифровані за допомогою OPE, тільки якщо користувачі запитують запити порядку цих стовпців.

Гомоморфне шифрування (Homomorphic encryption – HE) – це безпечна ймовірна схема шифрування (безпека рівня IND-CPA), що дозволяє серверу виконувати обчислення із зашифрованими даними, при цьому остаточний результат розшифровується на проксі-сервері. Хоча повністю гомоморфне шифрування (fully homomorphic encryption – FHE) дуже повіль-

не, гомоморфне шифрування для певних операцій є ефективним. Зокрема, для підтримки додавання був реалізований метод Пайе (Paillier) [13]. У Пайе множення двох зашифрованих значень призводить до шифрування суми значень, тобто $\mathrm{HE}_K(x)\cdot\mathrm{HE}_K(y)=\mathrm{HE}_K(x+y)$, де множення виконується за модулем деякого значення відкритого ключа. Для обчислення агрегатів SUM проксі-сервер замінює SUM викликами функції користувача (UDF), яка виконує множення Пайе в стовпці, зашифрованому за допомогою HE. HE також можна використовувати для обчислення середніх значень, якщо сервер СУБД повертає суму та кількість окремо, а також для збільшення значень (наприклад, SET *id=id+1*).

З'єднання (JOIN та OPE-JOIN). Для забезпечення рівності між двома стовпцями потрібна окрема схема шифрування, оскільки використовуються різні ключі для DET, щоб запобігти кореляції між стовпцями. JOIN також підтримує всі операції, дозволені DET, а також дозволяє серверу визначати значення, що повторюються між двома стовпцями. OPE-JOIN дає змогу виконувати з'єднання за відносинами порядку.

Пошук слова (SEARCH). SEARCH використовується для пошуку в зашифрованому тексті для підтримки таких операцій, як оператор LIKE в MySQL. Для кожного стовпця, який вимагає пошук, текст розбивається на ключові слова, використовуючи стандартні роздільники (або використовуючи спеціальну функцію вилучення ключових слів). Потім у цих словах видаляються повтори, випадково міняються місцями слова, а потім шифрується кожне зі слів, використовуючи схему [14], доповнюючи кожне слово до однакового розміру. SEARCH майже так само безпечний, як RND: шифрування не повідомляє серверу СУБД, чи повторюється певне слово в декількох рядках, але воно дає витік кількості ключових слів, зашифрованих за допомогою SEARCH. Зловмисник може оцінити кількість окремих або повторюваних слів (наприклад, шляхом порівняння розміру шифртекстів SEARCH і RND для тих самих даних).

Коли користувач виконує такий запит, як:

```
SELECT * FROM messages WHERE msg LIKE "%alice%",
```

проксі передає серверу СУБД токен (лазівку, Trapdoor), що є шифруванням "alice". Сервер не може розшифрувати токен, щоб визначити слово, що лежить в його основі. За допомогою функції, що визначається користувачем, сервер СУБД перевіряє, чи відповідає яке-небудь зашифроване слово в будь-якому повідомленні токену. У цьому підході все, що сервер дізнається в результаті пошуку, це те, чи відповідає токен повідомленню чи ні, і це відбувається тільки для токенів, запрошених користувачем. При цьому слід звернути увагу, що SEARCH дозволяє CryptDB шукати за ключовими словами лише за повним словом. Він не може підтримувати довільні звичайні вирази.

Таким чином, автори CryptDB [12] у своєму рішенні запропонували платформу, яка підтримує SQL-запити до зашифрованих даних. Це рішення ґрунтується на різних рівнях шифрування із збереженням властивостей, таких як детерміноване (DET) та шифрування із збереженням порядку (OPE), що застосовуються до стовпця таблиці SQL. Щоб запросити зашифровану базу даних, CryptDB перетворює незашифрований SQL-запит на його зашифрований еквівалент і розшифровує цільові шари. Основний недолік CryptDB полягає в тому, що щоразу, коли видаляється один шар, схема шифрування стає слабкою [9].

2. MONOMI

MONOMI – система безпечного виконання операцій над конфіденційними даними на ненадійному сервері бази даних [15]. MONOMI працює шляхом шифрування всієї бази даних та виконання запитів до зашифрованих даних. MONOMI була реалізована з використанням PostgreSQL та призначена для виконання аналітичних SQL-запитів.

Існуючі проблеми, які змушена вирішувати MONOMI:

– по-перше, запити до великих наборів даних часто обмежені можливостями системи введення-виводу, наприклад, читання даних із диска або потокове передавання через пам'ять.

В результаті схеми шифрування, які значно збільшують розмір даних можуть уповільнити обробку запитів;

– по-друге, аналітичні запити вимагають складних обчислень, які можуть бути неефективними для виконання над зашифрованими даними. Натомість на практиці повинні використовуватися ефективні схеми шифрування, які можуть виконувати лише певні обчислення (наприклад, використання шифрування із збереженням порядку для сортування та порівняння тощо). Завдання полягає в тому, щоб розділити запит на частини, які можуть бути виконані з використанням доступних схем шифрування на ненадійному сервері та частини, які повинні бути виконані на довіреному клієнті;

– по-третє, деякі методи обробки запитів за зашифрованими даними можуть прискорити одні запити, але уповільнити інші, що вимагає ретельного планування виконання кожного запиту для бази даних, що розглядається, і комбінації запитів.

Способи вирішення зазначених проблем:

а) вводиться роздільне клієнт-серверне виконання складних запитів, при якому виконується максимально можлива частина запиту за зашифрованими даними на сервері, а компоненти запитів, що залишилися, виконуються шляхом надсилання зашифрованих даних довіреному клієнту, який розшифровує дані та обробляє запити у звичайному режимі;

б) вводиться ряд методів, що підвищують продуктивність для певних типів запитів (але не обов'язково для всіх), включаючи попереднє обчислення для кожного рядка, ефективне за простором / економічне шифрування (space-efficient encryption), групове гомоморфне додавання (grouped homomorphic addition) та попередню фільтрацію (pre-filtering);

в) до архітектури додаються проєктувальник для оптимізації фізичного розміщення даних на сервері та планувальник для прийняття рішення про те, як розділити виконання запиту між клієнтом та сервером. Проєктувальник і планувальник необхідні, тому що «жадібне» застосування всіх методів або «жадібне» виконання всіх обчислень на сервері може призвести до надмірних накладних витрат пам'яті та / або неефективного виконання запитів.

Оскільки розробка MONOMI базувалася на CryptDB, вона має аналогічні властивості, що стосуються безпеки. Хоча ненадійний сервер зберігає лише зашифровані дані, він все ж таки може отримати інформацію про вихідні дані у вигляді відкритого тексту трьома способами:

1) деякі схеми шифрування розкривають інформацію, необхідну для обробки запитів (наприклад, детерміноване шифрування виявляє дублікати для перевірки рівності);

2) деякі схеми шифрування можуть пропускати більше інформації, ніж потрібно. Наприклад, схема із збереженням порядку дає частковий витік інформації про відкритий текст;

3) сервер дізнається, які рядки відповідають кожному предикату, обчисленому на сервері. Наприклад, рядки, які відповідають стовпцю LIKE '%keyword%'.

Комбінуючи ці джерела інформації, сервер, який є противником, може отримати додаткову інформацію про рядки або запити за допомогою статистичних методів.

MONOMI ніколи не зберігає текстові дані на сервері і використовує лише схеми шифрування, необхідні для роботи програми. MONOMI дозволяє адміністратору додатково обмежувати схеми шифрування, які використовуються для особливо чутливих стовпців. Наприклад, вимога шифрування із збереженням порядку (найслабша схема MONOMI) не повинна використовуватися для стовпців, у яких зберігаються номери кредитних карток або номерів соціального страхування.

Архітектура MONOMI представлена на рис. 2.

Під час налаштування системи проєктувальник MONOMI запускається на довіреній клієнтській машині та визначає її ефективну фізичну конфігурацію для ненадійного сервера. Щоб визначити основні характеристики робочого навантаження (workload – робочим навантаженням, як правило, є будь-яка програма, яка працює на комп'ютері; сьогодні терміни «робоче навантаження», «застосунок», «програмне забезпечення» та «програма» використовуються як взаємозамінні) для досягнення хорошої продуктивності проєктувальник приймає в

якості вхідних даних репрезентативну підмножину запитів і статистику за даними, наданими користувачем. Користувачі не зобов'язані застосовувати проєктувальник, і замість цього можуть вручну вводити стратегію шифрування або змінювати створену ними стратегію.

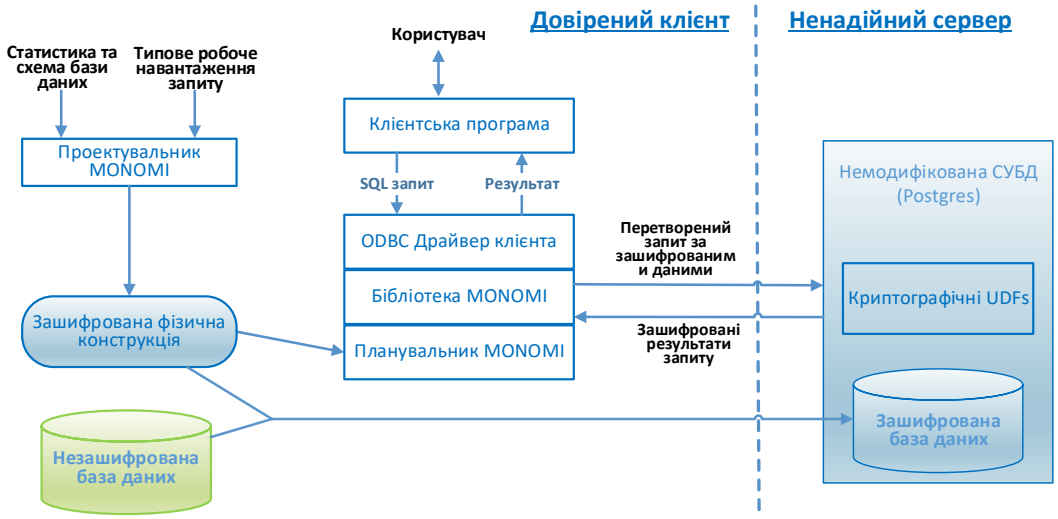

Рис. 2. Загальна архітектура MONOMI

При нормальній роботі програми видають немодифіковані запити SQL за допомогою бібліотеки MONOMI ODBC (Open Database Connectivity), яка є єдиним компонентом, що має доступ до ключів розшифрування. Бібліотека ODBC використовує планувальник, щоб визначити найкращий план виконання запиту з поділом для застосунків клієнт-сервер.

Враховуючи план виконання, бібліотека видає один або кілька запитів до зашифрованої бази даних, яка не має доступу до ключів розшифрування та може виконувати операції лише із зашифрованими даними. База даних запускає немодифіковане програмне забезпечення СУБД з декількома визначеними функціями користувача (UDFs), що надаються MONOMI, які реалізують операції із зашифрованими даними.

MONOMI шифрує всі дані, що зберігаються в базі даних, хоча на практиці неконфіденційні дані можна зберігати як відкритий текст для підвищення ефективності. Після того як клієнтська бібліотека отримає проміжні результати з бази даних, розшифрує їх і виконає всі операції, що залишилися, які не можуть бути ефективно виконані на сервері, результати надсилаються до застосунку, як якщо б вони виконувались у стандартній базі даних SQL.

Схеми шифрування, що використовуються MONOMI, приклади операцій SQL, які вони дозволяють над зашифрованими даними на сервері, та інформація, що розкривається зашифрованими текстами кожної схеми за відсутності будь-яких запитів, наведено у табл. 1.

Таблиця 1

Схеми шифрування, що використовуються у MONOMI

Схема шифрування	**SQL-операції**	**Витік (Leakage)**
Randomized AES + CBC	Жодної	Ні
Deterministic AES + режими: CMC [16] або FFX [17]	a = const, IN, GROUP BY, equi-join	Дублікати
OPE [18]	a > const, MAX, ORDER BY	Order + partial Незашифрований текст [19]
Paillier [13]	a + b, SUM(a)	Ні
SEARCH [12, 14]	шаблон LIKE	Ні

Основні характеристики MONOMI.

– реалізована поверх PostgreSQL;

– працює шляхом шифрування всієї бази даних та виконання запитів до зашифрованих даних;

– бібліотека MONOMI ODBC є єдиним компонентом, що має доступ до ключів розшифрування;

– представляє новий підхід, що базується на роздільному виконанні запиту на клієнт-сервері, що дозволяє виконати частину запиту на ненадійному сервері поверх зашифрованих даних. Для інших елементів запиту MONOMI завантажує проміжні результати на клієнт;

– бібліотека MONOMI ODBC отримує проміжні результати з бази даних, розшифровує їх і виконує всі операції, що залишилися, які не можуть бути ефективно виконані на сервері;

– реалізовано кілька методів, що покращують продуктивність: попереднє обчислення рядка, просторово-ефективне шифрування, групове гомоморфне додавання та попередня фільтрація;

– оскільки ці оптимізації добре працюють для одних запитів та неефективні для інших, MONOMI має планувальник, щоб визначити найкращий спосіб виконання запиту.

3. Seabed

Seabed – система, що забезпечує ефективну аналітику великих зашифрованих наборів даних [20]. Seabed використовує нову адитивно-симетричну схему гомоморфного шифрування (ASHE – additively symmetric homomorphic encryption) для ефективного виконання великомасштабних агрегацій. Крім того, Seabed представляє нову схему рандомізованого шифрування під назвою Splayed ASHE або SPLASHE, яка в деяких випадках може запобігти частотним атакам на основі допоміжних даних. ASHE і базовий варіант SPLASHE задовольняють стандартному поняттю семантичної безпеки (IND-CPA), у той час як розширений варіант SPLASHE доказово не пропускає більше інформації, ніж кількість значень вимірювань, які часто й рідко зустрічаються в базі даних. Прототип Seabed реалізований на базі Apache Spark (уніфікований аналітичний механізм (фреймворк) з відкритим кодом для великомасштабної обробки даних). На відміну від CryptDB та Monomi, система заснована не на реляційній базі даних, а на файловій системі, що призначена для зберігання неструктурованої інформації. Дані зберігаються в HDFS (Hadoop Distributed File System – файлова система, призначена для зберігання файлів великих розмірів, поблоково розподілених між вузлами обчислювального кластера) з використанням серіалізації Google Protobuf (Protobuf – це схема серіалізації, запропонована Google; ця схема не залежить від мови та платформи. Вона підтримує такі мови, як java, c, go, python та деякі інші, а також забезпечує підтримку файлів мультиплатформних бібліотек).

Схема Seabed представлена на рис. 3.

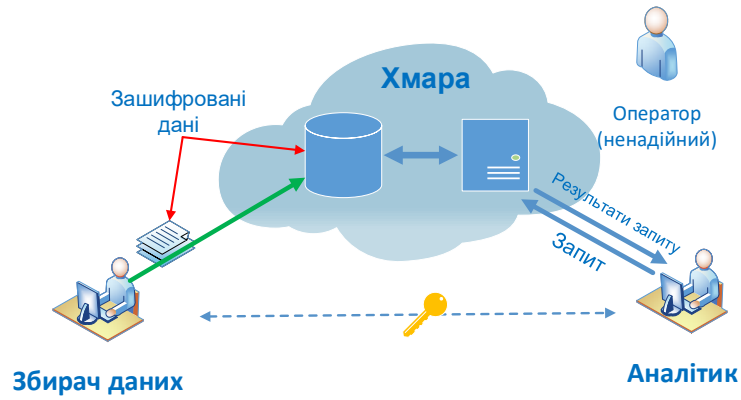

Рис. 3. Схема Seabed

Збирач даних (Data collector) збирає велику кількість даних, шифрує їх та завантажує у хмару, якій не довіряє. Аналітик може генерувати запити для обробника запитів у хмарі. Відповіді будуть зашифровані, але аналітик може розшифрувати їх за допомогою секретного ключа, який є спільним із збирачем даних. Робоче навантаження, яке передбачається підтримувати, складається із запитів у стилі OLAP (OnLine Analytical Processing – інтерактивна аналітична обробка) для великих наборів даних.

Основні компоненти Seabed наведено рис. 4.

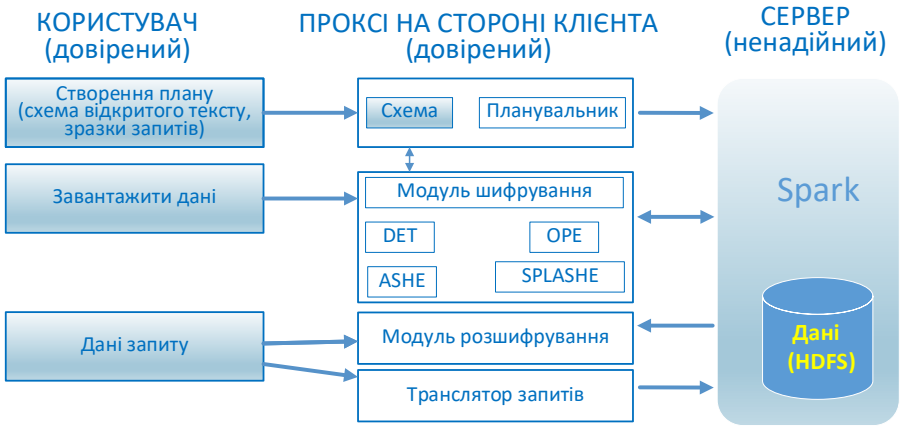

Рис. 4. Основні компоненти Seabed

Користувач взаємодіє з проксі-сервером клієнта Seabed, який працює у довіреному середовищі. Проксі, своєю чергою, взаємодіє з недовіреним сервером Seabed. Як і попередні системи, Seabed приховує від користувачів всі криптографічні операції, тому вони взаємодіють із системою так само, як зі стандартною системою Spark. Користувач може видавати три види запитів:

1. Створення плану (Create Plan). Спочатку користувач надає схему у вигляді відкритого тексту та зразок запиту, заданого для планувальника Seabed. Планувальник використовує їх і спеціальну процедуру визначення схем шифрування стовпців.
2. Завантаження даних (Upload Data). Потім користувач надсилає дані у вигляді відкритого тексту в модуль шифрування Seabed. Дані шифруються за допомогою необхідної схеми шифрування, а записи додаються до таблиці, що зберігається у хмарі. Це безперервний процес.
3. Запит даних (Query Data). Під час аналізу користувач надсилає сценарій запиту транслятору запитів Seabed, який модифікує запити обробки зашифрованих даних перед їх відправкою на сервер. Завдання транслятора запитів – перехоплювати немодифіковані запити клієнта та переписувати їх відповідно до схеми зашифрованого набору даних. При цьому в системі підтримуються принципи, введені в CryptDB та MONOMI. Єдина технічна відмінність від попередніх систем полягає в тому, що вони працюють з реляційними базами даних, тому і вихідною, і цільовою мовою транслятора є SQL. Однак Seabed працює на Spark, тому цільова мова – Scala та Spark API. Сервер виконує запити та відповідає модулю розшифрування проксі-сервера. Після розшифрування та подальшої обробки (якщо така потрібна) результати відправляються назад користувачеві.

Планувальник даних визначає, як шифрувати кожен стовпець у схемі з огляду на список конфіденційних стовпців, складений користувачем. Користувач також надає зразок набору запитів, який використовується планувальником для вибору алгоритму шифрування. Крім того, щоб використовувати розширений SPLASHE, користувач вказує кількість різних значень, які може набувати кожен стовпець, і частотний розподіл цих значень.

Модуль шифрування шифрує відкритий текст.

У разі використання адитивної симетричної схеми гомоморфного шифрування (ASHE) передбачається, що відкриті тексти беруться з адитивної групи $\mathbb{Z}_n = \{0,1,...,n-1\}$. Також

передбачається, що відправник і одержувач, що шифрують і розшифровують зашифрований текст, спільно використовують таємний ключ k, а також псевдовипадкову функцію (PRF) $F_k : I \to \mathbb{Z}_n$, що перетворює ідентифікатор з множини I у випадковий номер із $\mathbb{Z}_n$. Один із можливих варіантів PRF: $F_k = H(i \| k) \bmod n$, для $i \in I$, де H – криптографічна геш-функція (моделюється як випадкова функція), $\|$ позначає конкатенацію, розмір діапазону H кратний n. Іншим варіантом може бути використання AES, коли він використовується як псевдовипадкова перестановка.

На рис. 5 представлений загальний огляд ASHE у контексті Seabed. Наведена на рис. 5 схема відповідає стандартному поняттю семантичної безпеки (CPA).

Транслятор запитів перехоплює немодифіковані запити клієнта і переписує їх відповідно до обраного способу шифрування даних.

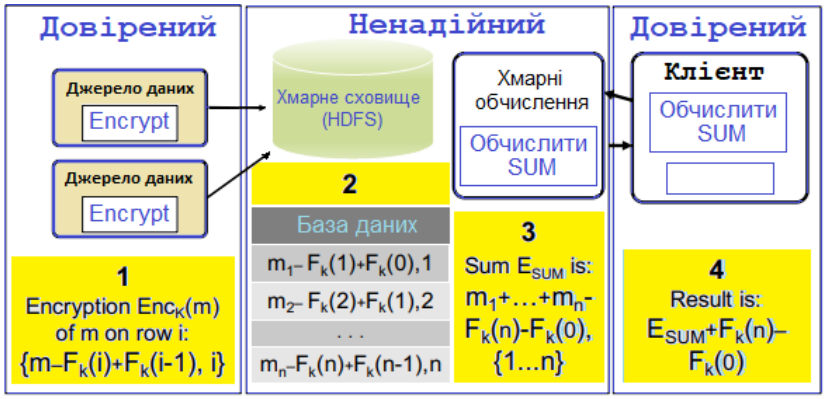

Рис. 5. Основні компоненти Seabed та схема ASHE

Передбачається, що противник є пасивним (чесним, але допитливим), тобто противник намагатиметься дізнатися про конфіденційні дані, але не буде їх активно спотворювати або іншим чином втручатися в роботу системи. Також передбачається, що зловмисник може спробувати провести частотну атаку. Особливо це актуально в тих випадках, коли стовпець може мати невелику кількість значень, а хмара знає, що якесь значення буде найбільш поширеним.

4. Arx

Arx – практично реалізована та багатофункціональна система баз даних, яка шифрує дані за допомогою семантично безпечних схем шифрування [20]. Система Arx реалізована на базі NoSQL СУБД MongoDB.

Передбачається, що сервер БД може бути розміщений у приватній або публічній хмарі.

Модель загроз Arx передбачає, що зловмисник не контролює і не спостерігає за даними або виконанням на стороні клієнта і може отримати доступ лише до сторони сервера, що складається з проксі-сервера Arx та серверів бази даних.

Arx вважає атакуючих сервер зловмисників пасивними (чесними, але допитливими): зловмисники вивчають дані на стороні сервера, щоб зібрати конфіденційну інформацію, але дотримуються протоколу і не змінюють бази даних або результатів запитів. Крім того, у моделі Arx зловмисник не може впровадити будь-які нові запити, оскільки він не має доступу до клієнтської програми або секретних ключів на клієнтському проксі, а лише до сервера. При цьому розглядаються два типи пасивних зловмисників, так званих офлайн та онлайн зловмисників із різними гарантіями для кожного з них.

Так офлайн зловмисник може викрасти одну копію бази даних, що складається із зашифрованих колекцій та індексів. Ця копія не містить даних у пам'яті, пов'язаних із виконанням поточних запитів (які підпадають під дію онлайн-зловмисника). Онлайн-зловмисник – це звичайний пасивний зловмисник: він може реєструвати та відстежувати будь-яку інформацію, доступну на сервері (тобто всі зміни в базі даних, весь стан у пам'яті та всі запити) у

будь-який момент часу протягом будь-якого періоду часу. При цьому Arx приховує параметри в запитах, але не операції, що виконуються.

Arx не покладається на якесь довірене обладнання на сервері. Основне завдання запропонованої архітектури (рис. 6) – не змінювати наявний сервер БД та застосунки, що з ним працюють. Застосунок та система бази даних залишаються незмінними. Натомість Arx вводить два компоненти між застосунком та сервером БД: довірений клієнтський проксі (client proxy) та недовірений проксі-сервер (server proxy).

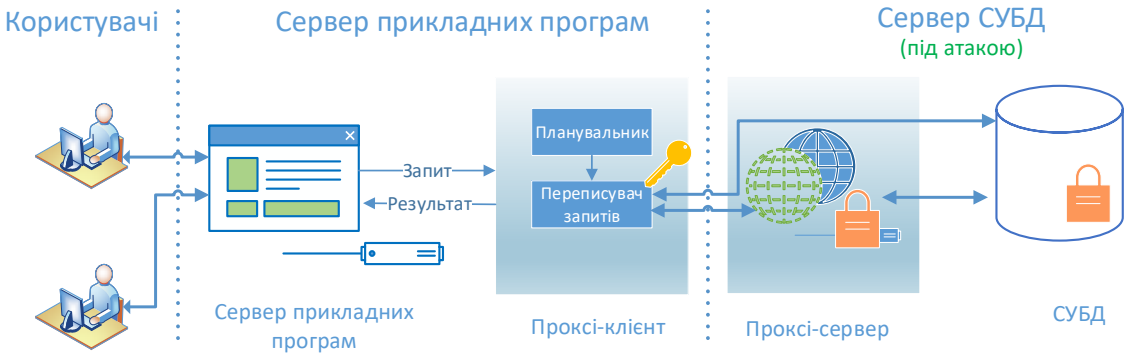

Рис. 6. Архітектура Arx

Довірений клієнтський проксі розгорнутий на сервері застосунків, ненадійний проксі-сервер розгорнутий на сервері СУБД, сервер СУБД розміщений у приватній або публічній хмарі. Клієнтський проксі перехоплює запити та шифрує конфіденційну інформацію. Проксі-сервер підтримує індекси зашифрованих даних та виконує вхідні запити. Сірі прямокутники зображують компоненти, представлені Arx, інші елементи представляють існуючі компоненти. Ключ вказує, що конфіденційні дані у компоненті завжди залишаються суворо зашифрованими.

Слід зазначити, що клієнтський проксі експортує до застосунку той же API, що і сервер БД, тому застосунок не потрібно змінювати. Проксі-сервер взаємодіє із сервером БД, викликаючи його незмінений API (наприклад, надаючи запити). Іншими словами, проксі-сервер веде себе як незмінний клієнт сервера БД. На відміну від CryptDB, Arx не може використовувати функції, що визначаються користувачем замість проксі-сервера, тому що проксі-сервер повинен взаємодіяти з сервером БД кілька разів для кожного запиту клієнта.

Клієнтський проксі зберігає головний (master) ключ. Він переписує запити, шифрує дані та перенаправляє переписані запити на проксі-сервер для виконання разом із допоміжними криптографічними токенами. Останній перенаправляє всі запити без будь-яких конфіденційних полів безпосередньо на сервер БД. Клієнтський проксі легковажний. Він не зберігає БД і робить набагато менше роботи, ніж сервер. Клієнтський проксі зберігає метадані (інформацію про схему), невелику кількість стану та, можливо, кеш. Сервер виконує дорогу частину запитів до БД, фільтруючи та поєднуючи безліч документів у невеликий набір результатів.

Щоб використовувати Arx, адміністратор вказує наступну інформацію під час налаштування системи:

1. Анотовану схему (необов'язково): унікальні поля, поля, які є конфіденційними (якщо є) і розміри полів.
2. Операції, які виконуються з конфіденційними полями.
3. Поля, які мають бути проіндексовані.

По-перше, адміністратор використовує наступний API:

$$collection = \{field_1: info_1, ..., field_n: info_n\},$$

щоб анотувати поля в колекції. Ця анотація необов'язкова, але, якщо вона надана, вона покращує продуктивність Arx. Для *info* слід вказати "*unique*", якщо значення в полі є унікальними, наприклад, SSN (Social Security Number). Arx автоматично визначає первинні ключі як

унікальні. У *info* також може вказуватись максимальна довжина поля, що допомагає Arx вибрати більш ефективну схему шифрування.

За замовчуванням Arx шифрує всі поля в БД. Однак адміністратор може явно перевизначити цю поведінку, вказавши інформацію як неконфіденційну для певного поля. Цю опцію слід використовувати тільки в тому випадку, якщо адміністратор вважає, що це поле не є конфіденційним, і хоче зменшити накладні витрати на шифрування, або якщо Arx не підтримує обчислення в цьому полі, але адміністратор все ще хоче використовувати Arx для інших полів. Однак при цьому слід враховувати, що хоча деякі поля самі по собі не можуть бути конфіденційними, вони можуть призвести до витоку допоміжної інформації про інші поля в базі даних. Отже, адміністратор має обирати такі поля з обережністю.

По-друге, Arx необхідно знати шаблони запитів, які будуть виконуватись у базі даних. Наприклад, для запиту `select * from T where age=10`, Arx необхідно знати, що проводитиметься перевірка на рівність за віком. Тобто Arx необхідно знати, які операції над якими полями виконуються, але не константи, які будуть запитуватись. Адміністратор може вказати ці операції безпосередньо, або надати трасування запуску застосунку, і Arx автоматично їх ідентифікує.

По-третє, Arx повинен знати перелік звичайних індексів, створених застосунком. Arx потрібна ця інформація, щоб забезпечити ті ж самі асимптотичні гарантії продуктивності, що й для незашифрованої бази даних.

Для пошуку даних Arx представляє два нові індекси бази даних:

– Arx-RANGE для запитів із діапазону.

– Arx-EQ для запитів на рівність.

Підтримувані Arx операції: INSERT, DELETE, UPDATE, SELECT (в тому числі з можливістю використання агрегуючих функцій: суми, суму квадратів, мінімуму і максимуму, середнього, агрегування, з мінімальною постобробкою на клієнтському проксі та інших).

Arx збільшує загальний обсяг даних, що зберігаються в базі даних, тому що: зашифровані тексти більше, ніж відкриті тексти для певних схем шифрування, і до документів додаються додаткові поля для виконання певних операцій, таких, наприклад, як перевірки на рівність з використанням EQ або токени для індексації ArxEq. Крім того, індекси ArxRange більші, ніж звичайні дерева B+, тому що кожен вузол у дереві індексів зберігає спотворені схеми. При цьому Arx підтримує менше запитів, ніж CryptDB.

5. CipherSweet

CipherSweet – серверна бібліотека, розроблена Paragon Initiative Enterprises для реалізації шифрування на рівні полів з можливістю пошуку [22]. CipherSweet використовує так зване сліпе індексування зі стратегіями нечіткої фільтрації та фільтрації Блума [23], щоб забезпечити швидкий пошук зашифрованих даних з мінімальним їх витоком. Фільтр Блума – це компактна імовірнісна структура даних, що підтримує запити на членство у множині. Фільтр Блума допускає хибні спрацьовування (твердження, що елемент є частиною множини, коли він таким не є), але ніколи не призводить до хибно негативних результатів (повідомлення про те, що існуючий елемент відсутній у множині). Сліпі індекси не слід плутати з традиційними індексами, введеними для підвищення продуктивності в системах управління базами даних.

Кожен сліпий індекс (аналог розглянутого вище зашифрованого індексу I) у кожному стовпці використовує ключ, відмінний від ключа шифрування та кожного іншого ключа сліпого індексу. Це не дозволяє використовувати оператори LIKE або пошук за звичайними виразами, але дозволяє індексувати перетворення (наприклад, підрядки) відкритого тексту, гешованого за допомогою окремого ключа.

Ідея техніки сліпого індексування, що використовується в CipherSweet, є досить простою. Розглянемо деяку таблицю A (табл. 2) бази даних з атрибутами: *id, field_1, field_2,*

де містяться дані (*data_1, data_2, data_3, data_4, data_5*). Для пошуку відповідних даних по атрибуту *field_1* можна використати, наприклад, наступний простий запит:

```
SELECT * FROM table WHERE field_1 = data_1,
```

результатом якого буде два картежі з *id* рівним 1 та 2.

Таблиця 2

Таблиця *A*

id	***field_1***	***field_2***
1	data_1	data_2
2	data_1	data_3
3	data_4	data_5

Однак при зашифруванні даних сервер не має змоги порівняти *field_1=data_1*. У зв'язку з чим виникає необхідність модифікації вихідної таблиці *A*. А саме потрібно введення додаткового атрибуту – індексу *field_1_index*, в якому будуть міститися так звані метадані, пов'язані з даними атрибуту *field_1* (табл. 3).

Таблиця 3

Таблиця A_1

id	***field_1_index***	***field_1***	***field_2***
1	index_1	data_1	data_2
2	index_1	data_1	data_3
3	index_4	data_4	data_5

Тепер можна зашифровувати відповідні дані стовпця *field_1*, використовуючи певний криптостійкий шифр. CipherSweet використовує автентифіковане шифрування із симетричним ключем та випадковим вектором ініціалізації (IV).

Наприклад, компонент CipherSweet FIPSCrypto надає безпечний інтерфейс AEAD (Authenticated Encryption with Associated Data – автентифіковане шифрування зі зв'язаними даними – клас блокових режимів шифрування, при якому частина повідомлення шифрується, частина залишається відкритою, і все повідомлення повністю автентифіковано), використовуючи FIPS 140-2 (та інші додаткові документи FIPS). Алгоритми FIPSCrypto забезпечують полегшене шифрування даних за допомогою алгоритму AES-256 у режимі лічильника (CTR). Зашифровані тексти автентифікуються за допомогою HMAC-SHA384. IV (вектори ініціалізації) / нонси (nonce – одноразовий номер) генеруються за допомогою стійкого криптографічно генератора псевдовипадкових чисел. Для розділення ключів використовується HKDF-HMAC-SHA384.

При цьому, щоб можна було використовувати операцію порівняння, необхідно виконати ще певні дії. А саме, у відповідні поля атрибуту *field_1_index* необхідно записати результат застосування до даних криптографічно стійкої псевдовипадкової функції (pseudorandom function family – PRF; наприклад, PBKDF2 або Argon2).

Для визначеності позначимо операцію шифрування за допомогою криптостійкого шифру, як Enc/Dec (зашифрування та розшифрування відповідно) та операцію обчислення псевдовипадкової функції PRF. Тоді результат перетворених даних на недовіреному сервері можна подати так (табл. 4).

Таблиця 4

Вміст таблиці A_1 після перетворень

id	***field_1_index***	***field_1***	***field_2***
1	index_1=PRF(data_1)	Enc(data_1)	data_2
2	index_1=PRF(data_1)	Enc(data_1)	data_3
3	index_4=PRF(data_4)	Enc(data_4)	data_5

Для реалізації описаного принципу потрібно: модифікувати запити для роботи з даними та інтегрувати три криптографічні функції Enc, Dec та PRF у логіку програми. При цьому слід зазначити, що алгоритми шифрування та псевдовипадкової функції використовують різні ключі. CipherSweet використовує ряд методів розширення ключів, щоб перетворити один ключ, який обробляється об'єктом Постачальник ключів і може бути отриманий зі сторонніх служб керування ключами, в:

– один окремий ключ шифрування для кожного зашифрованого поля у кожній таблиці;

– множина різних ключів для обчислення сліпих індексів, по одному для кожного індексу у кожному зашифрованому полі кожної таблиці.

На рис. 7 схематично представлено процес розширення ключів.

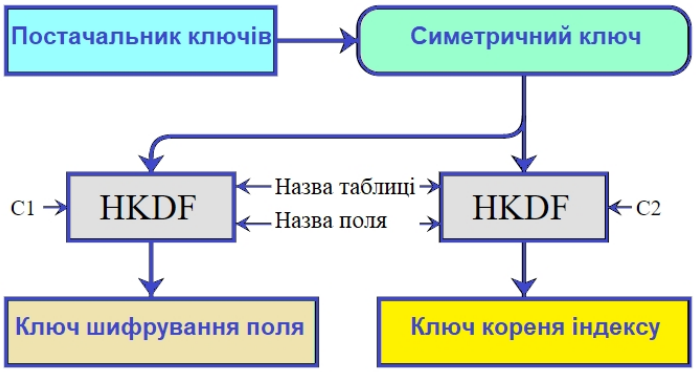

Рис. 7. Процес розширення ключів у CipherSweet

HKDF (HMAC (Hash-based Message Authentication Code) Key Derivation Function) – це проста функція формування ключів (KDF), заснована на коді автентифікації повідомлень HMAC; C1 і C2 – константи, які були обрані таким чином, щоб між ними була відстань Хеммінга 32*4=128b (C1 – це байт `0xB4` повторений 32 рази; C2 – байт `0x7E` повторений 32 рази); Ключ шифрування поля – секретний ключ шифрування даних відповідного атрибуту таблиці; Ключ кореня індексу – кореневий ключ для кожного сліпого індексу. Відповідний ключ кожного індексу обчислюється шляхом використання HMAC-SHA256 імені таблиці, імені атрибута / поля та імені індексу як вихідні дані.

Таким чином, наприклад, запит INSERT може виглядати так:

```
INSERT INTO A1 (id, field_1_index, field_1, field_2)
VALUES (1, PRF(data_1), Enc(data_1), data_2).
```

Запит SELECT для вибірки зашифрованих у стовпці *field_1* даних у цьому випадку перетворюється на такий вид:

```
SELECT * FROM table WHERE field_1_index=PRF(data_1).
```

Після отримання результату запиту програма може скористатися функцією Dec і розшифрувати дані атрибута *field_1* із записів з *id* 1 і 2.

При цьому у контексті безпеки слід відзначити важливу особливість. Результат застосування PRF буде завжди тим самим (значення індексів *field_1_index*) для двох однакових значень даних (табл. 5). Це потенційний витік, тому що сервер на основі аналізу зашифрованих даних та їх індексів, до яких він потенційно має доступ, може визначити, які записи в таблиці містять однакові значення зашифрованих даних за атрибутами, що підтримують пошук. Щоб зменшити негативні наслідки цього витоку, CipherSweet пропонує зберігати на сервері усічене значення індексів (табл. 6).

Таблиця 5

Вміст таблиці A_1 до усічення значення індексів

id	***field_1_index***	***field_1***	***field_2***
1	index_1[64]	Enc(data_1)	data_2
2	index_1[64]	Enc(data_1)	data_3
3	index_4[64]	Enc(data_4)	data_5

Де index_*i*=PRF(data_*i*).

Таблиця 6

Вміст таблиці A_1 після усічення значення індексів

id	***field_1_index***	***field_1***	***field_2***
1	index_1[<64]	Enc(data_1)	data_2
2	index_1[<64]	Enc(data_1)	data_3
3	index_4[<64]	Enc(data_4)	data_5

Припустимо, що псевдовипадкова функція генерує індекси розміром 64 байти. Тоді шляхом запису в таблицю A_1 БД усіченого значення індексу (наприклад, розмірністю менше 64 байт), можна знизити наслідки потенційного витоку схеми, тим самим підвищить безпеку. Однак сервер не зможе однозначно визначати записи, які необхідно включати до результату запиту. Тобто зростає можливість включення до результату запиту зайвих записів. У цьому випадку на застосунок покладається відповідальність фільтрації всіх нерелевантних записів після отримання результату запиту від сервера та розшифрування даних.

Параметр усічення (один із параметрів безпеки) може конфігуруватися користувачем. CipherSweet надає спеціальний планувальник для його розрахунку. Для використання планувальника необхідно оцінити очікувану кількість записів, що включають атрибути із зашифрованими даними, та розподілити дані, що підлягають захисту [24].

Підхід сліпого усічення індексу є одним з найбільш ефективних методів пом'якшення наслідків атаки витоку, оскільки він приховує шаблон пошуку регульованим чином за рахунок точності запиту. Коли сліпий індекс усікається до певної кількості біт, його можна розглядати як фільтр Блуму для пошуку в базі даних.

Основним параметром безпеки сліпих індексів є верхня межа витоку відкритого тексту (середня кількість рядків, що повертаються), яку можна виразити наступним чином [10]:

$$C = R / 2^{-S} \tag{1}$$

де $S = \sum_{i=0}^{n} \min(L_i, K_i)$; n – кількість сліпих індексів; L_i – довжина сліпого індексу (у бітах); K_i – довжина ключа (у бітах); R – низка зашифрованих записів, які використовують ці сліпі індекси. Зазвичай рекомендується підбирати параметри таким чином, щоб $2 \le C < \sqrt{R}$. Якщо $C < 2$, то зловмисник зможе зробити висновки, що деякі відкриті тексти ідентичні, що порушує стандартне поняття безпечної схеми. В іншому випадку, якщо $C > \sqrt{R}$, то буде мати місце занадто багато колізій, що сильно вплине на продуктивність (необхідна продуктивність не буде досягнута).

Насправді безпечною верхньою межею витоку відкритого тексту є максимальна кількість очікуваних збігів. У загальному випадку число n та довжину кожного індексу L_i слід мінімізувати. Чим більше індексів створено, тим більше впевненості набуває зловмисник. У той же час більші індекси корисніші, ніж більш короткі індекси.

Незважаючи на певні переваги та практичну реалізацію можливості здійснення пошуку за зашифрованими даними, CipherSweet підтримує мінімальну функціональність запитів (тільки на рівність – equality) і має відносно низьку продуктивність, хоча й надає високий рівень безпеки.

6. Acra

Acra – це набір інструментів забезпечення безпеки даних протягом усього їх життєвого циклу в сучасних розподілених системах [25]. Acra забезпечує шифрування на рівні застосунків, маскування, токенізацію, контроль доступу, запобігання витоку з бази даних та можливості виявлення вторгнень у зручному, дружньому для розробників пакеті.

Основні функції безпеки, що надаються Acra:

– Криптографічна безпека: дані шифруються під час зберігання та передачі за допомогою шифрування на рівні застосунків та на транспортному рівні з надійною взаємною автентифікацією.

– Шифрування із можливістю пошуку.

– Вибіркове шифрування: є можливість вибору, що, де і як потрібно зашифрувати.

– Маскування даних та токенізація: анонімізація або псевдонімізація даних із збереженням вихідного формату.

– Інструменти управління ключами: гнучке управління переносом / ротацією / відкликанням відповідно до наявних потреб у навантаженні та архітектурою даних.

– Брандмауер запитів SQL: запобігає SQL ін'єкції, блокує несанкціоновані та підозрілі запити.

– Система виявлення вторгнень: виявляє виток даних за допомогою шкідливих записів та інтеграції з SIEM (Security Information and Event Management).

– Моніторинг та реєстрація операцій: є можливість відстеження виконуваних операцій та подій, що контролюють дії з даними.

– Захищене від несанкціонованого доступу ведення журналу аудиту.

– Політики: виразна мова політик, що дозволяє настроювати поведінку Acra у великих інфраструктурах.

Компоненти забезпечення безпеки

– AcraServer також відомий як SQL-проксі, який працює як прозорий проксі-сервер зашифрування / розшифрування з базами даних SQL. Програма не знає, що дані зашифровані до того, як вони потраплять до бази даних, база даних також не знає, що хтось зашифрував дані. AcraServer підтримує шифрування, шифрування з можливістю пошуку, маскування, токенізацію, брандмауер SQL, ведення журналу та ведення журналу аудиту.

– AcraTranslator, також відомий як служба API. Це сервер API, який надає більшість функцій Acra у вигляді HTTP/gRPC API з клієнтськими SDK (software development kit) та захистом трафіку. AcraTranslator не залежить від бази даних і покладає на застосунок відповідальність за фактичне розміщення даних у сховищі. Він підтримує шифрування, створення гешів з можливістю пошуку, маскування, токенізацію, ведення журналу.

– AnyProxy. Це сервер API, який працює між кількома мікросервісами / застосунками, керованими API.

– Клієнтські SDK. Acra надає додаткові SDK для шифрування даних (AcraWriter), для розшифрування даних (AcraReader) або для роботи з AcraTranslator.

Компоненти Acra сумісні з численними реляційними СУБД (MySQL v5.7+, PosgtreSQL v9.4-v11, MariaDB v10.3; Google Cloud SQL, Amazon RDS), сховищами об'єктів та ключ-значення (key-value – KV), хмарними платформами, зовнішніми системами управління ключами (key management systems – KMS), системами балансування навантаження.

Далі детальніше розглянемо можливості Acra у контексті шифрування з можливістю пошуку, заснованого на підході «сліпого індексування», розробленого в рамках проєкту CipherSweet.

На відміну від існуючих рішень, дана система забезпечує суворий поділ обов'язків, що гарантує відсутність витоку криптографічних ключів із застосунку, безпечне зберігання та управління криптографічними ключами, а також набір додаткових функцій безпеки, які відповідають реальним загрозам. На рис. 8 представлений принцип проксі-опосередкованого пошуку зашифрованих даних зі сліпим індексуванням.

Основним компонентом схеми Acra SE є так званий AcraServer, який працює як реверсивний (reverse) проксі (прозорий проксі-сервер шифрування / розшифрування). Він знаходиться між застосунком та базою даних. Застосунок не знає, що дані зашифровані до того, як вони потраплять до бази даних, база даних також не знає, що хтось зашифрував дані (тому цей режим часто називають режимом прозорого шифрування).

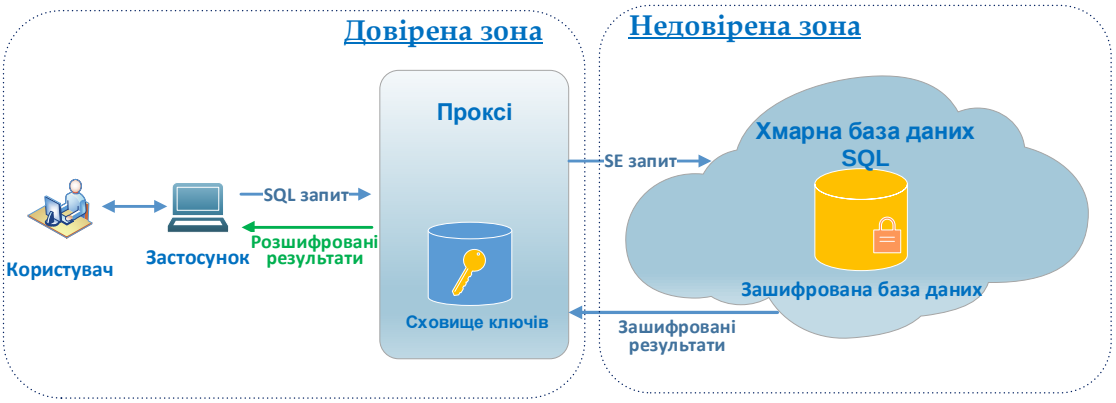

Рис. 8. Використання проксі-сервера між програмою
та ненадійним постачальником сховища

Криптографічна схема обчислення сліпого індексу є алгоритм T, на вхід якого надходять вихідні дані R і симетричний ключ K, на виході – рядок, що повертається: $UT_R = T(R, K)$. До кращих кандидатів для схеми розрахунку сліпого індексу належать алгоритми сімейства HMAC [26, 27].

На рис. 9 показані типові потоки даних (включаючи конфіденційні дані) з різних застосунків (Application 1, 2, …, N) до бази даних і навпаки. Застосунок видає запит, AcraServer виконує всі криптографічні операції (за потреби) і надсилає запит (який може бути змінено) далі до бази даних. База даних обробляє запит та надсилає результат назад. AcraServer отримує результат з бази даних, виконує (за потреби) криптографічні операції та відправляє результат (який також може бути змінений) далі у застосунок.

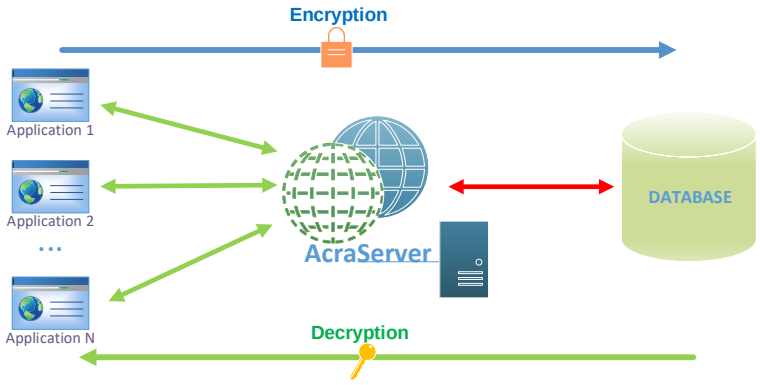

Рис. 9. Потоки даних

Слід зазначити, що один екземпляр AcraServer може працювати з кількома застосунками, але тільки з однією базою даних. AcraServer підтримує два режими роботи:

– стандартний;

– прозорий.

У стандартному режимі шифрування (створення AcraStructs) виконується на стороні застосунку. AcraStructs – це багатоцільовий криптографічний контейнер, у якому зберігаються зашифровані дані (і пов'язані з ними метадані) у певному форматі. AcraStruct може зберігатися як двійковий запис у таблиці бази даних або як великий двійковий об'єкт у файловій системі. Детальніше AcraStruct AS можна представити у вигляді наступної послідовності (масиву байтів) змінної довжини:

$$AS = Tag \,||\, PK_{rand} \,||\, K_{rand}^{PK_A} \,||\, len \,||\, Ciphertext \tag{2}$$

де Tag – спеціальний тег, що відзначає початок AcraStruct; PK_{rand} – відкритий ключ із випадково згенерованої пари ключів ECDH (Elliptic-Curve Diffie–Hellman): (PK_{rand}, SK_{rand}); $K_{rand}^{PK_A} = W(PK_A, SK_A)$ – випадковий симетричний ключ, PK_A – відкритий ключ із пари ключів ECDH (PK_{rand}, SK_{rand}), що належить клієнту A; len – представлення з прямим порядком слідування байтів (масив байтів довжиною 8) цілочисленної змінної, що визначає довжину зашифрованих даних; $Ciphertext = E_{K_{rand}}(R)$ – дані R, зашифровані за схемою AEAD.

У прозорому режимі шифрування виконується на AcraServer. Це дозволяє легко інтегрувати Acra SE в існуючу інфраструктуру без зміни вихідного коду застосунку.

Слід зазначити, що функції шифрування (і безпечного пошуку) AcraServer можна налаштувати кожного стовпця. Це означає, що кожна таблиця в базі даних може бути повністю зашифрована (кожний стовпець), частково (деякі стовпці зашифровані, деякі ні) або повністю не зашифрована.

Безпечний пошук вимагає запровадження двох процедур:

– безпечного завантаження (це модифікація типових запитів INSERT та UPDATE – застосування алгоритму SUpload) зашифрованих даних з індексами у ненадійне сховище;

– безпечного вилучення (це модифікація типового запиту SELECT – застосування алгоритму SSelect) даних з сховища.

Всі властивості безпеки шифрування Acra з можливістю пошуку дуже схожі на властивості безпеки CipherSweet, що створює ризик атак із частково відомим відкритим текстом. У зв'язку з чим автори роботи [10] дають практичні рекомендації для безпеки при використанні інструменту шифрування з можливістю пошуку Acra SE, а саме:

– не створювати сліпі індекси для надзвичайно конфіденційних даних (дані, які не повинні бути розкриті за будь-яку ціну);

– створювати тільки мінімальну кількість сліпих індексів (чим більше індексів – тим більше витоків метаданих);

– якщо дані, які необхідно проіндексувати, надзвичайно чутливі і мають дуже низьку ентропію, щоб їх можна було безпечно помістити в сліпий індекс, їх можна гешувати разом з деякими іншими даними (використання складового індексу);

– сліпий індекс доцільно перетворити до усіченого виду, щоб зменшити витік інформації за рахунок збільшення ймовірності колізій, що призведе до «неправильних» результатів виконання операції SELECT, які мають бути відфільтровані після розшифрування (рекомендується, щоб для будь-якого заданого значення вираз $2 \le C < \sqrt{R}$ завжди залишався істинним (див. вираз (1));

– повинен бути включений безпечний та автентифікований зв'язок між програмою та AcraServer

– та деякі інші.

Однак незважаючи на певні рішення, спрямовані на забезпечення безпеки зберігання та пошуку конфіденційних даних, Acra, як і CipherSweet, яка була взята як прототип схеми шифрування з можливістю пошуку, підтримує мінімальну функціональність запитів, а саме тільки на рівність.

Підбиваючи підсумки, наведемо в табл. 7 деякі узагальнені характеристики [2, 10] деяких систем SE, які мають реалізації та підтримують реляційні бази даних.

Доцільно також відзначити, що CryptDB дозволяє виконати більшість функцій СУБД із втратою продуктивності (накладними витратами) менше ніж 30 % [12]. Для Blind Seer [28] втрата продуктивності більшості запитів становить від 20 до 300 %. У SisoSPIR [33] повідомляється про уповільнення продуктивності на 500 % порівняно з базовою системою MySQL під час перевірки рівності ключових слів (keyword equality) та запитів діапазону (range). Acra SE має уповільнення на два порядки порівняно із звичайним виконанням операцій у PostgreSQL [10]. Найбільший вплив мають криптографічні операції (шифрування в SUpload і розшифрування в SSelect). Крім того, результати запитів SELECT можуть містити нерелевантні рядки при малих розмірах сліпого індексу (1-2 байти), що може призвести до подальшого зниження продуктивності.

Таблиця 7

Порівняльна характеристика деяких систем SE

Система	**Підтримувані операції**	**Додаткові можливості**	**Наявність відкритого вихідного коду**
CryptDB [12]	Equality, Boolean, Range, Sum, Join, Update	Автентифікація користувача, контроль доступу	Так
Blind Seer [28, 29]	Equality, Boolean, Keyword, Range, Update	Політика запитів	Ні
OSPIR-OXT [30, 31, 32]	Equality, Boolean, Keyword, Range, Substring, Wildcard, Update	Політика запитів	Ні
SisoSPIR [33]	Equality, Keyword, Range, Substring	Політика запитів	Ні
CipherSweet [22]	Equality	Управління ключами	Так
Acra [10]	Equality	Автентифікація, політика запитів, виявлення вторгнень, управління ключами, моніторинг та спостереження	Acra: Так Acra SE: Ні

Крім того, слід пам'ятати, що кожній з наведених вище схем властиві різні типи витоку даних. Наприклад, CryptDB є найшвидшим і найпростішим у розгортанні. Однак, як тільки якийсь стовпець БД використовується у запиті, CryptDB розкриває статистику по всьому набору даних для цього стовпця. Blind Seer та OSPIR-OXT також передають інформацію на сервер, але в основному про дані, що повертаються запитом. Таким чином, вони підходять для установок, де запитується невелика частина бази даних.

SisoSPIR підходить, якщо регулярно запитується велика частина даних. Однак SisoSPIR не підтримує логічні запити, що є обмеженням. CipherSweet підтримує лише запити на рівність, але пропонує просту та зрозумілу модель безпеки, яка, по суті, є компромісом між часом та пам'яттю, що створює ризик атак із частково відомим відкритим текстом. Acra заснована на шифруванні CipherSweet з можливістю пошуку, адаптованим до схеми з використанням проксі. Поряд з безпечним пошуком, в Acra забезпечується суворий поділ обов'язків, що гарантує відсутність витоку криптографічного ключа із застосунку, належне управління ключами та додаткові функції безпеки, що відповідають реальним загрозам для конфіденційних даних, що зберігаються у зовнішньому сховищі.

Висновки

1. Технологія захищених баз даних відкриває нові можливості у використанні хмарних сховищ, оскільки вселяє впевненість у власника даних у безпеці його збережених даних, у тому числі за рахунок використання можливостей захищеного пошуку. Захищені системи пошуку криптографічно ізолюють ролі читання, запису та адміністрування бази даних. Цей поділ обмежує непотрібний доступ адміністратора та захищає дані у разі злому системи.

2. Сьогодні пошук у захищених базах даних досяг переломного моменту у своїй зрілості. Однак, незважаючи на велику кількість існуючих академічних досліджень і практичних схем, в даний час немає домінуючого рішення для всіх випадків використання, не існує найбільш захищеної пошукової системи або набору методів. Користувачі повинні розуміти характеристики системи та компроміси для свого варіанта використання. Проєктування таких систем є балансом між безпекою, функціональністю, продуктивністю і зручністю використання. Коли схема покращується в одному аспекті, зазвичай доводиться жертвувати іншими. Ця задача ускладнюється постійною спеціалізацією баз даних, оскільки деяким користувачам потрібні функціональні можливості баз даних SQL, NoSQL або NewSQL. Ця еволюція баз даних продовжуватиметься, і спільнота захищеного пошуку повинна мати можливість швидко надавати функціональні можливості, сумісні з усіма типами баз, сховищ даних.

3. У всіх досить ефективних систем шифрування з можливістю пошуку є загальна проблема – вони пропускають шаблон пошуку, який показує, чи було виконано два пошукові запити по тому самому ключовому слову чи ні. Отже, шаблон пошуку надає інформацію щодо частоти появи кожного запиту. І ця інформація може бути додатково використана за допомогою статистичного аналізу, що дозволяє зловмиснику отримати повну інформацію про ключові слова відкритого тексту, що значно знижує переваги безпеки шифрування даних.

4. Результат аналізу розглянутих систем та практичних рішень показав, що не завжди запропоновані методи шифрування однаково придатні. Більше того, незважаючи на гнучкий вибір шифрування, кількість типів SQL запитів залишається обмеженою.

5. Для здійснення процедур шифрування та розшифрування даних, планування порядку виконання запитів, попередньої обробки даних в архітектуру системи обробки вводяться додаткові компоненти (довірені проксі-сервери, планувальники тощо), що призводить до ускладнення системи, збільшення обсягів необхідної пам'яті та збільшення часу виконання запитів. Усе це змушує проводити подальші дослідження альтернативних підходів задля забезпечення безпечної роботи з віддаленими базами, сховищами даних.

Список літератури;

1. Abadi D., Ailamaki A., Andersen D., Bailis P., Balazinska M., Bernstein P., Boncz P., Chaudhuri S., et al. The Seattle Report on Database Research // ACM SIGMOD Record. 2019. 48. P. 44–53.

2. Fuller B., Varia M., Yerukhimovich A., Shen E., Hamlin A., Gadepally V., Shay R., Mitchell J. D., Cunningham R. K. Sok: Cryptographically protected database search // 2017 IEEE Symposium on Security and Privacy (SP), 2017. P. 172–191. https://doi.org/10.1109/SP.2017.10.

3. General Data Protection Regulation GDPR. URL: https://gdpr-info.eu/ (дата звернення: 12.06.2022).

4. Payment Card Industry (PCI) Data Security Standard. Requirements and Testing Procedures Version 4.0. 2022. URL: https://www.pcisecuritystandards.org/documents/PCI-DSS-v4_0.pdf (дата звернення: 12.06.2022).

5. Atchinson B. K., Fox D. M. From the field: the politics of the health insurance portability and accountability act. Health affairs. 1997. 16(3). P. 146-150.

6. Scholl M., Stine K., Hash J., Bowen P., Johnson A., et al. NIST Special Publication 800-66 Revision 1. An Introductory Resource Guide for Implementing the Health Insurance Portability and Accountability Act (HIPAA) Securi-

ty Rule. 2008. URL: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-66r1.pdf (дата звернення: 12.06.2022).

7. Bösch, C., Hartel, P., Jonker, W., Peter, A. A survey of provably secure searchable encryption. ACM Computing Surveys (CSUR). 2014. 47(2). P. 1–51.

8. Єсін В. І., Вілігура В. В. Дослідження основних методів і схем шифрування з можливістю пошуку // Радіотехніка. 2022. № 209. С. 138–155.

9. Azraoui M., Önen M., Molva R. Framework for Searchable Encryption with SQL Databases. CLOSER. 2018. P. 57–67.

10.Pilyankevich E., Kornieiev D., Storozhuk A. Proxy-Mediated Searchable Encryption in SQL Databases Using Blind Indexes. Cryptology ePrint Archive. 2019.

11.Hacigümüş H., Iyer B., Li C., Mehrotra S. Executing SQL over encrypted data in the database-service-provider model // Proceedings of the 2002 ACM SIGMOD international conference on Management of data. 2002. P. 216–227. https://doi.org/10.1145/564691.564717.

12.Popa R. A., Redfield C. M., Zeldovich N., Balakrishnan H. CryptDB: protecting confidentiality with encrypted query processing // Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles. SOSP '11. 2011. P. 85–100. https://doi.org/10.1145/2043556.2043566.

13.Paillier P. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes // Stern, J. (eds) Advances in Cryptology - EUROCRYPT ’99. EUROCRYPT 1999. Lecture Notes in Computer Science, 1999. Vol 1592. P. 223–238. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-48910-X_16.

14.Song D. X., Wagner D., Perrig A. Practical techniques for searches on encrypted data // Proceeding 2000 IEEE symposium on security and privacy. S&P 2000. IEEE, 2000. P. 44–55. https://doi.org/10.1109/SECPRI.2000.848445.

15.Tu S. L., Kaashoek M. F., Madden S. R., Zeldovich N. Processing analytical queries over encrypted data // Proceedings of the VLDB Endowment. 2013. 6(5). P. 289–300. https://doi.org/10.14778/2535573.2488336.

16.Halevi S., Rogaway P. A Tweakable Enciphering Mode // Boneh, D. (eds) Advances in Cryptology - CRYPTO 2003. CRYPTO 2003. Lecture Notes in Computer Science, 2003. Vol 2729. P. 482–499. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-45146-4_28.

17.Bellare M., Rogaway P., Spies T. Addendum . The FFX mode of operation for format-preserving encryption // A parameter collection for enciphering strings of arbitrary radix and length, Draft 1.0, NIST. 2010. URL: https://csrc.nist.gov/CSRC/media/Projects/Block-Cipher-Techniques/documents/BCM/proposed-modes/ffx/ffx-spec2.pdf.

18.Boldyreva A., Chenette N., Lee Y., O’Neill A. Order-Preserving Symmetric Encryption // Joux, A. (eds) Advances in Cryptology - EUROCRYPT 2009. EUROCRYPT 2009. Lecture Notes in Computer Science, 2009. Vol 5479. P. 224–241. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-01001-9_13.

19.Boldyreva, A., Chenette, N., O’Neill, A. Order-Preserving Encryption Revisited: Improved Security Analysis and Alternative Solutions. In: Rogaway, P. (eds) // Advances in Cryptology – CRYPTO 2011. CRYPTO 2011. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg. 2011. Vol. 6841. P. 578–595. https://doi.org/10.1007/978-3-642-22792-9_33.

20.Papadimitriou A., Bhagwan R., Chandran N., Ramjee R., Haeberlen A., Singh H., Modi A., Badrinarayanan S. Big data analytics over encrypted datasets with seabed // 12th USENIX symposium on operating systems design and implementation (OSDI 16). 2016. P. 587–602.

21.Poddar R., Boelter T., Popa R. A. Arx: an encrypted database using semantically secure encryption // Proceedings of the VLDB Endowment. 12(11). 2019. P. 1664–1678. https://doi.org/10.14778/3342263.3342641.

22.CipherSweet. URL: https://ciphersweet.paragonie.com/ (дата звернення: 12.06.2022).

23.Tarkoma S., Rothenberg C. E., Lagerspetz E. Theory and practice of bloom filters for distributed systems // IEEE Communications Surveys & Tutorials. 2011. 14(1). P. 131–155.

24.Blind Index Planning. URL: https://ciphersweet.paragonie.com/node.js/blind-index-planning. (дата звернення: 12.06.2022).

25.Cossack Labs Knowledge Base. Acra in a nutshell. URL: https://docs.cossacklabs.com/acra/ (дата звернення: 12.06.2022).

26.Bellare M., Canetti R., Krawczyk H. Keying Hash Functions for Message Authentication. In: Koblitz, N. (eds) // Advances in Cryptology - CRYPTO ’96. CRYPTO 1996. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg. 1996. Vol 1109. P. 1–15. https://doi.org/10.1007/3-540-68697-5_1.

27.Turner J. M. The keyed-hash message authentication code (HMAC) // Federal Information Processing Standards Publication. 2008. 198(1). P. 1–13.

28.Pappas V., Krell F., Vo B., Kolesnikov V., Malkin T., Choi S. G., Bellovin S. Blind seer: A scalable private DBMS // 2014 IEEE Symposium on Security and Privacy. IEEE, 2014. P. 359-374.

29.Fisch B. A., Vo B., Krell F., Kumarasubramanian A., Kolesnikov V., Malkin T., Bellovin S. M. Malicious-client security in blind seer: a scalable private DBMS // 2015 IEEE Symposium on Security and Privacy. 2015. P. 395–410.

30.Cash D., Jarecki S., Jutla C., Krawczyk H., Roşu MC., Steiner M. Highly-Scalable Searchable Symmetric Encryption with Support for Boolean Queries. In: Canetti, R., Garay, J.A. (eds) // Advances in Cryptology – CRYPTO

2013. CRYPTO 2013. Lecture Notes in Computer Science. 2013. Vol. 8042. P. 353–373. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-40041-4_20.

31.Jarecki S., Jutla C., Krawczyk H., Rosu M., Steiner M. Outsourced symmetric private information retrieval // Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security. 2013. P. 875–888.

32.Faber S., Jarecki S., Krawczyk H., Nguyen Q., Rosu M., Steiner M. Rich Queries on Encrypted Data: Beyond Exact Matches. In: Pernul, G., Y A Ryan, P., Weippl, E. (eds) Computer Security – ESORICS 2015. ESORICS 2015. Lecture Notes in Computer Science. 2015. Vol 9327. P. 123–145. Springer, Cham. https://doi.org/10.1007/978-3-319-24177-7_7.

33.Ishai Y., Kushilevitz E., Lu S., Ostrovsky R. Private Large-Scale Databases with Distributed Searchable Symmetric Encryption // Sako, K. (eds) Topics in Cryptology - CT-RSA 2016. CT-RSA 2016. Lecture Notes in Computer Science. 2016. Vol. 9610. P. 90–107. Springer, Cham. https://doi.org/10.1007/978-3-319-29485-8_6.

Надійшла до редколегії 15.09.2022

Відомості про авторів:

Есін Віталій Іванович – д-р техн. наук, професор кафедри безпеки інформаційних систем і технологій, факультет комп'ютерних наук; Харківський національний університет імені В.Н. Каразіна, Україна; e-mail: v.i.yesin@karazin.ua; ORCID: https://orcid.org/0000-0003-1977-7269

Вілігура Владислав Вікторович – аспірант кафедри безпеки інформаційних систем і технологій, факультет комп'ютерних наук; Харківський національний університет імені В.Н. Каразіна, Україна; e-mail: viligura93@gmail.com; ORCID: https://orcid.org/0000-0002-1137-2382