DOI: 10.15514/ISPRAS-2025-37(3)-3

CC BY

Оптимизация конфиденциальных запросов к базе данных на облаке

[1] *Н.П. Варновский, ORCID: 0000-0002-2363-0254 <otd13isp@gmail.com>*
[2] *С.А. Мартишин, ORCID: 0000-0001-5437-4049 <mart@ispras.ru>*
[2] *М.В. Храпченко, ORCID: 0000-0002-5147-5132 <khrap@ispras.ru>*
[2] *А.В. Шокуров, ORCID: 0000-0002-6801-7728 <shok@ispras.ru>*

[1] *Институт проблем информационной безопасности МГУ имени М.В. Ломоносова. 119192, г. Москва, Мичуринский проспект, 1, офис 10.*
[2] *Институт системного программирования РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25.*

Аннотация. Авторами рассматривается задача PIR (Private Information Retrieval) обеспечения безопасных запросов к базе данных, размещенной на облаке при наличии активного противника, который не вмешивается в выполнение протокола, но может производить атаку с известными открытыми запросами. Для представления номера бита i в виде числа, все цифры которого различны, в предложенных алгоритмах применяется система счисления по основанию l с числом разрядов d. Использование перестановок цифр номера запрашиваемого бита в качестве секретных ключей шифрования уменьшает вероятность угадывания номера бита. Хранящиеся на облаке биты исходной базы данных группируются в виде массивов, что позволяет снизить коммуникационную сложность. Используется датчик псевдослучайных чисел для замены значения бита в зависимости от запрашиваемого клиентом номера бита i. Это позволяет в случае сговора пассивного противника, находящегося на облаке, и активного противника вне облака, затруднить сопоставление значения бита конкретному номеру. Приведены оценки коммуникационной сложности, вероятности угадывания номера бита при однократной атаке с известным открытым запросом номера бита i и при атаке с неограниченным числом известных открытых запросов.

Ключевые слова: база данных; облачные вычисления; задача безопасных запросов PIR.

Для цитирования: Варновский Н.П., Мартишин С.А., Храпченко М.В., Шокуров А.В. Оптимизация конфиденциальных запросов к базе данных на облаке. Труды ИСП РАН, том 37, вып. 3, 2025 г., стр. 39– 58. DOI: 10.15514/ISPRAS–2025–37(3)–3.

Благодарности: Авторы выражают особую благодарность Лазареву Д.О. за ценные замечания в процессе работы над статьей.

Optimizing Confidential Database Queries on the Cloud

[1] *N.P. Varnovskiy, ORCID: 0000-0002-2363-0254 <otd13isp@gmail.com>*
[2] *S.A. Martishin, ORCID: 0000-0001-5437-4049 <mart@ispras.ru>*
[2] *M.V. Khrapchenko, ORCID: 0000-0002-5147-5132 <khrap@ispras.ru>*
[2] *A.V. Shokurov, ORCID: 0000-0002-6801-7728 <shok@ispras.ru>*
[1] *Information Security Institute of Moscow State Lomonosov University, Office 10, 1, Michurinskiy prospect, 119192, Moscow, RUSSIA.*
[2] *Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.*

Abstract. The authors consider the PIR (Private Information Retrieval) problem to ensure secure requests to a database hosted on the cloud in the presence of an active adversary who does not interfere with the PIR protocol, but can carry out an attack with known open requests. To represent the bit number *i* as a number, all digits of which are different, the proposed algorithms use a base *l* number system with the number of digits *d*. The permutations of the digits of the requested bit number which are regarded as secret encryption keys were used. To reduce the communication complexity the bits of the source database stored on the cloud are grouped as arrays. A pseudorandom number sensor is used to replace the bit value depending on the number *i* requested by the client. This makes it difficult to match the bit value to a specific number in the case of collusion between a passive adversary located on the cloud and an active adversary outside the cloud. The communication complexity and the probability of guessing the bit number in a single attack with a known open query for bit number *i*, as well as in an attack with an unlimited number of known open queries, are estimated.

Keywords: database; cloud computing; PIR.

For citation: Varnovskiy N.P., Martishin S.A., Khrapchenko M.V., Shokurov A.V. Optimizing confidential database queries on the cloud. Trudy ISP RAN/Proc. ISP RAS, vol. 37, issue 3, 2025. pp. 39-58 (in Russian). DOI: 10.15514/ISPRAS-2025-37(3)-3.

Acknowledgements. The authors are especially grateful to D.O. Lazarev for valuable comments during the work on the article.

1. Введение

В настоящее время при использовании облачных сервисов для хранения и обработки информации наиболее актуальным является вопрос работы с конфиденциальными данными. В работах [1,2] авторы исследовали задачу организации конфиденциальных запросов к базе данных (PIR – Private Information Retrieval) в случае размещения копий базы данных на облаке и наличия активного противника, работающего по протоколу, но имеющему возможность производить атаку с известными открытыми запросами. В данной статье под облаком будем понимать сеть удаленных серверов, администрирование которых осуществляется только их владельцами и не доступно для других.

Классическая постановка задачи PIR была сформулирована в 1995 году Шором, Голдрайхом, Кушелевицем и Суданом [3]: имеется база данных – бинарная строка $X = (x_1, \dots, x_n)$ длины n, хранящаяся на сервере; клиент хочет получить один бит информации x_i из базы данных X так, чтобы никто, кроме клиента, обращающегося с запросом, не смог определить, для какого индекса i был запрошен бит.

Чтобы PIR-протокол можно было бы применять на практике, он должен удовлетворять условиям:

корректности: для любого номера клиент может сформировать такой запрос, что после завершения выполнения протокола клиент получит данные, по которым он может правильно вычислить значение бита;

конфиденциальности: в результате выполнения протокола PIR противник не получает никакой информации о номере запрашиваемого бита из базы данных;

лаконичности: коммуникационная сложность протокола (общее количество бит, которыми обмениваются участники протокола за время его работы) должна быть существенно меньше размера базы данных.

Было показано, что при размещении базы данных на единственном сервере, который полностью контролируется противником, теоретико-информационное условие конфиденциальности корректного протокола PIR может быть выполнено только в том случае, когда клиент запрашивает базу данных целиком. Однако при этом коммуникационная сложность протокола будет не меньше размера базы данных.

Для уменьшения коммуникационной сложности в работах [3, 4] была предложена модель, в которой несколько одинаковых копий строки $X = (x_1, \dots, x_n)$ размещалось на разных серверах. Предполагалось, что клиент имеет связь с каждым из этих серверов, но сами серверы не имеют связи друг с другом. Также предполагалось, что противник может наблюдать любой из серверов, на которых размещена база данных, причем, возможно, разные серверы во время выполнения разных сеансов протокола. Было доказано, что для моделей с k копиями базы данных возможно построение корректных конфиденциальных лаконичных протоколов PIR.

Дальнейшие исследования в этом направлении были сосредоточены на изучении методов улучшения коммуникационной сложности [5, 6]. Кроме того, в работах [7-10] были исследованы возможности построения и оценки коммуникационной сложности (или лаконичности) схемы PIR на одном сервере при помощи различных техник, в том числе гомоморфного шифрования.

Однако, если рассматривать задачу построения PIR в предположении о том, что копии базы данных хранятся на облаке, то очевидно, в облачных информационных системах нельзя обеспечить изолированность нескольких копий базы данных друг от друга. Также нельзя исключить обмен информацией между копиями базы данных и наличие противника. При этом ни владелец данных (пользователь), ни клиент (имеющий официальный доступ для получения информации из базы данных) не могут контролировать облако.

Для облачных вычислений исследовались различные применения протокола PIR на практике. Например, в [11] рассматривалась реализация простой схемы PIR с несколькими серверами, аналогично построению систем RAID (Redundant Array of Inexpensive Disks) с избыточными массивами недорогих дисков. При таком подходе каждый сервер хранит только часть базы данных, и серверы могут быть опрошены параллельно. Предполагается, что не все серверы вступают в сговор, поскольку серверы могут работать на разных платформах и/или под управлением различных облачных провайдеров.

Также большое внимание было уделено исследованию различных методов, повышающих эффективность применения на практике PIR (CPIR – computational PIR). Например, в [12] рассматривались два метода: эффективное использование центрального процессора сервера и кодирование данных PBC (probabilistic batch codes – вероятностные пакетные коды), который используется для создания схемы PIR, предназначенной для обработки множества запросов. На основе этих методов была разработана библиотека CPIR – SealPIR, которая объединяет наиболее эффективный в вычислительном отношении протокол CPIR и предлагаемый метод сжатия запросов, что уменьшает коммуникационные затраты.

Из практических реализаций также стоит отметить MuchPIR – поиск частной информации с использованием гомоморфного шифрования, реализованный как расширение C/C++ Aggregate для Postgres [13]. Также полезным на практике может оказаться использование Piano – односерверной схемы PIR с сублинейным временем [14]. В отличие от классической схемы PIR, которая не выполняет никакой предварительной обработки базы данных, и сервер просто хранит исходную копию базы данных DB, односерверная схема PIR под названием Piano (сокращение от «Private Information Access NOw»), выполняет предварительную обработку запросов в сети Интернет. Такая схема может использоваться для сохранения

конфиденциальной информации при извлечении данных из блокчейна или создания безопасного DNS сервера.

Существует также достаточно большое количество статей, посвященных использованию в алгоритмах схемы PIR и гомоморфного шифрования, оценки вычислительных затрат и затрат на коммуникацию. Однако основным отличием исследуемой ниже модели является изучение ее характеристик с точки зрения математики распределённых систем при наличии противника, который не только пассивно наблюдает, но и может производить атаку с известными открытыми запросами.

Как и в работах [1, 2] рассматривается модель, включающая в себя облако, состоящее из нескольких серверов, на каждом из которых хранится копия одной и той же базы данных, дилера, пользователей, клиентов, противника. Будем предполагать, что серверы соединены посредством незащищенных каналов связи друг с другом и дилером. Облачные серверы и незащищенные каналы связи доступны для стороннего наблюдателя. Дилер находится вне облака, противнику недоступен. Предполагается, что на облаке имеется противник, который не только пассивно наблюдает, но производит атаку с известными открытыми запросами.

В отличие от работ [1, 2], где основной функцией дилера являлось шифрование и дешифрование информации при работе с облаком, в данной статье предлагается вместо шифрования использовать перестановки цифр числа как одностороннюю функцию. Применяется новая система счисления для представления номера бита и далее производятся перестановки цифр в новой системе счисления для запрашиваемого номера бита. Этот подход будет использоваться при формировании запроса к облаку, обработке ответа облака и получения значения искомого бита.

Для сокрытия при запросе искомого номера бита, вместо одного запрашиваемого номера бита дилер будет запрашивать у облака множество битов. Запрашиваемые множества массивов битов строятся независимо друг от друга. По одному такому множеству противник не может сделать вывод о другом множестве и сделать вывод об алгоритме их построения и запрашиваемом номере бита.

Предлагается использовать датчик псевдослучайных чисел для замены значения бита на противоположное при хранении номера бита на облаке. Перестановка цифр в преобразованном новом номере бита затрудняет противнику поиск рядом стоящих запрашиваемых битов.

Получены оценки вероятности угадывания номера бита при однократной атаке с известным открытым запросом номера бита i и при атаке с неограниченным числом известных открытых запросов при наличии противника на облаке. Приведена оценка коммуникационной сложности алгоритма.

2. Модель вычислений, основные определения и постановка задачи

2.1 Состав модели

Модель вычислений, для которой в данной статье исследуется облачный вариант задачи PIR. Модель включает в себя:

Облако. Состоит из нескольких удаленных серверов, администрирование которых осуществляется только их владельцами и не доступно для других. На облаке хранятся копии одной и той же базы данных. Копии на серверах имеют разные перестановки номеров массивов битов. Серверы соединены посредством незащищенных каналов связи друг с другом и дилером.

Пользователь. Хранит данные на облаке. Предполагается, что на облаке хранится k копий баз данных. Для загрузки данных пользователь обращается к дилеру по каналу связи, использующему стандартный криптографический протокол для обмена зашифрованными данными.

Клиенты. Запрашивают некоторую информацию из базы данных. Обращаются для выполнения запроса к дилеру. С дилером соединены каналами связи, использующими стандартный криптографический протокол для обмена зашифрованными данными. Клиенты не доверяют облаку и доверяют только дилеру.
Дилер. Находится вне облака, противнику недоступен. Представляет собой доверенный сервер, который в том числе осуществляет аутентификацию пользователей и клиентов. Канал связи с облаком является незащищенным. Объем памяти дилера для постоянного хранения данных пренебрежимо мал по сравнению с n. Получив от клиента данные для размещения на облаке, дилер выполняет перестановку уникальную для каждой копии базы данных, размещает данные на облаке. При запросе клиента дилер возвращает клиенту значение бита. В дальнейшем для простоты будем считать, что база данных загружается одним пользователем. Предполагается, что после загрузки база данных не может быть изменена. Либо может быть изменена полностью. Также будем рассматривать случай запроса одного бита.

2.2 Основные определения

Противник. Не вмешивается в выполнение криптографического протокола. Имеет доступ к базе данных на облаке, к каждой ее копии, к каналам связи на облаке. Может создавать фальшивых клиентов, работающих по протоколу. То есть противник не только пассивно наблюдает, но и сам может производить атаку с известными открытыми запросами. Противник может через дилера отправить запрос на облако. При этом противник знает алгоритм формирования запроса к облаку, но не знает конкретных параметров. Противник также знает алгоритм формирования ответа клиенту дилером на основании ответа, полученного дилером от облака.
Угроза: угадывание исходного номера бита, запрашиваемого клиентом в базе данных.
Атака:

- Атака пассивного противника на облаке заключается в наблюдении за запросом от дилера и ответом облака дилеру. Эта информация используется для сбора статистики и анализа.
- Атака активного противника (атака с известными открытыми запросами) заключается в создании фальшивых клиентов и управление ими. При помощи фальшивых клиентов активный противник может сформировать произвольное число запросов, что позволит в коалиции с пассивным противником на облаке собрать и проанализировать информацию для всей базы данных.

2.3 Постановка задачи и основные обозначения

В отличие от классической постановки задачи PIR будем рассматривать базу данных (бинарная строка) $X = (x_0, \dots, x_{n-1})$ длины n, где элементы X нумеруются с нуля. Это делает более удобным работу с различными системами счисления.
Клиент хочет получить один бит информации x_i с номером i из базы данных X так, чтобы противник не узнал ничего о том, для какого индекса i был запрошен бит.
Будем размещать на облаке k копий баз данных. Пусть $k = 2^d$, где $d \geq 2$. На практике $4 \leq k \leq 32$. В дальнейшем будем предполагать, что на k наложены эти ограничения.
Без потери общности предполагается, что $n = l^d$, то есть, $d = log_2 k$ и $l = \sqrt[d]{n}$. Заметим, что поскольку l - основание системы счисления, то $l \geq 2$ и d подбирается так, чтобы $l \ll n$. Пусть $L_p = \frac{n}{l}$.
Так как $l = \sqrt[d]{n}$, округлим l до целого числа в большую сторону.
Пусть x – элемент группы Z_2, обозначим через h ($h \in \{1, \dots, k\}$) – номер копии базы данных.

3. Представление номера бита в l-ичной системе счисления

Представим номер бита как d разрядов в l-ичной системе счисления. Выбор l и d основывается на известной модели [1, 2], где номер i бита x_i представлен в l-ичной записи: $i = a_0^{(i)} + a_1^{(i)} l + \cdots + a_{d-1}^{(i)} l^{d-1}$. Цифры l-ичной записи представляют собой элементы кортежа $\left(a_0^{(i)}, \ldots, a_{d-1}^{(i)}\right)$ длины d, то есть на j–м месте кортежа стоит l-ичная цифра.

Эти d элементов кортежа можно интерпретировать как точку с целочисленными координатами в гиперкубе размерности d и длиной стороны l, где $l=\sqrt[d]{n}$. Такое множество точек куба размерности d можно рассматривать как множество слов длины d с алфавитом $(0, \ldots, l-1)$.

4. Загрузка данных на облако

4.1 Инициализация массива X^a

Напомним, что $n = l^d$ и $L_p = \frac{n}{l}$.

Выберем целые числа $d_u \geq 2$ и $l_b \geq 2$ такие, чтобы $l_b^{d_u} \leq L_p$ и обозначим $L_p^- = l_b^{d_u}$ ($L_p^- \leq L_p$).

Пусть $l^* = \left\lfloor \frac{n}{L_p^-} \right\rfloor$, где ($l^* \geq l$).

По аналогии с алгоритмом, изложенном в [2], для каждого из l^* интервалов применим алгоритм преобразования содержащихся в нем слов в слова без повторяющихся букв (цифр). В результате каждому номеру бита в l_b-ичной системе счисления будет соответствовать номер бита в $(l_b + d_u - 1)$-ичной системе счисления. То есть номер бита можно интерпретировать как точку в гиперкубе размерности d_u со стороной $l_u = (l_b + d_u - 1)$. Количество битов в каждом интервале увеличится и станет равным L_p^+ ($L_p^+ = l_u^{d_u}$ и $L_p^+ > L_p^-$). Пусть $n_+ = l^* \cdot L_p^+$.

Ниже будет показано, что коммуникационная сложность зависит от l^*. Для того, чтобы коммуникационная сложность не возрастала, нужно подобрать значения d_u и l_u так, чтобы минимизировать разность $L_p - L_p^-$. Конкретный выбор значений d_u и l_u, который не только минимизирует разность $L_p - L_p^-$, но и уменьшает количество вычислительных операций, производимых дилером, будет описан ниже.

Рассмотрим интервал, содержащий L_p^+ элементов. Каждый номер бита, попавший в этот интервал, в l_u-ичной записи состоит из элементов кортежа длины d_u. Для представления номера бита используются неповторяющиеся цифры.

Рассмотрим номера из одного интервала. В разрядах этих номеров стоят различные цифры из одного множества различных цифр. Заметим, что из множества d_u различных цифр можно получить $d_u!$ перестановок цифр.

Обозначим через $n_u^- = \frac{L_p^+}{d_u!}$ – количество множеств различных цифр мощностью d_u. Как было показано в [2] все преобразованные номера, содержащие одинаковые цифры, лежат в одном интервале мощностью L_p^+.

Значения битов для номеров с совпадающим набором цифр, можно хранить в виде массива.

Отсортируем перестановки (например, по возрастанию) цифр каждого номера. Каждая перестановка цифр будет иметь номер в соответствующем интервале. Этот номер ставится в соответствие массиву битов для номеров, имеющих одинаковые цифры.

Тогда значения битов, соответствующих данному номеру, будут располагаться в массиве битов в соответствии с сортировкой цифр номера.

Будем в дальнейшем хранить одну перестановку, соответствующую минимальному номеру.

Таким образом, вместо номеров с одинаковым набором различных цифр будем хранить один отсортированный набор цифр, что в $d_u!$ раз меньше количества номеров. Массив битов будет содержать $d_u!$ элементов.

Каждый элемент массива битов преобразуется с использованием датчика псевдослучайных чисел $PRNG1(db, i)$ в зависимости от запрашиваемого клиентом номера i. Датчик псевдослучайных чисел возвращает 0 или 1. Если возвращен 0, то значение бита не меняется, а если 1, то значение бита меняется на противоположное. Это необходимо, чтобы в случае сговора, пассивного противника, находящегося на облаке и видящего базу данных, и активного противника вне облака, нельзя было сделать вывод о соответствии значения бита конкретному номеру.

Пусть $\hat{n} = l^* \cdot n_u^-$.

Пусть $n_a = l_a^{d_a}$, где l_a и d_a целые числа. Выберем l_a и d_a такие, чтобы n_a было максимально близким к $\hat{n}$. Тогда $\hat{n} \le n_a$.

Обозначим массив битов, соответствующий группе номеров, состоящих из одинаковых цифр, через $x_{i_k}^a$. Обозначим через n_c количество битов, содержащихся во всех n_a массивах, тогда $n_c = n_a \cdot d_u!$.

Преобразуем номера массивов битов в l_a–ичную систему счисления.

На облако будет загружаться последовательность массивов битов $X^a = \left(x_0^a, \dots, x_{n_a-1}^a\right)$. Нумерация массива X^a начинается с нуля. Количество элементов массива битов $x_{i_k}^a$ равно $d_u!$. Каждому массиву битов $x_{i_k}^a$ соответствует $d_u!$ номеров битов.

Так как последовательность массивов битов X^a упорядочена, не нужно хранить номера массивов битов для доступа к ним.

4.2 Построение матрицы перестановок

При запросе клиентом номера бита i находится номер интервала, в котором будет содержаться i и номер массива битов в интервале. На основе этих двух номеров формируется номер массива в последовательности X^a.

Номер запрашиваемого клиентом массива битов в l_a–ичной системе счисления можно представить как целочисленную точку гиперкуба размерности d_a с длиной стороны l_a. При этом j-ая ($j \in \{1, \dots, d_a\}$) координата точки гиперкуба является j-ым разрядом представления числа в l_a-ичной системе счисления.

Изменим номера массивов битов путем изменения цифр в разрядах номеров в l_a–ичном представлении. Цифры в каждом разряде изменяются при помощи перестановок цифр от 0 до $l_a - 1$. Для каждого разряда вычисляется своя перестановка цифр. Поскольку перестановки являются случайными, то для разных разрядов возможно, что перестановки совпадают.

Выберем копию базы данных $h \in \{1, \dots, k\}$. Пусть β_{hj} - случайная перестановка из l_a чисел от 0 до $l_a - 1$, где $j \in \{1, \dots, d_a\}$ – номер разряда числа в номере массива битов в l_a–ичной системе счисления. Представим перестановку чисел в виде таблицы:

$$\begin{pmatrix} 0 & \dots & l_a - 1 \\ \beta_{hj}(0) & \dots & \beta_{hj}(l_a - 1) \end{pmatrix}.$$

Тогда $\beta_h = \{\beta_{h1}, \dots, \beta_{hd_a}\}$ - множество «поразрядных» перестановок числа в l_a–ичной системе счисления.

Построим матрицу M_β размера $k \times d_a$, где k – число копий базы данных, а d_a целое число, такое, что $n_a = l_a^{d_a}$. Элементом β_{ij} матрицы M_β является случайная перестановка целых чисел от 0 до $l_a - 1$. Таким образом, для каждой копии базы данных будет построено d_a случайных перестановок.

Для генерации перестановок цифр номера бита l_a–ичной системе счисления используется датчик псевдослучайных чисел $PRNG2(db, m)$. На вход $PRNG2(db, m)$ подается уникальное

значение db для конкретной базы данных и номер копии базы данных m. Датчик псевдослучайных чисел при одинаковой инициализации генерирует одинаковую перестановку.

Поскольку матрица M_β требует для хранения значительного объема памяти, матрица M_β при необходимости восстанавливается при помощи датчика псевдослучайных чисел.

4.3 Отображение $i_a \overset{\beta_h}{\to} i_c$

Для запрашиваемого клиентом номера бита i найдем соответствующий номер массива i_a в X^a.

Номер массива i_a в X^a представляем в системе счисления по основанию l_a с числом разрядов d_a:

$i_a = \sum_{j=1}^{d_c} a_j \cdot l_a^{j-1}$, где $a_j \in \{0, \dots, l_a - 1\}$.

Для копии базы данных с номером h выполним перестановки для разрядов числа i_a.

$i_c = \sum_{j=1}^{d_a} \beta_{hj}(a_j) \cdot l_a^{j-1}$.

Обозначим такое преобразование через $i_a \overset{\beta_h}{\to} i_c$, где $\beta_h = \{\beta_{h1}, \dots, \beta_{hd_a}\}$ – строка матрицы M_β.

Необходимые для работы алгоритма элементы матрицы M_β генерируются при помощи датчика псевдослучайных чисел.

4.4 Вспомогательные построения, необходимые для работы алгоритма запроса бита – алгоритм построения матрицы G

Построим матрицу G размера $n_a \times 2$.

$$G = \begin{pmatrix} 0 & x_0^a \\ \vdots & \vdots \\ n_a - 1 & x_{n_a-1}^a \end{pmatrix}$$

Первый столбец этой матрицы – последовательность n_a целых чисел $[0, n_a - 1]$. Второй столбец матрицы G – элементы $X^a = (x_0^a, \dots, x_{n_a-1}^a)$. Далее на основании матрицы G будут построены матрицы G_h, которые будут загружены на облако.

Алгоритм построения матрицы G

Вход: X^a.

Выход: Матрица G.

Шаг 1. В первый столбец матрицы G заносится последовательность n_a целых чисел $[0, n_a - 1]$.

Шаг 2. Во второй столбец матрицы G заносятся элементы $X^a = (x_0^a, \dots, x_{n_a-1}^a)$ ■.

4.5 Формирование матриц G_h для каждой копии базы данных из матрицы G

Алгоритм формирования матриц G_h

Вход: матрица G.

Выход: матрицы G_h для всех $h \in \{1, \dots, k\}$.

Шаг 1. Выберем копию базы данных h.

Шаг 2. Скопируем матрицу G во вспомогательную матрицу G_{aux}.

Шаг 3. Для каждого элемента i_a, где $i_a = 0, \dots, n_a - 1$ первого столбца матрицы G_{aux} выполним преобразование $i_a \overset{\beta_h}{\to} i_c$.

Шаг 4. Отсортируем строки матрицы G_{aux} по первому столбцу. Получим матрицу G_h.

Шаг 5. Выполним Шаги 1-4 для всех $h \in \{1, \dots, k\}$ ■.

В результате работы алгоритма получим G_h ($h \in \{1, \dots, k\}$) матриц для каждой копии базы данных.

4.6 Загрузка данных на облако

Алгоритм загрузки данных на облако аналогичен алгоритму, приведенному в [1, 2].
На вход алгоритма подается бинарный массив $X = (x_1, \dots, x_n)$. Для удобства будем предполагать, что нумерация элементов начинается с нуля $X = (x_0, \dots, x_{n-1})$. В результате работы алгоритма на облако загружается k преобразованных при помощи перестановки копий массивов битов баз данных.

Алгоритм загрузки данных на облако

Вход: X.
Выход: загрузка на облако 2-го столбца матриц G_h, где $h \in \{1, \dots, k\}$.

Шаг 1. Дилер аутентифицирует пользователя. Пользователь передает дилеру массив значений битов $X = (x_0, \dots, x_{n-1})$. Дилер преобразует массив $X = (x_0, \dots, x_{n-1})$. в массив $X^a = \left(x_0^a, \dots, x_{n_a-1}^a\right)$.
Шаг 2. Дилер формирует матрицу $G = \left\|g_{i,j}\right\|$, где $i = 0, .., n_a - 1,\ j = 1,2$.
Шаг 3. Дилер генерирует элементы матрицы M_β в процессе работы.
Шаг 4. Дилер создает матрицу G_h, где $h \in \{1, \dots, k\}$.
Шаг 5. Дилер загружает 2-ой столбец матрицы G_h на облако, где $h \in \{1, \dots, k\}$.
Шаг 6. Шаги 4-5 дилер выполняет в цикле для каждой копии базы данных. По окончании перебора k копий базы данных происходит выход из алгоритма ■.
Число битов, переданных каждой из k копий базы данных равно $n_a \cdot d_u!$.
Заметим, что поскольку в дальнейшем восстановление матрицы G не предполагается, матрицу G дилер не хранит.
Для выполнения перестановки дилеру требуется иметь объем памяти, сравнимый с объемом, необходимым для хранения одной копии базы данных. При этом предполагается, что дилер может работать с несколькими базами данных и нет необходимости их одновременного хранения. После загрузки базы данных на облако память дилера очищается для работы со следующей базой данных.

5. Выполнение дилером запроса клиента

Для сокрытия при запросе искомого номера бита, вместо одного запрашиваемого номера бита дилер будет запрашивать у облака множество битов мощностью $l^* \cdot d_u!$. Причем только один из этих битов был запрошен клиентом.
При запросе искомого номера бита вычисляется номер массива $x_{i_k}^a$, содержащего этот бит. Для сокрытия при запросе искомого номера массива $x_{i_k}^a$, вместо одного запрашиваемого номера массива дилер будет запрашивать у облака множество номеров массивов мощностью l^*.
Каждый номер запрашиваемого массива битов принадлежит только одному из интервалов мощности n_u^-.

5.1 Использование множества номеров массивов битов в запросе для сокрытия номера бита

Пусть клиент интересуется элементом $x_i \in X$, где $0 \le i < n$. Напомним, что $n_+ = l^* \cdot L_p^+$. Обозначим через $X^+ = \left(x_0^+, \dots, x_{n_+-1}^+\right)$.
Отрезок целых чисел от 0 до $n_+ - 1$ содержит l^* интервалов по L_p^+ элементов в каждом из них.
Номер бита i лежит ровно в одном из этих интервалов. Далее для массива в X^a, содержащего бит i, выберем по одному массиву битов в каждом из оставшихся $l^* - 1$ интервалов (алгоритм

выбора массивов описан в п. 5.2). Таким образом, для каждого массива битов выбираются $l^* - 1$ массивов битов, лежащих в различных интервалах и не попадающих в интервал, где лежит массив, содержащий число i.

Как было сказано выше, каждый интервал можно интерпретировать как дискретный гиперкуб размерности d_u с длиной стороны l_u.

Выберем интервал $m \in \{1, \dots, l^*\}$. Пусть σ_{mj} - случайная перестановка из l_u чисел от 0 до $l_u - 1$, где $j \in \{1, \dots, d_u\}$ – разряд числа в l_u–ичной системе счисления. Как и ранее, запишем перестановку чисел в виде:

$$\begin{pmatrix} 0 & \dots & l_u - 1 \\ \sigma_{mj}(0) & \dots & \sigma_{mj}(l_u - 1) \end{pmatrix}.$$

Тогда $\sigma_m = \{\sigma_{m1}, \dots, \sigma_{md_u}\}$ - множество «поразрядных» перестановок числа в l_u-ичной системе счисления d_u-разрядного числа.

Построим матрицу M_σ (по аналогии с матрицей M_β) размера $l^* \times d_u$, где l^* – число интервалов, а d_u целое число, такое, что $L_p^+ = l_u^{d_u}$. Элементом σ_{ij} матрицы M_σ является случайная перестановка целых чисел от 0 до $l_u - 1$. Таким образом, для каждого интервала на которые разбивается база данных будет построено d_u случайных перестановок.

Для генерации перестановок цифр номера бита матрицы M_σ используется датчик псевдослучайных чисел $PRNG3(db, m, j)$. На вход $PRNG3(db, m, j)$ подается уникальное значение db для конкретной базы данных, номер интервала m (всего l^* штук) и $j \in \{1, \dots, d_u\}$ – разряд числа в l_u–ичной системе счисления. Датчик псевдослучайных чисел при одинаковой инициализации генерирует одинаковую перестановку.

Поскольку матрица M_σ требует для хранения значительного объема памяти, элементы матрицы M_σ восстанавливаются при помощи датчика псевдослучайных чисел $PRNG3(db, m, j)$.

Поскольку отрезок целых чисел от 0 до $n_+ - 1$ разбивается на l^* интервалов по L_p^+ элементов в каждом интервале, то любое число из этого множества попадет в один из таких интервалов. Найдем номер интервала w, в который попало число i при разбиении отрезка целых чисел от 0 до $n - 1$ на l^* интервалов:

$$w = \left\lfloor \frac{i}{L_p^-} \right\rfloor + 1.$$

Заметим, что $w \in \{1, \dots, l^*\}$.

Приведем i по модулю L_p^- . Обозначим через $i^- = i \, mod L_p^-$ результат приведения. Тогда i^- является порядковым номером в интервале с номером w.

Представим порядковый номер i^- в интервале с номером w в l_u-ичной системе счисления: $i^+ = \sum_{j=1}^{d_u} a_j \cdot l_u^{j-1}$, где $a_j \in \{0, \dots, l_u - 1\}$.

Рассмотрим сумму $\sum_{j=1}^{d_u} (\sigma_{wj})^{-1}(a_j) \cdot l_u^{j-1}$, где $(\sigma_{wj})^{-1}(a_j)$ обратная перестановка, такая, что $\sigma_{wj}\left((\sigma_{wj})^{-1}(a_j)\right) = a_j$.

Обозначим через $D_j = (\sigma_{wj})^{-1}(a_j)$, величина D_j – секрет, известный только дилеру.

В каждом интервале m ($m \in [1, l^*]$) найдем число, такое, что $i^+(m) = \sum_{j=1}^{d_u} \sigma_{mj}(D_j) \cdot l_u^{j-1}$, где $j = 1, \dots, d_u$. Для всех интервалов таких чисел $i^+(m)$ будет l^*.Заметим, что для интервала w это будет число i^+.

Обозначим через Set_i полученное по алгоритму 5.2 (Алгоритм построения множества Set_i дилером) множество из l^* чисел.

5.2 Построение множеств Set_i дилером

Алгоритм построения множества Set_i дилером

Вход: i, l^*, d_u, l_u, d_a, l_a, L_p^-.
Выход: множество Set_i.
$Set_i = \emptyset$
// найдем номер интервала
$w \leftarrow \left\lfloor \frac{i}{L_p^-} \right\rfloor + 1$
// найдем порядковый номер числа i
// в соответствующем интервале разбиения
$i^- = i \; mod L_p^-$
// найдем представление числа i^- в l_u-ичной записи
$i^+ = \sum_{j=1}^{d_u} a_j \cdot l_u^{j-1}$, $a_j \in \{0, \dots, l_u - 1\}$
// найдем координаты $D_j \in \{0, \dots, l_u - 1\}$
for j← 1 to d_u do
    $D_j = \left(\sigma_{wj}\right)^{-1}\left(a_j\right)$
for m← 1 to l^* do
    $i^+(m) = \sum_{j=1}^{d_u} \sigma_{mj}\left(D_j\right) \cdot l_u^{j-1}$
    // находим номер массива в X^a
    // для номера элемента в X^+
    $i_m = \text{номер}(x_{i_k}^a) \leftarrow L_p^+ \cdot m + i^+(m)$
    $Set_i = Set_i \cup i_m$

Листинг 1. Алгоритм построения множества Set_i

5.3 Запрос клиентом бита с номером i. Предварительные замечания

Пусть клиент запрашивает бит с номером i.
После подтверждения полномочий клиента дилер разрешает клиенту отправить запрос.
Напомним, что физически в облаке хранятся 2-е столбцы матриц G_h, $h \in \{1, \dots, k\}$. Второй элемент в строке матрицы G_h представляет собой массив битов. Без потери общности (как было сказано выше) будем индексировать элементы базы данных с 0.
Также как и в [1, 2] для каждого элемента множества Set_i случайно и равномерно выбирается номер копии базы данных. Все элементы множества Set_i для которых выбрана копия базы данных h, обозначим через множества Set_i^h, где $h \in \{1, \dots, k\}$. Очевидно, что множества Set_i^h не пересекаются между собой, поскольку все элементы множества Set_i различны и каждому элементу ставится в соответствие ровно один номер h (номер копии базы данных). Объединение множеств Set_i^h содержит все элементы множества Set_i , то есть $Set_i = \cup Set_i^h$, где $h \in \{1, \dots, k\}$.
Пусть для запрашиваемого номера i в множестве Set_i была выбрана копия базы данных h.
Напомним, что элементами множества Set_i^h являются номера массивов битов множества Set_i для которых была выбрана копия базы данных h.
Для каждого элемента множества Set_i^h выполняется перестановка β^h, пусть $y = \beta^h(i_a)$. Полученные после перестановки элементы сортируются. Это множество обозначим через $Set_i^{\beta^h}$. Позиция i_{pos} для исходного запрашиваемого элемента $y = \beta^h(i_a)$ в множестве $Set_i^{\beta^h}$ запоминается.
Дилер на время выполнения запроса формирует вектор-строку $\boldsymbol{s}_{num}$ из четырех элементов $\boldsymbol{s}_{num} = (s_1, s_2, s_3, s_4)$:

- Первый элемент вектора-строки $\boldsymbol{s}_{num}$ содержит номер бита i.
- Второй элемент вектора-строки $\boldsymbol{s}_{num}$ содержит номер элемента в возвращаемом массиве.
- Третий элемент вектора-строки $\boldsymbol{s}_{num}$ содержит номер копии h базы данных, в которую попал номер i_a.
- Четвертый элемент вектора-строки $\boldsymbol{s}_{num}$ содержит позицию i_{pos} номера y в множестве $Set_i^{\beta^h}$.

Вектор-строка $\boldsymbol{s}_{num}$ позволяет дилеру после получения ответа от облака сразу отбросить данные, кроме данных, необходимых для выполнения запроса клиента.
Вектор-строка $\boldsymbol{s}_{num}$ формируется у дилера и известна только дилеру. Вектор-строка $\boldsymbol{s}_{num}$ является секретной.
На k копий базы данных дилер отправляет в общей сложности l^* номеров.
После выполнения запроса дилер при помощи вектора-строки $\boldsymbol{s}_{num}$ получает значение искомого элемента и отправляет его клиенту.

Алгоритм подготовки запроса к облаку

Вход: i, l^*, d_u, l_u, d_a, l_a, L_p^-.

Выход: $Set_i^{\beta^h}$, $\boldsymbol{s}_{num}$.

Шаг 1. Клиент обращается к дилеру и запрашивает значение бита с номером i.

Шаг 2. Дилер генерирует множество Set_i.

Шаг 3. Дилер на время выполнения запроса i-го бита резервирует память для вектора-строки $\boldsymbol{s}_{num}$ из четырех элементов.

Шаг 4. Дилер заносит номер бита в массиве i_a во второй элемент вектора-строки $\boldsymbol{s}_{num}$.

Шаг 5. Дилер выполняет разбиение множества Set_i на непересекающиеся подмножества Set_i^h, где $h \in \{1, \dots, k\}$, путем случайного и равномерного выбора номера копии базы данных для каждого элемента множества Set_i .

Дилер заносит в третий элемент вектора-строки $\boldsymbol{s}_{num}$ номер копии базы данных s_3, соответствующей искомому в множестве элементу i_a.

Шаг 6. Дилер выполняет перестановку β^h для элементов множества Set_i^h после чего полученные после перестановки новые элементы сортируются. Получаем множество $Set_i^{\beta^h}$. Этот шаг выполняется для всех $h \in \{1, \dots, k\}$.

Шаг 7. Дилер заносит в четвертый элемент вектора-строки $\boldsymbol{s}_{num}$ позицию i_{pos} номера $y = \beta^h(i_a)$ для множества $Set_i^{\beta^h}$ ■.

5.4 Выполнение запроса к облаку и ответ облака

Алгоритм обмена информацией с облаком

Вход: $Set_i^{\beta^h}$ $(h \in \{1, \dots, k\})$.

Выход: значения массивов битов в том же порядке, в котором передавались номера массивов битов множества $Set_i^{\beta^h}$ для каждой копии базы данных $(h \in \{1, \dots, k\})$.

Шаг 1. Для каждой копии базы данных дилер отправляет на облако множества $Set_i^{\beta^h}$.

Шаг 2. Для запрошенных номеров массивов битов облако возвращает дилеру значения массива битов, соответствующих второму столбцу матрицы G_h, где $h \in \{1, \dots, k\}$). Порядок возвращаемых массивов значений битов соответствует исходному порядку элементов множества $Set_i^{\beta^h}$ ■.

5.5 Обработка ответа облака

Алгоритм обработки ответа облака

Вход: $\boldsymbol{s}_{num}$, значения массивов битов в том же порядке, в котором передавались номера массивов битов множества $Set_i^{\beta^h}$ для каждой копии базы данных ($h \in \{1, \dots, k\}$).
Выход: значение бита i.
Шаг 1. Дилер при помощи элемента s_3 вектора-строки $\boldsymbol{s}_{num}$ выбирает возвращаемые значения из соответствующей базы данных h. Значения, возвращаемые из других баз данных, отбрасываются.
Шаг 2. Дилер при помощи элемента s_4 вектора-строки $\boldsymbol{s}_{num}$ выбирает возвращаемый массив битов из соответствующей базы данных h, стоящий на позиции i_{pos}. Остальные массивы значений битов отбрасываются.
Шаг 3. Дилер для выбранного массива значений битов при помощи элемента s_2 вектора-строки $\boldsymbol{s}_{num}$ получает значение бита. На основе датчика случайных чисел $PRNG1(db, i)$ дилер восстанавливает значение искомого бита.
Шаг 4. Дилер отправляет значение i бита клиенту ■.

6. Оценка требуемой памяти и сложности алгоритма

6.1 Объем информации, который необходимо хранить дилеру

На протяжении всего времени существования базы данных дилер хранит информацию для базы данных объема n:

- Значения n, k, l^*, d_a, l_a, d_u, l_u, L_p^-;
- Значение db для инициализации датчиков псевдослучайных чисел $PRNG1(db, i)$, $PRNG2(db, m)$ и $PRNG3(db, m, j)$;
- Вектор-строку $\boldsymbol{s}_{num}$.

Объем хранимой дилером информации много меньше n.

6.2 Характеристики предложенной схемы

В данной схеме рассматривается активный противник, работающий по протоколу. Противник имеет доступ к базе данных на облаке, к каждой ее копии, к каналам связи на облаке, может создавать фальшивых клиентов, работающих по протоколу. То есть противник не только пассивно наблюдает, но и сам может производить запросы с помощью созданных фальшивых клиентов, то есть производит атаку с известными открытыми запросами. Противник не имеет доступа к дилеру.

6.3 Основные результаты

Под коммуникационной сложностью для предложенной схемы будем понимать общее количество пересылаемых битов, необходимых для обмена информацией между дилером и облаком. То есть сумму числа битов, отправляемых дилером на облако и числа битов, получаемых от облака дилером, необходимых для нахождения дилером значения бита, запрашиваемого у дилера. Пусть s_a – число битов для представления номера массива битов. Напомним, что l^* является мощностью множества Set_i.
Утверждение 1. Коммуникационная сложность схемы получения значения номера бита дилером без раскрытия его номера равна $l^* \cdot (s_a + d_u!)$ битов.

Доказательство. Для запроса значения бита дилер посылает копиям базы данных l^* номеров массивов битов, каждый номер такого массива имеет длину s_a битов. Облако отвечает l^* массивами битов, длина каждого массива $d_u!$ битов ■.

Число битов для представления номера массива битов меньше в $d_u!$ раз числа битов для представления номера бита. Таким образом, коммуникационная сложность не меняется по сравнению с алгоритмом передачи номера бита и возвращением бита.

Проанализируем вероятность угадывания противником номера запрашиваемого бита.

Если запросы выполнены к одному множеству, то они неразличимы. Таким образом, если противник совершает n или более запросов, он не узнает номер конкретного бита. Максимум, какую информацию противник может получить – биты, содержащиеся массивах с номерами множества Set_i, хотя был запрошен только один бит. Что будет означать, что противник не знает, а только угадывает, какой именно бит для Set_i был запрошен.

Утверждение 2. При однократной атаке с известным открытым запросом номера бита i и предположении о наличии пассивного противника на облаке вероятность угадывания противником номера бита не более $\frac{1}{l^* \cdot d_u!}$.

Доказательство. Множество Set_i состоит из l^* массивов битов и для любого бита $j \in Set_i$ выполняется $Set_i = Set_j$, если множество Set_i тоже содержит этот бит. То есть существует одинаковое множество для $l^* \cdot d_u!$ различных номеров битов. Таким образом вероятность угадывания номера бита из множества Set_i равна $\frac{1}{l^* \cdot d_u!}$, так как массивы с номерами множества Set_i содержат в общей сложности $l^* \cdot d_u!$ битов ■.

Сделав n или более запросов, противник получит информацию о числе реальных битов в каждом множестве Set_i. Число этих битов будет меньше или равно $l^* \cdot d_u!$. Противник понимает, что запрошенный клиентом номер бита находится среди истинных номеров битов для множества Set_i. И вероятность того, что клиент запрашивал конкретный номер реального бита для всех реальных номеров битов для множества Set_i одинаковая.

Для того, чтобы оценить вероятность угадывания конкретного номера бита из множества Set_i, необходимо посчитать среднее число истинных битов, попадающих в массивы битов с номерами из множества Set_i.

Пусть $l^+ = l^* \cdot d_u!$ и $n_c = n_a \cdot d_u!$.

Мощность множества X равна n $(n < n_c)$. Из множества массивов битов X^a выбирается суммарно l^+ битов. В этом случае возникает вопрос: сколько элементов N из случайной выборки мощности l^+ в среднем соответствует элементам множества X?

Пусть P_r – вероятность получения в выборке мощности l^+ ровно r элементов, соответствующих элементам множества X. Тогда P_r находится по формуле:

$$P_r = \frac{\binom{r}{n} \cdot \binom{l^+ - r}{n_c - n}}{\binom{l^+}{n_c}}.$$

Тогда среднее число истинных битов, попадающих в множество Set_i, равно:

$$N = \sum_{r=1}^{l^*} r \cdot P_r.$$

Утверждение 3. При атаке с неограниченным числом известных открытых запросов и предположении о наличии пассивного противника на облаке вероятность угадывания номера бита противником в среднем не более $\frac{1}{N}$.

Доказательство. Выполнив n запросов, противник определяет биты, входящие в массивы с номерами из множества Set_i. Если фиктивные клиенты будут выполнять любое количество запросов большее n, противник все равно не получит никакой дополнительной информации. Количество реальных бит, которые попали в каждую выборку мощности l^+ в среднем равно N. Тогда вероятность угадывания номера i в среднем будет не более $\frac{1}{N}$ ■.

Как было сказано ранее, для того чтобы коммуникационная сложность не возрастала, нужно подобрать значения d_u и l_u так, чтобы L_p^+ было максимально близким к L_p^-. Следовательно, $l^* \ll n$.

Теорема 1. Для предложенной схемы организации базы данных размера n:

- Коммуникационная сложность запроса равна $l^* \cdot (s_a + d_u!)$ битов.
- При однократной атаке с известным открытым запросом номера бита i и предположении о наличии пассивного противника на облаке вероятность угадывания противником номера бита не более $\frac{1}{l^* \cdot d_u!}$.
- При атаке с неограниченным числом известных открытых запросов и предположении о наличии пассивного противника на облаке вероятность угадывания номера бита противником в среднем не более $\frac{1}{N}$.

Доказательство. Из Утверждения 1 следует, что коммуникационная сложность запроса равна $l^* \cdot (s_a + d_u!)$ битов.

Из Утверждения 2 следует, что при однократном запросе номера бита вероятность угадывания номера равна $\frac{1}{l^* \cdot d_u!}$.

Из Утверждения 3 следует, что при любом числе запросов номеров битов фиктивными клиентами и наличии противника на облаке вероятность угадывания номера бита в среднем не более $\frac{1}{N}$ ■.

7. Задача дискретной оптимизации

Напомним, что $n = l^d$, $l = \sqrt[d]{n}$. По приведенным выше построениям d – размер гиперкуба, а l – длина его стороны. Для минимизации вычислений дилером необходимо минимизировать размер вычисляемых матриц M_β и M_σ. Для этого надо минимизировать размер строки каждой из матриц, то есть величины $d_a \cdot l_a$ ($l_a = \sqrt[d_a]{n_a}$) и $d_u \cdot l_u$ ($l_u = \sqrt[d_u]{L_p^+}$), где l_a и l_u - основания счисления, а d_a и d_u - количество разрядов числа для представления номера.

Рассмотрим функцию $f(x) = \sqrt[x]{n} \cdot x$ описывающую соотношение между длиной стороны гиперкуба и его размерностью. Исследуем функцию $f(x)$ для того, чтобы минимизировать размер генерируемой дилером информации. Заметим, что $x \geq 1$.

Найдем производную функции $f(x)$.

$$f'(x) = \left(n^{\frac{1}{x}}x\right)' = \left(n^{\frac{1}{x}}\right)' x + n^{\frac{1}{x}}x' = n^{\frac{1}{x}} \ln n \left(-\frac{1}{x^2}\right) x + n^{\frac{1}{x}} = n^{\frac{1}{x}}\left(1 - \frac{\ln n}{x}\right).$$

Производная равна нулю в точках экстремума. Найдем координату x точек экстремума. Поскольку $n^{\frac{1}{x}} > 0$, приравняем к нулю второй сомножитель:

$$1 - \frac{\ln n}{x} = 0.$$

Тогда:

$$\frac{\ln n}{x} = 1$$
$$x = \ln n$$

Теперь выясним, является ли эта точка экстремума минимум или максимум.

$$f''(x) = \left(n^{\frac{1}{x}}\left(1 - \frac{\ln n}{x}\right)\right)' = \left(n^{\frac{1}{x}}\right)' \cdot \left(1 - \frac{\ln n}{x}\right) + n^{\frac{1}{x}} \left(1 - \frac{\ln n}{x}\right)'$$
$$= -n^{\frac{1}{x}} \cdot \frac{\ln n}{x^2}\left(1 - \frac{\ln n}{x}\right) + n^{\frac{1}{x}}\frac{\ln n}{x^2} = n^{\frac{1}{x}}\frac{\ln^2 n}{x^3} > 0$$

Таким образом, минимум функции $f(x) = \sqrt[x]{n} \cdot x$ находится в точке $x = \ln n$. Следовательно, минимальное количество элементов матриц M_β и M_σ, которое должен генерировать дилер, достигается при $x = \ln n$.

Найдем предел $\lim\limits_{n\to\infty} \frac{\sqrt[x]{n}}{x}$. Произведем замену $x = \ln n$:

$$\lim_{n\to\infty} \frac{\sqrt[\ln n]{n}}{\ln n}.$$

Произведем замену $n = e^t$. Тогда $t = \ln n$. Если $n \to \infty$, то и $t \to \infty$. Тогда:

$$\lim_{t\to\infty} \frac{(e^t)^{\frac{1}{t}}}{t} = \lim_{t\to\infty} \frac{e}{t} = 0$$

Так как предел $\lim\limits_{n\to\infty} \frac{\sqrt[x]{n}}{x}$ равен 0, то это говорит о том, что число разрядов числа больше основания счисления.

Для того, чтобы n_a было близким к $\hat{n}$, надо уменьшать число разрядов (большее число) при фиксированном основании счисления (меньшее число). В этом случае можно точнее приблизить n_a ($n_a = l_a^{d_a}$) к $\hat{n}$ и количество хранимой на облаке информации будет минимальной.

Ранее был найден минимум функции $f(x) = \sqrt[x]{n} \cdot x$ для случая, когда переменная x является непрерывной. В общем случае $x = \ln n$ является вещественным числом. На практике основание счисления l и число разрядов d являются натуральными числами. Сформулируем задачу дискретной оптимизации, соответствующую поиску минимума функции $f(x)$:

$$\begin{cases} l \cdot d \to min; \\ l^d \geq n; \\ l, d \geq 2. \end{cases}$$

В формуле $f(x) = \sqrt[x]{n} \cdot x$ в дискретном случае натуральное число l соответствует $\sqrt[x]{n}$, а натуральное число d соответствует x. Решение для $f(x)$ (абсолютный минимум) в действительных числах точнее целочисленного, поэтому $l^d \geq n$.

Так как $l \geq 2$ (и $d \geq 2$), то для d имеется оценка сверху $d \leq \lceil \log_2 n \rceil$.

На практике рассматривалась база данных размера 2^{40} бит и пусть по построению интервалов $l^* \approx 2^{10}$, на практике $d_u! = 24$, что требует 5 битов для записи числа.

Для этих параметров сначала найдем точное решение (абсолютный минимум) функции $f(x)$. Заметим, что дискретный минимум всегда больше абсолютного минимума. Поэтому далее будем искать целочисленное решение, максимально близкое к абсолютному минимуму.

Пусть $n_a \approx \frac{n}{(d_u-1)!}$.

Минимизируем объем хранящейся на облаке информации и используемой дилером в процессе вычислений памяти. Рассмотрим следующее условие:

$\frac{n_a}{\hat{n}} = \frac{l_a{}^{d_a}}{\hat{n}} = 1$, тогда $d_a = \lceil \log_{l_a} \hat{n} \rceil$.

Это условие соотношения количества элементов n_a гиперкуба к сумме количества элементов в каждом интервале. Чем ближе это соотношения к 1, тем меньше фиктивных массивов битов было добавлено.

Определим значение x для $\hat{n} = 2^{36}$.

$$x = \ln 2^{36} \approx 24{,}95, \text{ а } \log_2 2^{36} = 36$$

$$\sqrt[x]{n} = \sqrt[\ln 2^{36}]{(2^{36})} \approx 2{,}72.$$

Подберем целочисленные значения для l_a и d_a.

Пусть основание счисления $l_a = 2$, тогда:

$$\frac{n_a}{\hat{n}} = \frac{2^{d_a}}{2^{36}} = 1.$$

Следовательно, $d_a = 36$. Для $\hat{n} = 2^{36}$ нужно генерировать $36 \cdot 2 = 72$ цифры для вычисления нового номера при запросе к облаку.

Пусть основание счисления $l_a = 3$, тогда:

$$\frac{n_a}{\hat{n}} = \frac{3^{d_a}}{2^{36}} = 1$$

$$3^{d_a} = 2^{36}$$

$$d_a = \lceil \log_3 2^{36} \rceil = \lceil 22{,}71 \rceil = 23$$

Тогда:

$$\frac{n_a}{\hat{n}} = \frac{3^{23}}{2^{36}} \approx 1{,}37.$$

Количество разрядов для представления номера равно 23. В каждом разряде может находиться одна из трех цифр системы счисления.

Для $\hat{n} = 2^{36}$ нужно генерировать $23 \cdot 3 = 69$ цифр для вычисления нового номера при запросе к облаку.

Таким образом при $l_a = 2$ дилеру требуется генерировать 72 цифры, а при $l_a = 3$ дилеру требуется генерировать 69 цифр. Но при $l_a = 3$ на облаке придется хранить в 1.37 раза больше битов.

При генерации множества Set_i после получения нового номера в интервале память дилера освобождается.

Коммуникационная сложность предложенного алгоритма равна $l^* \cdot (s_a + d_u!)$. Заметим, что увеличение l^* приводит к увеличению коммуникационной сложности, а уменьшение увеличивает вероятность угадывания номера бита.

Число битов s_a для представления номера может быть найдено по формуле:

$$s_a = \left\lceil \log_2 \frac{\max(n, n_a)}{(d_u - 1)!} \right\rceil = \left\lceil \log_2 \frac{2^{40}}{2^4} \right\rceil = 36$$

Тогда коммуникационная сложность равна:

$$l^* \cdot (s_a + d_u!) = l^* \cdot (36 + 5) < 2^{10} \cdot 2^6 = 2^{16}.$$

Вероятность угадывания при однократном запросе с использованием перестановок:

$$\frac{1}{l^* \cdot d_u!} = \frac{1}{2^{10} \cdot 24}.$$

8. Заключение

Предложенный алгоритм обладает значительными преимуществами по сравнению с алгоритмами, описанными в [1, 2]:

- Ранее в алгоритмах использовалось шифрование номера бита. В предложенном алгоритме используются перестановки цифр номера бита в выбранной системе счисления, что на практике требует меньшего числа операций, чем шифрование и расшифрование номера бита.
- Также следует заметить, что в том случае, при загрузке БД на облако выполняется перестановка, то на практике биты, номера которых являются соседними числами, в исходной базе данных чаще всего не будут являться соседними при их размещении на облаке. Это затрудняет противнику поиск стоящих рядом запрашиваемых битов.
- Изменен алгоритм построения множества Set_i, который ранее использовал вектор $\boldsymbol{b}$, содержащий величину сдвига в интервале для каждого элемента любого множества Set_i. В предложенном алгоритме отсутствие вектора $\boldsymbol{b}$ не позволит противнику получить информацию о Set_j по Set_i для непересекающихся множеств Set_i и Set_j.

Заметим, что на практике при реализации вычислений легко организовать параллельные вычисления, что позволяет увеличить скорость выполнения алгоритма. Например, генерацию перестановок как для всей базы данных, так и интервалов, можно выполнять в разных потоках. Для каждого из l^* интервалов перестановки цифр числа при формировании множества Set_i могут выполняться параллельно.

Список литературы / References

[1]. Мартишин С.А., Храпченко М.В., Шокуров А.В. Организация безопасного запроса к базе данных на облаке. Труды Института системного программирования РАН. Том 34, № 3, 2022г., с. 173-188, ISSN 2079-8156 (Print), ISSN 2220-6426 (Online).

[2]. Варновский Н.П., Мартишин С.А., Храпченко М.В., Шокуров А.В. Организация конфиденциальных запросов к облаку. Труды Института системного программирования РАН. Том 35, выпуск 5, 2023, с. 37-54, ISSN 2220-6426 (Online), ISSN 2079-8156 (Print).

[3]. Chor B., Goldreich O., Kushilevitz E., Sudan M. Private Information Retrieval, in IEEE Annual Symposium on Foundations of Computer Science, 1995, pp. 41–50.

[4]. Chor B., Goldreich O., Kushilevitz E., Sudan M. Private Information Retrieval, Journal of the ACM, Vol. 45, No. 6, November 1998, pp. 965–982.

[5]. Gasarch W. A survey on private information retrieval, Bulletin of the EATCS, 2004 pp. 72-107

[6]. Yekhanin S. Locally Decodable Codes and Private Information Retrieval Schemes, Springer Heidelberg Dordrecht London New York, ISSN 1619-7100, ISBN 978-3-642-14357-1 e-ISBN 978-3-642-14358-8, DOI 10.1007/978-3-642-14358-8 2010, p.82

[7]. Kushilevitz E., Ostrovsky R. Replication is not needed: Single database, computationally-private information retrieval (extended abstract). In Proc. of the 38st IEEE Sym. on Found. of Comp. Sci., pages 364 373, 1997.

[8]. Kushilevitz E., Ostrovsky R. One-way trapdoor permutations are su cient for non-trivial single-server private information retrieval. In EUROCRYPT00, 2000.

[9]. Ostrovsky R., Skeith III W. E. A Survey of Single-Database Private Information Retrieval: Techniques and Applications". Proceedings of the 10th International Conference on Practice and Theory in Public-Key Cryptography. Springer-Verlag. pp. 393–411.

[10]. Aguilar-Melchor C., Barrier J., Fousse L. XPIR: Private Information Retrieval for Everyone, Proceedings on Privacy Enhancing Technologies 2016(2), pp. 155-174, DOI:10.1515/popets-2016-0010.

[11]. Demmler D., Herzberg A., Schneider T. RAID-PIR: Practical multi-server PIR CCSW '14: Proceedings of the 6th edition of the ACM Workshop on Cloud Computing Security, November 2014 Pages 45–56б https://doi.org/10.1145/2664168.2664181, DOI:10.1145/2664168.2664181.

[12]. Angel S., Chen H., Laine K., Setty S. PIR with compressed queries and amortized computation IEEE Symposium on Security and Privacy, S&P (Oakland) 2018, pp. 1-18.

[13]. "MuchPIR Demo". 14 September 2021. URL: https://github.com/ReverseControl/MuchPIR (дата обращения: 22.08.2022).

[14]. Zhou M., Park A., Shi E., Zheng W., Piano: Extremely Simple, Single-Server PIR with Sublinear Server Computation 2024 IEEE Symposium on Security and Privacy (SP) 2024, pp. 4296-4314 DOI Bookmark: 10.1109/SP54263.2024.00055

Информация об авторах / Information about authors

Николай Павлович ВАРНОВСКИЙ – старший научный сотрудник Института проблем информационной безопасности МГУ имени М.В.Ломоносова. Сфера научных интересов: математика и информационная безопасность, теория сложности.

Nikolay Pavlovich VARNOVSKIY – researcher of Mathematical Studies in Information Security Section of Information Security Institute of Moscow State Lomonosov University. His research interests are mathematics, Information Security and Cryptography, complexity theory.

Сергей Анатольевич МАРТИШИН – кандидат физико-математических наук, научный сотрудник отдела теоретической информатики Института системного программирования им. В.П. Иванникова РАН. Его научные интересы включают параллельные алгоритмы, базы данных, облачные вычисления.

Sergey Anatolievich MARTISHIN – Cand. Sci. (Phys.-Math.), researcher of the Department of Theoretical Computer Science of Ivannikov Institute for System Programming of the RAS. His research interests include parallel algorithms, databases, cloud computing.

Марина Валерьевна ХРАПЧЕНКО – научный сотрудник отдела теоретической информатики Института системного программирования им. В.П. Иванникова РАН. Её научные интересы включают параллельные алгоритмы, базы данных, облачные вычисления.

Marina Valerievna KHRAPCHENKO – researcher of the Department of Theoretical Computer Science of Ivannikov Institute for System Programming of the RAS. Her research interests include parallel algorithms, databases, cloud computing.

Александр Владимирович ШОКУРОВ – кандидат физико-математических наук, доцент, заведующий отделом теоретической информатики Института системного программирования им. В.П. Иванникова РАН с 2019 года. Сфера научных интересов: алгебраические структуры в полях Галуа, базисы Гребнера, модулярная арифметика, нейрокомпьютерные технологии, цифровая обработка сигналов, криптографические методы защиты информации.

Alexander Vladimirovich SHOKUROV – Cand. Sci. (Phys.-Math.), Professor, Head of the Department of Theoretical Computer Science of Ivannikov Institute for System Programming of the RAS since 2019. Research interests: algebraic structures in the Galois fields, modular arithmetic, neurocomputer technologies, Grobner bases, digital signal processing, cryptographic methods for protecting information.