DOI: 10.15514/ISPRAS–2020–33(2)–11

Исследование задачи обеспечения безопасности при хранении и обработке конфиденциальных данных

[1] *С.А. Мартишин, ORCID: 0000-0001-5437-4049 <mart@ispras.ru>*
[1] *М.В. Храпченко, ORCID: 0000-0002-5147-5132 <khrapm@gmail.com>*
[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, 141700, Россия, Московская область, г. Долгопрудный, Институтский пер., 9*

Аннотация. Приведен обзор современных подходов к работе с конфиденциальными данными в облачных вычислениях. Значительная часть хранилищ данных и систем их обработки используют облачные сервисы. Пользователи и организации рассматривают такие сервисы как поставщика услуг. В этом случае у них нет необходимости закупать, устанавливать и поддерживать дорогостоящее оборудование, они могут получить доступ к данным и результатам их обработки с любого устройства. Такое использование облачных сервисов несет в себе определенные риски, связанные с возможными угрозами раскрытия конфиденциальной информации, поскольку одним из участников протокола обеспечения безопасности доступа к облачным хранилищам данных может быть противник. Рассмотренные подходы предназначены для баз данных, в которых, с одной стороны, информация хранится в зашифрованном виде, а с другой, позволяют работать в привычной парадигме SQL-апросов. Такие подходы имеют, наряду с преимуществами, некоторые ограничения, в связи с необходимостью выбора метода шифрования, а также соблюдения баланса между его надежностью и необходимым пользователю набором запросов. Для случая, когда пользователь не ограничен рамками SQL-запросов, предложена организация облачных вычислений над конфиденциальными данными, основанная на использовании лямбда-архитектуры с ограничением на разрешенные дедуктивно безопасные запросы к базам данных и реализованная с использованием свободного программного обеспечения.

Ключевые слова: облачные вычисления; конфиденциальные данные; обеспечение безопасности; SQL; криптография; лямбда-архитектура; свободное программное обеспечение

Для цитирования: Мартишин С.А., Храпченко М.В., Шокуров А.В. Исследование задачи обеспечения безопасности при хранении и обработке конфиденциальных данных. Труды ИСП РАН, том 33, вып. 2, 2021 г., стр. 173-190. DOI: 10.15514/ISPRAS–2021–33(2)–11

Study of the problem of ensuring security in the storage and processing of confidential data

[1] *S.A. Martishin, ORCID: 0000-0001-5437-4049 <mart@ispras.ru>*
[1] *M.V. Khrapchenko, ORCID: 0000-0002-5147-5132 <khrapm@gmail.com>*
[1,2] *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., Dolgoprudniy, 141701, Russia*

Abstract. We introduce an overview of modern approaches to cloud confidential data processing. A significant part of data warehouse and data processing systems is based on cloud services. Users and organizations consider such services as a service provider. This approach allows users to take benefit from all of these technologies: they do not need to purchase, install and maintain expensive equipment, they can access the data and the calculation results from any device. Such data processing on cloud services carries certain risks because one of the participants of the protocol for securing access to cloud data storage may be an adversary. This leads to the threat of confidential information leakage. The above approaches are intended for databases in which information is stored in the encrypted form and they allow to work in the familiar paradigm of SQL queries. Despite the advantages such approach has some limitations. It is necessary to choose an encryption method and to maintain a balance between the reliability of encryption and the set of requests required by users. In the case if users are not limited by the framework of SQL queries, we propose another way of implementation of cloud computing over confidential data using free software. It is based on lambda architecture combined with certain restrictions on allowed deductively safe database queries.

Keywords: cloud computing; confidential data; security; SQL; cryptography; lambda architecture; free software

For citation: Martishin S.A., Khrapchenko M.V., Shokurov A.V. Study of the problem of ensuring security in the storage and processing of confidential data. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 2, 2021, pp. 173-190 (in Russian). DOI: 10.15514/ISPRAS–2021–33(2)–11.

1. Введение

С развитием интернет-технологий возможность распределенного хранения и обработки данных, ранее обсуждавшаяся только теоретически, стала реальной, например, на облачных серверах. Основным компонентом такого облака является хранилище данных. В нем данные хранятся на многочисленных распределённых в сети серверах, предоставляемых в пользование клиента. Клиент рассматривает облако как большой виртуальный сервер, вне зависимости от того, как он реализован и где физически расположен.

Развитие аппаратных и программных средств привело к появлению облачных вычислений, что позволило рассматривать облако как поставщика различных IT-услуг: SaaS (программное обеспечение как услуга, в которой клиент использует программное обеспечение (ПО) облачного провайдера), PaaS (платформа как услуга, в которой клиент имеет возможность размещать принадлежащее ему ПО на облаке), IaaS (инфраструктура как услуга, в которой клиент использует только облачную инфраструктуру, а задачу управления вычислительными ресурсами, хранением и обработкой данных решает при помощи системного и прикладного ПО), DBaaS (база данных как услуга, предназначенная для управления и хранения структурированных или неструктурированные данных).

Преимущества использования облака для хранения данных и операциями над ними очевидны. К данным можно получить доступ с любого устройства, организовать совместную работу, нет необходимости закупать, устанавливать и поддерживать дорогостоящее оборудование. Однако при работе с конфиденциальными данными, хранение и обработка

которых происходит на облаке, следует учитывать, что одним из участников протоколов обеспечения безопасности доступа к облачным хранилищам данных может быть противник.

В криптографии рассматриваются два типа противников.

- Активный противник, который может вмешиваться в ход выполнения криптографического протокола, как правило, может быть обнаружен путем полного анализа результатов однократного выполнения криптографического протокола.
- Пассивный противник – противник, который может получать некоторую информацию о процессе выполнения криптографического протокола, но не может в него вмешиваться. При этом полный анализ результатов неоднократного выполнения криптографического протокола не позволяет обнаружить присутствие пассивного противника.

Очевидно, что в подавляющем большинстве случаев наличие активного противника выявляется достаточно быстро. В то же время пассивный противник часто представляет даже большую опасность, поскольку он, собирая конфиденциальную информацию, может представлять значительную угрозу, оставаясь при этом необнаруженным.

Поэтому далее предполагается, что противник является пассивным, то есть может получать некоторую информацию о выполнении криптографического протокола, но не может в него вмешиваться.

Решению вопроса обеспечения безопасности хранения и обработки конфиденциальных данных посвящено большое количество работ, предложены различные подходы, основанные как на теоретических моделях, так и на готовых решениях, применяемых на практике. Ниже подробно рассмотрены примеры готовых решений, описаны их особенности, определенные недостатки, предложены пути использования теоретических наработок на практике.

2. Исследование задачи обеспечения безопасности при хранении и обработке конфиденциальных данных

Ниже приведена классификация типов облачных систем вычислений:

- частное облако (эксклюзивное использование одной организацией, состоящей из нескольких пользователей);
- облако сообщества пользователей (эксклюзивное использование определенным сообществом пользователей из нескольких организаций, которые решают задачи совместно);
- публичное облако (открытое использование широким кругом пользователей);
- гибридное облако (комбинация как минимум двух из трех, упомянутых выше).

Очевидно, что пользователи, хранящие данные на любом типе облака и использующие облачные сервисы для вычислений (хотя в первых двух случаях, возможно в меньшей степени) будут сталкиваться с проблемами, связанными с потерей данных, их искажением, утечкой результатов вычислений и пр.

Основными требованиями при хранении данных на облаке являются целостность данных, доступность и безопасность данных [1].

Под целостностью данных понимается свойство, при котором данные не могут быть изменены или уничтожены несанкционированным образом [2].

В облачном хранилище само облако рассматривается как противник, которому не доверяют. Данные могут быть неумышленно потеряны или умышленно стерты, или искажены. Поэтому необходимо иметь механизм проверки целостности данных, и регулярно проверять целостность данных на серверах облака. Таким образом проверка целостности данных направлена в первую очередь на защиту от активного противника, пытающегося внести нежелательные изменения в данные участников протокола.

Основными механизмами проверки целостности данных являются:

- POR (Proofs Of Retrievability, доказательство извлекаемости) и Provable Data Possession (PDP, доказательство владения данными) – схема подтверждает, что сохраненные данные не повреждены на сервере, в процессе хранения и извлечения клиентом;
- доказательство стираемости Proof of Erasability (POE), где облако обеспечивает полное уничтожение сохраненных данных в хранилище, когда клиент удаляет данные и разрывает соглашение с поставщиком услуг хранения.

Доступность означает, что данные хранятся в базе данных и могут использоваться по любому допустимому запросу авторизованных пользователей. Доступность может быть обеспечена при помощи различных методов резервного копирования данных.

Безопасность (конфиденциальность) данных означает, что информация, хранящаяся в базе данных и/или результаты вычислительного процесса, не предоставляется или не раскрывается неавторизованным пользователям. Основными способами защиты являются контроль доступа и шифрование данных (включая полностью гомоморфное шифрование).

Возможные атаки на облачное хранилище данных.

- Моментальный снимок (Snapshot). Можно получить моментальный снимок состояния атакованной системы. Угроза: атакующий может получить изображение виртуальной машины, выполняющей действия в системе управления базами данных (СУБД) или руткит (rootkit) операционной системы (ОС). Руткит – набор программных средств (например, исполняемых файлов, скриптов, конфигурационных файлов), которые злоумышленник устанавливает на взломанной им компьютерной системе, обеспечивающих в том числе сбор данных (параметров системы).
- Мониторинг (постоянный пассивный противник) – пассивно наблюдает за всеми операциями с базами данных включая выданные запросы, и также образом осуществляется механизм получения доступа к зашифрованным данным. Угроза, например, для зашифрованных баз – раскрытие принадлежности данных или схем шифрования, сохраняющих порядок зашифрованных данных.
- SQL-инъекция – инъекция произвольного SQL кода. Урон: возможна утечка данных за счет внедрения в SQL запрос вредоносного кода.
- Угроза всей системе в целом (нарушение данных) – рутинг (rooting, получение прав администратора) СУБД. Урон: получение полного доступа к ОС и БД.

Поскольку системы облачных вычислений получают все более широкое распространение, за последние несколько лет были предприняты усилия по разработке программных средств, которые предлагают различные подходы к проблеме безопасности облачных вычислений над конфиденциальными данными, сочетающие в себе определенные компромиссы между безопасностью, эффективностью и универсальностью.

Наиболее естественным подходом представляются теоретические схемы, основанные на современном криптографическом инструментарии:

- протокол конфиденциальных вычислений (также используют название *secure multi-party computation* – MPC);
- полностью гомоморфное шифрование (Fully Homomorphic Encryption, FHE);
- функциональное шифрование (Functional Encryption, FE).

Эти схемы могут обеспечить надежные гарантии безопасности, но их вычислительные и коммуникационные требования несовместимы с большинством практических приложений. Рассмотренные ниже системы управления базами данных (СУБД) и файловая система с возможностью хранения и обработки зашифрованных данных основаны на широко используемых СУБД MySQL, PostgreSQL, MongoDB и распределенной файловой системе HDFS.

Все перечисленные выше программные средства (ПС) являются свободным программным обеспечением (СПО), что подразумевает, в том числе, право на его изменение

(совершенствование), а также распространение копий и результатов изменения, что делает возможным адаптацию данных ПС для работы с зашифрованными данными. Таким образом, рассмотренные ниже продукты основаны на привычных и хорошо известных пользователям программных средствах. Кроме того, эти продукты практически полностью скрывают от пользователей особенности реализации шифрования данных и способы обработки зашифрованных данных за счет введения в архитектуру дополнительных модулей, что позволяет пользователям работать в привычной среде.

На практике предлагается использовать широко известные криптографические алгоритмы, если это возможно, модификацию этих алгоритмов с адаптацией к аппаратным особенностям и ограничениям или разработку новых специализированных решений.

Требования к такого рода алгоритмам хорошо известны: безопасность, стоимость и производительность. Под стоимостью понимается стоимость аппаратно-программной составляющей, необходимой для реализации алгоритмов шифрования. На практике легко оптимизировать любые две из трех целей проектирования: безопасность и стоимость, безопасность и производительность или затраты и производительность, но очень трудно оптимизировать все три цели проектирования одновременно.

3. ZeroDB

ZeroDB – СУБД со сквозным шифрованием, которая позволяет клиентам работать с зашифрованными данными (искать, сортировать, запрашивать и обмениваться) без предоставления ключей шифрования или открытого (незашифрованного) текста серверу базы данных. ZeroDB написана на Python (с некоторыми скомпилированными расширениями C) и предназначается для использования приложениями, написанными на Python. Проект больше не поддерживается (github.com/zerodb/zerodb).

ZeroDB [3] обеспечивает конфиденциальность данных, хранящихся на сервере базы данных. Предполагается, что клиент изначально владеет данными и ключами шифрования. Клиент работает с зашифрованными данными на ненадежном сервере, предполагая, что имеется пассивный (честный, но любопытный) противник, который только наблюдает, но не вмешивается в процесс хранения и обработки данных, и клиент получает правильные результаты.

Протокол запроса состоит в следующем. Клиент взаимодействует с сервером во время выполнения запроса в течение ряда циклов приема-передачи. Зашифрованный индекс хранится на сервере как B-Tree. BTree является обобщением бинарного дерева поиска; вместо того, чтобы хранить один ключ и иметь 2 дочерних узла, узлы B-Tree имеют n ключей и (n + 1) дочерних узлов. Клиент обходит дерево индексов, чтобы получить необходимые зашифрованные записи. Этот обход происходит постепенно, причем каждое следующее обращение соответствует шагу вниз по дереву B-Tree индексов. Индекс состоит из блоков, которые зашифровываются перед загрузкой на сервер и расшифровываются только на стороне клиента. Таким образом, сервер не знает, как отдельные объекты организованы в B-Tree.

Поскольку протокол запросов ZeroDB включает в себя несколько раундов между клиентом и сервером, часто используемые запросы кэшируются на стороне клиента, чтобы избежать ненужных сетевых вызовов.

Клиент запрашивает у сервера зашифрованный узел, расшифровывает его, принимает решение, на какой узел пройти дальше, а затем просит сервер вернуть следующий зашифрованный узел. Клиент проходит B-Tree удаленно, пока не получит требуемую запись/записи или объект/объекты. Тем не менее, этот протокол позволяет всю обработку данных производить на стороне клиента, в открытом тексте.

Вот сводка основных особенностей ZeroDB:

- система позволяет клиентам выполнять зашифрованные операции без предоставления ключей шифрования или незашифрованных данных серверу базы данных;
- поддерживаются операции поиска, сортировки, запросов, обмена;
- ZeroDB основана на объектной ZODB, которая вместо реляционных соединений таблиц используются ссылки на объекты;
- сервер БД действует как система хранения с гарантией согласованности; шифрование, дешифрование и сжатие данных происходят на стороне клиента, следовательно, сервер не имеет представления о типах хранимых данных, что исключает возможность раскрытия данных в результате утечки данных на стороне сервера (в случае успешного проникновения будут доступны только зашифрованные данные);
- индексы в ZeroDB хранятся в B-Tree, дерево состоит из зашифрованных сегментов, каждый из которых может быть корневым, ветвлением или листовым узлом; листовые узлы дерева указывают на фактические сохраняемые объекты; таким образом, поиск в базе данных представляет собой простой обход дерева;
- узлы B-Tree – $n + 1$ дочерних узлов; поскольку высота дерева меньше, чем у двоичного дерева, поиск требует гораздо меньше доступа к памяти; временная сложность – $O(\log n)$ для поиска, вставки и удаления.
- сервер, на котором хранятся данные, никогда не знает используемый ключ шифрования; объекты, на которые ссылаются конечные узлы индексов B-Tree, также зашифрованы на стороне клиента; в результате сервер не знает, как отдельные объекты организованы в B-Tree и распространяется ли на них индексирование вообще.

4. CryptDB

CryptDB – проект для решения проблемы безопасного хранения данных в БД, обслуживаемых в облачных сервисах и других неподконтрольных системах [4]. В основе лежит использование возможностей MySQL (или PostgreSQL).

Основное преимущество CryptDB заключается в выполнении запросов над зашифрованными данными, что делает возможным применение CryptDB на практике – использование четко определенного набора SQL-операторов, каждый из которых может эффективно обрабатывать зашифрованные данные.

В CryptDB рассматриваются две основные угрозы.

- любопытный администратор базы данных (DBA) – пассивный противник, который пытается узнать конфиденциальные данные (путем отслеживания на сервере СУБД); здесь CryptDB не позволяет администратору базы данных это сделать;
- противник, который может получить полный контроль над приложениями и серверов СУБД; в этом случае, CryptDB не может предоставить никаких гарантий для пользователей, которые работали с приложением во время атаки, но все же может обеспечить конфиденциальность данных остальных пользователей.

Архитектура CryptDB (рис. 1) состоит из двух частей: прокси-сервера базы данных и немодифицированной СУБД.

CryptDB использует функции, определяемые пользователем (User Defined Functions, UDF) для выполнения криптографических операций в СУБД. Прямоугольники и закругленные прямоугольники представляют процессы и данные соответственно. Затенение указывает компоненты, добавленные CryptDB. Пунктирные линии обозначают разделение между компьютерами пользователей, сервером приложений и сервером, на котором работает база данных CryptDB.

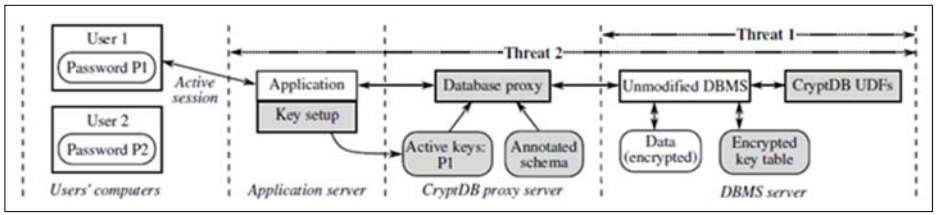

Рис.1. Архитектура CryptDB [4]
Fig. 1. CriptDB Architecture [4]

Прокси-сервер использует секретные ключи для шифрования всех вставленных данных. Основная идея вычислений над зашифрованными данными состоит в том, чтобы позволить серверу СУБД выполнять обработку запросов к зашифрованным данным, как это было бы с незашифрованной базой данных, то есть разрешить ему вычислять определенные функции над элементами данных на основе зашифрованных данных. Например, если СУБД необходимо выполнить команду группировки GROUP BY по некоторому столбцу, то сервер СУБД должен уметь определять, какие элементы в этом столбцы равны друг другу, но не фактическое значение каждого.

Противник может получить доступ к ключам, используемым для шифрования всей базы данных. Решение состоит в том, чтобы зашифровать различные элементы данных (например, данные, принадлежащие разным пользователям) с разными ключами. Противник, который атакует сервер приложений или прокси-сервер, теперь может расшифровать только данные пользователей, вошедших в систему в данный момент (данные, которые хранятся на прокси-сервере). Данные неактивных в настоящее время пользователей зашифрованы ключами, которые недоступны злоумышленнику и останутся конфиденциальными.

Для успешной работы CryptDB использует три основных подхода.

- Система выполняет SQL-запросы к зашифрованным базам данных. Поскольку CryptDB выполняет SQL-запросы на зашифрованных наборах данных с четко определенными операторами (такими как проверка равенства, сравнения порядка, агрегации/суммирования и объединения), это делает возможным осуществление на практике обработку зашифрованных данных (в основном используется шифрование с симметричным ключом).
- Производится настраиваемое шифрование на основе запросов. Некоторые схемы шифрования менее устойчивы к раскрытию информации, но требуются для обработки определенных запросов. Чтобы избежать этого, CryptDB тщательно настраивает схему шифрования для любого заданного элемента данных, используя луковичное шифрование.
- Ключи шифрования связываются с паролями пользователей, чтобы каждый элемент данных в базе данных можно было бы расшифровать только через цепочку ключей. Таким образом, если пользователь не работает в данный момент с приложением и если противник не знает пароль, он не сможет расшифровать данные пользователя, даже если СУБД и сервер приложений атакованы.

Поэтому CryptDB имеет возможность переключаться на лету между различными криптографическими схемами в зависимости от типа выполняемой операции. Это реализовано за счёт «луковичного» многоступенчатого шифрования, когда данные зашифрованы в несколько слоёв разными алгоритмами. У каждого слоя – свой ключ и свой список поддерживаемых операций. На нижнем слое используются самые надёжные алгоритмы, а операции в верхних слоях возможны без расшифровки нижних слоёв.

Таким образом, CryptDB

- способна эффективно обслуживать SQL-запросы к БД – поиск, сортировку,

математические функции и др. без расшифровки записей базы;
- выполняет запросы SQL над зашифрованными данными, используя набор эффективных схем шифрования с учетом SQL;
- может связывать ключи шифрования с паролями пользователей, так что элемент данных может быть расшифрован только с помощью пароля одного из пользователей, имеющих доступ к этим данным; в результате администратор базы данных никогда не получает доступ к расшифрованным данным, даже в том случае, когда все серверы скомпрометированы.

Однако в CryptDB количество запросов ограничено из-за используемых схем шифрования.

На практике предлагается использовать StealthyCRM: приложение для безопасной облачной CRM-системы (Customer Relationship Management, управление взаимоотношениями с клиентами), поддерживающее полностью гомоморфное шифрование баз данных [5] поверх среды шифрования базы данных CryptDB. StealthyCRM использует реализацию FHE под названием HELib (библиотека гомоморфного шифрования с поддержкой гомоморфного сложения и умножения) с открытым исходным кодом для интеграции FHE с CryptDB [6].

Система StealthyCRM направляет запросы на сервер CryptDB в случае простых запросов и к зашифрованным обработчикам объектов StealthyCRM для более сложных запросов. StealthyCRM анализирует запрос и преобразует его в два новых запроса – один для CryptDB (который анализирует текстовые запросы) и другой для обработчика зашифрованных объектов StealthyCRM. Прокси-сервер решает, CryptDB или StealthyCRM необходимы для выполнения конкретного запроса и пересылает зашифрованные запросы на сторону сервера. Сервер отправляет зашифрованный результат приложению после выполнения. Модуль дешифрования StealthyCRM расшифровывает полученный результат и передает его обратно в приложение, генерирующее запрос.

CryptDB использует для различных запросов к БД следующие типы шифрования.

- **Случайный (RND).** RND обеспечивает максимальную безопасность в CryptDB: IND-CPA (indistinguishability under chosen plaintext attack), неразличимость шифротекста – криптосистема надёжна в смысле IND-CPA, если любой вероятный злоумышленник за полиномиальное время имеет лишь пренебрежимо малое «преимущество» в различении шифротекстов над случайным угадыванием. Схема является вероятностной, что означает, что два равные значения отображаются в разные шифротексты с вероятностью почти единица. С другой стороны, в случае RND не существует эффективного выполнения вычислений над зашифрованным текстом. Обычно используется блочный шифр AES или Blowfish вместе со случайным вектором инициализации.
- **Детерминированный (DET).** DET имеет немного более слабую, надежную безопасность: возможна утечка только тех зашифрованных значений, которые соответствуют одному и тому же значению данных, детерминировано генерируя один и тот же зашифрованный текст для одного и того же открытого текста. Это шифрование позволяет серверу выполнять проверки равенства, что означает, что он может выполнять выборки с предикатами равенства, соединениями равенства, GROUP BY, COUNT, DISTINCT и т. д. В криптографических терминах DET должен быть псевдослучайной перестановкой (pseudo-random permutation, PRP).
- **Шифрование с сохранением порядка (OPE).** OPE (Order-Preserving Encryption) позволяет установить отношения порядка между элементами данных на основе их зашифрованных значений, не раскрывая самих данных. Если $x < y$, то $OPE_K(x) < OPE_K(y)$ для любого секретного ключа K. Следовательно, если столбец зашифрован с помощью OPE, сервер может выполнять запросы диапазона, когда заданы зашифрованные константы $OPE_K(c_1)$ и $OPE_K(c_2)$, соответствующие диапазону $[c1; c2]$. Сервер также может выполнять SQL запросы с операторами ORDER BY, MIN, MAX, SORT и т.д. OPE – более слабая схема шифрования, чем DET, потому что она раскрывает

порядок.

- **Гомоморфное шифрование (HOM)**. HOM – безопасная вероятностная схема шифрования (безопасность IND-CPA), позволяющая серверу выполнять вычисления с зашифрованными данными с расшифровкой окончательного результата на прокси-сервере. Хотя полностью гомоморфное шифрование очень медленное, гомоморфное шифрование для определенных операций является эффективным.
  В частности, для поддержки суммирования был реализован метод Пэйе [7]. Это метод, например, для произведения двух зашифрованных чисел дает зашифрованную сумму значений: $HOM_K(x)\,HOM_K(y) = HOM_K(x+y)$, где умножение выполняется по модулю некоторого значения открытого ключа.
- **Присоединение (JOIN и OPE-JOIN).** Отдельная схема шифрования, которая необходима для обеспечения проверки равенства между двумя столбцами, поскольку используются разные ключи для DET, чтобы предотвратить корреляцию между столбцами. JOIN также поддерживает все операции, разрешенные DET, а также позволяет серверу определить повторяющиеся значения между двумя столбцами. OPE-JOIN позволяет соединения по порядку отношений.
- **Поиск слова (ПОИСК).** ПОИСК используется для поиска соответствия зашифрованного текста, аналогично работе оператора MySQL LIKE.

На практике был реализовали криптографический протокол поиска по зашифрованным данным, разработанный в работе Суна (Dawn Xiaoding Song) и др. [8]. Для каждого столбца, требующего ПОИСКа, текст разбивается на ключевые слова с использованием стандартных разделителей. Далее убираются повторяющиеся слова, произвольно изменяются позиции слов, а затем каждое слово шифруется по Суну [8], так, что все слова имеют одинаковый размер. ПОИСК почти как же безопасен как RND: шифрование не раскрывает серверу СУБД, повторяется ли определенное слово в нескольких строках. При этом он может знать количество ключевых слов, зашифрованных с помощью ПОИСКа. Противник может уметь оценивать количество различных или повторяющихся слов (например, сравнивая размер шифротекстов SEARCH и RND для тех же данных).

В предлагаемой реализации прокси-сервер CryptDB состоит из библиотеки C++ и модуля Lua. Библиотека C++ состоит из парсера запросов; блока шифрования/перезаписи запросов, который шифрует поля или включает UDF в запрос; и модуль дешифрования.

5. MONOMI

MONOMI – система для безопасного выполнения операций над конфиденциальными данными на ненадежном сервере базы данных. MONOMI работает путем шифрования всей базы данных и выполнения запросов к зашифрованным данным. MONOMI была реализована с использованием PostgreSQL и предназначена для выполнения аналитических SQL-запросов [9]. По-прежнему сервер рассматривается как пассивный противник.

Существующие проблемы, для решения которых предназначена MONOMI:

- во-первых, запросы к большим наборам данных часто ограничены возможностями системы ввода-вывода, например, чтение данных с диска или потоковая передача через память; в результате схемы шифрования, которые значительно увеличивают размер данных, могут замедлить обработку запросов.
- во-вторых, аналитические запросы требуют сложных вычислений, которые могут быть неэффективными для выполнения над зашифрованными данными; вместо этого на практике должны использоваться эффективные схемы шифрования, которые могут выполнять только определенные вычисления (например, использование шифрования с сохранением порядка для сортировки и сравнения и пр.); задача состоит в том, чтобы разделить запрос на части, которые могут быть выполнены с использованием доступных схем шифрования на ненадежном сервере, и части, которые должны быть выполнены на

доверенном клиенте;

- в-третьих, некоторые методы обработки запросов по зашифрованным данным могут ускорить одни запросы, но замедлить другие, что требует тщательного планирования выполнения каждого запроса для рассматриваемой базы данных и комбинации запросов.

MONOMI решает эти проблемы тремя способами:

- во-первых, вводится разделенное клиент-серверное выполнение сложных запросов, при котором выполняется столько запросов, сколько возможно, по зашифрованным данным на сервере, а остальные компоненты запросов выполняются путем отправки зашифрованных данных доверенному клиенту, который расшифровывает данные и обрабатывает запросы в обычном режиме;
- во-вторых, вводится ряд методов, которые улучшают производительность для определенных типов запросов (но не обязательно для всех), включая предварительное вычисление каждой строки, эффективное шифрование, групповое гомоморфное сложение и предварительную фильтрацию;
- в-третьих, в архитектуру добавляются проектировщик для оптимизации физического размещения данных на сервере и планировщик для принятия решения о том, как разделить выполнение запроса между клиентом и сервером.

Поскольку разработка MONOMI основывалась на CryptDB, то она имеет аналогичные свойства, касающиеся безопасности. Хотя ненадежный сервер хранит только зашифрованные данные, он все же может получить информацию об исходных данных в виде открытого текста тремя способами:

- во-первых, некоторые схемы шифрования раскрывают информацию, необходимую для обработки запросов (например, детерминированное шифрование выявляет дубликаты для выполнения проверки равенства);
- во-вторых, некоторые схемы шифрования могут пропускать больше информации, чем необходимо; например, схема с сохранением порядка дает частичную утечку информации об открытом тексте;
- в-третьих, сервер узнает, какие строки соответствуют каждому предикату, вычисленному на сервере, например, строки, соответствующие столбцу LIKE '%keyword%'.

Комбинируя эти источники информации, сервер, являющийся противником, может получить дополнительную информацию о строках или запросах, используя статистические методы.

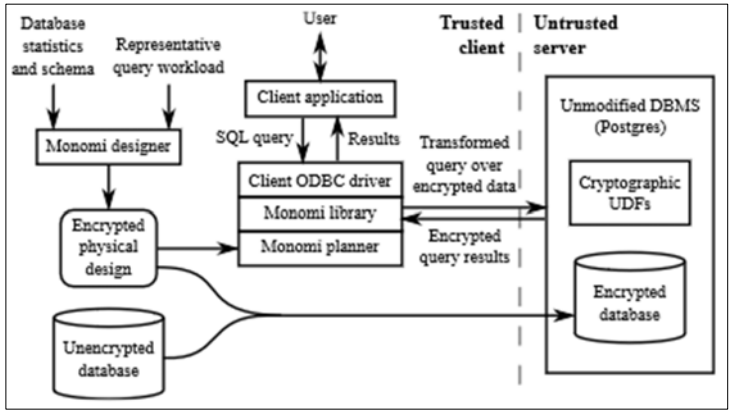

Рис. 2. Архитектура MONOMI [9]
Fig. 2. MONOMI Architecture [9]

MONOMI никогда не хранит текстовые данные на сервере и использует только схемы шифрования, необходимые для работы приложения. MONOMI позволяет администратору

дополнительно ограничивать схемы шифрования, используемые для особо чувствительных столбцов: например, требование шифрования с сохранением порядка (самая слабая схема MONOMI) не должна использоваться для столбцов, в которых хранятся номера кредитных карт или номеров социального страхования.

Архитектура MONOMI представлена на рис. 2.

- Во время настройки системы проектировщик MONOMI запускается на доверенной клиентской машине и определяет ее эффективную физическую конфигурацию для ненадежного сервера. Чтобы определить основные характеристики рабочей нагрузки для достижения хорошей производительности, проектировщик принимает в качестве входных данных репрезентативное подмножество запросов и статистику по данным, предоставленным пользователем. Пользователи не обязаны применять проектировщик, и вместо этого могут вручную вводить стратегию шифрования или изменять стратегию, им созданную.
- При нормальной работе приложения выдают немодифицированные запросы SQL с использованием библиотеки MONOMI ODBC, которая является единственным компонентом, имеющим доступ к ключам дешифрования. Библиотека ODBC использует планировщик, чтобы определить наилучший план выполнения запроса с разделением для приложений клиент/сервер.
- Учитывая план выполнения, библиотека выдает один или несколько запросов к зашифрованной базе данных, которая не имеет доступа к ключам дешифрования и может выполнять операции только с зашифрованными данными. База данных запускает немодифицированное программное обеспечение СУБД, такое как PostgreSQL, с несколькими определяемыми пользователем функции (UDF), предоставляемые MONOMI, которые реализуют операции с зашифрованными данными. MONOMI шифрует все данные, хранящиеся в базе данных, хотя на практике неконфиденциальные данные могут быть сохранены в виде открытого текста для повышения эффективности. После того, как клиентская библиотека получает промежуточные результаты из базы данных, расшифровывает их и выполняет любые оставшиеся операции, которые не могут быть эффективно выполнены на сервере, результаты отправляются в приложение, как если бы они выполнялись в стандартной базе данных SQL.

Вот сводка основных характеристик MONOMI:

- прототип MONOMI реализован поверх PostgreSQL;
- работает путем шифрования всей базы данных и выполнения запросов к зашифрованным данным;
- библиотека MONOMI ODBC является единственным компонентом, имеющим доступ к ключам расшифровки;
- MONOMI представляет новый подход, основанный на раздельном исполнении запроса на клиент-сервере; это позволяет выполнить часть запроса на ненадежном сервере поверх зашифрованных данных; для других частей запроса MONOMI загружает промежуточные результаты на клиента;
- библиотека MONOMI ODBC получает промежуточные результаты из базы данных, расшифровывает их и выполняет все оставшиеся операции, которые не могут быть эффективно выполнены на сервере;
- реализовано несколько методов, которые улучшают производительность: предварительное вычисление строки, пространственно-эффективное шифрование, групповое гомоморфное сложение и предварительная фильтрация;
- поскольку эти оптимизации хорошо работают для одних запросов и неэффективны для других, MONOMI имеет планировщик, чтобы определить лучший способ выполнения запроса.

6. Seabed

В Seabed используются HDFS и Spark. То есть, в отличие от CryptDB и Monomi, система основана не на реляционной базе данных, а на файловой системе, предназначенной для хранения неструктурированной информации. Прототип реализован на Apache Spark [10].

Цель – поддержка Business intelligence (BI) – компьютерных методов, которые обеспечивают перевод транзакционной деловой информации в удобную для восприятия человеком форму, пригодную для бизнес-анализа, а также средства для массовой работы с такой обработанной информацией.

Предполагается, что противник является пассивным (честным, но любопытным), то есть противник будет пытаться узнать конфиденциальные данные, но не будет их повреждать или иным образом вмешиваться в работу системы.

В плане шифрования Seabed

- использует новую, аддитивно-симметричную схему гомоморфного шифрования (Additively Symmetric Homomorphic Encryption Scheme, ASHE) для эффективного выполнения крупномасштабных агрегаций;
- представляет новую рандомизированную схему шифрования под названием Splayed ASHE, или SPLASHE.

Схема Seabed приведена ниже (рис. 3).

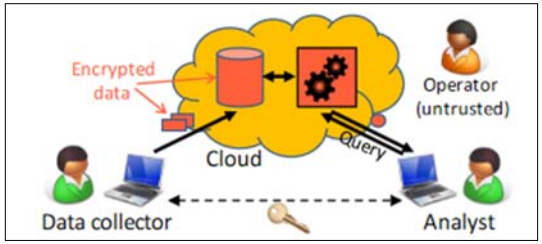

Рис. 3. Схема Seabed [10]
Fig. 3. Seabed outline [10]

Сборщик данных (Data collector) собирает большое количество данных, шифрует их и загружает на облако, которому не доверяет. Аналитик может генерировать запросы для обработчика запросов в облаке. Ответы будут зашифрованы, но аналитик может расшифровать их с помощью секретного ключа, который является общим со сборщиком данных. Рабочая нагрузка, которую предполагается поддерживать, состоит из запросов в стиле OLAP для больших наборов данных. OLAP (OnLine Analytical Processing, интерактивная аналитическая обработка) – технология обработки данных, заключающаяся в подготовке суммарной (агрегированной) информации на основе больших массивов данных, структурированных по многомерному принципу. Реализации технологии OLAP являются компонентами программных решений класса Business Intelligence.

Один из распространенных подходов к решению вышеуказанной проблемы – использовать гомоморфное шифрование. Например, есть криптосистемы с аддитивным гомоморфизмом, такие как Пэйе (Pascal Paillier) [7], что позволяет облаку выполнять агрегирование непосредственно на зашифрованных данных.

Однако бывают ситуации, когда необходимо, чтобы облако имело доступ к какому-то свойству зашифрованных значений (шифрование с сохранением свойств). Например, чтобы вычислить соединение (JOIN), облако должно иметь возможность согласовывать зашифрованные значения. В этом случае можно использовать детерминированное шифрование, где каждое значение отображается ровно на один зашифрованный текст. Однако такие схемы подвержены частотным атакам, особенно если столбец может принять только небольшое количество значений, и облако знает, что какое-то значение будет

наиболее распространенным. Еще одним примером операции, доступной облаку в случае использования схемы шифрования с сохранением свойств, является выбор строки на основе диапазона значений в зашифрованном столбце. Здесь можно использовать шифрование OPE. Очевидно, что в этих схемах есть компромисс между конфиденциальностью, производительностью и функциональностью.

Примеры операций в Seabed, которые могут быть полностью выполнены на сервере, – вычисление суммы; среднего, минимума и т.д. Пример операции, которые могут быть выполнены при наличии предварительной обработки клиента, – квадратичные вычисления, необходимые для более сложной аналитики, такой как обнаружение аномалий, линейной регрессии в одном измерении, и деревьев решений.

ASHE предполагает, что открытые тексты взяты из аддитивной группы $Z_n = \{0, 1, \dots, n-1\}$. Также предполагается, что отправитель и получатель, шифрующие входную и выходную информацию соответственно, имеют общий секретный ключ k, а также общую псевдослучайную функцию (PRF) $F_k: I \to Z_n$, преобразующая идентификатор из набора I в случайный номер из Z_n.

Один из возможных вариантов PRF – $F_k(i) := H(i \,||\, k) \bmod n$, где $i \in I$, H – криптографическая хеш-функция (моделируется как случайная функция), $||$ обозначает конкатенацию, размер диапазона H кратен n. Другим вариантом может быть использование AES (Advanced Encryption Standard – симметричного алгоритма блочного шифрования), когда он используется как псевдослучайная перестановка.

Ниже рассмотрена архитектура Seabed (рис. 4).

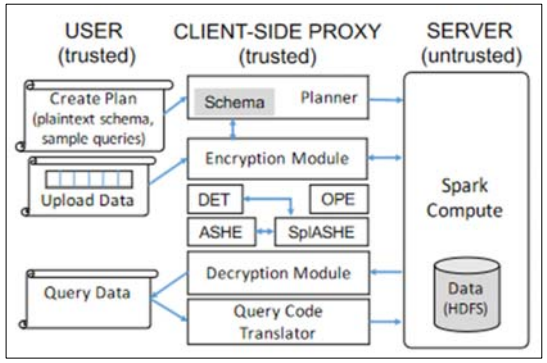

Рис. 4. Архитектура Seabed [10]
Fig. 4. Seabed architecture [10]

Пользователь взаимодействует с клиентским прокси Seabed, который является доверенным. Прокси, в свою очередь, взаимодействует с сервером Seabed, которому не доверяют.

Пользователь может отправлять запросы трех типов.

- **Создание плана:** сначала пользователь предоставляет схему в виде открытого текста и образец запроса, заданный для планировщика Seabed. Планировщик использует их и специальную процедуру для определения схем шифрования столбцов.
- **Загрузка данных.** Затем пользователь отправляет данные в виде открытого текста в Модуль шифрования Seabed. Данные шифруются с использованием необходимой схемы шифрования и записи добавляются в таблицу, хранящуюся в облаке. Это непрерывный процесс; вставки в базу данных обрабатываются таким же образом.
- **Запрос данных:** во время анализа пользователь отправляет скрипт запроса в транслятор запросов Seabed, который изменяет запросы для обработки зашифрованных данных перед их отправкой на сервер. Сервер выполняет запросы и выдает ответ модулю

дешифрования прокси. После расшифровки и дальнейшей обработки (если она требуется) результаты отправляются обратно пользователю.

Планировщик данных определяет, как зашифровать каждый столбец в схеме, учитывая список конфиденциальных столбцов. Пользователь также предоставляет образец набора запросов, который используется планировщиком для выбора алгоритмов шифрования.

Модуль шифрования шифрует открытый текст.

Транслятор запросов предназначен для переписывания запросов пользователей, чтобы адаптировать их к выбранному способу шифрования данных.

7. Arx

Система Arx реализована на базе MongoDB, СУБД категории NoSQL [11]. Декларация разработчиков:

- отказ от слабых схем шифрования;
- в основе схемы шифрования для Arx используется AES, (что связано с наличием соответствующей аппаратуры); возможно использование других методов шифрования;
- уровень безопасности данных IND-CPA.

Предполагается, что сервер БД может быть размещен в частном или публичном облаке. Модель угроз Arx предполагает, что противник не имеет доступа к данным на стороне клиента (пользователи, приложения и клиентский прокси Arx), а имеет доступ только к информации на стороне сервера (прокси-сервера Arx и серверы баз данных).

Предполагается, что

- противником, например, может быть любопытный администратор облачного провайдера;
- противник может видеть всю информацию на сервере – все содержимое базы данных, любые хранящиеся данные или ключи, в памяти, и любые сетевые сообщения; следовательно, если секретный ключ находится в основной памяти, злоумышленник может расшифровать базу данных;
- противник пассивен, то есть не изменяет содержимое базы данных или результаты запроса.

Arx не полагается на какое-либо доверенное оборудование на сервере.

Основная задача предложенной архитектуры – не менять имеющийся сервер БД и приложения. Ниже показана архитектура Arx (рис. 5).

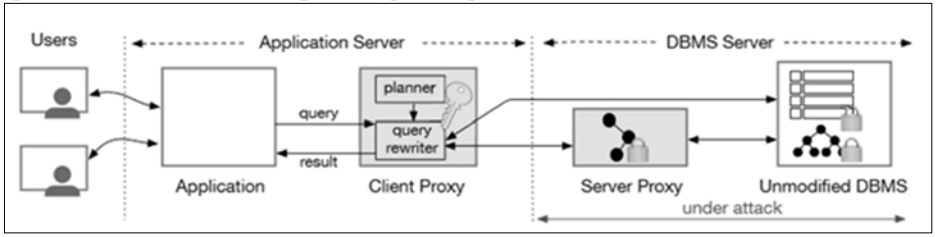

Рис. 5. Архитектура Arx [11]
Fig. 5. Arx's architecture [11]

- Доверенный клиентский прокси развернут на сервере приложений, ненадежный прокси-сервер развернут на сервере СУБД, сервер СУБД размещен на частном или публичном облаке.
- Клиентский прокси перехватывает запросы и шифрует конфиденциальную информацию. Прокси-сервер поддерживает индексы зашифрованных данных и выполняет входящие запросы.
- Серые прямоугольники изображают компоненты, представленные Arx, а остальные

прямоугольники представляют существующие компоненты.

- Ключ указывает, что конфиденциальные данные в компоненте всегда остаются строго зашифрованными

Заметим, что клиентский прокси экспортирует в приложение тот же API, что и сервер БД, поэтому приложение не нужно менять. Прокси-сервер взаимодействует с сервером БД, вызывая его неизмененный API (например, выдавая запросы); другими словами, прокси-сервер ведет себя как постоянный клиент сервера БД.

- Клиентский прокси хранит ключ (master key). Переписывает запросы, шифрует конфиденциальные данные и пересылает зашифрованные запросы на прокси-сервер для выполнения
- Прокси-сервер помогает серверу выполнять зашифрованную обработку данных.
- Прокси клиента не хранит базы данных, а хранит метаданные (информацию о схеме) и небольшой дополнительный кеш. Почти во всех случаях клиентский прокси обрабатывает только результаты запросов (например, для их расшифровки).

Мастер-ключ – ключ шифрования для симметричного алгоритма шифрования (например, в Windows 10 доступен AES).

Arx представляет два новых индекса базы данных:

- Arx-RANGE для запросов из диапазона;
- Arx-EQ для запросов на равенство.

Оба индекса являются структурами данных, построенными на основе AES.

Индекс – бинарное дерево поиска. Arx-RANGE позволяет серверу самостоятельно обходить дерево при сохранении безопасности.

Для обеспечения сложных вычислений с высокой степенью безопасности, Arx-RANGE использует одноразовую обфускацию на каждом узле в дереве индексов для выполнения сравнения.

Планировщик Arx принимает в качестве входных данных:

- набор шаблонов запросов, специфичных для Arx;
- список регулярных индексов

и создает:

- план шифрования данных;
- список индексов Arx;
- план выполнения запроса для каждого шаблона запроса.

Чтобы использовать Arx, разработчик приложения должен указать:

- какие поля являются конфиденциальными и должны быть зашифрованы;
- операции, выполняемые на конфиденциальных полях.

Поддерживаемые операции:

- чтение:
  - ВЫБОРКА (сумма, сортировка, группировка, агрегирование, например, с минимальной постобработкой на клиентском прокси, среднее или стандартное отклонение);
- запись:
  - INSERT;
  - DELETE;
  - UPDATE.

8. Обсуждение

Напомним, что во всех случаях сервер рассматривается как пассивный противник.

Следует заметить, что все рассмотренные выше инструменты для работы с базами данных основываются на хорошо известных СУБД (MySQL, PostgreSQL, известных NoSQL-базах) или структурах (Btree), а также на распределенной файловой системе (HDFS).

Основная задача – сохранить конфиденциальность данных при работе на облаке, которому не доверяют, и оставить возможность использовать запросы SQL-типа на зашифрованных данных.

Для этого применяются различные методы шифрования. Так, например, в CryptDB используются различные схемы шифрования (детерминированное, гомоморфное, шифрование, сохраняющее порядок, шифрование, позволяющее выполнять операции присоединения). Следует заметить, что не всегда эти методы шифрования одинаково пригодны. Например, при использовании шифрования, сохраняющего порядок, пассивный противник путем наблюдения может собрать статистику. При этом он не должен определять точное значение данных p, соответствующее зашифрованному значению c. Нарушение конфиденциальности может произойти даже в том случае, если противник может c определенной вероятностью оценить, что значение данных p находится в интервале $[p_1; p_2]$. Более того, несмотря на гибкий выбор шифрования, количество типов SQL запросов остается ограниченным.

Кроме того, для осуществления шифрования и расшифровки данных, планирования порядка выполнения запросов, предварительной обработки данных в архитектуру вводятся дополнительные компоненты (доверенные прокси-сервера, планировщики и пр.).

Заметим, что все это замедляет выполнение запросов. При этом некоторые схемы шифрования, необходимые для выполнения определенных запросов, не обеспечивают достаточную конфиденциальность данных, то есть при длительном наблюдении пассивный противник может получить значительную часть информации о конфиденциальных данных.

Таким образом, все системы, описанные выше, позволяют пользователям работать в привычной парадигме SQL-запросов. Однако при этом ограничивается спектр возможных запросов, а для некоторых разрешенных запросов заведомо повышается возможность утечек конфиденциальных данных за счет выбора менее надежных схем шифрования. Кроме того, большая часть запросов выполняется не в один раунд, что сильно замедляет время его выполнения.

Все это заставляет исследовать альтернативные алгоритмы для работы с базами данных, особенно тогда, когда пользователь не ограничен рамками SQL-запросов, а имеет возможность задавать запрос на вычисление некоторых разрешенных для вычисления функций над зашифрованными данными.

В этом случае протокол и модели облачных вычислений на основе криптосерверов, предложенные в [12,13], являются более предпочтительным способом работы с конфиденциальными данными, поскольку для этой модели доказана ее стойкость [14] и дедуктивная безопасность запросов к базам данных [15].

Кроме того, в [16] были предложены методы реализации вычислений над конфиденциальными данными для предложенной модели при помощи пакетной обработки данных на основе СПО. Это позволяет использовать хорошо известное программное обеспечение (HDFS, PostgreSQL и т.д.) для работы с конфиденциальными данными, а также выполнять их быструю обработку.

Такой подход прекрасно работает, когда, например, необходимо отнести человека (пациента) к определенной группе лечения по некоторому заболеванию. При этом имеется значительное число показателей, которые должны быть учтены. Для этого потребуется вычислить только статистические функции, которые являются дедуктивно безопасными. Соответствующая выборка должна производиться на основе некоторого идентификатора личности, который необходимо хранить в зашифрованном виде. Идентификатором может быть, например, UUID (Universally Unique Identifier – универсальный уникальный идентификатор), который позволяет распределенным системам уникально идентифицировать информацию без обращения к единому центру координации. Во многих СУБД СПО, например, в MySQL, существует встроенная функция, генерирующая UUID и возвращающая значение, которое представляет собой 128-битное число, в виде строки формата utf8 из пяти

шестнадцатеричных чисел.

В этом случае в СУБД хранится таблица, аналогичная таблице паролей. Этот первичный ключ позволит сгруппировать записи, относящиеся к человеку, и собрать статистическую информацию, не раскрывая конфиденциальные данные. Далее производится обработка больших объемов информации при помощи упомянутого СПО (уровень пакетной обработки) и результат возвращается пользователю.

9. Заключение

Вопрос сохранения конфиденциальности данных на облаке является одним из наиважнейших, поскольку одним из участников протокола доступа к облачным хранилищам данных может быть противник.

В предложенном обзоре рассмотрены различные подходы к работе с конфиденциальными данными на облаке. Эти подходы базируются на широко используемых СУБД (MySQL, PostgreSQL, NoSQL базе данных MongoDB), структурах (Btree), а также на распределенной файловой системе (HDFS).

Программные средства ZeroDB, CryptDB, MONOMI, Arx и Seabed, разработанные на основе вышеупомянутых СУБД и файловой системы, позволяют работать в парадигме SQL-запросов. Конфиденциальность обеспечивается при помощи использования различных схем шифрования (в зависимости от разрешенных запросов) и введения в архитектуру дополнительных элементов, таких как доверенные прокси-сервера, планировщики и некоторые другие, которые позволяют организовать вычислительный процесс.

Из недостатков рассмотренных средств следует отметить, что в некоторых случаях при выполнении запросов приходится использовать схемы шифрования, которые не могут обеспечить полную конфиденциальность данных. Также работа планировщиков и других вспомогательных элементов архитектуры замедляет выполнение запросов.

Предложены методы работы с конфиденциальными данными для тех случаев, когда пользователь не ограничен парадигмой SQL-запросов.

Список литературы / References

[1] Huang C-T, Huan L, Qin Z, Yuan H, Zhou L, Varadharajan V, Jay Kuo C.-C. Survey on securing data storage in the cloud. APSIPA Transactions on Signal and Information Processing, vol. 3, 2014, article e7.

[2] X.800: Security architecture for Open Systems Interconnection for CCITT applications. URL: https://www.itu.int/rec/T-REC-X.800-199103-I/en, accessed 25.12.2020.

[3] Egorov M., Wilkison M. ZeroDB white paper. arXiv:1602.07168, 2016, 11 p.

[4] Popa A., Redfield C., Zeldovich N., and Balakrishnan H. CryptDB: Protecting Confidentiality with Encrypted Query Processing. In Proc. of the Twenty-Third ACM Symposium on Operating Systems Principles, 2011, pp 85-100.

[5] Felipe M.R., Mi Aung K.M., Ye X. and Yonggang W. StealthyCRM: A Secure Cloud CRM System Application that Supports Fully Homomorphic Database Encryption. International Conference on Cloud Computing Research and Innovation (ICCCRI), 2015, pp. 97-105.

[6] Halevi S. HElib. URL: https://github.com/shaih/HElib, accessed 25.12.2020.

[7] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. Lecture Notes in Computer Science, vol. 1592, 1999, pp. 223-238.

[8] Song D.X., Wagner D., and Perrig A. Practical techniques for searches on encrypted data. In Proc. of the 21st IEEE Symposium on Security and Privacy, 2000, pp. 44-55.

[9] Tu S., Kaashoek M. F.,Madden S., Zeldovich N. Processing Analytical Queries over Encrypted Data, Proceedings of the VLDB Endowment, vol. 6, no. 5, 2013, pp. 289-300.

[10] Papadimitriou A., Bhagwan R., Chandran N., Ramjee R., Singh H., Modi A. Big Data Analytics over Encrypted Datasets with Seabed. In Proc. of the 12th USENIX conference on Operating Systems Design and Implementation, 2016, pp. 587–602

[11] Poddar R., Boelter T., Popa A. Arx: An Encrypted Database using Semantically Secure Encryption. Proceedings of the VLDB Endowment, vol. 12, no. 11, 2019, pp. 1664-1678.

[12] Варновский Н.П., Мартишин С.А., Храпченко М.В., Шокуров А.В. Методы пороговой криптографии для защиты облачных вычислений, Труды ИСП РАН, том 26, вып. 2, 2014 г., с. 269-274 / Varnovskij N.P., Martishin S.A., Khrapchenko M.V., Shokurov A.V. A Threshold Cryptosystem in Secure Cloud Computations. Trudy ISP RAN/Proc. ISP RAS, vol. 26, issue 2, 2014, pp. 269-274 (in Russian). DOI: 10.15514/ISPRAS-2014-26(2)-12.

[13] Варновский Н.П., Мартишин С.А., Храпченко М.В., Шокуров А.В. Пороговые системы гомоморфного шифрования и защита информации в облачных вычислениях. Программирование, том 41, no. 4, 2015 г., стр. 47-51 / Secure cloud computing based on threshold homomorphic encryption. Varnovskiy N.P., Martishin S.A., Khrapchenko M.V., Shokurov A.V. Programming and Computer Software, vol. 41, no. 4, 2015, pp. 215-218.

[14] Варновский Н.П., Захаров В.А., Шокуров А.В. К вопросу о существовании доказуемо стойких систем облачных вычислений. Вестник Московского университета. Сер. 15. Вычислительная математика и кибернетика, no. 2, 2016 г., стр. 32-45. / Varnovsky N.P., Zakharov V.A., Shokurov A.V. On the existence of provably secure cloud computing systems. Moscow University Computational Mathematics and Cybernetics, vol. 40, no. 2, 2016, pp. 83-88

[15] Варновский Н.П., Захаров В.А., Шокуров А.В. О дедуктивной безопасности запросов к базам конфиденциальных данных в системе облачных вычислений. Вестник Московского университета. Серия 15: Вычислительная математика и кибернетика, no. 1, 2017 г., стр. 38-44 / Varnovsky N.P., Zakharov V.A., Shokurov A.V. On the deductive security of queries to confidential databases in cloud computing systems. Moscow University Computational Mathematics and Cybernetics, vol. 41, no. 1, 2017, pp. 38-43.

[16] Мартишин С.А., Храпченко М.В. Организация облачных вычислений над конфиденциальными данными на СПО. Труды 15-й конференции «Свободное программное обеспечение в высшей школе», 2020 г., стр. 171-174 / Martishin S.A., Khrapchenko M.V. Organization of cloud computing over confidential data on open source software. In Proc/ of the 15th Conference on Free Software in Higher Education, 2020, pp. 171-174 (in Russian).

Информация об авторах / Information about authors

Сергей Анатольевич МАРТИШИН, кандидат наук, научный сотрудник. Научные интересы: защита информации, облачные вычисления, вычисления над зашифрованными данными, гомоморфные вычисления, базы данных, СУБД.

Sergey Anatolyevich MARTISHIN, Candidate of Science, Research Fellow. Research interests: information security, cloud computing, computing over encrypted data, homomorphic computing, databases, DBMS.

Marina Valeryevna KHRAPCHENKO, Research Fellow. Research interests: cryptography, cloud computing, free software.

Марина Валерьевна ХРАПЧЕНКО, научный сотрудник. Научные интересы: криптография, облачные вычисления, свободное программное обеспечение.

Александр Владимирович ШОКУРОВ, кандидат наук, доцент, ведущий научный сотрудник ИСПРАН, доцент МФТИ. Сфера научных интересов: гомоморфное шифрование, облачные вычисления, криптография на решетках, алгебра.

Alexander Vladimirovich SHOKUROV, Candidate of Physics and Mathematics, Associate Professor, Leading Researcher, ISP RAS, Associate Professor at MIPT. Research interests: homomorphic encryption, cloud computing, lattice cryptography, algebra.