Министерство науки и высшего образования Российской Федерации
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ (НИ ТГУ)
Институт прикладной математики и компьютерных наук
Кафедра защиты информации и криптографии

ДОПУСТИТЬ К ЗАЩИТЕ В ГЭК
Руководитель ООП
канд. техн. наук, доцент
_____ С. А. Останин
“ 22 ” января 2020 г.

ДИПЛОМНАЯ РАБОТА

РАЗРАБОТКА БИБЛИОТЕКИ ДЛЯ ОБРАБОТКИ ЗАПРОСОВ НАД ЗАШИФРОВАННОЙ БАЗОЙ ДАННЫХ

на основной образовательной программе подготовки специалистов
«Анализ безопасности компьютерных систем»
специальности подготовки 10.05.01 Компьютерная безопасность

Салосин Иван Анатольевич

Руководитель ВКР
канд. техн. наук, доцент
_____ Тренькаев В. Н.

Автор работы
студент группы №1145
_____ Салосин И. А.

Томск — 2020

Министерство науки и высшего образования Российской Федерации
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ (НИ ТГУ)
Институт прикладной математики и компьютерных наук
Кафедра защиты информации и криптографии

УТВЕРЖДАЮ
Руководитель ООП
канд. техн. наук, доцент
_____ С. А. Останин
« 9 » сентября 20 19 г.

ЗАДАНИЕ

по подготовке ВКР специалистра студенту Салосину Ивану Анатольевичу группы 1145.

1. Тема ВКР: «Разработка библиотеки для обработки запросов над зашифрованной базой данных».

2. Срок сдачи студентом выполненной дипломной работы:

а) на кафедре: 22.01.2020

б) в ГЭК: 31.01.2020

3. Исходные данные к работе: Цель работы: разработка высокопроизводительной библиотеки для поддержки обработки SQL-запросов над зашифрованной базой данных без расшифрования конфиденциальных данных на стороне недоверенного сервера БД. Объекты исследования: схемы гомоморфного шифрования, схемы шифрования, сохраняющие порядок, архитектуры защищенных облачных СУБД. Методы исследования: аналитический обзор современной научно-технической литературы, затрагивающей проблематику построения защищеных облачных СУБД, сравнительное тестирование производительности предложенной схемы поддержки операций над зашифрованными данными.

4. Краткое содержание работы: изучение существующих способов обрабатываемого шифрования (шифрование, сохраняющее порядок, гомоморфное шифрование и пр.), а также архитектур СУБД, использующих эти виды шифрования (к 01.10.2019), изучение существующих библиотек с поддержкой гомоморфного шифрования (к 01.11.2012), разработка архитектуры и реализация прототипа библиотеки для поддержки обработки запросов над зашифрованной базой данных (к 20.12.2012), сравнительное тестирование предложенного решения, анализ полученных результатов и написание текста дипломной работы (к 21.01.2020).

5. Работа выполняется по заданию кафедры защиты информации и криптографии Томского государственного университета.

6. Дата выдачи задания « 5 » сентября 20 19 г.

Руководитель дипломной работы,
канд. техн. наук, доцент _____ / Тренькаев В. Н./
Задание принял к исполнению,
студент группы №1145 _____ / Салосин И. А./

РЕФЕРАТ

В работе рассматриваются способы защиты облачной СУБД. Предметом исследования являются алгоритмы и протоколы, применяемые в защищенных СУБД для реализации операций над зашифрованными данными. Рассмотрены полностью и частично гомоморфные схемы шифрования, детерминированные шифры и шифры, сохраняющие порядок шифртекстов. Приводится описание реализации библиотеки на языке Python для выполнения SQL запросов над зашифрованной базой данных без расшифровки самих данных и прототипа веб приложения, использующего библиотеку. Проведено тестирование производительности библиотеки.

Ключевые слова: гомоморфные шифры, шифры с сохранением порядка, защищенная субд, шифрованная бд.

Работа содержит 34 страницы, 13 рисунков, 12 литературных источников.

ОГЛАВЛЕНИЕ

Введение 5

1 Задача обеспечения безопасности веб-приложения с облачной БД 7

2 Шифры, позволяющие производить операции над зашифрованными данными 9

2.1 Гомоморфные схемы шифрования 9

2.2 Детерменированные шифры 11

2.3 Сохраняющие порядок шифры 12

2.4 Частично гомоморфные шифры 13

3 Защищенные облачные СУБД 15

3.1 CryptDB 15

3.2 ZQL 16

3.3 Monomi 17

4 Библиотека для обработки SQL-запросов над зашифрованной БД 19

4.1 Архитектура библиотеки 19

4.2 Основные классы реализации 20

4.3 Функции на стороне СУБД 26

4.4 Прототип веб-приложения медицинского учреждения 27

4.5 Исследование производительности библиотеки 28

Заключение 33

Список использованных источников и литературы 34

ВВЕДЕНИЕ

Современные информационные технологии невозможно представить без облачных вычислений, когда клиенту (подписчику) предоставляется третьей стороной (провайдером) сетевой доступ к распределенным и совместно используемым вычислительным ресурсам. Среди многочисленных облачных сервисов, можно выделить база данных как сервис (DBaaS), при котором потребителю предоставляется база данных (БД), развернутая на облачной платформе, так называемая облачная БД.

При всех достоинствах, в частности возможности масштабирования БД, активному использованию облачных БД препятствует отсутствие доверия клиентов провайдеру сервиса. Возможна ситуация, когда провайдер корректно выполняет протокол всех действий необходимых для предоставления услуги, но при этом ведет наблюдение за данными, обращаемыми в облачной среде, что может привести к раскрытию конфиденциальной информации.

Как известно, задача обеспечения конфиденциальности информации может быть решена с помощью шифрования, т.е. такого преобразования, которое делает прочтение информации невозможным лицам, которые не обладают секретным ключом.

Шифрование БД — это распространенная опция современных коммерческих СУБД. Например, в продуктах от Microsoft и Oracle используется так называемое «прозрачное» (Transparent Database Encryption, TDE) шифрование, когда данные шифруются перед записью на диск и расшифровываются во время чтения в память, т.е. прозрачно для приложений. Такой подход небезопасен для случая облачной БД, т.к. ключ при этом становится доступен облачному провайдеру. Возможен другой вариант, когда данные шифруются перед тем, как передаваться в облачный сервис. В этом случае возможно существенное снижение быстродействия обработки запросов к БД, в частности из-за невозможности использовать при поиске механизма индексирования данных, необходимости расшифровки при манипуляции с данными.

Наиболее перспективным является использование шифров, позволяющих производить операции над зашифрованными данными (гомоморфные шифры, сохраняющие порядок шифры и др.) [1—5]. Однако, обычно данные шифры позволяют эффективно выполнять только ограниченный набор вычислений над зашифрованными данными. Также такие шифры часто требуют дополнительных условий для обеспечения своей работы и безопасности, не всегда выполнимых на практике. Более того, на данный момент не предложено доказуемо криптостойкого шифра, позволяющего эффективно вычислять над зашифрованными данными все необходимые для БД операции (сложение, умножение, сравнение, поиск и т.д.). Возможным кандидатом для использования в защищенной облачной СУБД является полностью гомоморфный шифр на базе схемы Джентри [6] и его модификации, например [7], но препятствием служит большой коэффициент расши-

рения объема данных при шифровании таких схем, что не позволяет использовать их в реальных приложениях.

Тем не менее, существует достаточно много исследовательских проектов по разработке СУБД [8—10], которые поддерживают специализированные схемы шифрования, позволяющие обрабатывать запросы над зашифрованными данными. Данные СУБД основаны на применении комбинации различных видов шифров, часто трех вариантов: шифра с сохранением порядка, частично гомоморфного шифра и традиционного шифра. Отличие заключается в наборах поддерживаемых запросов и наборах используемых шифров, в особенностях протоколов взаимодействия приложения и сервера БД. Данная работа также посвящена задаче практического применения шифров, позволяющих производить операции над зашифрованными данными. Предлагается схема внедрения специализированных схем шифрования в веб-приложения, серверная часть которых хранит данные в облачной БД. Предложенная схема реализует концепцию разнотипного шифрования, когда для поддержания разных операций над данными используются разные шифры. Для обработки условных запросов к БД с предикатами сравнения на равенство используется детерминированный блочный шифр AES в режиме SIV [11], а для предикатов сравнения порядка – шифр с сохранениям порядка BCLO [5]. Обработка агрегированных арифметических запросов (сумма, среднее и т.д.) базируется на аддитивно гомоморфном шифре Пэйе [12] и поддержки операции умножения (деления) на стороне приложения.

Предложенная схема реализована в виде библиотеки для серверной части веб-приложения (бэкенда). Считается, что библиотека подключается на стороне доверенного веб-сервера. Перед работой требуется сконфигурировать окружение библиотеки, определив адрес БД, схему таблицы БД, выбрав операции над зашифрованными данными. Клиентская часть веб-приложения (фронтенд) передает запрос к серверной части веб-приложения (бэкенда) в открытом виде. На основе конфигурации библиотека шифрует необходимые данные, после чего формируется SQL-запрос к облачной СУБД, которая выполняет его уже над зашифрованными данными. Библиотека написана на языке Python, в качестве СУБД выбрана популярная и свободно распространяемая PostrgreSQL. Для поддержки шифра Пэйе на стороне СУБД используются пользовательские функции.

Дипломная работа состоит из четырех разделов. В первом разделе ставится задача обеспечения безопасности веб-приложений при использовании облачных БД. Во втором разделе рассматриваются схемы шифрования, позволяющие производить операции над зашифрованными данными, в том числе, использованные при разработке библиотеки. Третий раздел посвящен обзору существующих защищенных облачных СУБД. Четвертый раздел отводится под описание архитектуры и подробностей реализации библиотеки.

1 Задача обеспечения безопасности веб-приложения с облачной БД

Угрозы безопасности в СУБД, как и угрозы безопасности информации в целом классически разделяют на три вида.

1. Угроза конфиденциальности. Эта угроза заключается в том, что доступ к информации получает тот, кто на это не имеет права.

2. Угроза целостности. Под нарушением целостности понимают неправомерное изменение данных.

3. Угроза доступности. Угрозы нарушения доступности — это угрозы, связанные с действиями, которые влекут за собой полную или частичную потерю доступа к информации теми, кто имеет на это право.

В современных веб приложениях СУБД является неотъемлемой частью архитектуры. Зачастую СУБД физически находится на доверенном сервере, подконтрольном владельцу веб приложения.

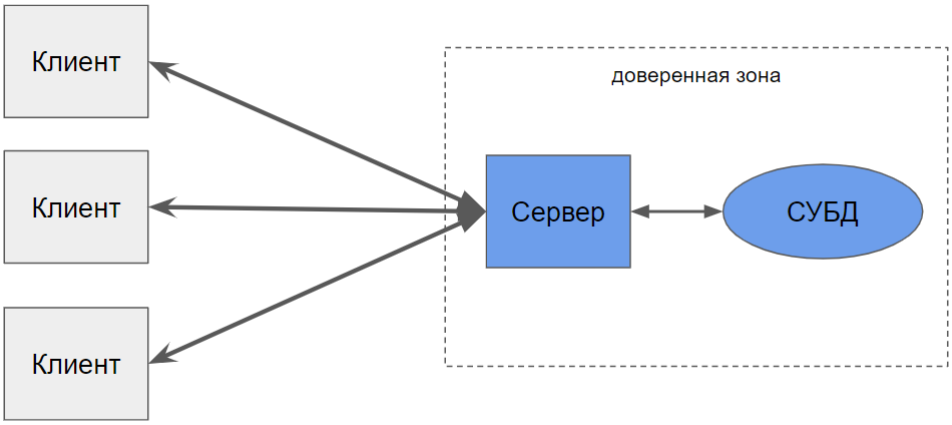

Рисунок 1 — Архитектура веб приложения с СУБД на доверенном сервере

В такой архитектуре для защиты конфиденциальности данных, в частности, применяют шифрование данных. В дальнейшем для обработки запросов к СУБД необходимые данные расшифровываются, выполняется запрос и данные снова шифруются.

Однако, не всегда СУБД находится на доверенном сервере. Она может быть перенесена в облачный севрис, далее «облако».

Облако — сервис, предоставляющий сетевой доступ к некоторому набору вычислительных ресурсов. В случае СУБД в качестве ресурса рассматривается хранилище и обработка данных. Физическая структура сервера, на котором хранятся данные, клиенту в общем случае неизвестна. Это может быть как один сервер, так и распределенная сеть серверов. Чаще всего облако предоставляется клиенту третьей стороной.

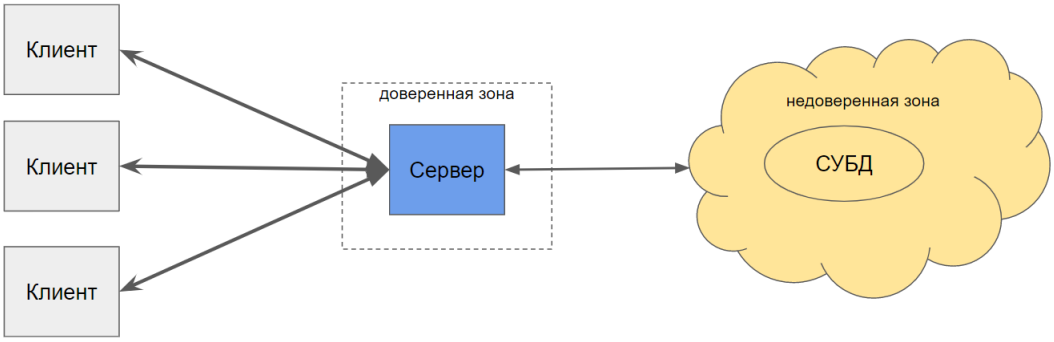

Рисунок 2 — Архитектура веб приложения с СУБД на облачном сервере

Такой подход применяют при нехватке места на доверенном сервере, недостатке вычислительных мощностей или в целях финансовой экономии.

При такой архитектуре становится невозможно привычное шифрование. Расшифровка данных на стороне СУБД, находящейся на недоверенном сервере, означает открытие этих данных тому, кто может получить доступ к недоверенному серверу. Это может быть не только злоумышленник, получивший доступ к серверу путем эксплуатации уязвимости, но и недобросовестный владелец сервера. В рамках данной работы рассматриваются способы обеспечения конфиденциальности данных в удаленной СУБД.

2 Шифры, позволяющие производить операции над зашифрованными данными

2.1 Гомоморфные схемы шифрования

Невозможность расшифрования данных на недоверенном сервере ставит задачу поиска иных путей обеспечения конфиденциальности данных в удаленной СУБД. Одним из решений может стать использование схем шифрования, сохраняющих возможность производить операции над шифртекстами. Такое шифрование называют гомоморфными схемами шифрования.

Гомоморфная схема шифрования — это схема шифрования, позволяющая производить некоторые математические операции над шифртекстами. Результат применения операций над шифртекстами соответствует зашифрованному результату применения операций к открытым текстам. Дадим математическое определение.

Пусть k — ключ шифрования, m — открытый текст, который необходимо зашифровать. Функция $Enc(m, k) = cm$ выполняет шифрование сообщения, а $Dec(cm, k) = m$ выполняет расшифрование.

Функция $Enc(m, k)$ является гомоморфной относительно операции $*$, если существует алгоритм M, требующий полиномиального числа ресурсов и работающий за полиномиальное время, который, получив на вход два шифртекста $Enc(m_1, k)$ и $Enc(m_2, k)$ выдаст на выходе шифртекст $z = M(Enc(m_1, k), Enc(m_2, k))$ такой, что $Enc(m_1 * m_2, k) = z$.

Обозначим символом $\oplus$ операцию сложения над шифртекстами, соответствующую операции сложения над открытыми текстами. Аналогично, символом $\otimes$ обозначим операцию умжножения над шифртекстами.

Говорят, что шифрсистема является:

1. *гомоморфной относительно сложения*, если $Enc(m, k)$ гомоморфна относительно $\oplus$, то есть $Dec(Enc(m_1, k) \oplus Enc(m_2, k)) = m_1 + m_2$.

2. *гомоморфной относительно сложения*, если $Enc(m, k)$ гомоморфна относительно $\otimes$, то есть $Dec(Enc(m_1, k) \otimes Enc(m_2, k)) = m_1 * m_2$.

3. *полностью гомоморфной*, если если $Enc(m, k)$ гомоморфна относительно $\oplus$ и $\otimes$ одновременно, то есть

$$\begin{cases} Dec(Enc(m_1, k) \oplus Enc(m_2, k)) = m_1 + m_2 \\ Dec(Enc(m_1, k) \otimes Enc(m_2, k)) = m_1 * m_2 \end{cases}$$

Операции сложения и умножения над битами образуют полный по Тьюрингу базис, поэтому, если шифрсистема может зашифровать два бита, то становится возможным вычислить любую булеву функцию над шифртекстами.

Первой полностью гомоморфной схемой шифрования стала криптосистема Крейга Джентри, описаная им в 2009 году [6]. Приведем ее упрощенный пример.

Алгоритм шифрования:

1. Выбирается случайное нечетное число $p = 2k+1$. Это число является секретным ключом. Пусть необходимо зашифровать сообщение $m \in \{0, 1\}$.

2. Выбирается произвольное r, называемое шумом. Составляется $z = 2r + m = m \ mod \ 2$.

3. В результате шифрования m ставится в соответствие $c = z + pq$, где q является произвольным числом.

Алгоритм расшифрования:

Пусть даны секретный ключ p и зашифрованноле сообщение c. Для получения сообщения необходимо

1. Произвести вычисление $z = c \ mod \ p$:

$$c \ mod \ p = (z + pq) \ mod \ p = z \ mod \ p$$

2. Получить открытый текст $m = z \ mod \ 2$.

Стоит отметить, что для успешного расшифрования необходимо, чтобы при шифровании выполнялось $c = z \ mod \ p$. Легко заметить, что это происходит при условии

$$r \in \left(-\frac{p}{2}, \frac{p}{2}\right).$$

Покажем гомоморфизм этой схемы шифрования по сложению. Пусть есть два открытых сообщения $m_1, m_2 \in \{0, 1\}$, а секретный ключ $p = 2k+1$. Произведем шифрование сообщений:

$$c_i = z + i + pq_i, i \in \{1, 2\}.$$

Вычислим сумму шифртекстов:

$$c_1 + c_2 = z_1 + pq_1 + z_2 + pq_2 = z_1 + z_2 + p(q_1 + q_2) = 2(r_1 + r_2) + m_1 + m_2 + p(q_1 + q_2).$$

Расшифруем сумму:

$$((c_1 + c_2) \ mod \ p) \ mod \ 2) = (2(r_1 + r_2) + m_1 + m_2) \ mod \ 2 = m_1 + m_2.$$

Аналогичным образом покажем гомоморфизм по умножению. Вычислим произведение шифртекстов:

$$\begin{aligned} c_1 * c_2 = (z_1 + pq_1)(z_2 + pq_2) = z_1 z_2 + p(z_1 q_2 + z_2 q_1) + p^2 q_1 q_2 = \\ = (2r_1 + m_1)(2r_2 + m_2) + p(z_1 q_2 + z_2 q_1) + p^2 q_1 q_2 = \\ = (4r_1 r_2 + 2(r_1 m_2 + r_2 m_1) + m_1 m_2) + p(z_1 q_2 + z_2 q_1) + p^2 q_1 q_2. \end{aligned}$$

Расшифруем произведение:

$$((c_1 * c_2) \ mod \ p) \ mod \ 2 = (4r_1 r_2 + 2(r_1 m_2 + r_2 m_1) + m_1 m_2) \ mod \ 2 = m_1 * m_2.$$

Эта схема является неприменимой на практике из-за быстрого роста шума, что делает невозможным многократное выполнение операций над шифртекстами. Крейг Дженти разработал алгоритм уменьшения этой ошибки (bootstrapping). Также эта схема претерпела множество доработок и модификаций, в результате которых появлялись улучшенные ее версии.

Сейчас в сети Интернет можно найти множество открытых реализаций полностью гомоморфных шифрсистем. Наиболее известные из них:

1. HELib от IBM.
2. SEAL от Microsoft.
3. PALISADE.
4. HEAAN.
5. FHEW.

К сожалению, на сегодняшний момент схемы полного гомоморфного шифрования в некоторых случаях показывают серьезное падение в производительности, что недопустимо для веб приложений, где разработчики борются за каждую миллисекунду ожидания пользователя.

В веб приложениях определяющую роль играет производительность. Пользователи редко готовы ждать длительного ответа от сервера. Альтернативой полностью гомоморфному шифрованию может стать использование таких схем, которые сохраняю лишь часть математических операций с шифртекстами, но обладают гораздо большей производительностью. В этой главе будут рассмотрены такие схемы шифрования. Также будут приведены некоторые подробности схем, использованных в реализованной библиотеке.

2.2 Детерменированные шифры

Детерменированный шифр — такой шифр, который одинаковые открытые тексты на одинаковом ключе шифрует в одинаковые криптограммы.

Другими словами, если x и y — открытые тексты, k — ключ шифрования, а $x = y$, то $Enc(x, k) = Enc(y, k)$.

Примером такого шифра может шифр Цезаря — один из самых известных методов шифрования. В шифре Цезаря ключом является число k. Чтобы зашифровать сообщение, необходимо каждый его символ сдвинуть право по алфавиту на k символов вправо, а чтобы расшифровать криптограмму, необходимо каждый ее символ сдвинуть на k символов влево.

Например, пусть $k = 3$, сообщение $m = attackatdown$. Тогда криптограммой будет $Enc(attackatdown, 3) = dwwdfndwgrzq$.

В СУБД такие шифры можно использовать для сохранения отношения эквивалентности. Например, веб приложению необходима запись из таблицы пользователей с именем «Алиса». Если весь столбец имен зашифрован детерменированным шифром с одинаковым ключом, то веб приложению необходимо зашифровать искомое имя «Алиса» на том же ключе и просто заменить его в запросе к СУБД на полученную криптограмму.

Кроме запросов SELECT с предикатами равенства (или неравенства) использование детерменированных шифров в СУБД позволяет выполнять запросы, содержащие DISTINCT, GROUP BY и COUNT.

В реализации библиотеки, описанной в данной работе, в качестве детерменированного шифра был использован AES-SIV [11]. Этот режим шифрования не использует случайный *nonce*, благодаря чему одинаковые открытые тексты шифруются в одинаковые шифртексты. Для генерации вектора инициализации используется AES-CMAC, а непосредственно шифрование происходит в режиме CTR. $H_1, ..., H_m$ являются необязательными заголовками, участвующими в формирвовании вектра инициализации.

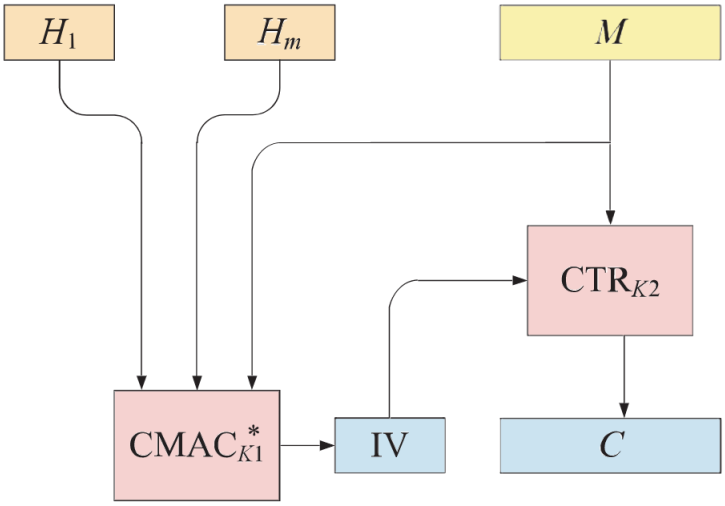

Рисунок 3 — Режим AES-SIV

2.3 Сохраняющие порядок шифры

Дать определение сохраняющему порядок шифру нагляднее всего следующим образом: если x и y — открытые тексты, k — ключ шифрования и $x > y$, то $Enc(x, k) > Enc(y, k)$. Такие шифры называют классом OPE шифров (или просто OPE шифрами) от английского *order preserving encryption*.

Сохраняющий порядок шифр может быть успешно интегрирован в СУБД. Например, веб прилоению необходимы все записи из таблицы товаров, стоящих меньше определенной суммы. Если весь столбец цены зашифрован шифром, сохраняющим порядок, то веб приложению необходимо зашифровать искомую сумму этим шифром и просто заменить настоящую сумму в запросе к СУБД на полученную криптограмму.

Кроме обычных SELECT запросов с предикатами $<$ и $>$ использование шифров, сохраняющих порядок в СУБД позволят выполнять запросы, содержащие оператор сортировки ORDER BY, а также BETWEEN, MAX, MIN, RANK.

Примером простой схемы шифрования может быть биективное отображение с сохранением порядка. Пусть M и N два положительных целых числа, $M < N$. Обозначим $[M] = \{1, 2, ..., M\}$, а $[N] = \{1, 2, ..., N\}$. Случайно выберем M элементов из $[N]$, упорядочим их. Чтобы зашифровать $i \in [M]$ необходимо поставить ему в соответствие i-й элемент из выбранного множества, для расшифрования необходимо проделать обратную операцию.

Одной из проблем этой схемы шифрования является необходимость хранить отображение целиком. Решение нашла Александра Болдырева. Она предложила способ «ленивого представления» данного отображения, используя свойства отрицательного гипергеометрического распределения [5]. Эту схему шифрования назвали BCLO и использовали в реализациях защищенных облачных СУБД, описанных в следующей главе. Схема шифрования является эффективной, но предоставляет дополнительную информацию о половине бит открытого текста.

Решением проблемы раскрытия половины бит может стать использование шифра mOPE. Испоьзование этого шифра предполагает наличие клиента и сервера. Добавление нового значения в уже имеющуюся базу данных требует перешифрования некоторых записей, а поиск необходимой записи требует постоянного общения между клиентом и сервером, из-за чего производительность этого шифра может оказаться недостаточной.

2.4 Частично гомоморфные шифры

Во второй главе уже давалось полное определение полностью гомоморфной схемы шифрования. Напомним, что *полностью гомоморфной схемой шифрования* называется такая схема, которая гомоморфна относительно операций сложения $\oplus$ и умножения $\otimes$.

$$\begin{cases} Dec(Enc(m_1, k) \oplus Enc(m_2, k)) = m_1 + m_2 \\ Dec(Enc(m_1, k) \otimes Enc(m_2, k)) = m_1 * m_2 \end{cases}$$

В отличии от полностью гомоморфной, *частично гомоморфная схема шифрования* обязана быть гомоморфной относительно только одной операции. Соответственно, выполняется только одно из равенств.

Примером частично гомоморной схемы шифрования является знакомая всем RSA, гомоморфная по умножению. Пусть (e, n) — открытый ключ, m_1 и m_2 — открытые сообщения, функция шифрования $E(m) = m^e \ mod \ n$. Покажем гомоморфизм по умножению:

$$D(E(m_1) * E(m_2)) = D(m_1^e * m_2^e \ mod \ n) = D((m_1 * m_2)^e \ mod \ n) = D(E(m_1 * m_2)) = m_1 * m_2.$$

Частично гомоморфные схемы шифрования лишены огромного плюса схем полного гомоморфного шифрования — способности вычислять разнообразные функции над шифртекстами. Вместе с этим они обладают гораздо большей производительностью, что является ключевым фактором для веб приложений.

Одной из самых популярных схем шифрования, гомоморфных по сложению, является криптосистема Пайе [12]. Она, как и RSA, основана на сложности факторизации составного числа, являющегося произведением двух простых чисел.

Алгоритм генерации ключей:

1. Генерируются два больших простых числа p и q таким образом, чтобы выполнялось равенство $gcd(pq, (p-1)(q-1)) = 1$.

2. Вычисляются $\lambda = lcm(p-1, q-1)$ и $n = pq$.

3. Генерируется случайное целое число g, такое что $0 \leqslant g < n^2$.

4. Вычисляется $\mu = (L(g^\lambda \ mod \ n^2))^-1 \ mod \ n$, где $L(x) = \frac{x-1}{b}$.

5. Пара чисел (n, g) будут открытым ключом, а (λ, μ) закрытым.

Алгоритм шифрования:

1. Пусть шифруется открытое сообщение m, такое что $0 < m < n$.

2. Генерируется случайное число r, $0 \leqslant r < n$.

3. В соответствие числу m ставится $c = g^m * r^n \ mod \ n^2$.

Алгоритм расшифрования:

1. Имеется шифртекст c, $0 \leqslant c < n^2$.

2. Исходным сообщением будет $m = (L(c^\lambda \ mod \ n^2)) * \mu) \ mod \ n$.

Гомоморфизм по сложению:

$$Enc(m_1) * Enc(m_2) = (g^{m_1} * r_1^n) * (g^{m_2} * r_2^n) \ mod \ n^2 =$$
$$= g^{m_1+m_2} * (r_1 * r_2)^n \ mod \ n^2 = Enc(m_1 + m_2).$$

Криптосистема Пайе хорошо себя зарекомендовала и используется в реализациях защищенных облачных СУБД, описанных в следующей главе.

3 Защищенные облачные СУБД

Идея реализации защищенной облачной СУБД, способной выполнять запросы над зашифрованными данными, была раскрыта в некоторых работах, описаных в этой главе.

3.1 CryptDB

Попыткой реализации СУБД, выполняющей запросы над зашифрованными данными, является CryptDB [8]. Разработчики этой системы озадачились проблемой недоверенного сервера, на котором находится СУБД. В архитектуре веб приложения, использующего CryptDB можно выделить следующие ключевые компоненты:

1. Сервер веб приложения.

2. Прокси-сервер, находящийся в доверенной зоне и имеющий доверенный канал связи с сервером веб приложения.

3. СУБД, находящуюся в недоверенной зоне, данные в которой зашифрованы. Канал между СУБД и прокси сервером предполагается недоверенным.

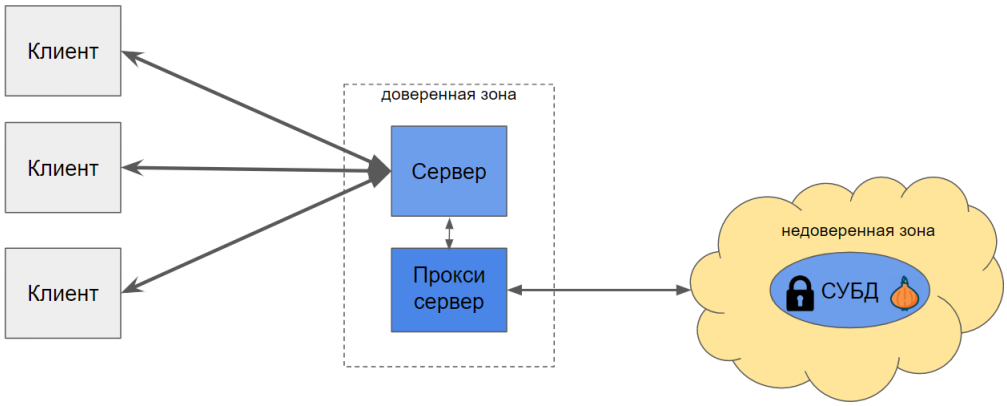

Рисунок 4 — Архитектура CryptDB

Запросы к СУБД идут от сервера веб приложения через прокси-сервер. Задачей прокси сервера является сформировать запрос таким образом, чтобы СУБД могла его выполнить над зашифрованными данными. Таким образом, для СУБД запросы выглядят как обычный SQL запрос, а все данные передаются по недоверенному каналу в зашифрованном виде.

Для того чтобы СУБД могла выполнять запросы над зашифрованными данными, для шифрования столбцов используются схемы шифрования, сохраняющие операции над шифртекстами. Для поддержания частично гомоморфного шифрования на стороне СУБД необходимо определить некоторые функции. Авторы CryptDB использовали MySQL, которая позволяет определять пользовательские функции (UDF), решающие эту задачу. Для

поддержки большого числа операций столбцы дублируются и каждая копия шифруется разными схемами шифрования. Кроме того, данные шифруются несколько раз, образуя «луковые» слои шифрования.

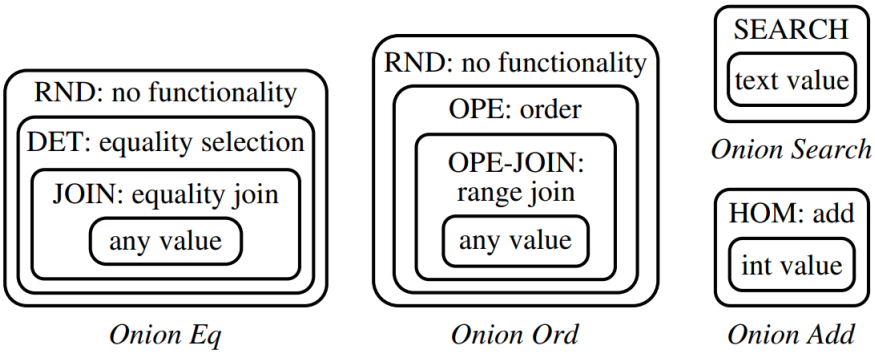

Рисунок 5 — «Луковые» слои шифрования [8]

Стоит отметить, что CryptDB это не готовый продукт, выпущенный на рынок, а только прототип, разработанный в целях исследования области.

3.2 ZQL

Одним из недостатков CryptDB было наличие прокси-сервера, усложняющего архитектуру веб приложения. На это обратили внимание и разработчики ZQL [10], защищенной СУБД, также производящей запросы над зашифрованными данными. Авторы отказались от необходимости в прокси-сервере, перенеся шифрование данных и формирование запроса к БД на сторону самого приложения, что позволило исключить лишние слои абстракции и добиться более высокой эффективности выполнения запросов.

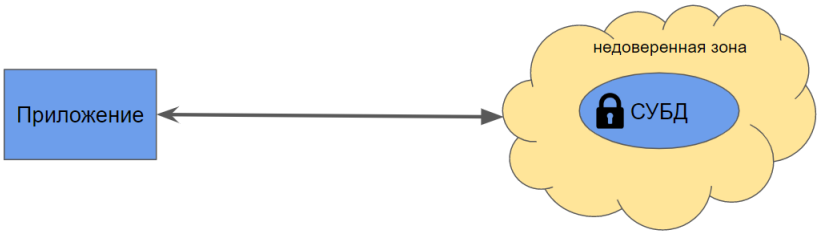

Рисунок 6 — Архитектура ZQL

Авторы ZQL в своей работе использовали только схему шифрования, сохраняющую порядок шифртекстов, чего было достаточно для их предметной области. В качестве такой схемы они выбрали шифр mOPE, обладающий большей криптостойкостью по сравнению с BCLO. Однако, этот шифр требует активного общения между приложением и

СУБД в процессе выполнения запросов. Авторам понадобилось спроектировать протокол такого общения. Если СУБД находится на облачном сервере, то передача данных от веб приложения к этому серверу может занимать существенное количество времени, что отрицательно сказывается на общей производительности системы.

3.3 Monomi

Другим недостатком CryptDB была плохая поддержка аналитических запросов. Авторы Monomi [9] указали на то, что CryptDB может выполнить только четыре из двадцати двух запросов TCP-H — бенчмарка для сравнения производительности аналитических систем и хранилищ данных. Они постарались решить эту проблему, представив прототип системы для работы с СУБД, находящейся на недоверенном сервере.

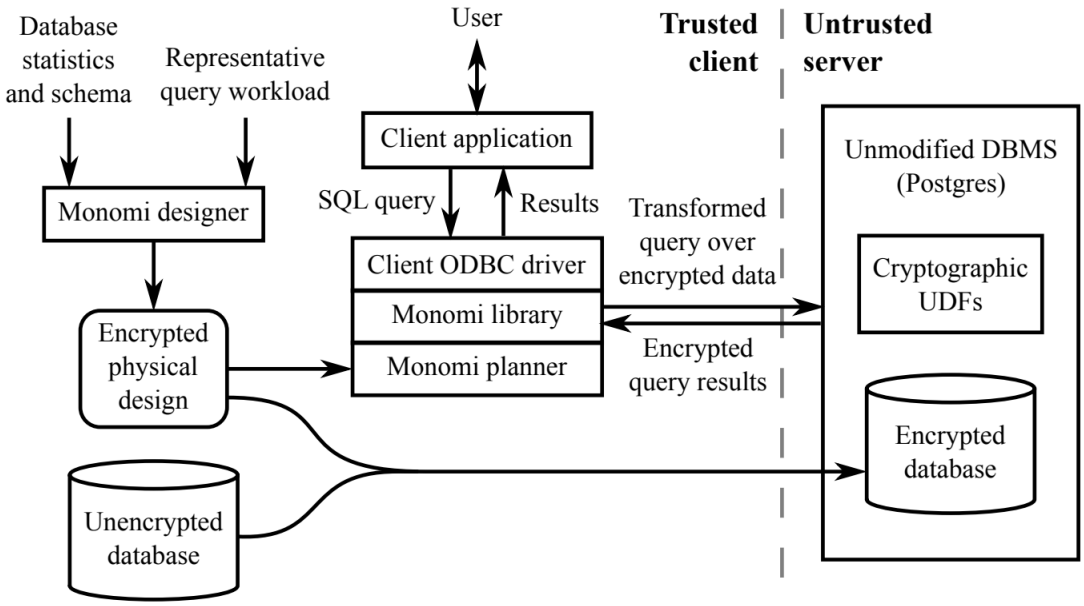

Рисунок 7 — Архитектура Monomi [9]

Основными элементами архитектуры являются проектировщик, библиотека и зашифрованная база данных, находящаяся на недоверенном сервере.

Проектировщик находится на доверенном сервере. Он необходим для составления архитектуры защищаемой базы данных. Для корректного составления архитектуры ему требуются типовые запросы, которые приложение планирует выполнять над базой данных. Авторы отмечают, что для получения хорошей эффективности набор этих операций не должен быть слишком велик.

ODBC интерфейс принимает немодифицированный SQL запрос. Для того чтобы запрос мог выполниться эффективно, его может быть необходимо разделить на несколько частей. Также выполнение запроса может проходить в несколько этапов: сначала СУБД, находящаяся на недоверенном сервере, делает некоторую выборку, далее Monomi исполь-

зует результат работы СУБД, расшифровывает необходимые данные и составляет новый запрос. Это происходит для достижения большей производительности.

4 Библиотека для обработки SQL-запросов над зашифрованной БД

Для реализации библиотеки был выбран один из самых популярных и широко распространенных языков программирования Python. Этот язык позволяет ускорить процесс разработки за счет минималистичного синтаксиса и огромного набора библиотек с готовыми реализациями типовых задач.

В качестве СУБД была выбрана PostgreSQL. Она популярна, свободно распространяется, легко настраивается и обладает широким функционалом.

Для сохранения возможности выполнять операции над шифртекстами были использованы схемы шифрования, упомянутые в третьей главе.

1. **AES-SIV** для отношения эквивалентности.
2. **BCLO** для отношения порядка.
3. **Paillier** для сложения.

4.1 Архитектура библиотеки

За основу была взята архитектура CryptDB. Необходимость прокси сервера была устранена за счет того, что библиотека подключается прямо на сервере веб приложения. Однако сервер обязан быть доверенным так как библиотека обеспечивает хранение ключей шифрования и производит само шифрование защищаемых данных.

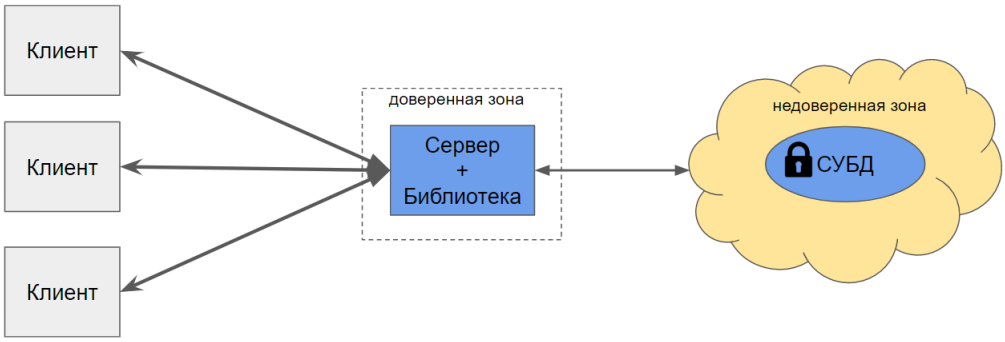

Рисунок 8 — Архитектура библиотеки

Передача данных по каналу связи может являться слабым местом при проверке производительности системы, поэтому в процессе выполнения запроса данные передаются по каналу между доверенным сервером и СУБД лишь один раз.

Библиотека не решает задачи доставки данных до клиента, поэтому, если веб приложение также не решает этой задачи рекомендуется расширить доверенную зону, поместив туда клиентов. Это можно сделать, например, при помощи VPN.

4.2 Основные классы реализации

Ключевым классом в библиотеке является *RequestBuilder*. Это класс для построения запросов к таблице, столбцы которой зашифрованы схемами шифрования.

```
1  class RequestBuilder:
2      def __init__(self):
3          self.fields = {}
4          self.fields_array = []
5
6      def set_table_name(self, name):
7          self.table_name = name
8
9      def add_field(self, name, operation):
10         field = Field(name, operation)
11         self.fields[name] = field
12         self.fields_array.append(field)
13         return field
14
15     ...
```

Листинг 1. Класс RequestBuilder

В веб приложении с помощью этого класса программист опишет шифруемую таблицу, передав в функцию *set_table_name* имя таблицы, а в функцию *add_field* имя колонки таблицы и сохраняемую в ней операцию. В качестве операции можно передать =, >, + или *none* для того, чтобы данные в столбце не шифровались. Таким образом, инициализация в веб приложении примет следующий вид:

```
1 request_builder = RequestBuilder()
2 request_builder.set_table_name("patients")
3 request_builder.add_field("id", "none")
4 request_builder.add_field("name", "=")
5 request_builder.add_field("age", ">")
6 request_builder.add_field("therapy_duration", ">")
7 request_builder.add_field("gen_before", "+")
8 request_builder.add_field("gen_after", "+")
9 request_builder.add_field("is_effective", "=")
```

Листинг 2. Пример инициализации RequestBuilder

Функция *add_field* возвращает объект, который веб приложение либо сохранит, либо впоследствии получит у *RequestBuilder* по имени столбца таблицы.

Когда веб приложению понадобится вставить новую запись в базу, необходимо будет создать *InsertRequest* и передать в его функцию *add_value* объекты, соответствующие столбцам таблицы, и значение, записываемое в базу.

```
1  class InsertRequest:
2      def __init__(self):
3          self.values = {}
4
5      def add_value(self, field, value):
6          self.values[field] = value
```

Листинг 3. Класс InsertRequest

Все данные между веб приложением и СУБД будут передаваться по потенциально незащищенному каналу, поэтому значения полей в запросе должны быть зашифрованы уже на стороне приложения. Для этого необходимо правильно сформированный *InsertRequest* передать в функцию *build_insert_args* объекта *RequestBuilder*.

```
1  def build_insert_args(self, insert_request):
2      column_names = ""
3      args = []
4      for field in insert_request.values.keys():
5          column_names += field.name + ", "
6          enc_value = None
7          if field.operation == "=":
8              enc_value = det_enc(insert_request.values[field])
9          elif field.operation == ">":
10             enc_value = ope_enc(insert_request.values[field])
11         elif field.operation == "+":
12             enc_value = he_enc(insert_request.values[field])
13         else:
14             enc_value = insert_request.values[field]
15         args.append(enc_value)
16     column_names = column_names[:-2]
17     args_str = ('%s, ' * len(insert_request.values))[:-2]
18     db_request = "INSERT INTO " + self.table_name
19     db_request += " (" + column_names + ") VALUES ("
20     db_request += args_str + ")"
21     return (db_request, args)
```

Листинг 4. Функция формирования insert запроса

Эта функция вернет кортеж, первый элемент которой является строкой (SQL запросом), а второй аргументами, которые нужно в него подставить при обращении к СУБД.

Как можно заметить, аргументы массива $args$ являются результатами шифрования значений, необходимых для вставки в таблицу. В зависимости от первоначального выбора сохраняемой операции, применяется детерминированная схема шифрования (det_enc), шифр, сохраняющий отношение порядка (ope_enc) или частично гомоморфная схема шифрования (he_enc). Также учитывается желание разработчика не шифровать столбец.

Для запросов типа SELECT необходимо создать объект $SelectRequest$.

```
1  class SelectRequest:
2      def __init__(self):
3          self.where = {}
4          self.avg_requests = []
5          self.search_fields = []
6
7      def add_condition(self, field, condition):
8          self.where[field] = condition
9
10     def add_avg_request(self, field):
11         self.avg_request.append(field)
12
13     def add_search_field(self, field):
14         self.search_fields.append(field)
```

Листинг 5. Класс SelectRequest

1. $add_condition$ используется для наложения условий на записи, входящие в выборку. В финальном запросе эти условия попадут в блок WHERE.

2. $add_avg_request$ является аналогом AVG в SQL запросе. В нужно передать $field$, операцией которого является $>$.

3. add_search_field используется для поиска определенных столбцов в таблице. Эту функцию нельзя одновременно использовать с $add_avg_request$, так как это не имеет смысла. Если эту функцию не вызвать, то в поиске будут участвовать все столбцы таблицы.

Для того, чтобы сформировать по $SelectRequest$ запрос к СУБД нужно вызвать функцию $build_select_request$ объекта $RequestBuilder$.

```
1  def build_select_request(self, select_request):
2      request = "SELECT "
3      avg_requests = select_request.avg_requests
4      where = select_request.where.items()
```

```
5     if (len(avg_requests) == 0):
6         if (len(select_request.search_fields) == 0):
7             request += "* "
8         else:
9             for field in search_fields:
10                request += field.name + " "
11    else:
12        he_cipher = get_he_cipher()
13        he_pub = cipher.pub.n
14        for field in avg_requests:
15            request += " paillier_sum(" + str(he_pub)
16            request += ", " + field.name + ")"
17        request += ", count(*) "
18    request += "FROM " + self.table_name + " "
19    if (len(where) != 0):
20        request += "WHERE 1=1"
21        for item in where:
22            field = item[0]
23            condition = item[1]
24            enc_value = None
25            if (field.operation == "="):
26                enc_value = det_enc(condition.value)
27            elif (field.operation == ">"):
28                enc_value = ope_enc(condition.value)
29            else:
30                raise Exception('Invalid operation')
31            request += ' AND ' + field.name + ' '
32            request += condition.condition + ' '
33            request += str(enc_value)
34    return request
```

Листинг 6. Функция формирования select запроса

В результате работы этой функции формируется запрос к СУБД, где значения полей уже зашифрованы соответствующими схемами шифрования.

Стоит обратить внимание на то, как формируется запрос в случае, когда необходимо найти среднее значение. Вызывается функция $paillier_sum(he_pub, field_name)$, которая производит операцию над шифртекстами, аналогичную сложению над открытыми текстами. Это доопределенная функция на стороне СУБД, описание которой можно найти в следующем разделе. Также поисходит посчет всех записей, подходящих под условия с помощью $count(*)$. Для того чтобы посчитать среднее значение, библиотека расшифрует полученную сумму и разделит ее на количество записей.

Сформированный запрос веб приложение может отправить СУБД своими средствами. Однако, для упрощения этой задачи в библиотеке есть класс *PostgreRepository*.

```
1  class PostgreRepository:
2      def __init__(self, dbname, user, host, password):
3          self.dbname = dbname
4          self.user = user
5          self.host = host
6          self.password = password
7
8      def select(self, request):
9          conn = psycopg2.connect(create_conn_args())
10         cursor = conn.cursor()
11         cursor.execute(request)
12         rows = cursor.fetchall()
13         cursor.close()
14         conn.close()
15         return rows
16
17     def insert(self, insert_args):
18         conn = psycopg2.connect(create_conn_args())
19         cursor = conn.cursor()
20         cursor.execute(insert_args[0], insert_args[1])
21         conn.commit()
22         cursor.close()
23         conn.close()
24
25     def create_conn_args(self):
26         conn_args = "dbname='" + self.dbname
27         conn_args += "' user='" + self.user
28         conn_args += "' host='" + self.host
29         conn_args += "' password='" + self.password + "'"
30         return conn_args
```

Листинг 7. Класс PostgreRepository

Этот класс использует библиотеку *psycopg*2 для соединения с PostgreSQL и предлагает разработчикам две функции:

1. *select* принимает на вход SQL запрос SELECT и возвращает таблицу, полученную в результате его выполнения.

2. *insert* принимает массив из двух элементов. Это строка, являющаяся SQL запросом и аргументы, которые подставляются в него. Функция позволяет выполнять транзакционные запросы INSERT.

Таблица, полученная в результате работы функции *select*, содержит зашифрованные данные. Веб приложению необходимо выполнить еще один шаг для того, чтобы получить открытую информацию. Для расшифрования данных необходимо вызвать функцию *decrypt_rows* объекта *RequestBuilder*:

```
1  def decrypt_rows(self, rows, select_request):
2      if (len(rows) == 0):
3          return rows
4      operations = self.form_operations(select_request)
5      d_rows = []
6      if (len(operations) != len(rows[0])):
7          raise Exception("Wrong length")
8      if len(select_request.avg_requests) > 0:
9          j = 0
10         last_index = len(select_request.avg_requests)
11         row = rows[0]
12         d_row = []
13         while j < last_index:
14             d_row.append(he_dec(row[j]) / row[last_index])
15             j += 1
16         d_rows.append(d_row)
17         return d_rows
18     j = 0
19     for row in rows:
20         d_row = []
21         d_rows.append(d_row)
22         i = 0
23         while i < len(operations):
24             dec_value = None
25             if operations[i] == "=":
26                 dec_value = det_dec(row[i])
27             elif operations[i] == ">":
28                 dec_value = ope_dec(row[i])
29             else:
30                 dec_value = row[i]
31             d_row.append(dec_value)
32             i += 1
```

```
33          j += 1
34      return d_rows
```

Листинг 8. Функция расшифрования ответа

Функция принимает на вход *SelectRequest*, на основе которого был сформирован запрос к СУБД. По нему формируется список полей, которые входят в ответ. Также определяются схемы шифрования, которыми были зашифрованы столбцы.

В случае нахождения среднего значения в столбце, соответствующие суммые расшифровываются и делятся на количество записей. Таким образом, в результате выполнения этой функции в таблице будут средние значения вместо сумм и общего количества.

4.3 Функции на стороне СУБД

Как уже было сказано выше, в качестве СУБД была выбрана PostgreSQL. Для корректной работы библиотеки на стороне СУБД доопределяется пара функций, необходимых для выполнения сложения над шифртекстами.

Первой функцией является функция агрегатор, которая будет возвращать сумму всего столбца

```
1 CREATE AGGREGATE paillier_sum (DECIMAL, DECIMAL) (
2     initcond = '1',
3     stype = DECIMAL,
4     sfunc = state_sum
5 );
```

Листинг 9. Функция нахождения среднего значения

1. *initcond* является начальным состоянием. Состояние сохраняется и передается в функцию состояния.

2. *stype* является типом состояния.

3. *sfunc* является функцией состояния. Эта функция вызывается на каждую новую запись в базе в процессе вычисления значения агрегатора. Она использует текущее состояние и очередную запись из базы данных для вычисления следующего состояния.

Второй функцией является функция состояния:

```
1 CREATE OR REPLACE FUNCTION state_sum (state DECIMAL, pub
    DECIMAL, elem DECIMAL)
2 RETURNS DECIMAL
3 LANGUAGE SQL
4 AS $$
```

```
5     SELECT (state * elem % (pub * pub))
6 $$;
```

Листинг 10. Функция суммирования шифртекстов

Она выполняет операцию сложения над шифртекстами, полученными в результате схемы Пайе. Напомним, что в схеме Пайе

$$Enc(m_1 + m_2, k) = Enc(m_1) * Enc(m_2) \ mod \ k^2.$$

Также от таблицы требуется, чтобы все шифруемые столбцы имели тип DECIMAL так как результат схем шифрования библиотека конвертирует в числа.

4.4 Прототип веб-приложения медицинского учреждения

В рамках данной работы было разработан прототип веб приложения, использующий библиотеку. Предметной областью для приложения стал процесс работы НИИ онкологии, выглядящий следующим образом:

1. Пациент приходит лечиться в центр онкологии. Врач заполняет данные о пациенте в веб приложении.

2. Научные сотрудники получают из приложения данные о пациентах и планируют на их основе эксперименты.

Для хранения в базе данных была сформирована таблица пациентов, являющаяся сильно урезанной версией реальной таблицы.

1. **id** идентификатор записи. Первичный ключ.

2. **name** имя пациента.

3. **age** возраст пациента.

4. **therapy_duration** общая продолжительность лечения.

5. **gen_before** количество гена X в анализе до начала лечения.

6. **gen_after** количество гена X в анализе после лечения.

7. **is_effective** было ли личение эффективно.

Под геном X подразумевается некоторая количественная величина, измеряющаяся до и после лечения.

В рамках прототипа был реализован функционал вставки записи в базу данных, а также функционал поиска записей, удовлетворяющих следующим условиям:

1. Имя совпадает с искомым.

2. Возраст попадает в искомый интервал.

3. Продолжительность лечения попадает в искомый интервал.

4. Лечение было эффективно (не эффективно).

5. Любая комбинация.

Также был реализован функционал поиска суммы или среднего значения гена X до или после лечения среди записей, удовлетворяющих этим условиям.

Таким образом, в конечной таблице столбец id не был зашифрован, столбцы $name$ и $is_effective$ были зашифрованы детерминированным шифром, age и $therapy_duration$ шифровались шифром, сохраняющим порядок, а gen_before и gen_after схемой шифрования, сохраняющей сложение.

4.5 Исследование производительности библиотеки

В рамках работы был произведен тест производительности библиотеки для различных запросов к СУБД. Для тестирования использовалась база данных из прототипа веб приложения, стукрура которой описана в предыдущем разделе и ее аналог для незашифрованных данных.

Алгоритм тестирования производительности вставки данных:

1. Генерируются случайные данные для записи в открытой таблице.

2. Запись вставляется в открытую таблицу, замеряется время вставки.

3. Данные шифруются и составляется запись для зашифрованной таблицы.

4. Запись вставляется в зашифрованную таблицу, замеряется время вставки и шифрования.

В процессе тестирования было выяснено, что нет видимого различия между временем вставки шифрованных данных от временем вставки открытых данных, которое составляло в среднем **0.01** секунд на запись. Разница заключалась лишь во времени шифрования, которое было решено протестировать отдельно.

На рисунке 9 можно видеть, что время шифрования записи составляет примерно **0.1** секунду на запись. Таким образом, запись в зашифрованную СУБД занимает **0.11** секунды.

Процесс тестирования аналитических запросов состоял из следующих шагов:

1. Выбирается тестируемый функционал (выбор по точному совпадению, выбор интервала, поиск среднего значения).

2. Генерируются данные, необходимые для составления запроса к СУБД.

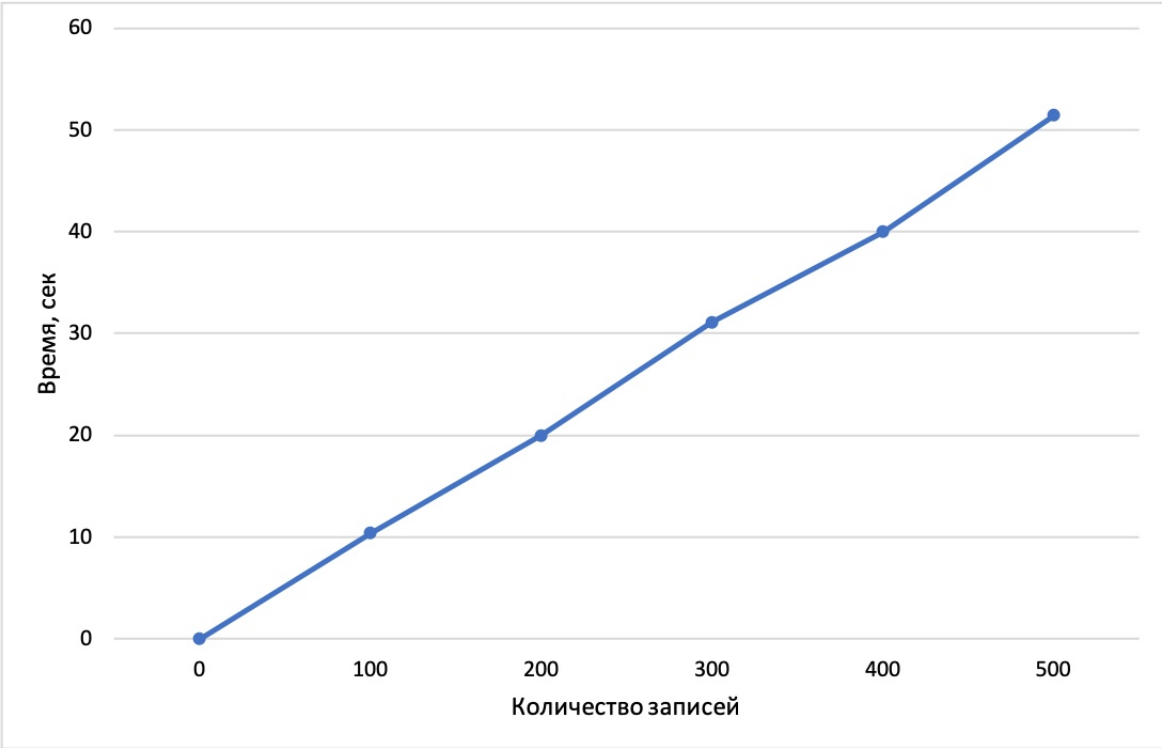

Рисунок 9 — Время шифрования записей

3. Замеряется время выполнения запроса.

4. Замеряется время составления и выполнения запроса к защищенной базе данных.

В процессе тестирования был сделан вывод о том, что количество возвращаемых записей существенно влияет на время, которое проходит между формированием SQL запроса и получения расшифрованного реультата веб приложением. Поэтому время расшифрования записей было решено протестировать отдельно.

На рисунке 10 можно видеть, что время расшифрования одной записи составляет **0.01** секунду на запись. Соит помнить, что результат выполнения аналитического запроса может содержать множество записей, поэтому необходимо его составлять таким образом, чтобы требовалось минимальное количество расшифрований.

Тестирование аналитических запросов, содержащих только предикат равенства, показало хорошие результаты, что можно увидеть на рисунке 11. Время выполнения запроса практически не отличается, а разница обуславливается необходимостью предварительно зашифровать искомое значение детерминированным шифром.

Результаты тестирования производительности запросов, содержащих отношения порядка оказались несколько хуже. В первую очередь это связано с тем, что размер шифр-текстов значительно увеличивается. Результаты тестирования отражены на рисунке 12.

На рисунке 13 отражен результат тестирования скорости выполнения запросов на поиск среднего среди всех значений определенного столбца. Следует обратить внимание

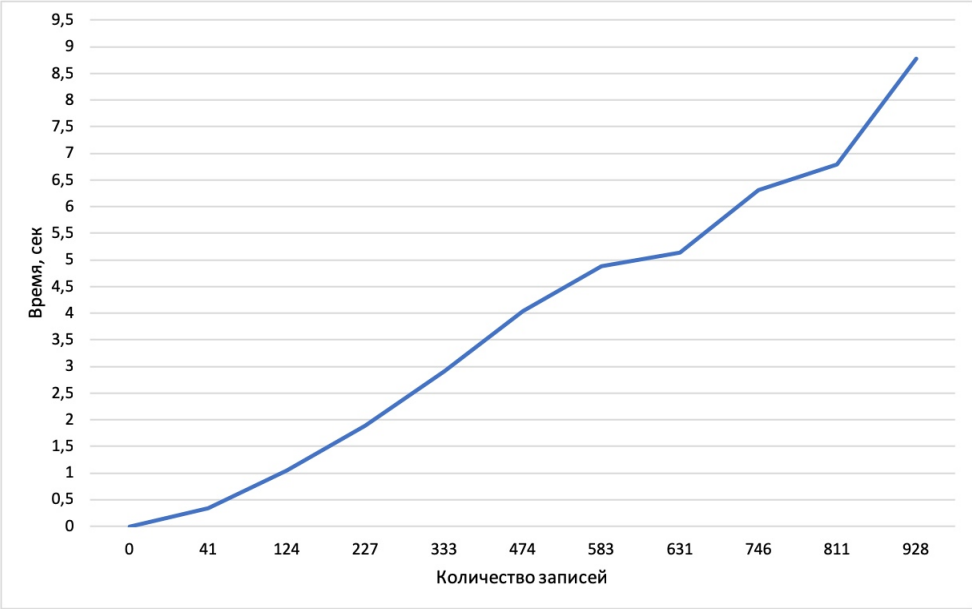

Рисунок 10 — Время расшифрования записей

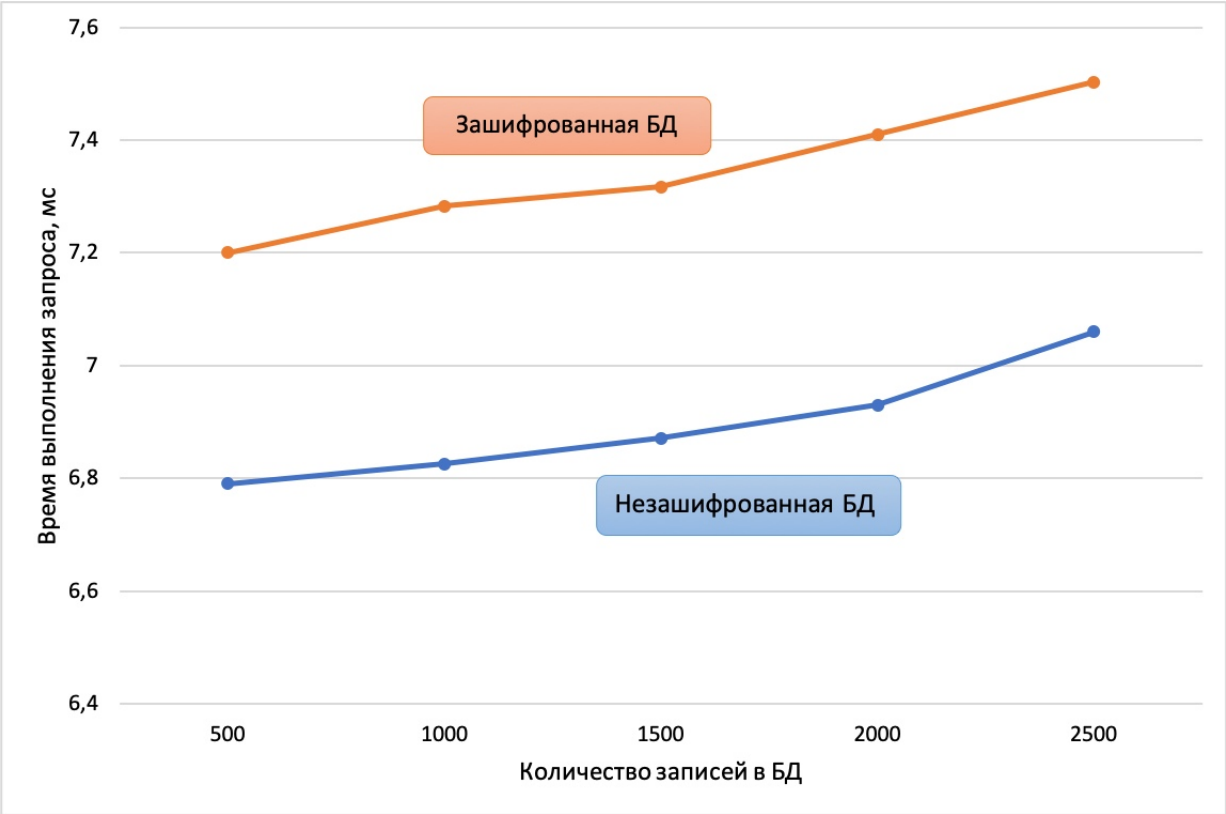

Рисунок 11 — Тестирование запросов, содержащих предикаты равенства

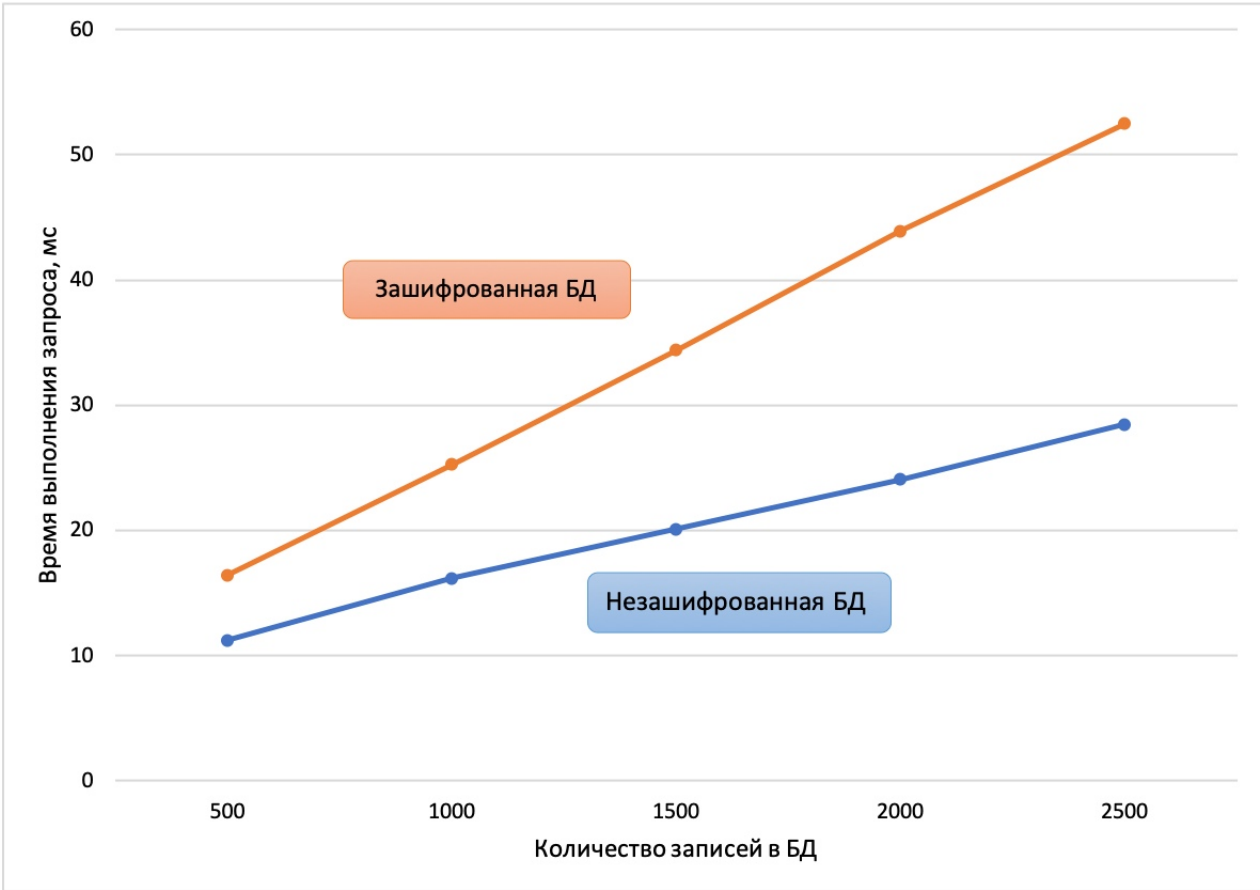

Рисунок 12 — Тестирование запросов, содержащих предикаты неравенства

на то, что данный запрос не требует дополнительной расшифровки полученных данных. Хотя скорость выполнения этого запроса и оказалась худшей, она все равно приемлема для веб приложений.

Финальным этапом тестирования стала проверка библиотеки в условиях, приближенных к реальности. Была сгенерирована база данных из 10000 записей, что удовлетворяет требованиям медицинского учреждения. Было измерено время выполнения получения расшифрованных данных от СУБД с использованием библиотеки:

1. Поиск пациента по совпадению имени — **0.0192** с.

2. Поиск пациентов от 25 до 30 лет — **0.242** с (было получено 669 записей).

3. Поиск среднего значения гена X в анализе крови до начала лечения среди всех пациентов — **0.122** с.

Все запросы отрабатывают за приемлимое время, позволяющее пользователям комфортно работать с веб приложением. Дополнительным выводом стало то, что хотя поиск среднего значения и проигрывает запросам, содержащим предикаты неравенста, но на практике они отрабатывают быстрее из-за необходимости расшифровывать тысячи записей. В реальных приложениях для сокращения выдачи используют механизм пагинации,

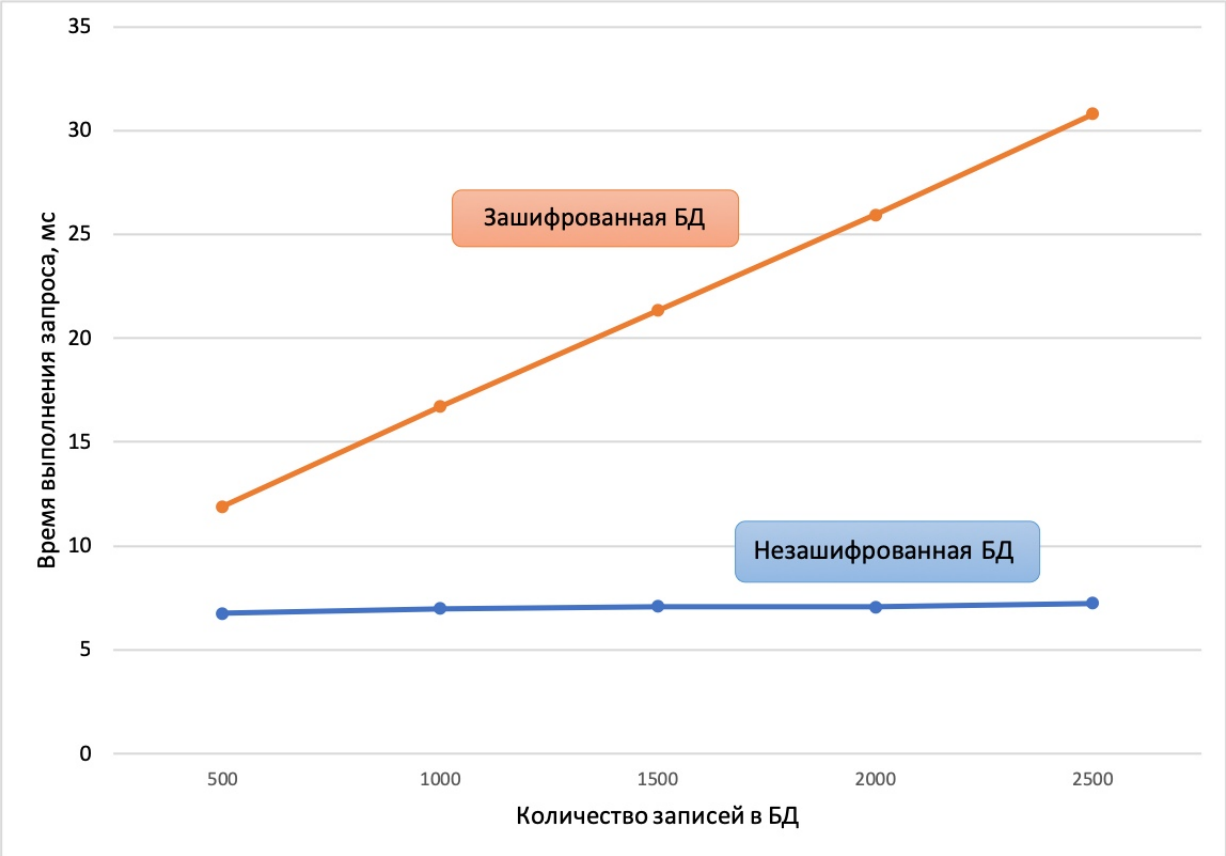

Рисунок 13 — Тестирование запросов на поиск среднего значения

когда данные пользователю выдаются порционно, например, по 30 записей. Это позволяет существенно сократить время получения расшифрованных данных от библиотеки.

ЗАКЛЮЧЕНИЕ

В процессе выполнения дипломной работы получены следующие результаты:

1. Разработана схема поддержки SQL-запросов над зашифрованной базой данных на основе частично гомоморфного шифрования и шифрования, сохраняющего порядок, когда часть операций над данными реализуется в доверенном окружении.

2. Реализована библиотека на языке Python для серверной части веб-приложения, которая обеспечивает обработку SQL-запросов над зашифрованной БД без расшифрования конфиденциальных данных на недоверенной стороне.

3. Разработан прототип защищенного веб-приложения взаимодействующего с облачной БД, предоставляющего доступ к конфиденциальным данным медицинского учреждения.

4. Проведено сравнительное тестирование производительности предложенного решения, которое показало, что по сравнению с нешифрованным вариантом библиотека дает некритичное замедление обработки SQL-запросов, позволяя пользователю работать с веб-приложением в режиме реального времени.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ И ЛИТЕРАТУРЫ

1. Полностью гомоморфное шифрование (обзор) / Л. К. Бабенко [и др.] // Вопросы защиты информации. — 2015. — № 3. — С. 3— 26.

2. Варновский Н. П., Шокуров А. В. Гомоморфное шифрование // Труды институтасистемного программирования РАН. — 2007. — No 12. — С. 27— 36.

3. Жиров А. О., Жирова О. В., Кренделев С. Ф. Безопасные облачные вычисления спомощью гомоморфной криптографии // Безопасность информационных техноло-гий. — 2013. — No 1. — С. 6— 12.

4. Popa R. A., Li F. H., Zeldovich N. An Ideal-Security Protocol for Order-Preserving Encoding // 2013 IEEE Symposium on Security and Privacy. — 2013. — С. 463— 477.

5. Order-Preserving Symmetric Encryption / A. Boldyreva [и др.] // Proc. 28th Internat. Conference. on Advances in Cryptology (EUROCRYPT'09). Cologne. — 2009. — С. 224— 241.

6. Gentry C. Fully Homomorphic Encryption Using Ideal Lattices // Proceedings of the forty-first annual ACM symposium on Theory of computing. — 2009. — С. 169— 178.

7. Gentry C., Halevi S., Smart N. P. Homomorphic evaluation of the AES circuit // Advances in Cryptology. CRYPTO 2012. Springer Berlin Heidelberg. — 2012. — С. 850— 867.

8. Popa R. A., Redfield C. M. S., Zeldovich N. CryptDB: Protecting Confidentiality with Encrypted Query Processing // Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles (SOSP'11). — 2011. — С. 85— 100.

9. Processing Analytical Queries over Encrypted Data. — URL: http://db.csail.mit.edu/pubs/tu-monomi-cr-vldb13.pdf — Дата доступа: 19 января 2020. Электрон. дан.

10. Глотов Н. И., Овсянников С. В., Тренькаев В. Н. Защищённая СУБД с сохранением порядка // Прикладная дискретная математика. — 2014. — No 7. — С. 81— 82.

11. RFC 5297 — Synthetic Initialization Vector (SIV) Authenticated Encryption Using the Advanced Encryption Standard (AES). — URL: https://tools.ietf.org/html/rfc5297 — Дата доступа: 19 января 2020. Электрон. дан.

12. Paillier P. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes // Advances in cryptology. EUROCRYPT'99. — 1999. — С. 223— 238.

АНТИПЛАГИАТ
ТВОРИТЕ СОБСТВЕННЫМ УМОМ

Отчет о проверке на заимствования №1

Автор: Салосин Иван salonkasoli@gmail.com / ID: 5776044
Проверяющий: Салосин Иван (salonkasoli@gmail.com / ID: 5776044)

Отчет предоставлен сервисом «Антиплагиат»- http://users.antiplagiat.ru

ИНФОРМАЦИЯ О ДОКУМЕНТЕ

№ документа: 1
Начало загрузки: 26.01.2020 14:18:17
Длительность загрузки: 00:00:01
Имя исходного файла: _____ (8).pdf
Название документа: _____ (8)
Размер текста: 1297 кБ
Символов в тексте: 50298
Слов в тексте: 6380
Число предложений: 465

ИНФОРМАЦИЯ ОБ ОТЧЕТЕ

Последний готовый отчет (ред.)
Начало проверки: 26.01.2020 14:18:18
Длительность проверки: 00:00:06
Комментарии: не указано
Модули поиска: Модуль поиска Интернет

ЗАИМСТВОВАНИЯ	САМОЦИТИРОВАНИЯ	ЦИТИРОВАНИЯ	ОРИГИНАЛЬНОСТЬ
4,08%	0%	0%	95,92%

Заимствования — доля всех найденных текстовых пересечений, за исключением тех, которые система отнесла к цитированиям, по отношению к общему объему документа.
Самоцитирования — доля фрагментов текста проверяемого документа, совпадающий или почти совпадающий с фрагментом текста источника, автором или соавтором которого является автор проверяемого документа, по отношению к общему объему документа.
Цитирования — доля текстовых пересечений, которые не являются авторскими, но система посчитала их использование корректным, по отношению к общему объему документа. Сюда относятся оформленные по ГОСТу цитаты, общеупотребительные выражения, фрагменты текста, найденные в источниках из коллекций нормативно-правовой документации.
Текстовое пересечение — фрагмент текста проверяемого документа, совпадающий или почти совпадающий с фрагментом текста источника.
Источник — документ, проиндексированный в системе и содержащийся в модуле поиска, по которому проводится проверка.
Оригинальность — доля фрагментов текста проверяемого документа, не обнаруженных ни в одном источнике, по которым шла проверка, по отношению к общему объему документа.
Заимствования, самоцитирования, цитирования и оригинальность являются отдельными показателями и в сумме дают 100%, что соответствует всему тексту проверяемого документа.
Обращаем Ваше внимание, что система находит текстовые пересечения проверяемого документа с проиндексированными в системе текстовыми источниками. При этом система является вспомогательным инструментом, определение корректности и правомерности заимствований или цитирований, а также авторства текстовых фрагментов проверяемого документа остается в компетенции проверяющего.

№	Доля в отчете	Доля в тексте	Источник	Ссылка	Актуален на	Модуль поиска	Блоков в отчете	Блоков в тексте
[01]	0,96%	1,14%	ДИПЛОМНАЯ РАБОТА ОБОБ...	https://doc.player.ru	08 Апр 2019	Модуль поиска Интернет	6	8
[02]	0,53%	0,86%	[IEEE 2014 International Conf...	https://doi.org	15 Окт 2019	Модуль поиска Интернет	4	5
[03]	0,23%	0,79%	Дерандомизационная крип...	http://yandex.ru	17 Окт 2019	Модуль поиска Интернет	1	5

Еще источников: 13
Еще заимствований: 2,34%

С результатами ознакомлен.

Руководитель ВКР В.Н. Третьяков