DOI: 10.15514/ISPRAS-2022-34(3)-12

Организация безопасного запроса к базе данных на облаке

[1] С.А. Мартишин, ORCID: 0000-0001-5437-4049 <mart@ispras.ru>
[1] М.В. Храпченко, ORCID: 0000-0002-5147-5132 <khrap@ispras.ru>
[1,2] А.В. Шокуров, ORCID: 0000-0002-6801-7728 <shok@ispras.ru>

[1] Институт системного программирования им. В.П. Иванникова РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25
[2] Московский физико-технический институт,
141701, Россия, Долгопрудный, Институтский пер., 9

Аннотация. Развитие облачных вычислений, включающее хранение и обработку конфиденциальных данных пользователей на серверах, которые могут быть атакованы, выдвигает новые требования к защите информации. В статье исследуется задача получения клиентом информации из базы данных таким образом, чтобы никто, кроме самого клиента не обладал сведениями о том, какую именно информацию он запросил (задача PIR – Private Information Retrieval). Эта задача в информационно-теоретической постановке была сформулирована в 1995 году Шором, Голдрайхом, Кушелевицем и Суданом. Предложена модель облачных вычислений, включающая облако, пользователя, клиентов, доверенное лицо (дилера), пассивного противника на облаке. Также предполагается, что у атакующей стороны имеется возможность создания фальшивых клиентов для формирования неограниченного числа запросов. Предложен алгоритм формирования и размещения базы данных на облаке и алгоритм запроса требуемого бита. Приведены оценки коммуникационной сложности и вероятности угадывания номера бита при совершении любого количества запросов клиентами, созданными для организации атаки.

Ключевые слова: базы данных; облачные вычисления; PIR

Для цитирования: Мартишин С.А., Храпченко М.В., Шокуров А.В. Организация безопасного запроса к базе данных на облаке. Труды ИСП РАН, том 34, вып. 3, 2022 г., стр. 173-188. DOI: 10.15514/ISPRAS–2022–34(3)–12.

Благодарности: Работа выполнена при поддержке Министерства науки и высшего образования Российской Федерации (грант 075-15-2020) и ИСП РАН. Авторы выражают особую благодарность Лазареву Д.О. за ценные замечания в процессе работы над статьей.

Organization of a secure query to a database in the cloud

[1] S.A. Martishin, ORCID: 0000-0001-5437-4049<mart@ispras.ru>
[1] M.V. Khrapchenko, ORCID: 0000-0002-5147-5132 <khrap@ispras.ru>
[2] A.V. Shokurov, ORCID: 0000-0002-6801-7728 <shok@ispras.ru>

[1] Ivannikov Institute for System Programming of the Russian Academy of Sciences,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia
[2] Moscow Institute of Physics and Technology,
9 Institutskiy per., Dolgoprudny, Moscow Region, 141701

Abstract. The development of cloud computing, including the storage and processing of confidential user data on servers that can be attacked, puts forward new requirements for information protection. The article explores the problem of obtaining information from the database by the client in such a way that no one except the client himself get any information about the information the client is interested in (PIR - Private Information Retrieval). The problem was introduced in 1995 by Chor, Goldreich, Kushilevitz and Sudan in the information-theoretic setting. A model of cloud computing is proposed. It includes a cloud, a user, clients, a trusted dealer, a passive adversary in the cloud. Also, the attacking side has the ability to create fake clients to generate an unlimited number of requests. An algorithm for the organization and database distribution on the cloud and an algorithm for obtaining the required bit were proposed. Communication complexity of the algorithm was estimated. The probability of revealing required bit's number in the case when fake clients perform unlimited requests was estimated too.

Keywords: database; cloud computing; PIR

For citation: Martishin S.A., Khrapchenko M.V., Shokurov A.V. Organization of a secure query to a database in the cloud. Trudy ISP RAN/Proc. ISP RAS, vol. 34, issue 3, 2022, pp. 173-188 (in Russian). DOI: 10.15514/ISPRAS–2022–34(3)–12

Acknowledgements. The work was supported by the Ministry of Science and Higher Education of the Russian Federation (grant 075-15-2020) and ISP RAS. The authors are especially grateful to D.O. Lazarev for valuable comments during the work on the article.

1. Введение

Безопасное использование облачных вычислений невозможно без решения задачи защиты данных. Хорошо известна задача получения информации из базы данных с открытым доступом, размещенной на облаке, таким образом, чтобы никто, кроме клиента, обращающегося с запросом, не обладал сведениями о запрашиваемой информации.

Неформально задача может быть сформулирована следующим образом. Алиса делает запрос к базе данных, но хочет, чтобы никто, в том числе любопытный администратор базы данных, не узнал, какие именно данные запрашиваются. Очевидным решением является запрос Алисой всей базы данных. Это решение приводит к чрезмерным затратам на передачу информации. Возникает вопрос: можно ли получить ответ с меньшей коммуникационной сложностью?

Протокол поиска конфиденциальной информации (Private information retrieval – PIR) позволяет пользователю получить интересующую его частную информацию с сервера. В области исследования PIR протокола значительный вклад был сделан в [1,2].

Сформулированная в этих работах постановка задачи приведена ниже:

Имеется база данных – бинарная строка $X = (x_1, \dots, x_n)$ длины n, хранящаяся на сервере.

Клиент хочет получить один бит информации x_iиз базы данных X так, чтобы сервер не смог определить, с какой позиции i был запрошен бит.

В первую очередь, протокол PIR должен удовлетворять **условию корректности**: для любого номера $i, 1 \le i \le n,$ клиент может сформировать такой запрос, что после завершения выполнения протокола клиент получит данные, по которым он может правильно вычислить значение бита x_i.

Помимо условия корректности, протокол PIR должен удовлетворять **условию конфиденциальности**: в результате выполнения протокола PIR противник не получает никакой информации о номере i запрашиваемого бита x_i из базы данных.

Если база данных размещена на единственном сервере, который полностью контролируется противником, то теоретико-информационное условие конфиденциальности корректного протокола PIR может быть выполнено только том случае, когда клиент запрашивает базу данных целиком. Это означает, что по каналам связи придется передавать объем информации, не меньший того, который содержится в самой базе данных. Очевидно, что такой протокол практически неприемлем из-за больших коммуникационных издержек [2].

Под коммуникационной сложностью протокола понимают общее количество бит, которыми обмениваются участники протокола за время его работы. Чтобы PIR-протокол можно было

бы применять на практике, он должен удовлетворять **условию лаконичности**: коммуникационная сложность протокола должна быть существенно меньше размера базы данных.

Таким образом, если база данных размещена только на одном сервере, и этот сервер контролируется противником, то не существует PIR-протокола, который одновременно удовлетворял бы условиям корректности, конфиденциальности и лаконичности. Возникает вопрос: можно ли построить протоколы PIR, удовлетворяющие всем трем указанным условиям одновременно?

Одной из таких перспективных моделей является модель реплицированной базы данных, в которой несколько одинаковых копий строки $X = (x_1, \dots, x_n)$ размещены на разных серверах. Предполагается, что клиент имеет связь с каждым из этих серверов, но сами серверы не имеют связи друг с другом. Также предполагается, что противник может наблюдать любой из серверов, на которых размещена база данных, причем, возможно, разные серверы во время выполнения разных сеансов протокола.

В работах [1, 2] было доказано, что для моделей с k реплицированными копиями базы данных при $k \geq 2$ возможно построение корректных конфиденциальных лаконичных протоколов PIR. Дальнейшие исследования в этом направлении были сосредоточены на изучении методов улучшения коммуникационной сложности [3, 4]. Кроме того, в работах [5-8] 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 реплицированная база данных хранится на облаке, то очевидно, что ни владелец данных (пользователь), ни клиент (имеющий официальный доступ для получения информации из базы данных) не могут контролировать облако. Таким образом, в облачных информационных системах в отличие от [1, 2] нельзя обеспечить изолированность копий распределенной базы данных друг от друга, то есть нельзя исключить обмен информацией между копиями базы данных. Кроме того, говоря о хранении и использовании конфиденциальной информации, требуется знать, является ли противник активным или пассивным.

Активный противник в облаке, который способен вмешиваться в ход выполнения протокола, как правило, может быть выявлен достаточно быстро путем полного анализа результатов выполнения протокола [9].

Пассивный противник, который наделен способностью получать некоторую информацию о процессе выполнения протокола, но не в праве в него вмешиваться, не может быть обнаружен даже при исчерпывающем анализе результатов многократного выполнения протокола. Такой противник представляет даже большую опасность, поскольку он собирает и анализирует конфиденциальную информацию, не обнаруживая себя.

Для облачных вычислений, исследовались различные возможности реализации. Например, реализация простой схемы PIR с несколькими серверами, основанной на подходе, аналогичном подходу построения систем RAID (Redundant Array of Inexpensive Disks) с избыточными массивами недорогих дисков [10]. В этом случае каждый сервер хранит только часть базы данных, серверы могут быть опрошены параллельно. Предполагается, что не все они вступают в сговор, поскольку эти серверы могут работать на разных платформах или под управлением различных облачных провайдеров.

Также исследовались различные методы, делающие практическое применение PIR (CPIR - computational PIR) эффективнее. Например, в [11] рассматривалось два метода. Первый направлен на эффективное использование центрального процессора сервера. Второй метод основан на кодировании данных PBC (probabilistic batch codes – вероятностные пакетные коды) и 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 [12].

Таким образом, за все время изучения протокола PIR было получено большое количество результатов для различных условий его применения.

Однако, если базы данных хранятся на облачных серверах, то условия задачи PIR существенно изменяются, и ранее известные методы ее решения не подходят. Чтобы найти решение задачи PIR в облачной вычислительной среде, предлагается рассмотреть иную модель взаимодействия участников протокола с базой данных.

2. Состав модели и постановка задачи

2.1 Состав модели

Модель вычислений, в рамках которой мы исследуем облачный вариант задачи PIR, включает несколько участвующих в ней процессов.

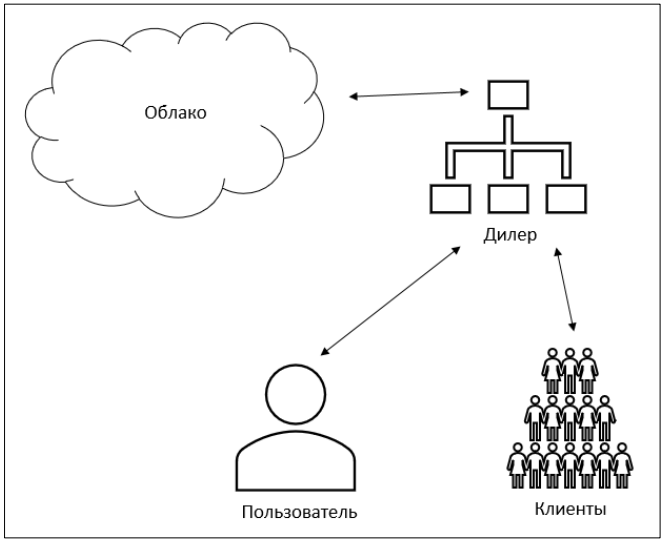

Рис. 1. Состав модели
Fig. 1. Model components

1) **Облако.** Состоит из нескольких серверов, на каждом из которых хранится копия одной и той же базы данных. Эти копии могут отличаться друг от друга при использовании различных ключей шифрования одинаковых данных. Облачные серверы способны выполнять любые вычислительные операции, присущие системам управления базами данных. Считается, что эти серверы соединены посредством незащищенных каналов связи друг с другом и дилером. По этим каналам облачные серверы обмениваются по запросу информацией (получают и передают) с дилером. Облачные серверы и все примыкающие к ним каналы связи доступны для стороннего наблюдателя (противника).
2) **Пользователь.** Хранит данные на облаке. Предполагается, что на облаке хранится k копий баз данных. Для загрузки данных пользователь обращается к дилеру, с которым соединен защищенным каналом связи. Этот канал недоступен для наблюдения.

3) **Клиенты.** Запрашивают некоторую информацию из базы данных. Обращаются для выполнения запроса к дилеру. С дилером соединены защищенными каналами связи.
4) **Дилер.** Находится вне облака, противнику недоступен. Дилер имеет защищенные каналы связи с пользователем и клиентами. Канал связи с облаком является незащищенным. Объем памяти дилера для постоянного хранения данных пренебрежимо мал по сравнению с n. Функции дилера:
   - аутентифицирует пользователя и клиентов;
   - получает от пользователя данные для размещения на облаке;
   - генерирует ключи шифрования/расшифрования;
   - осуществляет обмен данными с облаком в процессе размещения данных на облаке и в процессе обработки запроса;
   - выполняет шифрование/расшифрование;
   - производит сортировку шифротекстов.

Модель рассматривается в предположении, что на облаке имеется **противник**, который 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. Таким образом, запрос к одной из копий баз данных сравнивается с запросом к другой копии базы данных и собирается статистическая информация. Противник не имеет доступа к дилеру.

Под **угрозой** будем понимать угадывание исходного номера бита, запрашиваемого клиентом в базе данных.

Атака включает в себя:

1) Сбор информации противником в облаке (наблюдение, сбор статистики и анализ).
2) Фальшивые клиенты. Такие клиенты с помощью следующей процедуры могут сформировать произвольное число запросов, что позволит в коалиции с облаком собрать и проанализировать информацию для всей базы данных.

2.2 Постановка задачи и условные обозначения

Имеется база данных – бинарная строка $X = (x_1, \dots, x_n)$ длины n.

Клиент хочет получить один бит информации x_iс номером i из базы данных X так, чтобы противник не узнал ничего о том, с какой позиции i был запрошен бит.

Будем размещать на облаке k копий баз данных ($k \geq 2$). Пусть $k = 2^d$,где $d \geq 1$. Без потери общности предполагается, что $n = l^d$, то есть, $d = log_2 k$ и $l = \sqrt[d]{n}$. Пусть $L_p = \frac{n}{l}$.

Так как $l = \sqrt[d]{n}$, то $L_p \geq l$ или $\frac{n}{l} \geq l$, что эквивалентно $n \geq l^2$. Учитывая, что $n = l^d$, отсюда следует, что $d \geq 2$ и $k \geq 4$.

Пусть x – элемент группы Z_2, а K, K_{enc}, K_{dec} – ключи шифрования и расшифрования, alg - алгоритм шифрования или расшифрования, $h \in \{1, \dots, k\}$ – номер соответствующей копии базы данных. $DEnc(x, K, alg, h)$ - функция однозначного шифрования. $REnc(x, K_{enc}, alg, h)$ и $RDec(x, K_{dec}, alg, h)$ – функции вероятностного шифрования и расшифрования.

3. Размещение базы данных на облаке

3.1 Предварительные замечания

Для простоты изложения будем считать, что база данных загружается одним пользователем. Предполагается, что после загрузки база данных не может быть изменена. Также будем рассматривать случай запроса одного бита.

На начальном этапе пользователь производит загрузку базы данных. Он обращается к дилеру. Дилер проводит аутентификацию пользователя. Если полномочия пользователя подтверждены, то пользователь может передавать информацию дилеру.

Предполагается, что каналы связи с дилером защищены, поэтому пользователь передает данные в незашифрованном виде.

Построим матрицу M (размера $n \times d$)

$$M = \begin{pmatrix} m_{11}m_{12} & \cdots & m_{1d} \\ \vdots & \ddots & \vdots \\ m_{n1}m_{n2} & \cdots & m_{nd} \end{pmatrix}$$

с помощью следующей процедуры. Строка с номером i матрицы M имеет вид: в первом столбце m_{i1} стоит номер i, в остальных $d-1$ столбцах – различные натуральные числа из отрезка [1,n], которые сгенерированы при помощи датчика псевдослучайных чисел PRNG(*db*,*i*) и не равные i.

Для инициализации датчика случайных чисел задаются два параметра. Первый параметр *db* соответствует конкретной базе данных $X = (x_1, \dots, x_n)$ и хранится все время существования этой базы данных. Второй соответствует запрашиваемому номеру бита i. Эти параметры обеспечивают повторяемость датчика псевдослучайных чисел.

Строка матрицы M должна содержать различные числа. Если при генерации псевдослучайных чисел для строки матрицы M получаем число, которое было сгенерировано ранее, датчик *PRNG(db,i)* запускается до тех пор, пока не получим число, которое не было сгенерировано ранее. Таким образом, строка матрицы M будет содержать d различных чисел. Разные строки могут содержать одинаковые числа. Помимо этого, требуется, чтобы датчик псевдослучайных чисел *PRNG(db,i)* позволял бы восстановить любую строку матрицы M. Тогда дилеру нет необходимости хранить матрицу M в памяти.

Формирование матрицы M происходит следующим образом.

Построим l интервалов, в которые попадут все n элементов:

$[1, L_p], [L_p + 1{,}2 \cdot L_p], \dots, [(l-1) \cdot L_p + 1, l \cdot L_p]$.

Число элементов в каждом интервале $L_p = \frac{n}{l}$. Рассмотрим, например, первый интервал и первую строку, то есть $i = 1$. Датчик псевдослучайных чисел *PRNG(db,i)* генерирует случайные числа в диапазоне $[2, L_p]$. Числа в строке различны, всего получаем d чисел. Она состоит из элементов, $m_{11}, m_{12}, .., m_{1d}$ ($m_{11} = 1$, значения всех этих элементов попадают в диапазон $[1, L_p]$.

Далее для каждой строки из первого интервала вычисляется $l-1$ строк, принадлежащих другим интервалам. На основе каждой строки из первого интервала генерируется по одной строке для каждого из оставшихся интервалов. Эти строки заполняются значениями по алгоритму, представленному ниже.

Далее осуществляется переход к следующей строке первого интервала и алгоритм повторяется. То есть для строк с номерами $i \in \{1, \dots, L_p\}$ формируем соответственно по l строк для каждой строки i.

Сформируем вектор-столбец $\boldsymbol{b}$, состоящий из различных случайных натуральных чисел b_m значения которых лежат в диапазоне $[1, L_p], m \in \{1, \dots, l\}$. Числа b_m генерируются датчиком псевдослучайных чисел. На вход датчика *PRNG2(db,m)* подается номер интервала m, для которого вычисляется b_m. Вектор-столбец $\boldsymbol{b}$ восстанавливается каждый раз одинаково. Хранить вектор-столбец $\boldsymbol{b}$ нет необходимости, поскольку при одинаковом начальном значении параметра m датчик срабатывает одинаково.

3.2 Алгоритм построения матрицы *M*

На вход алгоритма подается: L_p, l, d.

```
for i ← 1 to L_p do
/* при помощи датчика псевдослучайных чисел PRNG(db,i)
   генерируется i-я строка первого интервала */
    for m ← 2 to l do
        z ← 0 /* инициализация глобальной переменной */
        for j← 1 to d do
            /* при помощи датчика псевдослучайных чисел
               PRNG2(db,1) генерируется b_1*/
            a ← (m_ij − b_1)modL_p
             /* при помощи датчика псевдослучайных чисел
               (db,m) генерируется b_m */
            c ← (a + b_m)modL_p
            if c = 0 then
                c ← L_p
            y ← L_p · (m − 1) + c
            if j = 1 then
                z ← y
            m_zj = y
```

В цикле по *i* последовательно генерируются L_p строк, где первым элементом является номер строки i, а остальные $d-1$ элементов отличны от *i* и различны между собой.

Для каждой из этих строк по приведенному выше алгоритму в цикле по *m* генерируются *l*-1 строк для интервалов с номерами от 2 до *l*. Каждая из формируемых *l*-1 строк попадает в свой интервал. В каждый интервал попадает ровно одна строка.

Все элементы *i*-й генерируемой строки различны. Если датчик псевдослучайных чисел *PRNG(db,i)* генерирует число, которое встречалось ранее в этой строке, то генерируется следующий элемент до тех пор, пока не получим элемент, не совпадающий с предыдущими.

Таким образом, на *i*-м шаге ($i \in [1, L_p]$) алгоритма вычисляется *l* строк, причем первой формируется *i*-я строка матрицы *M*. Остальные *l*-1 строк попадают случайным образом в различные интервалы (по одной строке в интервал), в зависимости от значения вектора-столбца $\boldsymbol{b}$. Вектор-столбец $\boldsymbol{b}$ позволяет вычислять номера строк в каждом интервале не по упрощенной формуле $y = L_p \cdot (m-1)$, что позволило бы противнику легко определить номера строк в каждом интервале. Благодаря вектору-столбцу $\boldsymbol{b}$ номер строки в каждом интервале определяется случайным образом.

Поскольку матрица *M* требует для хранения значительного объема памяти, дилер не хранит матрицу *M*. Элементы матрицы *M*, которые требуются для вычисления, формируются при помощи датчика псевдослучайных чисел *PRNG(db,i)*. Датчик псевдослучайных чисел при одинаковой инициализации генерирует одинаковую последовательность чисел и восстановление *i*-ой строки матрицы *M* возможно.

3.3 Построение матрицы *G*

Построим матрицу *G*, размера $n \times 2$

$$G = \begin{pmatrix} g_{11} g_{12} \\ \vdots \quad \vdots \\ g_{n1} g_{n2} \end{pmatrix}.$$

Первый столбец матрицы *G* генерируется как случайная последовательность из элементов 0 и 1.

Во второй столбец поместим корректирующие биты с помощью следующего алгоритма.

Каждая строка матрицы *M* состоит из *d* чисел. Каждое число можно рассматривать как номер, которому поставлен в соответствие случайным образом выбранный бит g_{i1}.

Положим:

$g_{i2} = \bigoplus_{j=1}^{d} g_{m_{ij}1} \oplus x_i$, где m_{ij} – элементы *i*-й строки матрицы *M*, а x_i – элемент бинарной строки $X = (x_1, \dots, x_n)$ длины *n*.

Тогда:

$x_i = \bigoplus_{j=1}^{d} g_{m_{ij}1} \oplus g_{i2}$.

Дилер осуществляет поэлементное шифрование матрицы *G* для отправки ее на облако. Шифротексты, полученные из элементов матрицы *G* хранятся на облаке. При хранении матрицы *G* на облаке для доступа к ее строкам будем использовать шифротекст, полученный из номера строки.

Шифротексты номеров строк матрицы *G* будем рассматривать как указатели на шифротексты ее строк.

Заметим, что поскольку в дальнейшем расшифровка шифротекста номера строки матрицы *G* не выполняется, в качестве алгоритма шифрования можно использовать одностороннюю функцию $DEnc(x, K_{enc}, alg, h)$, где K_{enc} является аргументом этой функции. Для возможности поиска по базе данных односторонняя функция должна быть однозначной, то есть без коллизий (различным открытым текстам соответствуют различные шифротексты).

Для шифрования матрицы *G* используется вероятностное шифрование. Дилер применяет вероятностное шифрование/расшифрование значения. Вероятностное шифрование позволяет для исходных значений 0 или 1 получить различные шифротексты в случае одинаковых исходных значений.

3.4 Размещение *k* копий базы данных на облаке

Для размещения *k* копий базы данных на облаке дилер генерирует:

1. *k* ключей $K(h)(h \in \{1, \dots, k\})$ для шифрования номера строки матрицы *G* (односторонняя функция) по одному для каждой копии базы данных;
2. *k* пар ключей $K_{enc}(h)$ и $K_{dec}(h)$, где $h \in \{1, \dots, k\}$, для вероятностного шифрования/расшифрования значений строк матрицы *G*, по одной паре для каждой копии базы данных.

Дилер шифрует номера строк матрицы *G* сгенерированными ключами $K(h)(h \in \{1, \dots, k\})$. Дилер шифрует матрицу *G* своим набором ключей для каждой копии базы данных.

Обозначим через $G_{enc}(h)(h \in \{1, \dots, k\})$, матрицу, где первый столбец – зашифрованные номера строк матрицы *G*, а второй и третий столбцы – зашифрованные столбцы матрицы *G* для *k* копий базы данных. Таким образом размер матрицы $G_{enc}(h)$ равен $n \times 3$.

Дилер осуществляет перестановку строк матриц $G_{enc}(h)$ $(h \in \{1, \dots, k\})$ путем сортировки их по 1 столбцу и передает матрицы $G_{enc}(h)$ облаку. Это необходимо, чтобы скрыть истинный порядок номеров для каждой копии базы данных. Поскольку для каждой копии базы данных используется свой ключ шифрования, порядок строк матриц будет различным.

Для выполнения шифрования и последующей перестановки дилеру требуется иметь объем памяти, достаточный для хранения одной базы данных. При этом предполагается, что дилер работает с несколькими базами данных и все эти базы данных одновременно дилер не хранит.

Так как матрицы $G_{enc}(h)$ хранятся на облаке, после загрузки базы данных на облако память дилера очищается для работы со следующей базой данных и в памяти дилера остаются только ключи.

В приведенном ниже алгоритме дилер должен выполнять Шаги 4-6 для каждой копии базы данных последовательно. Иначе ему придется хранить в памяти все *k* копий базы данных, что приводит к большим расходам памяти.

3.5 Описание алгоритма загрузки данных на облако

Шаг 1. Пользователь передает дилеру полученный массив номеров битов и значений, т.е. массив $X = (x_1, \dots x_n)$. Дилер аутентифицирует пользователя и принимает данные.

Шаг 2. Дилер формирует матрицу $G = \|g_{ij}\|$, где $i = 1,..,n, j = 1,2$.

Шаг 3. Дилер генерирует ключи $K(h)$, $K_{enc}(h)$ и $K_{dec}(h)$, где $h \in \{1, \dots, k\}$.

Ключ $K(h)$ (реализующий одностороннюю функцию) предназначен для шифрования номера строки матрицы *G*.

Открытый $K_{enc}(h)$ и секретный $K_{dec}(h)$ ключи предназначены для вероятностного шифрования/расшифрования значений столбцов матрицы *G*.

Шаг 4. Дилер шифрует ключом $K(h)$ номер строки матрицы *G* $DEnc(g_{i1}, K(h), alg, h)$, где $i = 1,..,n,\ h \in \{1, \dots, k\}$ и получает шифротексты, являющиеся указателями на строки матрицы *G*.

Дилер шифрует ключом $K_{enc}(h)$ столбцы матрицы *G*: $REnc(g_{ij}, K_{enc}(h), alg, h)$ где $i = 1,..,n, j = 1,2, h \in \{1, \dots, k\}$ и получает шифротексты, соответствующие элементам столбцов матрицы *G* для каждой копии базы данных.

Дилер получает матрицу $G_{enc}(h)$, где $h \in \{1, \dots, k\}$.

Шаг 5. Матрицу $G_{enc}(h)$, где $h \in \{1, \dots, k\}$, дилер сортирует по первому столбцу.

Шаг 6. Дилер загружает матрицу $G_{enc}(h)$, где $h \in \{1, \dots, k\}$ на облако.

Шаг 7. Шаги 4-6 дилер выполняет в цикле для каждой копии базы данных ■.

Число шифротекстов, переданных каждой из k копий базы данных равно $3n$.

4. Описание выполнения алгоритма запроса клиента дилером

4.1 Использование множества номеров в запросе для сокрытия номера *i*

Пусть клиент интересуется элементом $x_i \in X$.

С целью сокрытия номера, запрашиваемого клиентом, запрос дилера к облаку будет формироваться как множество номеров.

Поскольку предполагается наличие фальшивых клиентов, запрашивающих конкретный номер, запрос будет выполнен следующим образом: дилер передает в запросе к облаку не только этот номер (в зашифрованном виде), а также множество номеров, содержащих запрашиваемый номер.

Напомним, что отрезок [1,n] разбивается на l интервалов по $L_p = \frac{n}{l}$ элементов в каждом интервале.

По построению матрицы *M* каждой строке *i* соответствует единственная строка из первого интервала. Соответственно, каждому элементу m_{i1} матрицы M соответствует строка из первого интервала, по которой *i*-я строка была построена. Таким образом, по строке *i* может быть полностью восстановлена искомая строка из первого интервала. После восстановления строки из первого интервала по алгоритму построения матрицы M можно восстановить те *l*-1 строк, которые формируются на основе искомой строки из первого интервала. Столбцы этих восстановленных *l* строк формируют множества $Set_{ij}, j = 1, \dots, d$.

Для запрашиваемого *i*-го бита дилер формирует множество номеров $Set_i = \cup Set_{ij}$, содержащее *i*-ю строку матрицы *M*.

Дилер шифрует и отправляет облаку запрос, включающий в качестве параметра это множество. Далее после ответа облака, дилер получает в качестве ответа также множество значений.

Для построения множества Set_i используем следующую процедуру, которая основана на восстановлении l строк матрицы *M*. Для восстановления строк матрицы *M* по элементу m_{i1} необходимо найти:

- порядковый номер интервала (*w*), в который попал этот элемент для определения элемента вектора-столбца b_w;
- порядковый номер элемента в интервале для вычисления числа *a*;
- порядковый номер элемента, соответствующего элементу m_{i1} в первом интервале, поскольку восстановление матрицы *M* выполняется на базе строк первого интервала.

Приведем m_{i1} (номер искомого бита) по модулю L_p. Обозначим через $r = m_{i1}\ mod L_p$ результат приведения.

Определяется номер интервала, в который попало число m_{i1}. По определению матрицы *M* номер интервала определяется индексом *i*.

$$w = \left\lfloor \frac{m_{i1} - 1}{L_p} \right\rfloor + 1$$

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

$$m'_{i1} = \begin{cases} r, \text{если } r \neq 0 \\ L_p, \text{если } r = 0 \end{cases}$$

порядковый номер в интервале.

Пусть $a = (m'_{i1} - b_w) mod L_p$. В этом случае номер строки i' в первом интервале вычисляется по формуле $i' = (a + b_1) mod L_p$, если i'=0, то i'=L_p. Для номера строки i' при помощи датчика псевдослучайных чисел *PRNG(db,i)* получим *d-1* различных чисел от 1 до L_p аналогично тому, как строилась матрица *M*.

Для номера строки i' из диапазона $[1, L_p]$, по аналогии с алгоритмом генерации матрицы *M*, построим *l*-1 строк.

Столбцами этих l строк будут множества $Set_{ij}, j = 1, \dots, d$, где множество Set_{i1} содержит элемент m_{i1}.

Мощность множества $|Set_{ij}|$ равна *l*. Следовательно, мощность множества $|Set_i|$ не более $l \cdot d$.

4.2 Описание алгоритма построения множества Set_i

На вход алгоритма подается номер запрашиваемого бита *i*. На выходе алгоритм формирует множество Set_i.

Шаг 1. Дилер находит номер интервала *w*, в который попал номер запрашиваемого бита *i* и его порядковый номер в этом интервале m'_{i1}.

Шаг 2. Дилер находит число a для номера интервала w, куда попал номер *i*: $a = (m'_{i1} - b_w) mod L_p$.

Шаг 3. Дилер находит номер строки i' в диапазоне $[1, L_p]$ по формуле: $i' = (a + b_1) mod L_p$, если i'=0, то i'=L_p.

Шаг 4. Дилер при помощи датчика псевдослучайных чисел *PRNG(db,i)* восстанавливает строку матрицы *M* для номера i', состоящую из *d* различных натуральных чисел, аналогично алгоритму построения строки матрицы *M*.

Шаг 5. Дилер строит множество: $Set_i = \cup Set_{ij}$, где $j = 1, \dots, d$ ■.

4.3 Запрос клиентом *i*-го бита. Предварительные замечания

Запрос клиента на получение данных поступает дилеру. Дилер проверяет полномочия клиента и, если они подтверждены, то выполнение запроса будет санкционировано.

Противник в облаке собирает статистику запросов. Также предполагается наличие фальшивых клиентов, которые могут вступать в коалицию с облаком. Поэтому дилер скрывает запрашиваемый клиентом номер среди множества номеров.

При запросе клиентом *i*-го бита дилер строит множество Set_i, которое содержит элементы *i*-й строки матрицы *M*. Напомним, что в *i*-й строке элемент m_{i1} является искомым номером. Множество Set_i является избыточным, чтобы скрыть от противника информацию о запрашиваемом бите.

При генерации множества Set_i дилер при помощи датчика псевдослучайных чисел *PRNG(db,i)* восстанавливает *i*-ю строку матрицы *M*, состоящую из *d* различных натуральных чисел.

Из множества элементов Set_i только элементы *i*-й строки матрицы *M* нужны дилеру для получения значения *i*-го бита.

Напомним, что физически в облаке хранятся матрицы $G_{enc}(h), h \in \{1, \dots, k\}$. Строка матрицы $G_{enc}(h)$ состоит из шифротекстов номера строки, произвольного и корректирующего битов.

Разобьем множества Set_i на непересекающиеся подмножества.

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. Объединение множеств Set_i^h содержит все элементы множества Set_i, то есть $Set_i = \cup\, Set_i^h$, где $h \in \{1, \dots, k\}$.

Далее каждое множество Set_i^h шифруется ключом $K(h)$, полученные шифротексты сортируются для каждого множества Set_i^h отдельно. В результате получим множества $Set_i^{enc(h)}$ – зашифрованные соответствующим ключом $K(h)$ и затем отсортированные.

Для каждого множества $Set_i^{enc(h)}$ исходному номеру бита будет соответствовать номер шифротекста в перестановке. Это соответствие необходимо запомнить и хранить на время выполнения запроса, чтобы дилер мог восстановить значение запрашиваемого бита.

Дилер на время выполнения запроса формирует матрицу S_{num} из d строк и трех столбцов $(i = 1, \dots, d, j = 1,2,3)$.

$$S_{num} = \begin{pmatrix} s_{11} & s_{12} & s_{13} \\ \vdots & \vdots & \vdots \\ s_{d1} & s_{d2} & s_{d3} \end{pmatrix}.$$

Первым столбцом матрицы S_{num} являются элементы восстановленной *i*-ой строки матрицы *M*.

Ранее множество Set_i, содержащее элементы *i*-й строки матрицы *M*, было разбито на множества Set_i^h. Для каждого множества Set_i^h, а следовательно, для элементов *i*-й строки матрицы *M* были выбраны копии базы данных. Второй столбец матрицы S_{num} содержит выбранные номера копий баз данных для элементов первого столбца матрицы S_{num}.

Третий столбец матрицы S_{num} – номер шифротекста в перестановке множества $Set_i^{enc(h)}$ для соответствующего элемента первого столбца. Таким образом, s_{j3} является функцией от трех параметров, которая позволяет получить соответствие между исходным номером и номером шифротекста в перестановке:

$$s_{j3} = F\left(s_{j1}, s_{j2}, Set_i^{enc(s_{j2})}\right).$$

Матрица S_{num} позволяет дилеру после получения ответа от облака без выполнения расшифрования сразу отбросить данные, кроме данных, необходимых для выполнения запроса.

Матрица S_{num} формируется у дилера и известна только дилеру. Матрица S_{num} является секретной.

На k копий базы данных дилер отправляет в общей сложности $l \cdot d$ шифротекстов для элементов множества Set_i.

После выполнения запроса дилер при помощи матрицы S_{num} определяет значения элементов для строки матрицы *M* из копий базы данных. После расшифрования значений, дилер определяет значение запрашиваемого бита и отправляет его клиенту.

4.4 Подготовка дилером запроса к облаку

На входе алгоритм получает номер запрашиваемого бита *i*. На выходе дилер получает зашифрованные и отсортированные множества $Set_i^{enc(h)}$ и матрицу S_{num}.

Шаг 1. Клиент обращается к дилеру и передает ему номер бита *i*, значение которого хочет получить. Дилер аутентифицирует клиента.

Шаг 2. Дилер восстанавливает *i*-ю строку матрицы *M* и генерирует множество Set_i.

Шаг 3. Дилер на время выполнения запроса *i*-го бита резервирует память для матрицы S_{num} из d строк и трех столбцов $(i = 1, \dots, d, j = 1,2,3)$.

Шаг 4. Дилер заполняет первый столбец матрицы S_{num} элементами *i*-й строки матрицы *M*.

Шаг 5. Дилер выполняет разбиение множества Set_i на непересекающиеся подмножества Set_i^h путем случайного и равномерного выбора номера копии базы данных для каждого элемента множества Set_i.

Дилер заполняет второй столбец матрицы S_{num} номерами копий базы данных, соответствующих номерам в первом столбце.

Шаг 6. Дилер шифрует элементы множеств Set_i^h в соответствии с выбранной копией базы данных ключом $K(h), h \in \{1, \dots, k\}$, получает шифротексты и сортирует их (множества $Set_i^{enc(h)}$).

Шаг 7. Дилер заполняет третий столбец матрицы S_{num} номерами шифротекстов в перестановке множеств $Set_i^{enc(h)}$ для элементов первого столбца ■.

4.5 Выполнение запроса дилера к облаку и ответ облака

На вход алгоритма подаются зашифрованные и отсортированные множества $Set_i^{enc(h)}$. На выходе дилер получает шифротексты значений битов и корректировочных битов для элементов множеств $Set_i^{enc(h)}$.

Шаг 1. Для каждой копии базы данных дилер отправляет на облако отсортированные шифротексты номеров битов для множеств $Set_i^{enc(h)}$.

Шаг 2. Для запрошенных шифротекстов номеров битов облако возвращает дилеру шифротексты значений битов и корректировочных битов (второй и третий столбцы матрицы $G_{enc}(h)$, где $h \in \{1, \dots, k\}$). Порядок возвращаемых шифротекстов соответствует исходной сортировке шифротекстов для множеств $Set_i^{enc(h)}$ ■.

4.6 Обработка дилером ответа облака

На входе дилер получает второй и третий столбцы матриц $G_{enc}(h)$ для шифротекстов множеств $Set_i^{enc(h)}$. На выходе значение *i*-го бита.

Шаг 1. Дилер при помощи матрицы S_{num} выбирает зашифрованные значения бита и корректировочного бита для номеров первого столбца матрицы S_{num}. Остальные шифротексты отбрасываются.

Шаг 2. Дилер расшифровывает значения битов для номеров первого столбца матрицы S_{num} и корректировочный бит для элемента s_{11} матрицы S_{num}. По построению матрицы S_{num} запрашиваемый номер является элементом s_{11}.

Шаг 3. Дилер использует формулу $x_i = \bigoplus_{j=1}^{d} g_{m_{ij}1} \oplus g_{i2}$ для вычисления значения *i*-го бита и отправляет клиенту полученное значение ■.

5. Оценка требуемой памяти и сложности алгоритма

5.1 Объем информации, который необходимо хранить дилеру

В процессе работы на время загрузки базы данных дилер генерирует k ключей $K_{enc}(h)$. Эти ключи служат для вероятностного шифрования значений битов. После загрузки ни ключи, ни шифротексты дилер не хранит.

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

- значение для инициализации датчиков случайных чисел *PRNG(db,i)* и *PRNG2(db,m)*;
- k ключей $K(h), h \in \{1, \dots, k\}$ для шифрования односторонней функцией;
- k ключей $K_{dec}(h)$, где $h \in \{1, \dots, k\}$ для вероятностного расшифрования значений битов.

Заметим, что на практике число *n* битов в базе данных традиционно предполагается 2^{30} - 2^{40}, а число копий баз данных k не превышает 16 (то есть 2^4). Таким образом, k значительно меньше n, а хранение k ключей для шифрования номеров битов и k ключей для расшифрования значений битов занимает $2 \cdot k \cdot l_{key}$, где l_{key} - длина ключа. Длина ключа на практике чаще всего ограничена 256 байтами [13]. Следовательно, дилер хранит объем информации значительно меньше *n*.

5.2 Основные результаты

Утверждение 1. Общая коммуникационная сложность для получения значения номера бита без раскрытия его номера равна $l \cdot d \cdot 3$ шифротекстов.

Доказательство. Дилер посылает копиям базы данных $l \cdot d$ (где $l = \sqrt[d]{n}$) шифротекстов для запроса значений. Облако отвечает $l \cdot d \cdot 2$ шифротекстами значений ■.

Проанализируем вероятность угадывания противником номера запрашиваемого бита.

Если запрошены два номера бита, то они могут принадлежать либо одному, либо разным множествам Set_i. Противник может увидеть, выполнен ли запрос к одному из множеств Set_i, на которые разбита база данных или к разным. Если запросы выполнены к одному множеству, то они неразличимы. Таким образом, если противник совершает n или более запросов, он не узнает номер бита. Максимум, какую информацию противник может получить - такое подмножество множества Set_i, что при запросе к каждому элементу из подмножества, на облаке осуществляется доступ ко всем битам из множества Set_i и только к ним. Что будет означать, что противник не знает, а только угадывает, какой именно бит из данного множества был выбран ■.

Заметим, что так как любым двум элементов x_{i_1} и x_{i_2}, $x_{i_1}, x_{i_2} \in Set_{i1}$ соответствует единственная строка из первого интервала, то множества $Set_{i2}, \dots, Set_{id}$ совпадают для элементов x_{i_1} и x_{i_2}. Следовательно, для всех элементов множества Set_{i1} элементы множества Set_i совпадают. Поэтом запросы к облаку будут неотличимы для всех элементов множества Set_{i1}.

Утверждение 2. При запросе фиктивными клиентами не менее *n* номеров битов и наличии пассивного противника на облаке вероятность угадывания противником номера бита не более $\frac{1}{l}$.

Доказательство. Множество Set_{i1} ($Set_{i1} \subset Set_i$) порождается одинаково для всех l чисел, в него входящих. То есть существует одинаковое множество для l различных номеров битов. Вероятность угадывания номера из множества Set_{i1} равна $\frac{1}{l}$, так как Set_{i1} имеет мощность l.

Выполнив n запросов, противник определяет элементы, входящие в каждое множество Set_i (в предположении, что $k < ld$).

Если фиктивные клиенты будут выполнять любое количество запросов большее n, противник все равно не получит никакой дополнительной информации.

В этом случае вероятность угадывания номера *i* будет не более $\frac{1}{l}$ ■.

Так как значение k по сравнению с n мало, хранение k пар ключей для шифрования/расшифрования значений битов не требует много памяти и занимает $2 \cdot k \cdot l_{key}$.

Докажем, что предложенный протокол удовлетворяет условию лаконичности.

Теорема. Предложенная схема организации базы данных размера n и запроса к ней удовлетворяет следующим свойствам.

На время загрузки базы данных дилер генерирует:

- k ключей $K_{enc}(h)$ для вероятностного шифрования значений битов, которые после окончания загрузки не хранятся.

На протяжении всего времени существования базы данных дилер хранит только следующую информацию для данной базы данных объема *n*:

- значение для инициализации датчиков случайных чисел *PRNG(db,i)* и *PRNG2(db,m)*;
- k ключей $K(h), h \in \{1, \dots, k\}$ для шифрования односторонней функцией;
- k ключей $K_{dec}(h)$, где $h \in \{1, \dots, k\}$, для вероятностного расшифрования значений битов.

Объем хранимой информации много меньше n.

Коммуникационная сложность запроса $l \cdot d \cdot 3$ шифротекстов.

При атаке с использованием фальшивых клиентов, выполняющих любое число запросов и предположении о наличии пассивного противника на облаке вероятность угадывания номера бита противником не более

$$\frac{1}{l}.$$

Доказательство. Из Утверждения 1 следует, что коммуникационная сложность запроса равна $l \cdot d \cdot 3$ шифротекстов.

Из Утверждения 2 следует, что при любом числе запросов номеров битов фиктивными клиентами и наличии пассивного противника на облаке вероятность угадывания номера бита не более $\frac{1}{l}$ ■.

Комментарий. Так как k – число копий баз данных, и это число мало на практике, а дилер хранит не более $O(k)$ информации, то дилер в состоянии обслуживать запросы к значительному числу баз данных. То факт, что $k \ll n$ позволяет эффективно использовать облако как место хранения значительного объема информации.

Список литературы / References

[1] Chor B., Goldreich O. et al. Private Information Retrieval. In Proc. of the IEEE Annual Symposium on Foundations of Computer Science, 1995, pp. 41-50.

[2] Chor B., Goldreich O. et al. Private Information Retrieval. Journal of the ACM, vol. 45, no. 6, 1998, pp. 965-982.

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

[4] Yekhanin S. Locally Decodable Codes and Private Information Retrieval Schemes. Springer-Verlag Berlin Heidelberg, 2010, 82 p.

[5] Kushilevitz E., Ostrovsky R. Replication is not needed: Single database, computationally-private information retrieval (extended abstract). In Proc. of the 38th Annual Symposium on Foundations of Computer Science, 1997, pp. 364-373.

[6] Kushilevitz E., Ostrovsky R. One-way trapdoor permutations are sufficient for non-trivial single-server private information retrieval. Lecture Notes in Computer Science, vol. 1807, 2000, pp. 104-121.

[7] Ostrovsky R., Skeith III W. E. A Survey of Single-Database Private Information Retrieval: Techniques and Applications. Lecture Notes in Computer Science, vol. 4450, 2007, pp. 393-411.

[8] Aguilar-Melchor C., Barrier J., Fousse L. XPIR: Private Information Retrieval for Everyone, Proceedings on Privacy Enhancing Technologies Symposium, 2016, issue 2, pp. 155-174.

[9] Варновский Н.П., Нестеренко Ю.В., Ященко В.В. Введение в криптографию. Из-во МЦНМО, 2012, 348 стр. / Varnovsky N.P., Nesterenko Yu.V., Yashchenko V.V. Introduction to cryptography. MCCME, 2012, 348 p. (in Russian).

[10] Demmler D., Herzberg A., Schneider T. RAID-PIR: Practical multi-server PIR. In Proc. of the 6th edition of the ACM Workshop on Cloud Computing Security, 2014, pp. 45-56.

[11] Angel S., Chen H. et al. PIR with compressed queries and amortized computation. In Proc. of the IEEE Symposium on Security and Privacy, 2018, pp. 1-18.

[12] "MuchPIR Demo". URL: https://github.com/ReverseControl/MuchPIR, accessed 22.08.2022.

[13] Wahid M.N.A., Ali A. et al. A Comparison of Cryptographic Algorithms: DES, 3DES, AES, RSA and Blowfish for Guessing Attacks Prevention. Journal of Computer Science Applications and Information Technology, vol. 3, no. 2, 2018, pp. 1-7.

Информация об авторах / Information about authors

Сергей Анатольевич МАРТИШИН – кандидат физико-математических наук, научный сотрудник отдела теоретической информатики. Его научные интересы включают параллельные алгоритмы, базы данных, облачные вычисления,

Sergey Anatolievich MARTISHIN – PhD, researcher of the Department of Theoretical Computer Science. His research interests include parallel algorithms, databases, cloud computing.

Марина Валерьевна ХРАПЧЕНКО – научный сотрудник отдела теоретической информатики. Её научные интересы включают параллельные алгоритмы, базы данных, облачные вычисления,.

Marina Valerievna KHRAPCHENKO – researcher of the Department of Theoretical Computer Science. Her research interests include parallel algorithms, databases, cloud computing.

Александр Владимирович ШОКУРОВ – кандидат физико-математических наук, доцент, заведующий отделом теоретической информатики. Сфера научных интересов: алгебраические структуры в полях Галуа, базисы Гребнера, модулярная арифметика, нейрокомпьютерные технологии, цифровая обработка сигналов, криптографические методы защиты информации.

Alexander Vladimirovich SHOKUROV – PhD of Physical and Mathematical Sciences, Professor, Head of the Department of Theoretical Computer Science. Research interests: algebraic structures in the Galois fields, modular arithmetic, neurocomputer technologies, Grobner bases, digital signal processing, cryptographic methods for protecting information.