DOI: 10.15514/ISPRAS-2023-35(5)-3

Организация конфиденциальных запросов к облаку

[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. В статье исследуется известная криптографическая задача получения клиентом данных из базы, размещенной на сервере, таким образом, чтобы никто из имеющих доступ к серверу, кроме самого клиента, не смог получить информацию о содержании этого запроса. Задача, известная как PIR (Private Information Retrieval), была сформулирована в информационно-теоретической постановке в 1995 году Шором, Голдрайхом, Кушелевицем и Суданом. Предложена модель облачных вычислений, включающая облако, центр аутентификации, пользователя, клиентов, доверенное лицо (дилера), активного противника, работающего по протоколу, на облаке. Предполагается, что у атакующей стороны имеется возможность создания фальшивых клиентов для формирования неограниченного числа запросов. Предложен алгоритм размещения базы данных на облаке и алгоритм запроса требуемого бита. Применяется инъективное преобразование номеров битов, представленных в l-ичной системе счисления словами длины d, в слова без повторяющихся цифр той же длины с алфавитом из $\hat{\boldsymbol{l}}$ цифр, то есть $\{\mathbf{0},\dots,\boldsymbol{l}-\mathbf{1}\}^{\boldsymbol{d}} \to \{\mathbf{0},\dots,\hat{\boldsymbol{l}}-\mathbf{1}\}^{\boldsymbol{d}}$, что позволяет уменьшить вероятность угадывания противником номера бита. Приведены оценки коммуникационной сложности и вероятности раскрытия запрашиваемого бита с учетом выполненного преобразования.

Ключевые слова: базы данных; облачные вычисления; PIR.

Для цитирования: Варновский Н.П., Мартишин С.А., Храпченко М.В., Шокуров А.В. Организация конфиденциальных запросов к облаку. Труды ИСП РАН, том 35, вып. 5, 2023 г., стр. 37–54. DOI: 10.15514/ISPRAS–2023–35(5)–3.

Благодарности: Работа выполнена при финансовой поддержке Российской Федерации в лице Минобрнауки России (соглашение № 075-15-2020-788) и ИСП РАН. Авторы выражают особую благодарность Лазареву Д. О. за ценные замечания в процессе работы над статьей.

About Cloud Request Protection

[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 article examines the well-known cryptographic problem of obtaining data from a database by a client so that no one with access to the server except the client himself could obtain information about this request. This problem known as PIR (Private Information Retrieval) was formulated in 1995 by Chor, Goldreich, Kushilevitz and Sudan in the information-theoretic setting. A model of cloud computing is proposed. It includes a cloud, an authentication center, a user, clients, trusted dealer, an active adversary executing the protocol in the cloud. 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. An injective transformation of bit numbers represented in the *l*-ary number system by words of length *d* into words without repeating digits of the same length with an alphabet of $\hat{\boldsymbol{l}}$ digits is used, i.e. a transformation $\{\mathbf{0}, \dots, \boldsymbol{l}-\mathbf{1}\}^{\boldsymbol{d}} \to \{\mathbf{0}, \dots, \hat{\boldsymbol{l}}-\mathbf{1}\}^{\boldsymbol{d}}$ was constructed. This transformation reduces the probability of disclosure of the requested bit number. The communication complexity and probability of revealing required bit were estimated, taking into account the performed transformation.

Keywords: database; cloud computing; PIR.

For citation: Varnovskiy N.P., Martishin S.A., Khrapchenko M.V., Shokurov A.V. About cloud request protection. *Trudy ISP RAN/Proc. ISP RAS*, vol. 35, issue 5, 2023. pp. 37-54 (in Russian). DOI: 10.15514/ISPRAS-2023-35(5)-3.

Acknowledgements. This work was funded by Russian Federation represented by the Ministry of Science and Higher Education (grant number 075-15-2020-788) and ISP RAS. The authors are especially grateful to D.O. Lazarev for valuable comments during the work on the article.

1. Введение

Задача (PIR – Private Information Retrieval) в информационно-теоретической постановке была сформулирована в 1995 году Шором, Голдрайхом, Кушелевицем и Суданом [1]. Задача заключается в получении 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 при наличии активного противника, работающего по протоколу через создаваемых фальшивых клиентов и за счет этого получающего больше информации.

Классическая постановка задачи PIR:

- имеется база данных – бинарная строка $X = (x_1, \dots, x_n)$ длины n, хранящаяся на сервере в облаке;
- клиент хочет получить один бит информации x_i из базы данных X так, чтобы сервер не смог определить, с какой позиции i был запрошен бит.

В работах [1-2] было показано, что при размещении базы данных на единственном сервере с пассивным противником, конфиденциальность запроса может быть обеспечена только тогда, когда клиент запрашивает базу данных целиком. Однако такой подход на практике потребует значительных ресурсов (времени скачивания информации, памяти для ее загрузки и пр.), поэтому исследования были сосредоточены на поиске решений, в которых коммуникационная сложность протокола должна быть существенно меньше размера базы

данных. Под коммуникационной сложностью протокола понимают общее количество бит, которыми обмениваются участники протокола за время его работы.

В работах [1-2] была предложена модель реплицированной базы данных с размещением нескольких копий строки $X = (x_1, \dots, x_n)$ 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. Работы [3-4] были сосредоточены на изучении методов уменьшения коммуникационной сложности, в работах [5-8] были исследованы возможности построения схемы PIR на одном сервере при помощи различных техник, в том числе гомоморфного шифрования и оценки коммуникационной сложности предложенной схемы.

Схемы реализации PIR исследовались и для облачных вычислений. Например, в [9] предложена реализация аналогичная построению систем RAID (Redundant Array of Inexpensive Disks), где каждый сервер хранит только часть базы данных, а серверы могут быть размещены на разных платформах и иметь разных провайдеров. Предполагается, что часть из них не вступает в сговор. Серверы могут быть опрошены параллельно.

Ранее авторами в [10] была рассмотрена задача, в которой в отличие от классического PIR реплицированная база данных хранится на облаке. В этом случае ни владелец данных (пользователь), ни клиент (имеющий официальный доступ для получения информации из базы данных) не могут контролировать облачную коммуникационную среду. Таким образом, в облачных информационных системах нельзя обеспечить изолированность копий распределенной базы данных друг от друга. Кроме того, говоря о хранении и использовании конфиденциальной информации, требуется знать, с каким противником, активным или пассивным, приходится иметь дело.

Активный противник в облаке, который способен вмешиваться в ход выполнения криптографического протокола, может быть выявлен достаточно быстро путем полного анализа результатов однократного выполнения криптографического протокола. Однако активный противник может работать по протоколу и проводить атаки не нарушая протокол.

Пассивный противник получает некоторую информацию о процессе выполнения криптографического протокола путем наблюдения, но не вмешивается в процесс выполнения протокола. Пассивный противник не может быть обнаружен даже при исчерпывающем анализе результатов многократного выполнения криптографического протокола.

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

Ниже будут рассмотрена модель, основанная на представлении номера бита в системе счисления, зависящей от числа битов и количества копий базы данных на облаке. Предложены алгоритмы представления номера бита в новой системе счисления, формирования запроса к облаку в виде множества шифротекстов, обработка ответа облака и получение значения искомого бита. Даны оценки коммуникационной сложности алгоритмов и вероятности угадывания противником номера запрошенного бита.

Вводится инъективное преобразование номеров битов, представленных в l-ичной системе счисления словами длины d, в слова без повторяющихся цифр той же длины с алфавитом из $\hat{l}$ цифр, то есть $\{0, \dots, l-1\}^d \to \{0, \dots, \hat{l}-1\}^d$. Это позволяет уменьшить вероятность угадывания противником номера бита. Приведены оценки коммуникационной сложности и вероятности раскрытия запрашиваемого бита с учетом выполненного преобразования.

2. Модель вычислений, основные определения и постановка задачи

2.1 Состав модели

Модель организации хранения и запросов к данным, предложенная в данной работе, имеет ряд отличий по сравнению с моделью, предложенной в [10]. В модель введен центр аутентификации пользователей и клиентов, через который посредством стандартных протоколов осуществляется их аутентификация. Центр аутентификации связан с дилером каналом связи, использующим стандартный криптографический протокол для обмена зашифрованными данными. Кроме того, вместо защищенных каналов [10] для связи между дилером и пользователями/клиентами достаточно иметь каналы связи, использующие стандартный криптографический протокол обмена зашифрованными данными.

Состав модели:

Облако. Состоит из нескольких серверов, на каждом из которых хранится копия одной и той же базы данных. Эти копии отличаются друг от друга при использовании различных ключей шифрования. Облачные серверы способны выполнять любые вычислительные операции, присущие системам управления базами данных. 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).

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 конфиденциальной передачи информации.

Клиенты. Запрашивают некоторую информацию из базы данных. Обращаются для выполнения запроса к дилеру. С дилером соединены каналами связи, использующими стандартный криптографический протокол конфиденциальной передачи информации.

Центр аутентификации. Аутентифицирует пользователя и клиентов.

Дилер. Находится вне облака, противнику недоступен. Канал связи с облаком является незащищенным. Объем памяти дилера для постоянного хранения данных пренебрежимо мал по сравнению с n. Функции дилера при работе с пользователем

- вырабатывает совместно с пользователем секретный ключ для шифрования базы данных;
- получает от пользователя данные для размещения на облаке в зашифрованном виде.

Функции дилера при работе с клиентом:

- публикует открытые ключи шифрования.

При формировании запроса к облаку дилер:

- осуществляет обмен данными с облаком в процессе размещения данных на облаке и в процессе обработки запроса;
- выполняет расшифрование запроса;
- производит сортировку шифротекстов.

В дальнейшем для простоты будем считать, что база данных загружается одним пользователем. Предполагается, что после загрузки база данных не может быть изменена. Либо изменена полностью. Также будем рассматривать случай запроса одного бита.

На начальном этапе пользователь обращается в центр аутентификации. Центр проводит аутентификацию пользователя. Если полномочия пользователя подтверждены, то пользователь может передавать информацию дилеру и производить загрузку базы данных на облако.

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 (активный противник), работающих по протоколу. То есть противник не только пассивно наблюдает, но и сам может производить запросы с помощью созданных фальшивых клиентов. Противник производит атаку с известными открытыми запросами. Противник может заставить 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

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

Клиент хочет получить один бит информации x_i с 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 ($k \geq 4$). Пусть $k = 2^d$, где $d \geq 2$. 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 $l = \sqrt[d]{n}$, округлим l до целого числа в большую сторону.

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.

Пусть $Len(K)$ – длина шифротекста в битах при шифровании номера бита БД в облаке односторонней функцией. Такая функция необратима, то есть при ее использовании задача восстановления номера бита по шифротексту является вычислительно сложной.

Пусть $Len(K_{enc})$ – длина шифротекста в битах при вероятностном шифровании значений битов БД в облаке. Пусть $Len(K, K_{enc})$ – сумма длин $Len(K)$ и $Len(K_{enc})$.

3. Алгоритм преобразования слов в слова без повторяющихся букв

3.1 Постановка задачи

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: $i = a_0^{(i)} + a_1^{(i)} l + \dots + a_{d-1}^{(i)} l^{d-1}$. Цифры l-ичной записи представляют собой кортежи $\left(a_0^{(i)}, \dots, a_{d-1}^{(i)}\right)$ длины d. Эти d элементов кортежа можно интерпретировать как точку с целочисленными координатами в кубе размерности d и длиной стороны l, где $l=\sqrt[d]{n}$. На каждом месте в строке стоит цифра в l-ичной системе счисления от 0 до l-1. Такое множество

точек куба размерности d можно рассматривать как множество слов длины d с алфавитом $(0,\dots,l-1)$.

Обозначим через $N_d(l)$ множество слов длины d с алфавитом $(0,\dots,l-1)$, т. е. множество $\{0,\dots,l-1\}^d$. Элементы множества $N_d(l)$ можно рассматривать как номера чисел от 0 до l^d-1, то есть l-ичную запись d-разрядного числа.

Обозначим через $\widehat{N}_d(l)$ – множество слов длины d с алфавитом $(0,\dots,l+d-2)$, где все буквы в слове различны. Элементы множества $\widehat{N}_d(l)$ можно рассматривать как подмножество номеров битов от 0 до $(l+d-1)^d-1$, где все цифры числа различны.

Построим отображение f: $N_d(l)\to\widehat{N}_d(l)$, то есть преобразуем слова множества $N_d(l)$ в слова множества $\widehat{N}_d(l)$, все буквы в которых различны.

Ниже будет доказано, что отображение f является инъективным.

Значение функции $f(a)$ для каждого $a\in N_d(l)$ будет определяться матрицей $A(a)$ размера $d\times 3$. Зафиксировав элемент a, будем эту матрицу обозначать через A.

3.2 Первый этап построения матрицы A

Сопоставим каждому элементу $a=(a_0,\dots,a_{d-1})$ (d-разрядной l-ичной записи числа) из $N_d(l)$ матрицу A размера $d\times 3$ с помощью следующей процедуры. На первом этапе строим матрицу A_0, размера $d\times 2$, первым столбцом которой будет последовательность $0,1,\dots,d-1$. Второй столбец заполним последовательно цифрами $a_0,\dots,a_{d-1}$ в l-ичном представлении числа a. Далее упорядочим строки полученной матрицы A_0 по неубыванию элементов второго столбца. При наличии последовательностей совпадающих элементов во втором столбце, содержащие эти элементы строки упорядочиваем по возрастанию элементов первого столбца. Получаем матрицу A_1.

3.3 Второй этап построения матрицы A - алгоритм формирования третьего столбца матрицы A

На вход алгоритма подается матрица A_1. На выходе алгоритма получаем матрицу A размера $d\times 3$ (листинг 1).

```
// вводим счетчик c, начальное значение c = l − 1
c ← l − 1
// вводим индикатор σ, начальное значение σ = −1
σ ← −1
// последовательно перебираем строки матрицы A1
for i ← 1 to d do
// для i-ой строки матрицы A1 выбираем элемент a_{i,2}, и
// сравниваем со значением σ.
// если σ и a_{i,2} различны, то положим σ = a_{i,2} и a_{i,3} = a_{i,2}
      if σ ≠ a_{i,2} then
          σ ← a_{i,2}
          a_{i,3} ← a_{i,2}
// если σ = a_{i,2}, то присваиваем a_{i,3} текущее значение счетчика c
      else
          a_{i,3} ← c
      c ← c + 1
// упорядочиваем строки матрицы A1 по возрастанию значений первого
// столбца и получаем матрицу A
A ← Sort(A1, 1, Ascending)
// конец алгоритма
```

Листинг 1. Алгоритм формирования третьего столбца матрицы A.

Listing 1 Algorithm for forming the third column of matrix A.

Третий столбец матрицы A будет искомой последовательностью в $\widehat{N}_d(l)$, соответствующий элементу a.

Лемма 1. Приведенный выше алгоритм переводит l-ичную d-разрядную запись числа в $(l + d - 1)$-ичную d-разрядную запись, в которой кратность каждой цифры равна единице.

Доказательство.

Во втором столбце стоят упорядоченные по неубыванию цифры. Если кратность цифры равна единице, то $a_{i,3} = a_{i,2}$ и эти цифры в третьем столбце различны: цифры $a_{i,2}$ находятся в диапазоне $[0, l-1]$.

Для тех цифр, кратность которых больше единицы первый раз цифра заменяется, как если бы ее кратность была равна единице. Начиная со второго вхождения происходит замена последующих вхождений этих цифр, в зависимости от номера строки. Производится замена $a_{i,3} = c$, счетчик c увеличивается на единицу при переходе на следующую строку. То есть для каждой строки c различно. Поскольку $c \in [l, l + d - 2]$, то не совпадает с цифрами из $N_d(l)$, принадлежащих диапазону $[0, l-1]$ ■.

Лемма 2. В матрице A всегда выполняется одно из двух соотношений:

$a_{i,3} = a_{i,2}$, если $a_{i,2} \neq a_{i-1,2}$

или

$a_{i,3} = l + i - 2$, если $a_{i,2} = a_{i-1,2}$.

Доказательство.

На первом этапе алгоритма определим $c = l - 1$. В конце каждого шага алгоритма счетчик c изменяется по формуле $c = c + 1$.

Если рассматриваем первую строку или при переборе строк матрицы A цифра $a_{i,2}$ встречается первый раз, то есть $a_{i,2} \neq a_{i-1,2}$, то в соответствии с алгоритмом $a_{i,3} = a_{i,2}$.

Если цифра $a_{i,2}$ встречалась ранее во втором столбце, то есть $a_{i,2} = a_{i-1,2}$, то в соответствии с алгоритмом $a_{i,3} = c$.

Теорема 1. Отображение f: $N_d(l) \to \widehat{N}_d(l)$ является инъективным.

Доказательство.

Возьмем два числа a' и a''. Построим для этих чисел матрицы A' и A'' согласно алгоритму. Пусть c_i - значение величины c при формировании третьего столбца i-й строки матрицы A.

Последовательно сравниваем строки матриц A' и A'' и ищем первое несовпадение элементов строк. Пусть они отличаются во втором столбце. Так как предыдущие строки совпадали, то $\sigma' = \sigma'' = \sigma$ и $c' = c'' = l + i - 2$ для матриц A' и A''. Поскольку $a'_{i,2} \neq a''_{i,2}$, без нарушения общности можно предполагать, что $a'_{i,2} < a''_{i,2}$. Поскольку вторые столбцы матриц A' и A'' отсортированы по неубыванию, по лемме 2 возможны два варианта:

1. $a'_{i,2} > \sigma$. Тогда $a''_{i,2} > \sigma$. Тогда $a'_{i,3} = a'_{i,2}$, а $a''_{i,3} = a''_{i,2}$. Следовательно, так как матрица A'' отсортирована по неубыванию, элемент $a'_{i,2} < a''_{k,2}, k \in [i, d]$ и в силу этого неравенства цифра $a'_{i,2}$ из матрицы A' отсутствует в третьем столбце матрицы A'', следовательно, образы элементов a' и a'' различаются.
2. $a'_{i,2} = \sigma$. Тогда $a'_{i,3} = c = l + i - 2$. По алгоритму цифра $l + i - 2 = c_i$ может стоять только в i-ой строке матрицы A'' в третьем столбце. В силу того, что $a'_{i,2} < a''_{i,2}$, а элементы строк были равны до i-ой строки, то в i-ой строке матрицы A'' во втором столбце стоит цифра $a''_{i,2}$ (в i-ой строке будет $a''_{i,3} = a''_{i,2}$, а $a''_{i,2} \in [0, l-1]$). То есть цифра $l + i - 2$ отсутствует в записи второго числа.

Пусть теперь при первом несовпадении строк элементы вторых столбцов матриц совпадают, а первые различаются. В этом случае $a'_{i,3} = a''_{i,3}$, но эти цифры находятся в разных разрядах образов элементов a' и a'', и, следовательно, согласно лемме 1 не могут совпадать ■.

В последующих алгоритмах матрица A использоваться не будет.

4. Загрузка данных на облако

4.1 Инициализация массива $\hat{X}$

Напомним, что $n = l^d$ и $\hat{l} = l + d - 1$.

Обозначим: $\hat{n} = \hat{l}^d$.

Ранее было определено преобразование f такое, что

$$f\colon N_d(l) \to \widehat{N}_d(l).$$

Определим массив $\hat{X} = (\hat{x}_1, \dots, \hat{x}_{\hat{n}})$.

Таким образом каждому номеру бита в l-ичной системе счисления будет соответствовать номер в $(l + d - 1)$-ичной системе счисления (точка в кубе размерности d со стороной $\hat{l}$). Цифры записи номера бита из $N_d(l)$ в $\widehat{N}_d(l)$ будут представлены различными целыми числами от 0 до $\hat{l} - 1$.

Заметим, что $\widehat{N}_d(l) \subset N_d(\hat{l})$, поэтому не для всех элементов $N_d(\hat{l})$ существует прообраз в множестве $\widehat{N}_d(l)$.

Массив $(x_1, \dots, x_n)$ это биты. Тогда массив $\hat{X}$ также массив битов.

В массиве $\hat{X}$ заполним только элементы, для которых в множестве $N_d(\hat{l})$ существует прообраз в множестве $\widehat{N}_d(l)$ по следующей формуле:

$\hat{x}_{f(i)} = x_i$, $(i = 1, \dots, n)$.

На остальных местах могут стоять случайные значения битов.

Заметим, что нам не требуется, чтобы существовало обратное преобразование $f^{-1}\colon N_d(\hat{l}) \to N_d(l)$, поскольку из облака возвращается значение бита, а не его номер.

4.2 Вспомогательные построения, необходимые для работы алгоритма запроса бита - алгоритм построения матрицы G

Построим матрицу G, состоящую из $\hat{n}$ строк и 2 столбцов ($i = 1,..,\hat{n},\ j = 1,2$).

$$G = \begin{pmatrix} 1 & g_{1,2} \\ \vdots & \vdots \\ \hat{n} & g_{\hat{n},2} \end{pmatrix}$$

Первый столбец этой матрицы – последовательность первых $\hat{n}$ натуральных чисел $[1, \hat{n}]$. Второй столбец матрицы G – элементы $\hat{X} = (\hat{x}_1, \dots, \hat{x}_{\hat{n}})$.

4.3 Алгоритм загрузки данных на облако

Обозначим через $G_{enc}(h)$ $(h \in \{1, \dots, k\})$ матрицу, хранящую зашифрованные элементы матрицы G. В первом столбце в i строке матрицы $G_{enc}(h)$ хранятся зашифрованные значения $g_{i,1} - 1$ матрицы G. Во втором столбце в i строке матрицы $G_{enc}(h)$ хранятся зашифрованные значения $g_{i,2}$ матрицы G. Матрицы $G_{enc}(h)$ формируются для k копий базы данных.

Алгоритм загрузки данных на облако аналогичен алгоритму, приведенному в [10].

На вход алгоритма подается бинарный массив $X = (x_1, \dots, x_n)$. В результате работы алгоритма на облако загружается k зашифрованных копий баз данных.

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 $G = \|g_{i,j}\|$, где $f(i) = 1,..,\hat{n},\ j = 1,2$.

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 первого столбца 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 $K(h)$ первый столбец матрицы G $DEnc(f(i), K(h), alg, h)$, где $f(i) = 1, .., \hat{n}, h \in \{1, \dots, k\}$ и получает столбец шифротекстов, соответствующих номерам строк матрицы G.

Дилер шифрует ключом $K_{enc}(h)$ второй столбец матрицы G: $REnc\left(g_{ij}, K_{enc}(h), alg, h\right)$ где $f(i) = 1, .., \hat{n},\ h \in \{1, \dots, k\}$ и получает шифротексты, соответствующие элементам второго столбца 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. По окончании перебора k копий базы данных происходит выход из алгоритма.

Конец алгоритма.

Число шифротекстов, переданных каждой из k копий базы данных равно $2 \cdot \hat{n}$.

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

Для шифрования второго столбца 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.

Для выполнения шифрования и последующей перестановки дилеру требуется иметь объем памяти, сравнимый для хранения одной 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 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.

5. Алгоритм выполнения запроса клиента дилером

Напомним, что имеется отображение:

$$f\colon N_d(l) \rightarrow \widehat{N}_d(l).$$

Также напомним, что элементы множества $N_d(l)$ будем также интерпретировать как l-ичную запись d-разрядного числа. Тогда множество $N_d(l)$ представляет все целые числа от 0 до $l^d - 1$.

После преобразования f в представлении номера бита в $\hat{l}$-ичной системе счисления у нас все цифры различны. Следовательно, можно получить $d!$ различных перестановок из различных цифр для преобразованного числа из $N_d(l)$. Каждая такая перестановка будет соответствовать единственному числу, так как все цифры различны.

Пусть $\{\sigma_1, \dots \sigma_{d!}\}$ - множество элементов симметрической группы S_d, причем σ_1 является тождественной перестановкой, то есть σ_1 - единица. σ_i, где $i > 1$ порождаются идентично при помощи генератора перестановок и σ_i не требуют хранения.

Для каждого элемента $\sigma_i \epsilon S_d$ определено отображение

$$h_\sigma: N_d(\hat{l}) \to N_d(\hat{l}).$$

по формуле

$$h_{\sigma_j}(a_0, \dots, a_{d-1}) = (\sigma_j(a_o), \dots, \sigma_j(a_{d-1})), \text{ где } a_i \epsilon N_d(\hat{l}).$$

Согласно такому определению, множество $N_d(\hat{l})$ преобразуется в себя. Ввиду интерпретации элементов множества $N_d(\hat{l})$ как целых чисел из отрезка $[0, \hat{l}^d - 1]$ можно определить $\hat{l} \times d!$-матрицу $M = \| m_{i,j} \|$, где

$$m_{i,1} = i - 1, i = 1, .., \hat{n},$$
$$m_{i,j} = h_{\sigma_j}(i-1), i = 1, .., \hat{n}, \ j = 2, \dots, d!$$

Матрица M также не хранится в памяти, поскольку однозначно генерируется при помощи перестановок σ и параметра i.

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

Пусть клиент интересуется элементом $x_i \in X$. По построению номеру i соответствует строка матрицы M с номером $g(i) = f(i-1)$, где $g(i) \in [1, \hat{n}]$.

Для сокрытия при запросе искомого номера бита, вместо одного запрашиваемого зашифрованного номера будем передавать множество зашифрованных номеров. Выбираем $f(i)$ строку матрицы M ($m_{g(i),1} = g(i)$).

Разобьем отрезок $[1, \hat{n}]$ на $\hat{l}$ интервалов по $\hat{L}_p = \frac{\hat{n}}{\hat{l}}$ элементов в каждом интервале.

Строка матрицы M состоит из $d!$ элементов. Для каждого элемента выберем по одному числу y в каждом из оставшихся $\hat{l} - 1$ интервалов (по построению числа все будут различные). Таким образом каждому числу $m_{g(i),j}$ $(j = 1, .., d!)$ в $g(i)$-й строке матрицы M поставим в соответствие $\hat{l} - 1$ чисел, таким образом, чтобы эти числа лежали в различных интервалах и не попадали в интервал, где лежит $m_{g(i),j}$.

Поскольку отрезок $[1, \hat{n}]$ был разбит на $\hat{l}$ интервалов по $\hat{L}_p = \frac{\hat{n}}{\hat{l}}$ элементов в каждом интервале, то любое число, принадлежащее $[1, \hat{n}]$ попадет в один из интервалов. Необходимо найти номер интервала w, в который попало число $m_{f(i),j}$.

Определяется номер интервала, в который попало число $m_{g(i),j}$ при разбиении отрезка $[1, \hat{n}]$ на $\hat{l}$ интервалов:

$$w = \left\lfloor \frac{m_{g(i),j} - 1}{\hat{L}_p} \right\rfloor + 1.$$

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

Приведем $m_{g(i),j}$ по модулю $\hat{L}_p$. Обозначим через $r = m_{g(i),j} \ mod \hat{L}_p$ результат приведения.

Положим

$$m'_{g(i),j} = \begin{cases} r, \text{если } r \neq 0 \\ \hat{L}_p, \text{если } r = 0 \end{cases}$$

порядковый номер в интервале с номером w.

Сформируем вектор-столбец $\boldsymbol{b}$, состоящий из различных случайных натуральных чисел b_m значения которых лежат в диапазоне $[1, \hat{L}_p]$, $m \in \{1, \dots, \hat{l}\}$. Числа b_m генерируются датчиком псевдослучайных чисел. На вход датчика *PRNG(db,m)* подается номер интервала m, для которого вычисляется b_m. Вектор-столбец $\boldsymbol{b}$ восстанавливается каждый раз одинаково. Элементы вектора-столбца в общем случае могут совпадать.

Хранить вектор-столбец $\boldsymbol{b}$ нет необходимости, поскольку при одинаковом начальном значении параметра m датчик срабатывает одинаково. Вектор $\boldsymbol{b}$ противнику неизвестен.

Пусть $a = \left(m'_{g(i),j} - b_w\right) mod \hat{L}_p$. Число a противнику неизвестно.

Для всех интервалов (где $m \in [1, \hat{l}]$) найдем $\hat{l}$ элементов y с учетом a по формуле:

$$c = \begin{cases} (a + b_m) mod L_p, & \text{если } c \neq 0 \\ L_p, & \text{если } c = 0 \end{cases}$$

$$y = L_p \cdot (m - 1) + c$$

Обозначим через множество $Set_{g(i),j}$ числа y для элемента $m_{g(i),j}$.

Обозначим через множество $Set_{g(i),j}$ числа y для элемента $m_{g(i),j}$. По построению множество $Set_{g(i),j}$ включает в себя элемент $m_{g(i),j}$ и $\left|Set_{g(i),j}\right| = \hat{l}$.

Пусть $Set_{g(i)} = \cup Set_{g(i),j}$, где $j = 1, .., d!$. Следовательно, мощность множества $\left|Set_{g(i)}\right|$ равна $\hat{l} \cdot d!$.

Число a и вектор $\boldsymbol{b}$ позволяют осуществлять сдвиг в каждом интервале. Сдвиг нужен для того, чтобы не дать противнику возможность определить состав множества $Set_{g(i)}$ за один шаг.

Алгоритм формирования множества $Set_{g(i)}$

На вход алгоритма (листинг 2) подается: $\hat{L}_p$, $\hat{l}$, i, d. На выходе множество $Set_{g(i)}$.

```
g(i) ← i
// формируем строку матрицы M с номером f(i)
// построим множества Set_{g(i),j} для каждого элемента строки матрицы M с
// номером g(i)
for j← 1 to d! do
Set_{g(i),j} ← ∅
        // найдем номер интервала
        w ← ⌊(m_{g(i),j} − 1)/L̂_p⌋ + 1
        // приведем m_{g(i),j} по модулю L̂_p
        r ← m_{g(i),j} mod L̂_p
        // найдем порядковый номер числа в интервале с номером w
        m'_{g(i),j} = { r, если r ≠ 0
                      { L̂_p, если r = 0
        // при помощи датчика псевдослучайных чисел PRNG(db,1)
        // генерируется b_w
        // найдем число a
        a = (m'_{g(i),j} − b_w) mod L̂_p
        for m ← 1 to l̂ do
                // при помощи датчика псевдослучайных чисел PRNG(db,m)
                // генерируется b_m
                c ← (a + b_m) mod L_p
                if c = 0 then
                        c ← L_p
                y ← L_p · (m − 1) + c
                Set_{g(i),j} ← Set_{g(i),j} ∪ y
```

Листинг 2. Алгоритм формирования множества $Set_{g(i)}$

Listing 2. $Set_{g(i)}$ formation algorithm.

5.2 Описание алгоритма построения множеств $Set_{f(i)}$ дилером

Дилер шифрует множество $Set_{g(i)}$ и отправляет запрос облаку, состоящий из $\hat{l} \cdot d!$ шифротекстов. Дилер получает ответ от облака как множество шифротекстов значений мощностью $\hat{l} \cdot d!$

На вход алгоритма подается: $\hat{L}_p$, $\hat{l}$, i, d. На выходе алгоритм формирует множество $Set_{g(i)}$.

Шаг 1. Дилер получает номер запрашиваемого бита i, преобразует его в $g(i)$ и строит строку матрицы M из $d!$ элементов.

Шаг 2. Дилер находит номер интервала w, в который попал номер запрашиваемого бита $m_{g(i),j}$ и его порядковый номер в этом интервале $m'_{g(i),j}$. Для вектора-столбца $\boldsymbol{b}$ дилер при помощи датчика псевдослучайных чисел *PRNG(db,w)* находит элемент b_w.

Шаг 3. Дилер находит число a для номера интервала w, куда попал номер $m_{g(i),j}$: $a = \left(m'_{g(i),j} - b_w\right) mod \hat{L}_p$.

Шаг 4. Для всех интервалов (где $m \in \left[1, \hat{l}\right]$) дилер находит $\hat{l}$ элементов y по формулам:

$$c = \begin{cases} (a + b_m) mod L_p, & \text{если } c \neq 0 \\ L_p, & \text{если } c = 0 \end{cases}$$

$$y = L_p \cdot (m - 1) + c$$

$$Set_{g(i),j} \cup y$$

Шаги 2-4 выполняются для каждого элемента построенной строки матрицы M.

Шаг 5. Дилер строит множество: $Set_{g(i)} = \cup\, Set_{g(i),j}$, где $j = 1, \dots, d!$ ■

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

После подтверждения центром аутентификации полномочий клиента дилер разрешает клиенту отправить запрос.

При запросе клиентом i-го бита дилер строит множество $Set_{g(i)}$, которое содержит элементы $g(i)$-й строки матрицы M. Напомним, что в $g(i)$-й строке элемент $m_{g(i),1}$ является искомым номером. Множество $Set_{g(i)}$ содержит $\hat{l} \cdot d!$, элементов, чтобы скрыть от противника информацию о запрашиваемом бите.

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

По аналогии с [10] для каждого элемента множества $Set_{g(i)}$ случайно и равномерно выбирается номер копии базы данных. Все элементы множества $Set_{g(i)}$ для которых выбрана копия базы данных h, обозначим через множества $Set^h_{g(i)}$, где $h \in \{1, \dots, k\}$. Очевидно, что множества $Set^h_{g(i)}$ не пересекаются между собой, поскольку все элементы множества $Set_{g(i)}$ различны и каждому элементу ставится в соответствие ровно один номер h. Объединение множеств $Set^h_{g(i)}$ содержит все элементы множества $Set_{g(i)}$, то есть $Set_{g(i)} = \cup\, Set^h_{g(i)}$, где $h \in \{1, \dots, k\}$.

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

Номер искомого бит попадает в одно из множеств $Set^{enc(h)}_{g(i)}$. Дилер запоминает номер h и соответствующий 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 вектор-строку $\boldsymbol{s}_{num}$ из трех элементов $\boldsymbol{s}_{num} = (s_1, s_2, s_3)$.

Первый элемент вектора-строки $\boldsymbol{s}_{num}$ содержит искомый номер элемента i_{cnv}.

Второй элемент вектора-строки $\boldsymbol{s}_{num}$ содержит номер копии базы данных, в которую попал шифротекст номера s_1.

Третий элемент вектора-строки $\boldsymbol{s}_{num}$ содержит номер шифротекста в перестановке элементов множества $Set_{g(i)}^{enc(h)}$. Таким образом, s_3 является функцией от трех параметров. Эта функция устанавливает соответствие между исходным номером $f(i)$ и номером шифротекста в перестановке:

$$s_3 = F\left(s_1, s_2, Set_{g(i)}^{enc(s_2)}\right).$$

Вектор-строка $\boldsymbol{s}_{num}$ позволяет дилеру после получения ответа от облака без выполнения расшифрования сразу отбросить данные, кроме данных, необходимых для выполнения запроса.

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 $\hat{l} \cdot d!$ шифротекстов для элементов множества: $Set_{g(i)}$.

После выполнения запроса дилер при помощи вектора-строки $\boldsymbol{s}_{num}$ определяет значение искомого элемента для строки матрицы M из копии базы данных. После расшифрования шифротекста, дилер получает значение запрашиваемого бита и отправляет его клиенту.

5.4 Подготовка запроса к облаку

На входе алгоритм получает номер запрашиваемого бита i. На выходе дилер получает зашифрованные и отсортированные множества $Set_{g(i)}^{enc(h)}$ и вектор-строку $\boldsymbol{s}_{num}$.

Шаг 1. Клиент запрашивает у дилера значение бита с номером i.

Шаг 2. Дилер восстанавливает $f(i)$-ю строку матрицы M и генерирует множество $Set_{g(i)}$.

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

Шаг 4. Дилер заносит $g(i)$ в первый элемент вектора-строки $\boldsymbol{s}_{num}$.

Шаг 5. Дилер выполняет разбиение множества $Set_{g(i)}$ на непересекающиеся подмножества $Set_{g(i)}^{h}$ путем случайного и равномерного выбора номера копии базы данных для каждого элемента множества $Set_{g(i)}$.

Дилер заносит во второй элемент вектора-строки $\boldsymbol{s}_{num}$ номер копии базы данных, соответствующей элементу s_1.

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

Шаг 7. Дилер заносит в третий элемент вектора-строки $\boldsymbol{s}_{num}$ номер шифротекста соответствующего элементу s_1 в перестановке множества $Set_{g(i)}^{enc(s_2)}$ ■.

5.5 Выполнение запроса к облаку и ответ облака

На вход алгоритма подаются зашифрованные и отсортированные множества $Set_{g(i)}^{enc(h)}$. На выходе дилер получает шифротексты значений битов в том же порядке, в котором передавались шифротексты, соответствующие номерам битов.

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

Шаг 2. Для запрошенных шифротекстов номеров битов облако возвращает дилеру шифротексты значений битов из второго столбца матрицы $G_{enc}(h)$, где $h \in \{1, \dots, k\}$).

Порядок возвращаемых шифротекстов соответствует исходной сортировке шифротекстов для множеств $Set_{g(i)}^{enc(h)}$ ■.

5.6 Обработка ответа облака

На входе дилер получает второй столбец матриц $G_{enc}(h)$ для шифротекстов множеств $Set_{g(i)}^{enc(h)}$. На выходе значение i-го бита.

Шаг 1. Дилер при помощи вектора-строки $\boldsymbol{s}_{num}$ выбирает зашифрованное значения бита. Остальные шифротексты отбрасываются.

Шаг 2. Дилер расшифровывает значение бита для первого элемента вектора-строки $\boldsymbol{s}_{num}$. По построению вектора-строки $\boldsymbol{s}_{num}$ запрашиваемый номер является элементом s_1.

Шаг 3. Дилер отправляет значение i-го бита клиенту ■.

6. Оценка требуемой памяти и сложности алгоритма

6.1 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 датчика случайных чисел *PRNG (db, i)*;
- 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. 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 [11]. Следовательно, дилер хранит объем информации значительно меньше n.

6.2 Характеристики предложенной схемы

Напомним, что $Len(K)$ – длина шифротекста в битах при шифровании номера бита БД в облаке односторонней функцией, $Len(K_{enc})$ – длина шифротекста в битах при вероятностном шифровании значений битов БД в облаке и $Len(K, K_{enc})$ – сумма длин $Len(K)$ и $Len(K_{enc})$.

Предложенная схема организации базы данных размера n и запроса к ней удовлетворяет следующим свойствам. На время загрузки базы данных дилер генерирует генерирует k ключей $K_{enc}(h)$. Эти ключи служат для вероятностного шифрования значений битов. После загрузки ни ключи, ни шифротексты дилер не хранит.

Как было сказано ранее, на протяжении всего времени существования базы данных объема n дилер хранит значение для инициализации датчика случайных чисел *PRNG(db,i)*, k ключей $K(h), h \in \{1, \dots, k\}$ для шифрования односторонней функцией и k ключей $K_{dec}(h)$, где $h \in \{1, \dots, k\}$ для вероятностного расшифрования значений битов.

Объем хранимой информации много меньше n.

Напомним, что $Len(K)$ – длина шифротекста в битах при шифровании номера бита БД в облаке односторонней функцией, $Len(K_{enc})$ – длина шифротекста в битах при вероятностном шифровании значений битов БД в облаке и $Len(K, K_{enc})$ – сумма длин $Len(K)$ и $Len(K_{enc})$.

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.

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 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.

Утверждение 1. Коммуникационная сложность схемы получения значения номера бита дилером без раскрытия его номера равна $\hat{l} \cdot d! \cdot Len(K, K_{enc})$ битов.

Доказательство. Для запроса значения бита дилер посылает копиям базы данных $\hat{l} \cdot d!$ (где $\hat{l} = \sqrt[d]{\hat{n}}$) шифротекстов номеров битов длиной $Len(K)$. Облако отвечает $\hat{l} \cdot d!$ шифротекстами значений битов длиной $Len(K_{enc})$ ■.

Проанализируем вероятность угадывания противником номера запрашиваемого бита.

Если запрошены два номера бита, то они могут принадлежать либо одному, либо разным множествам $Set_{g(i)}$. Противник может увидеть, выполнен ли запрос к одному из множеств $Set_{g(i)}$, 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 $Set_{g(i)}$, что при запросе к каждому элементу из подмножества, на облаке осуществляется доступ ко всем битам из множества $Set_{g(i)}$ и только к ним. Что будет означать, что противник не знает, а только угадывает, какой именно бит из данного множества был выбран ■.

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 $\frac{1}{\hat{l} \cdot d!}$.

Доказательство.

Множество $Set_{g(i),j}$ ($Set_{g(i),j} \subset Set_{g(i)}$) порождается одинаково для всех $\hat{l}$ чисел, в него входящих. То есть существует одинаковое множество для $\hat{l}$ различных номеров битов. Таким образом вероятность угадывания номера из множества $Set_{g(i),j}$ равна $\frac{1}{\hat{l}}$, так как $Set_{g(i),j}$ имеет мощность $\hat{l}$. Так как $Set_{g(i)} = \cup Set_{g(i),j}$ где $j = 1, \dots, d!$, вероятность угадывания номера i будет не более $\frac{1}{\hat{l} \cdot d!}$ ■.

Сделав n или более запросов, противник получит информацию о числе реальных битов в каждом множестве $Set_{g(i)}$. Число этих битов будет меньше или равно $\hat{l}$. Таким образом, противник понимает, что запрошенный клиентом номер находится среди истинных номеров битов. И вероятность того, что клиент запрашивал конкретный номер реального бита для всех реальных номеров битов из $Set_{g(i)}$ одинаковая.

Для того, чтобы оценить вероятность угадывания конкретного номера бита из множества $Set_{g(i)}$, необходимо посчитать среднее число истинных битов, попадающих в множество $Set_{g(i)}$.

Мощность множества $\hat{X}$ равна $\hat{n}$, где элементам множества X соответствует n $(n < \hat{n})$ элементов. Из множества $\hat{X}$ выбирается $\hat{l}$ элементов. В этом случае возникает вопрос: сколько элементов N из случайной выборки мощности $\hat{l}$ в среднем соответствует элементам множества X?

Пусть P_r - вероятность получения в выборке мощности $\hat{l}$ ровно r элементов, соответствующих элементам множества X. Тогда P_r находится по формуле [12]:

$$P_r = \frac{C_n^r \cdot C_{\hat{n}-n}^{\hat{l}-r}}{C_{\hat{n}}^{\hat{l}}}.$$

Тогда среднее число истинных битов, попадающих в множество $Set_{g(i)}$, равно:

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

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 $\frac{1}{N \cdot d!}$.

Доказательство.

Выполнив *n* запросов, противник определяет элементы, входящие в каждое множество $Set_{g(i)}$ (в предположении, что $k < ld$). Если фиктивные клиенты будут выполнять любое количество запросов большее n, противник все равно не получит никакой дополнительной информации.

Количество реальных бит, которые попали в каждый интервал в среднем равно N. Так как $Set_{g(i)} = \cup\, Set_{g(i),j}$ где $j = 1, \dots, d!$, вероятность угадывания номера *i* в среднем будет не более $\frac{1}{N \cdot d!}$ ■.

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 2. Для предложенной схемы организации базы данных размера *n*:

- Коммуникационная сложность запроса равна $\hat{l} \cdot d! \cdot Len(K, K_{enc})$ 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 $\frac{1}{\hat{l} \cdot d!}$.
- 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 $\frac{1}{N \cdot d!}$.

Доказательство. Из Утверждения 1 следует, что коммуникационная сложность запроса равна $\hat{l} \cdot d! \cdot Len(K, K_{enc})$ битов.

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

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 $\frac{1}{N \cdot d!}$ ■.

7. Заключение

Целью этого исследования было уменьшение вероятности угадывания номера бита при незначительном увеличении коммуникационной сложности. В результате проведенного исследования было показано, что использование другого основания системы счисления и инъективного отображения для представления номера бита позволяет получить следующие результаты.

Так как число копий баз данных мало (на практике обычно $k \leq 16$), а дилер хранит не более $O(k)$ бит информации, то дилер в состоянии обслуживать запросы к значительному числу баз данных. В этом случае имеется возможность эффективно использовать облако как место хранения значительного объема информации.

В ранее предложенном алгоритме коммуникационная сложность составляет $l{\cdot}d{\cdot}(Len(K)+2\ Len(K_{enc}))$ бит. А вероятность угадывания номера - $\frac{1}{l}$. Так как $\hat{l} = l + d - 1$, а d мало (практике d обычно не превышает 4 для 16 копий баз данных), то можно считать, что l и $\hat{l}$ отличаются незначительно. Если предположить, что $Len(K)$ и $Len(K_{enc})$ мало различаются, то сравнивая результаты предыдущего и нового алгоритма, получаем увеличение коммуникационной сложности в $\frac{2}{3}(d-1)!$ То есть при $d = 4$ получим увеличение коммуникационной сложности в 4 раза. Вероятность угадывания уменьшается в $d!$ раз, то есть в 24 раза при $d = 4$.

При $d \geq 2$ вероятность угадывания уменьшается быстрее, чем растет коммуникационная сложность.

Таким образом, предложенные методы для рассматриваемой схемы организации базы данных просты в реализации и снижают вероятность угадывания номера и значения бита.

Список литературы / References

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

[2]. Chor B., Goldreich O., Kushilevitz E., Sudan M. Private Information Retrieval, Journal of the ACM, Vol. 45, No. 6, November 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 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

[5]. 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.

[6]. Kushilevitz E., Ostrovsky R. One-Way Trapdoor Permutations Are Sufficient for Non-trivial Single-Server Private Information Retrieval. In EUROCRYPT00, 2000.

[7]. 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.

[8]. 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.

[9]. 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.

[10]. 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).

[11]. Wahid M.N.A., Ali A., Esparham B., Marwan M. A Comparison of Cryptographic Algorithms: DES, 3DES, AES, RSA and Blowfish for Guessing Attacks Prevention. Journal of Computer Science Applications and Information Technology, 2018, 3(2); 1-7.

[12]. Ширяев А. Н. Вероятность – 1, Москва, Из-во МЦНМО, 2021, изд. 7, стереот., 552 с. ISBN 978-5-4439-1557-9.

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.