Содержание

ВВЕДЕНИЕ

Использование облачных баз данных вместо локальных хранилищ может давать организациям и частным лицам ряд преимуществ. Перенося данные на сервера облачных сервис провайдеров можно обеспечить высокую степень доступности и надёжности базы данных. Делегируя проблемы администрирования и обслуживания сторонней компании, можно снизить собственные расходы на управление данными. Широкое распространение облачных баз данных в последние годы (Windows SQL Azure, Amazon RDS и пр.), показывает высокую степень эффективности принципов облачных баз данных.

Существуют разные виды информации, которые могут храниться в облачной базе данных. Это могут быть как общедоступные или любые другие не конфиденциальные данные, так и персональные или любые другие конфиденциальные данные. Не конфиденциальные данные можно хранить в облачных базах данных, не беспокоясь о том, что они будут скомпрометированны. Однако существует множество угроз конфиденциальным данным в таких базах данных (БД): база данных может быть атакована инсайдером из числа администраторов базы данных([1]); она может быть подвергнута атаке ввиду обнаружения новой уязвимости в программном обеспечении используемом на сервере облачного хранилища; существует ещё множество угроз приватным данным в облачной БД ([2], [3]). Для того чтобы бороться с этими угрозами, и таким образом делать облачные технологии более привлекательными для хранения конфиденциальных данных, важно разработать систему, позволяющую защищать разные виды данных, предоставляя высокую степень гибкости для удовлетворения различных потребностей владельцев данных.

Одним из способов обеспечения приватности данных в удалённых БД, является шифрование всех конфиденциальных данных, до их передачи на третье-сторонний сервер. Таким образом, оператор данных видит лишь зашифрованные данные, не

зная их истинного значения.В свою очередь, владелец данных, может при необходимости расшифровать данные, и оперировать ими. Данный подход широко используется и имеет множество различных реализаций ([4], [5], [6], [8]).

Шифрование данных влечёт за собой накладные расходы на используемые ресурсы. Накладные расходы по времени, включают в себя по крайней мере время, которое необходимо потратить на шифрование данных до их передачи в облачную БД, так же, будут затраты по времени, на расшифровку данных. Обычно, так же наблюдаются накладные расходы по дисковому пространству, занимаемому БД, т.к. стойкие, современные алгоритмы шифрования, после шифрования дынных, увеличивают их размер. Временные накладные расходы могут быть критичны для систем реального времени, и для систем, которые проводят частые операции над данными – шифрование может значительно увеличить время проведения операций в БД. В худшем случае для проведения операции в облачной зашифрованной БД, от владельца данных потребуется полная выгрузка зашифрованных данных, проведение необходимой операции над дешифрованными данными, шифрование результата, и загрузка результата обратно на сервер. Такой сценарий оперирования данными, противоречит основам облачных систем. Во избежание такого сценария оперирования данными, можно применять специальные алгоритмы шифрования.

Использование особых алгоритмов шифрования позволяет проводить операции над зашифрованными данными на сервере, без необходимости их дешифровки. Однако, ввиду того, что зашифрованные данные отличаются от первоначальных, операции над зашифрованными данными могут тоже отличаться. К примеру, если пользователь захочет выбрать данные из зашифрованного столбца в реляционной БД, находящиеся в определённом диапазоне, от него может потребоваться "перевернуть"диапазон, т.к. зашифрованные данные могут храниться в "перевёрнутом"порядке. Операции могут отличаться и значительнее. Некоторые алгоритмы шифрования, позволяющие проводить операции над зашифрованными данными, возвращают в результате изменения сообщения, вектор значений, которые ввиду особенностей алгоритмов проведения операций над зашифрованными данными, не могут быть объединены в одно значение. Таким образом, в базе данных, зашифрованное значение будет представлено данными из разных колонок, и для проведения операции над таким значением, придётся оперировать различными столбцами из БД.

Этими особыми алгоритмами шифрования являются так называемые алгоритмы

сохраняющие порядок, гомоморфные шифрования и полностью гомоморфные алгоритмы ([9]). Сохраняющие порядок алгоритмы, позволяют сравнивать данные друг с другом, сортировать их и проводить другие операции требующие упорядочивания данных. Это может быть полезно к примеру данных, по которым следует проводить поиск (большинство распространённых алгоритмов поиска на первой стадии упорядочивают или частично упорядочивают данные). Гомоморфне шифрования позволяют проводить над зашифрованными данными некоторые арифметические операции, а полностью гомоморфные алгоритмы позволяют проводить все арифметические операции над данными.

То, как проводятся операции над зашифрованными данными отличается от того, как проводятся операции над первичными данными, таким образом, для работы системы использующей описанные выше алгоритмы шифрования, нужно специальное клиентское приложение, которое изменяет то, как проводятся операции, а так же шифрует и дешифрует данные. Некоторые подобные системы уже были разработаны. К примеру в работе [5], предлагает архитектура основанная на "луковичной"структуре хранения зашифрованных данных. В работе [7], представлена модель, разделяющая данные на зашифрованные подмножества, и хранящая номера этих множеств в незашифрованном виде (что предоставляет потенциальную уязвимость их системы, как отмечено в работе [5]). Модель представленная в работе [8], использует полностью гомоморфный алгоритм шифрования, однако, как признаёт сам автор, данная схема слишком медленна для практического применения. Так же существуют и другие работы на заданную тему ([10], [11]. [12]).

Данная работа представляет верхне-уровневую архитектуру клиентского приложения-прототипа, которое позволяет проводить арифметические операции, и операции требующие сохранения порядка над зашифрованными данными в облачной БД, не компрометируя их значение. Архитектура основана на гомоморфных и сохраняющих порядок алгоритмах шифрования, придуманных специально для применения в этой системе. В ходе построения данной архитектуры были:

- проанализированы решения, используемые в схожих системах, выявлены их схожие черты, участки которые могут и не могут применяться в нашей системе;
- выявлены функциональные и не функциональные требования предьявляемые к системе;

- решены задачи выбора, языка разработки, среды разработки, целевой ОС, выбора библиотек для написания прототипа;
- на основании функциональных и не функциональных требований были выявлены основные действующие лица в системе и была построена модель прецедентов использования системы;
- базируясь на модели прецедентов, была построена пакетная архитектура системы;
- пакеты были разбиты на подсистемы, были решены вопросы взаимодействия подсистем (определены интерфейсы взаимодействия);
- были выявлены ключевые места в верхне уровневой архитектуре и построены классовые архитектуры для этих ключевых мест;
- был построен прототип системы, и протестирована его работа.

Работа выполнена при финансовой поддержке Минобрнауки РФ (договор № 02.G25.31.0054).

Приложение позволяет выбирать пользователям какие данные какими алгоритмами шифрования они хотят шифровать. В сравнении с подходом использованным в [5], предлагаемая модель не использует "луковичную"структуру хранения данных, не хранит как в работе [7], номера множеств разбиения в открытом виде, и вообще не проводит подобного разбиения на множества. Так же, ввиду того, что в представляемой архитектуре пользователю представляется самостоятельно выбирать какими свойствами должен обладать алгоритм шифрования (какие операции он хочет проводить над данными), достигается высокая гибкость системы, а разработанные алгоритмы шифрования уникальны, и применяются в подобных системах впервые.

Прототип работает не со всем множеством SQL запросов, т.к. основной целью его построения, является доказательство концептуальной применимости предлагаемой системы, замеры времени работы алгоритмов шифрования и проведения операций над зашифрованными БД, и прочих тестов. Для решения этих задач, всё множество SQL запросов не нужно. Готовый прототип отвечает предъявленным к нему требованиям. В дальнейшем, данный прототип может быть расширен до полноценной системы работы с зашифрованной облачной БД. Прототип работает с SQLite ([16])

базой данных, однако при дальнейшем развитии проекта может быть расширен до работы с другими SQL-подобными СУБД (Oracle, MariaDB, Postgres и т.п.).

1 Основные задачи системы

1.1 Требования к системе

1.1.1 Функциональные требования

Система клиентского приложения защищённой базы данных как сервиса в "облаке выполняет две главные функции: шифрование и дешифрование конфиденциальных данных; и преобразование операций над не зашифрованными данным и в операции над зашифрованными данными. Дополнительными функциями являются: хранение информации о зашифрованных данных и их структурах в БД, предоставление подключения к БД, управление подсистемой пользователей, создание и модификация зашифрованной схемы данных. Данных функций достаточно для решения задач поставленных перед прототипом — доказательства концептуальной применимости предлагаемой системы, замера времени работы алгоритмов шифрования и проведения операций над зашифрованными данными, и прочих тестов. На основе описанных функций, были определены следующие функциональные требования к системе:

- `Modify RW Queries`: Изменение операций чтения/записи в БД над не зашифрованными данными в операции над зашифрованными данными. Данный прецедент включает в себя так-же шифрование данных.
- `Modify Responses`: Модификация ответов БД, полученных в зашифрованном виде, и с изменённой схемой данных. Включает в себя дешифрование данных.
- `Communicate with DB`: Всё что связано с взаимодействием с БД. Получение от неё ответов, отправка в неё SQL запросов.
- `Modify Schema`: Изменение запросов связанных с изменением схемы данных в

БД, таких как создание новых таблиц, добавление/удаление столбцов в существующих таблицах и т.п.

- `Manage Users`: Создание пользователей, удаление, разделение их привилегий, переходы между пользователями, аутентификация и авторизация и т.п.
- `Handle Meta-data`: Хранение информации о зашифрованных данных в БД — имена новых колонок, ключи которыми шифровались данные, алгоритмы шифрования которые были использованы на тех или иных данных, свойства этих алгоритмов шифрования, версия и тип СУБД с которой осуществляется работа и т.п.

Данный список описывает обязательства прототипа на верхнем уровне абстракции.

1.1.2 Не функциональные требования

Так же существуют не функциональные требования. Ввиду того, что описывается прототип системы, не функциональных требования всего два:

- Наличие простого способа расширения списка алгоритмов шифрования, используемых в системе.
- Возможность развёртывания с уже существующей БД.

В соответствии с описанными выше требованиями можно выделить три ключевых действующих лица в системе:

- `User`: авторизованный пользователь, производящий операции не изменяющие схему данных в БД.
- `Root`: авторизованный пользователь, который может менять схему данных в БД и имеет прямой доступ к хранилищу информации о зашифрованных данных.
- `BD`: СУБД, позволяющая проводить SQL операции в БД и хранить там данные.

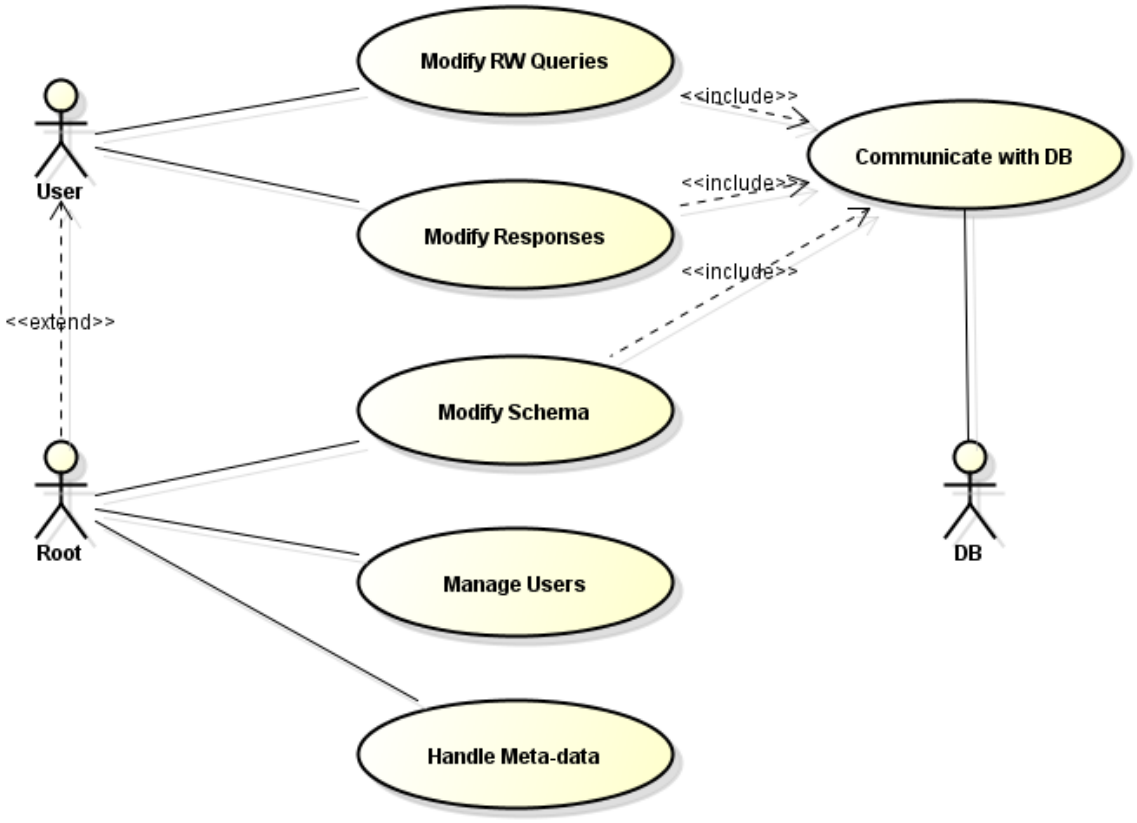

Рис. 1.1: Общая диаграмма прецедентов

1.2 Прецеденты использования системы

На основании анализа приведённых функциональных и нефункциональных требований, а так же, выявленных в соответствии с ними действующих лиц, можно построить диаграмму прецедентов ([18]) использования системы (см. рис. 1.1). Она демонстрирует функциональные возможности, доступные конкретным действующим лицам, кроме того, иллюстрирует организационные связи между прецедентами.

На данной диаграмме показано, что действующее лицо `Root` обладает всеми теми же функциональными возможностями, что и действующее лицо `User`. `Root` так же может модифицировать схему данных, создавать других пользователей системы, и имеет прямой доступ к мета -данным БД. Так же, из схемы следует, что пользователь взаимодействует с БД посредством прецедентов отправки запросов чтения записи в БД, изменения схемы и получения результата. Все эти прецеденты включают в себя модификацию проходящих через них данных, таким образом работа с алгоритмами шифрования, изменение запросов, хранение ключей и прочие сложные операции ин-

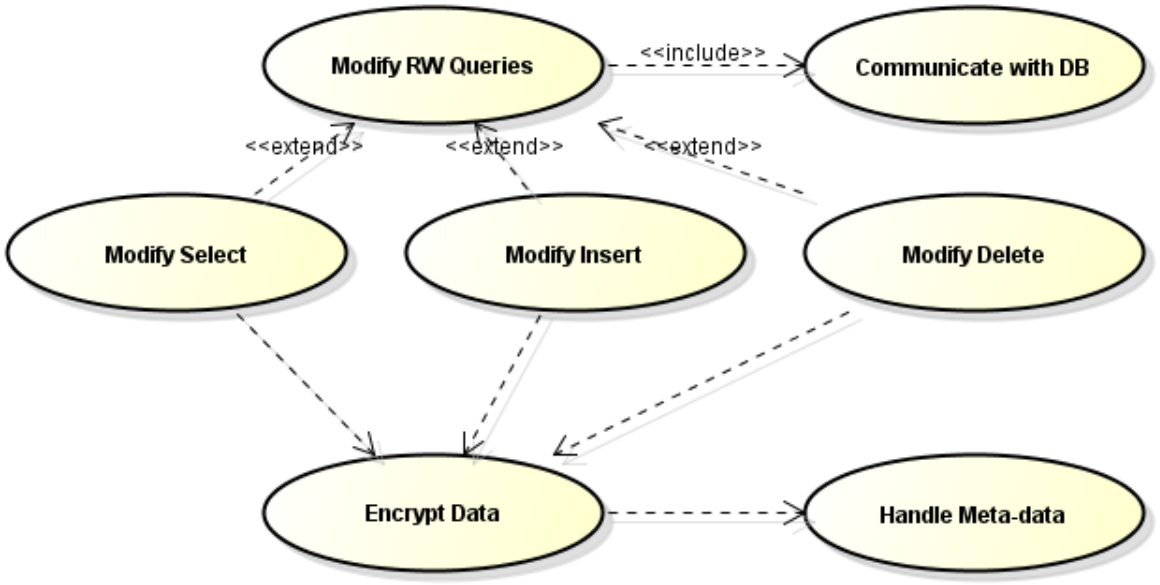

Рис. 1.2: RW диаграмма прецедентов

капсулируются от пользователя: он работает с зашифрованной БД, как с обычной БД.

1.2.1 `DML` прецеденты

Рассмотрим подробнее прецедент использования `Modify RW Queries`. Он позволяет пользователю выполнять `DML` SQL запросы типов `SELECT`, `INSERT`, `DELETE`, то есть запросы чтения записи данных в БД. Этот прецедент использования, исследует пользовательские запросы, выделяет из них данные, которые необходимо зашифровать (если такие имеются — пользователь может не шифровать какие -то данные, если обрабатываемый запрос работает только с этими данными, то шифровать ничего не надо). Процесс шифрования данных, включает в себя взаимодействие с хранилищем мета -данных т.к. работа с уже зашифрованными столбцами требует знания о том, каким ключом данные в них были зашифрованны, и каким образом были модифицированны имена столбцов. Данные зависимости иллюстрируются диаграммой прецедентов на рис. 1.2.

Следует отметить, что для проведения операций над данными в одном зашифрованном столбце, все данные в этом столбце должны быть зашифрованы одним алгоритмом шифрования с использованием одного и того же ключа. То -есть, к примеру,

для проведения операций требующих сохранения порядка на множестве зашифрованных данных и получения верного результата, данное множество должно быть зашифровано одним и тем же ключом и и одним и тем же алгоритмом шифрования. Такое ограничение накладывается на систему свойствами используемых алгоритмов шифрования. Подобное ограничение типично для такого рода систем ([5], [7], [8], [10], [11], [12]).

В дальнейшем исследовании, планируется провести анализ влияния разделения данных в одном столбце на подмножества шифруемые различными ключами, на производительность системы и на крипто-стойкость системы. Крипто-стойкость предположительно должна возрасти, т.к. если предположить крайний случай — все данные зашифрованы разными ключами, такая система гораздо меньше уязвима к разным видам атак, таким как к статистическому анализу данных, и другим. С другой стороны, производительность системы должна падать с увеличением количества подмножеств шифруемых разными ключами т.к. для проведения операции над всем множеством будет необходимо проводить действий, по крайней мере в количество подмножеств раз больше, чем в такой же системе без разбиения на подмножества. Возможно, разбиения на подмножества можно проводить основываясь на статистике обращения к разным блокам данных в столбце, что позволит снизить эффект падения производительности. Однако в данной работе подобный анализ не проводился.

1.2.2 `DDL` прецеденты

Подробного рассмотрения заслуживает прецедент использования `Modify Schema`, при изменении схемы данных — создании зашифрованных структур в БД, изменения алгоритма их шифрования, удаление зашифрованных структур, происходит множество операций. Сохраняются данные о ключах, генерируются новые ключи, новые названия для столбцов и таблиц, изменяются старые и т.п. Всё это иллюстрирует рис. 1.3. Здесь показаны основные SQL запросы по изменению схемы: `CREATE`, `DROP`, `UPDATE`. И их взаимодействие с изменением хранимой в приложении мета - информации. При Использовании пользователем SQL запроса типа `CREATE`, в котором должны быть зашифрованные данные, надо создать ключи для шифрования этих данных, выбрать подходящий алгоритм шифрования, и создать новые имена для колонок, которые будут хранить зашифрованные данные. Имена колонок изме-

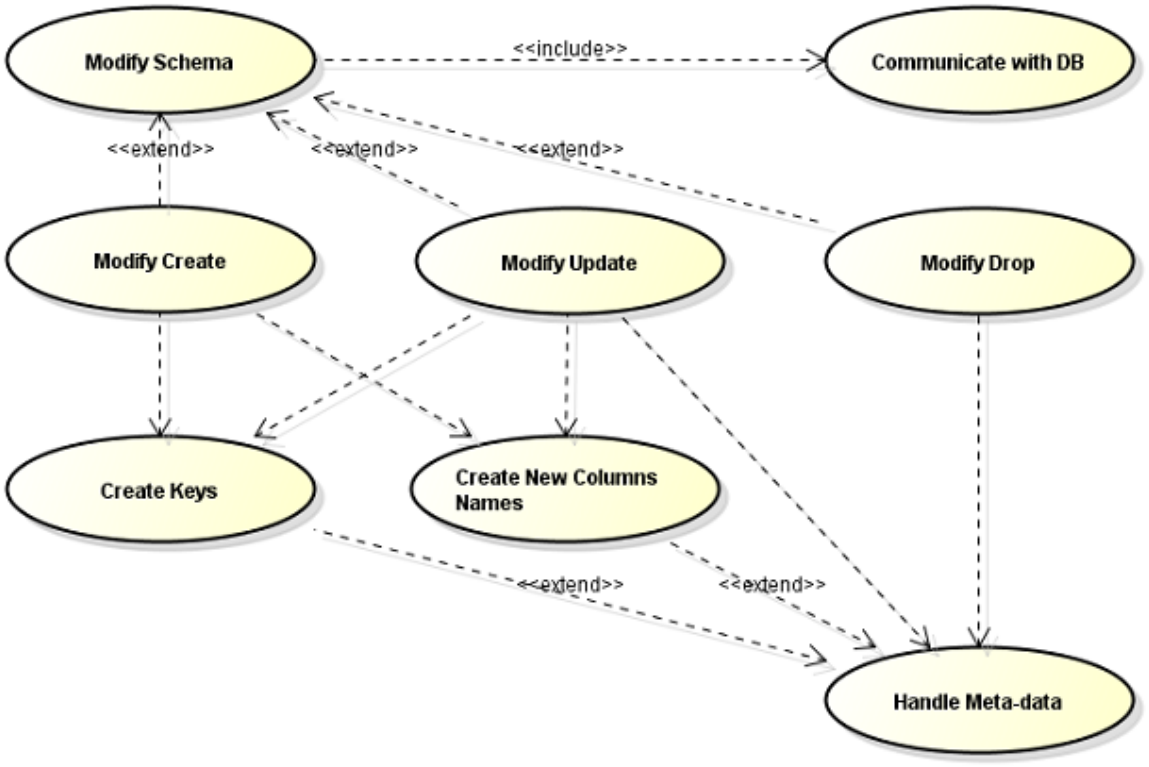

Рис. 1.3: Диаграмма прецедентов по изменению схемы

няются для того чтобы обеспечить конфиденциальность их названия, в описываемом прототипе они генерируются случайно, гарантируя что никакие две колонки не будут названы одинаково. SQL запросы `UPDATE`, требуют таких же изменений в хранимой мета -информации как и запросы типа `CREATE`. Запросы типа `DROP`, лишь удаляют уже существующую информацию.

2 Компоненты системы

2.1 Модульная архитектура

После анализа основных требований предъявляемых к системе, и описания прецедентов использования системы, была построена модульная архитектура. Каждый модуль, решает часть задач описанных в разделе 1.2. В данной архитектуре, для поставленных задач можно выделить шесть основных модулей, которые в свою очередь разделяются на другие подсистемы. Данными модулями являются: пользовательский интерфейс (`UI`), модуль изменяющий команды, посылаемые в БД и ответы из БД (`Syntax Modifier`), модуль шифрований данных (`Encryption System`), модуль подключения к БД (`DB Connection`), хранилище мета-данных (`Meta-data holder`), и подсистема пользователей (`(User System)`). Интерфейсы данных модулей и их взаимодействие друг с другом представлено на рисунке 2.1.

Рассмотрим работу данной модульной архитектуры на нескольких примерах, показывающих как с помощью данной архитектуры можно реализовать прецеденты использования описанные в разделе 1.2.

Рассмотрим для начала прецедент использования `Modify RW Queries`, данный прецедент описывает функцию системы по умению изменять SQL запросы чтения/записи, и шифровать данные в них. Данный прецедент использования тесно контактирует с прецедентом использования `Modify Responses`, поэтому `Modify Responses`, будет рассмотрен следующим.

Для того, чтобы произвести SQL операцию чтения записи, пользователь должен написать соответствующий SQL запрос в `UI` модуле. `UI` модуль определит что данный запрос является `RW` запросом, проверит что пользователь обладает достаточными правами для выполнения такого рода запросов, и отправит его в пакет `Syntax Modifier`, через соответствующий интерфейс `DB Commands`. Теперь обработ-

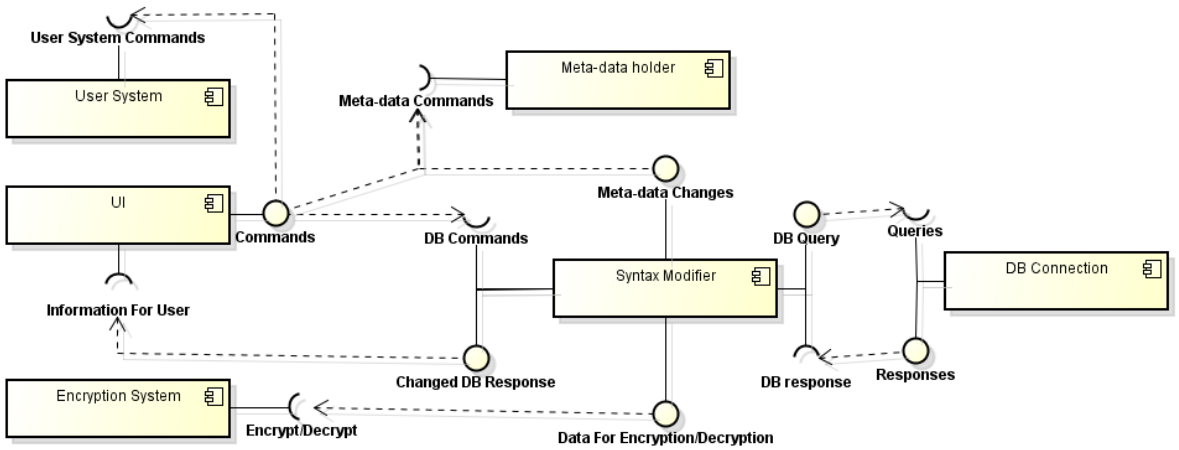

Рис. 2.1: Модульная архитектура системы

кой запроса, шифрованием данных и изменением операции займётся модуль `Syntax Modifier`. С помощью пакета `Meta-data holder`, `Syntax Modifier` определит, есть ли в SQL запросе обращение к зашифрованным данным. Если да, он приступит к соответствующему изменению запроса. К примеру в случае `SELECT` запроса, необходимо изменить запрос к не зашифрованной схеме данных в запрос к зашифрованной схеме данных, помимо этого, если в запросе производится сравнение с константами или какими либо полями, необходимо проверить существует ли возможность провести такую операцию и зашифровать все константы. Для шифрования используется модуль `Encryption System`. Проверка возможности проведения операции, является важным этапом в работе `Syntax Modifier`, дело в том, что к примеру, сравнивать данные зашифрованные разными алгоритмами шифрования, или даже одним алгоритмом шифрования но с разными ключами, не получится. Это было отмечено ранее на странице 12. После того, как запрос был модифицирован, он может быть отправлен в БД, для этого существует пакет `DB Connection`, который отправляет сообщения в БД, получает от неё ответы, и поддерживающей единый интерфейс работы с разными СУБД.

После отправки SQL запроса чтения записи, пользователь может ожидать ответ от БД. К примеру при отправке `SELECT` SQL запроса, пользователь ожидает получить ответ в виде выборки, соответствующей его запросу. При этом пользователь ожидает получить ответ в не зашифрованном виде, и так, будто бы он работает с не изменённой схемой БД.Таким требованиям соответствует прецедент использова-

ния Modify Responses. Данный функционал реализован схоже с Modify Queries, но протекает в обратном направлении. Когда пакет DB Connection получает от БД информацию, данная информация направляется в Syntax Modifier, который в свою очередь определяет необходимо ли что-то менять в ответе от БД, к примеру в случае если БД прислала ошибку, скорее всего ничего менять не надо. Для определения того, надо ли что-то менять, Syntax Modifier, пользуется информацией из Meta-data holder, изменяет то что необходимо изменить, к примеру, дешифрует и объединяет данные из разных столбцов, и отправляет всё это в UI. UI, соответственно отображает результат пользователю.

Прецедент использования Modify Schema, отличается от Modify RW Queries, тем, что его не может использовать действующее лицо User, а может только использовать Root, и в данном прецеденте создаются и/или удаляются новые записи в Meta-data holder. В данном прецеденте использования так же генерируются ключи, для шифрования при создании новых зашифрованных сущностей в БД. Это осуществляется подсистемой Encryption System, ответственной за создание ключей, аналогичная подсистема, но уже в рамках пакета Meta-data holder, создает новые имена зашифрованных колонок и следит за тем, чтобы они не повторялись.

Следующие два прецедента использования значительно отличаются от описанных выше. Они проходят без использования подключения к БД. Это прецеденты Manage Users, в результате которого создаются и удаляются пользователи и им присваиваются различные права, и Handle meta-data, в результате которого пользователь может напрямую оперировать данными связанными с шифрованием. Прецедент Handle Meta-data, должен быть доступен лишь доверенному лицу, хорошо понимающему, что именно оно делает, зачем и как это можно делать, поэтому данный прецедент доступен только пользователю Root. В дальнейшем планируется ввести отдельного пользователя, который будет способен выполнять операции с мета-информацией о зашифрованных данных в БД, а пользователю Root, оставить, только возможности обычного пользователя и модификации схемы данных.

Прецедент Manage Users, реализован пользовательской подсистемой посредством пакета User System. Он обеспечивает переключение между пользователями, авторизацию, аутентификацию, проверку прав на выполнение тех или иных операций в БД. Для авторизации используется логин, пароль и зашифрованный файл мета-информации. Из логина и пароля комбинируется ключ для расшифровки файла

мета-информации, предоставленного пользователем,если это удаётся, пользователь считается авторизованным. При этом, если пользователь предоставит неверный файл мета-информации, и такой логин и пароль, которым он сможет его расшифровать, он всё равно будет авторизован, однако он не сможет проводить никаких операций над зашифрованной БД, т.к. в не верном файле мета-информации будут храниться не верные ключи. в дальнейшем, есть идеи по вынесу системы аутентификации пользователя на доверенный прокси сервер, туда-же планируется вынести и подсистему создающую ключи для шифрований.

Прецедент `Handle Meta-data`, реализован внутри пакета `Meta-data holder`, он представляет собой SQLite базу данных, хранящую в себе всю необходимую информацию о шифрованиях, зашифрованных данных, пользователях, имена зашифрованных данных, свойствах шифрований, версии и типу СУБД, с которой осуществляется работа.

2.2 Описание пакетов

Далее приведено подробное описание пакетов, затронута их классовая архитектура и взаимодействие друг с другом, а так же, в некоторых случаях разобраны вопросы их дальнейшего развития и разработки.

2.2.1 Пакет `UI`

Данный пакет предоставляет графический интерфейс для работы с описываемой системой, базовая классовая архитектура приведена на рисунке 2.2. На данном рисунке показаны основные классы, то как они организационно связан и их взаимодействие с другими пакетами в системе.

Ключевыми классами здесь являются `MessageDisplayer` и `CommandReader`. Они умеют соответственно отображать на дисплее (`IDisplay`), сообщения (`IMessage`), и считывать с него команды (`ICommand`). Управление командами их перенаправление другим пакетам, частичное определение возможности их исполнения, выполняется классом `CommandHandler`. С помощью сообщений типа `IMessage`, происходит внутреннее общение в системе в целом.

Более подробно следует остановиться на интерфейсах приведённых на рисунке,

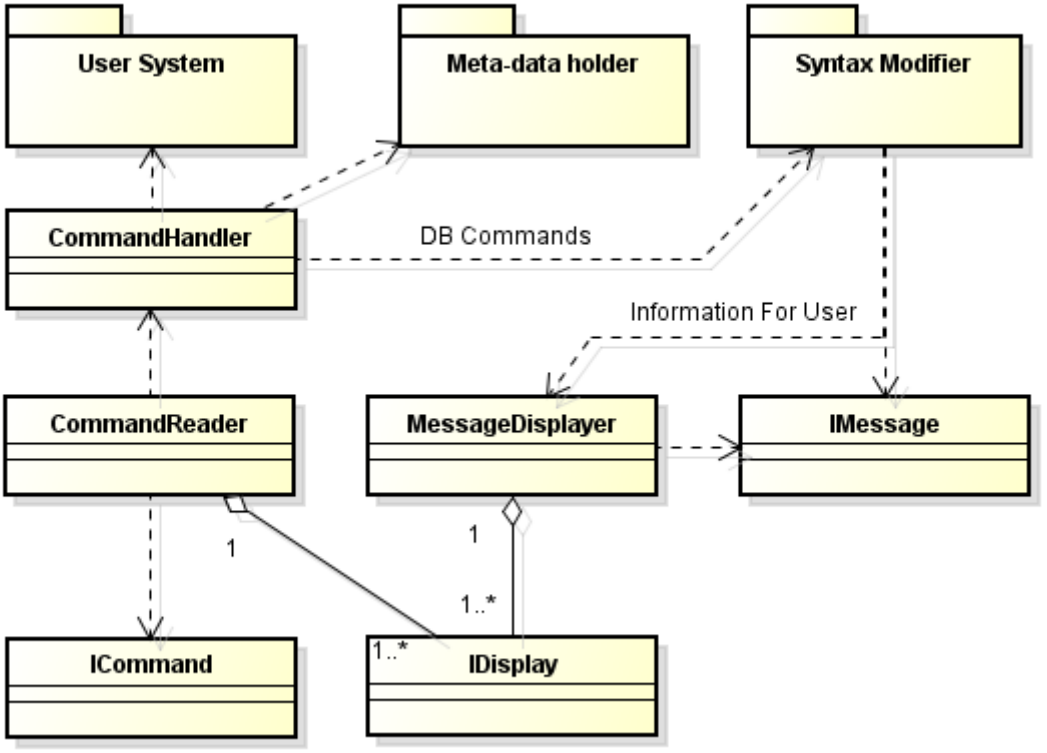

Рис. 2.2: Базовая классовая архитектура `UI`

2.2. Их иерархия наследования подробно продемонстрирована на рисунке 2.3. Как видно, в данный момент поддерживается только один вариант `IDisplay`, а именно `Terminal`, это обусловлено тем, что для задач прототипа достаточно терминала, однако в дальнейшем, прототип можно будет с лёгкостью расширить графическим интерфейсом, или каким либо другим интерфейсом работы. Ввиду того, что работа производится с базой данных, основным классом сообщений является `Table` — таблица. Так же, Сообщением может является строка — `Line` и ошибка, которая должна обрабатываться иначе чем строка, и может быть представлена не только строкой — `Error`. Сообщения, благодаря соответствия интерфейсу `IMessage`, корректно отображаются классом `MessageDisplayer`. Третьим представленным интерфейсом является `ICommand`, он объединяет сходства трёх классов. Как видно было из Диаграммы прецедентов (рис. 1.2, стр. 11), можно выделить команды, которые относятся к БД (соответствующие `RW Queries` и `Schema Queries`), запросы относящиеся к системе в целом (команды `Manage Users`, команда выхода из системы и т.п.), и команды относящиеся к мета-информации (прецедент `Handle Meta-data`).

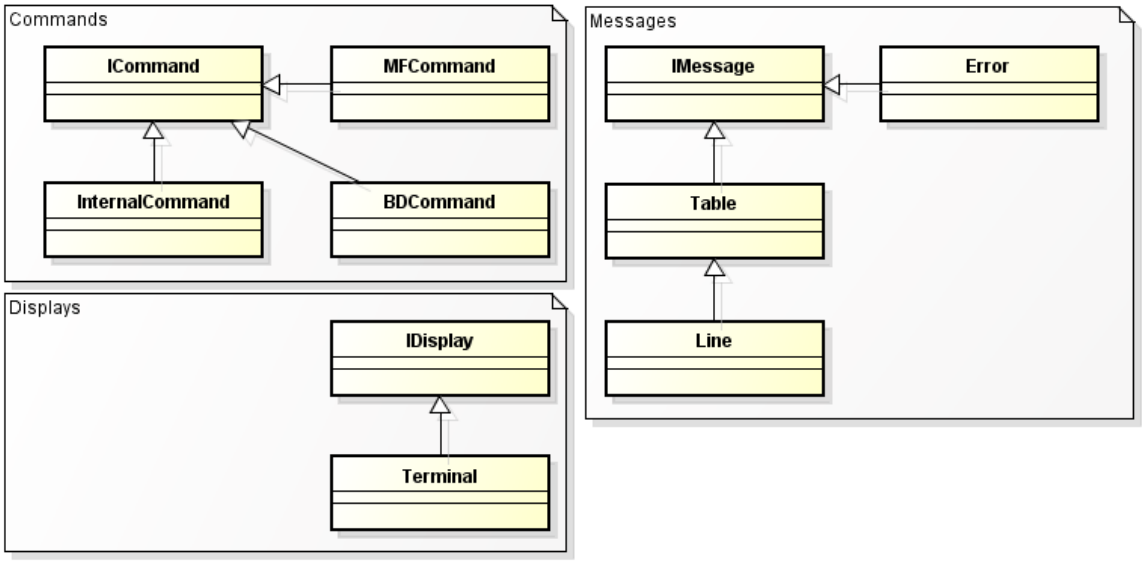

Рис. 2.3: Иерархия интерфейсов в `UI`

В целом, представленная классовая модель решает все поставленные перед ней на данный момент задачи, и её можно сравнительно легко расширить, если понадобиться вносить изменения. Вся внутренняя структура сообщений и команд инкапсулирована в соответствующих классах, работа с графическим интерфейсом делегирована отдельному классу, и существуют классы прослойки, между сообщениями и командами и интерфейсом — `CommandReader` и `MessageDisplayer`.

Данный пакет довольно много взаимодействует с пакетом `User System`, и в целом, пакет `User System`, на определённом уровне абстракции можно воспринимать как подсистему `UI`, поэтому следующим рассматриваемым пакетом будет `User System`.

2.2.2 Пакет `User System`

Данный пакет инкапсулирует в себе переключения между пользователями и авторизацию пользователя. Он содержит в себе сложную подсистему авторизации, включающую в себя, шифрование и дешифрование файла хранящего мета-информацию, валидацию корректности этих действий, систему резервного копирования данного файла, представленную отдельным процессом демоном, обеспечивающую сохранность данных при экстренном выключении системы. Однако на столько подробно данный пакет здесь не описывается. Подсистема авторизации будет представлена нами одним классом, который инкапсулирует в себе описанные выше задачи (хоть на

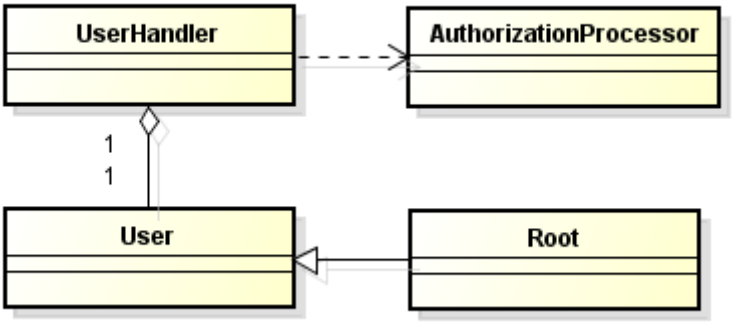

Рис. 2.4: Базовая классовая архитектура `User System`

самом деле, этот класс лишь внешний интерфейс подсистемы пакет `User System`). Помимо этого класса ключевыми классами данного пакета являются класс `User`, класс `Root` и класс `UserHandler`. Их взаимосвязь понятным образом получается из описанной в начале модели прецедентов, и выявленных действующих лиц. Их организационное взаимодействие проиллюстрировано рисунком 2.4.

2.2.3 Пакет `Meta-data holder`

Данный пакет представляет собой локальную SQLite базу данных и API к ней, в которой созданы соответствующие всем необходимым для хранения данным таблицы. API представляет широкий набор методов инкапсулирующих в себе обращения к этой БД. Таким образом замена SQLite базы данных на другой способ хранения мета-информации должен проходить сравнительно без проблемно. Самыми важными данными, хранимыми в этой БД, являются ключи к зашифрованным данным, хранимым в облачной БД, и соответствие имён зашифрованных структур данных в облачной БД, не зашифрованным структурам данных. Так-же данная локальная SQLite база данных, должна предоставляться пользователем при авторизации, в зашифрованном виде. В данный момент вся работа с ней происходит на диске, что замедляет работу и потенциально не безопасно, в рамках прототипа, такое решение допустимо, однако в дальнейшем решение следует изменить.

2.2.4 Пакет `Syntax Modifier`

Это очень обширный пакет, как видно из 2.1 на странице 16, данный пакет взаимодействует со всеми другими пакетами кроме `User System`. Подробное его описание можно найти в [14], а так же в соответствующей выпускной квалификационной работе бакалавра, этого же года. В данном разделе опишем лишь основной функционал данного пакета. Ввиду того, что данному пакету посвящена объёмная работа, его архитектура будет описана словесно.

Основной задачей данного пакета является изменение запросов к БД, и ответов от неё. Когда в этот модуль попадает информация через интерфейс `DB Commands`, описываемый модуль в первую очередь определяет вид SQL запроса, определив вид, т.е. то, к какому семейству относится данный запрос, после этого он направляется в соответствующий этому семейству анализатор.

Анализатор знает структуру SQL запроса той СУБД (облачной) с которой осуществляется работа, и определяет с какими данными данный запрос работает. Затем, анализатор проверяет в `Meta-data holder`, являются ли эти данные конфиденциальными. Все анализаторы наследуются от единого интерфейса, и используют соответствующие им подсистемы разбора SQL выражений, основанные на `Boost spirit` ([17]), грамматиках. Грамматики в свою очередь тоже имеют развитую классовую архитектуру.

После того, как анализатор определил есть ли в запросе зашифрованные данные, он может либо просто послать запрос в БД (если не надо изменять запрос, шифровать данные и т.п.), либо начать преобразовывать запрос. Преобразователи, тоже все наследуются от единого интерфейса, и используют в качестве основы грамматики `Boost Spirit`. В этом процессе, могут создаваться новые записи в `Meta-data holder`, могут удалятся там записи, создаваться новые ключи и т.п. Так же, проходит проверка выполнимости указанных запросов в БД.

При разборе результатов, `Syntax Modifier`, определяет есть ли зашифрованные данные в ответе, расшифровывает их, конструирует корректный для пользователя ответ от БД. обычно этот ответ представлен в виде таблицы, ошибки, или сообщения от БД.

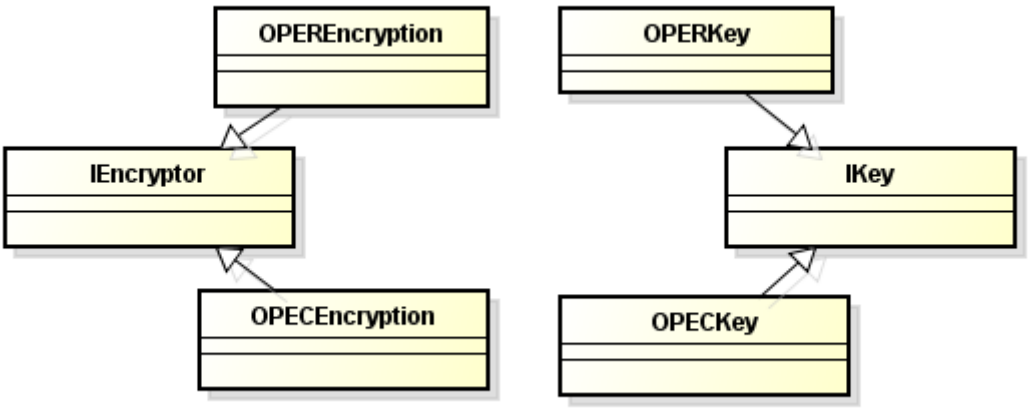

Рис. 2.5: Наследование алгоритмов шифрования

2.2.5 Пакет `Encryption System`

Данный пакет отвечает за шифрования данных, конкретные описания шифрований можно найти у примеру [15]. Однако, все шифрования используемые в системе наследуются от единого интерфейса, и все ключи тоже, таким образом, подключение нового шифрования осуществляется посредством создания обёртки внешнего шифрования в классы удовлетворяющие данным интерфейсам. Интерфейсы для шифрований, по сути содержат в себе методы для шифрования, дешифрования, получения информации о типе результирующих данных и информации о том, как создавать ключ. Примерно, как выглядит данное наследование на примере двух разных шифрований показано на рисунке 2.5, шифрований больше, но двух, достаточно чтобы показать наследование алгоритмов шифрования. Как видно, для того, чтобы можно было использовать новое шифрование, надо создать обёртку и для самого алгоритма и для ключа.

Так же данный модуль включает в себя генератор случайных чисел. Данный модуль включает в себя класс который инкапсулирует в себе SQL типы специфичные для используемых СУБД. Шифрование своим результатом может возвращать разные типы данных, и эти типы данных могут быть совсем по разному быть представлены в СУБД, поэтому для конструирования запроса пакетом `Syntax Modifier`, необходимо

2.2.6 Пакет `DB Connection`

Последний рассматриваемый пакет, это пакет обеспечивающий подключение к СУБД. Т.к. в рамках прототипа мы работаем с SQLite базой данных, этот пакет представлен набором методов, обеспечивающих работу с ней. В дальнейшем он должен быть заменён модулем поддерживающим безопасный, зашифрованный канал с СУБД, а так же инкапсулирующий в себе различия между СУБД. Подробная разработка данного модуля будет проводится в дальнейшей работе над проектом.

3 Возможности прототипа

Прототип позволяет проводить операции языка SQL семейств `SELECT`, `INSERT`, `DELETE`, `DROP`, `CREATE`. При этом если пользователь шифровал данные с использованием алгоритма сохраняющего порядок, над зашифрованными данными можно проводить эффективный поиск и сравнение данных, а так же, если при создании таблиц пользователь указал, что по заданным зашифрованным столбцам он хочет проводить `JOIN` операции `SELECT`, эти операции будут ему так же доступны. В рамках разработки данного прототипа была придумана следующая реализация `JOIN` операций — вводится понятие `JOIN` групп, внутри которых данные шифруются одним, общим алгоритмом шифрования, сохраняющего порядок, с одним и тем же ключом. При создании таблиц, пользователь указывает что над данными столбцами ему необходимо проводить `JOIN` операции в соответствующей `JOIN` группе. Так-же, при проведении `SELECT`, операций, пользователю доступны арифметические операции, такие как плюс и умножить, другие арифметические операции на текущем этапе развития прототипа не доступны, однако сейчас находятся в стадии разработки, тестирования и реализации (разные арифметические операции).

В дальнейшем планируется широкое развитие прототипа. Существует множество задач, которые ещё не решены в рамках прототипа, далее описаны, как мне кажется ключевые из них.

Задача кэширования данных является одной из наиболее важных потому, что её решение потенциально позволит значительно сократить время доступа пользователя к данным в БД. В результате шифрования, дешифрования и изменения запросов, время доступа к данным сокращается, что может негативно сказываться на пользовательском опыте работы с подобной БД, поэтому кэширование может заметно улучшить работу защищённой БД с точки зрения пользовательского опыта.

Так-же, одной из наиболее важных задач, стоящих в данный момент перед про-

ектом является расширение списка допустимых для работы СУБД и предоставление функционала для работы с ними в "облаках". В данный момент, прототипная модель работает с локальной БД, хранимой на пользовательской машине вместе с приложением, что не отвечает требованиям предьявляемым к системе в целом. Помимо этого, подзадачей данной задачи является организация и поддержка защищённого канала связи с облачной СУБД.

Другим важным направлением развития является введение мандатного управления доступом к данным в БД, т.е. развитие подсистемы пользователей, которая позволяла бы доверенным лицам работать лишь с теми данными, для которых у них есть ключи. Для развития такой системы, так же требуется введение горизонтального разделения таблиц в БД на зашифрованные разными ключами подмножества, что требует серьёзной разработки с точки зрения модуля `Syntax Modifier`.

4 Заключение

В рамках данной работы был проведён анализ существующих систем подобного типа. Были выявлены их общие черты, схожести и недостатки. Были исследованы применяемые в подобных системах алгоритмы шифрования, и алгоритмы шифрования, которые потенциально могли бы быть использованы в системе. Были выявлены функциональные и нефункциональные требования предьявляемые к системе. Исследованы угрозы которым подвергаются облачные базы данных. На основании функциональных и не функциональных требований были выяснены ограничения накладываемые на систему, основные действующие лица в системе и их взаимодействие. Были разработаны две концептуальные модели развёртывания системы — на клиентской машине и на доверенном прокси сервере. Основываясь на полученных результатах была построена прецедентная модель работы с системой, а так же выявлены допустимые ограничения на построение прототипа системы. Были поставлены цели и задачи которые должен решать прототип, разработана модель работы с ним. Основываясь на модели работы с прототипом, и выясненных ограничениях, была построена прецедентная модель работы с прототипом. Основываясь на ней была построена пакетная архитектура прототипа, состоящая из шести модулей. Данные модули были разбиты на подсистемы, решающие конкретные задачи. Для решения поставленных перед подсистемами задач были разработаны классовые архитектуры этих подсистем. Для реализации этих подсистем был проведён анализ инструментария с помощью которого их можно было бы создавать. Были изучены функциональные возможности различных библиотек, а так же выбран основной язык на котором пишется прототип. В соответствии с ограничениями на прототип была выбрана среда разработки, а так же система тестирования функциональности системы.

Основываясь на пакетной и классовой архитектуре, был разработан и создан прототип, который реализует основной функционал системы, показывает концеп-

туальную применимость описываемых подходов к защите данных в облачных БД, и позволяет проводить тестирование работы такого рода системы. Прототип позволяет шифровать конфиденциальные данные различными алгоритмами шифрования и проводить над ними операции в БД, без необходимости дешифрования данных в облаке. Пользователь сам решает какими свойствами должны обладать алгоритмы шифрования для тех или иных данных, и надо ли эти данные шифровать.

Тестирование работы прототипа показало, что он отвечает поставленным перед ним требованиям, а так же подтверждает концептуальную применимость описываемой схемы обеспечения конфиденциальности данных в облачной БД.

Литература

1. William R Claycomb, Alex Nicoll: Insider Threats to Cloud Computing: Directions for New Research Challenges / CERT 2012.

2. Cloud Security Alliance: Top Threats to Cloud Computing V1.0 // Cloud Security Alliance 2010.

3. Oracle White Paper: Security in Private Database Clouds 2012.

4. Sabrina De Capitani di Vimercati, Sara Foresti, Sushil Jajodia: Overencryption: Management of Access Control Evolution on Outsourced Data / 2007.

5. R.A.Popa, C.M.S.Redeld, N.Zeldovich, and H.Balakrishnan: CryptDB: Protecting Confidentiality with Encrypted Query. // Processing proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles, 2011, pp. 851-900.

6. Oracle White Paper: Oracle Advanced Security Transparent Data Encryption Best Practices 2012.

7. Hakan Hacgumus, Bala Iyer, Chen Li, Sharad Mehrotra: Executing SQL over Encrypted Data in the Database-Service-Provider Model. // In proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, Madison, WI, June 2002.

8. C.Gentry: Protecting Confidentiality with Encrypted Query Processing. // In Proceedings of the 41st Annual ACM Symposium on Theory of Computing, 2009, pp. 169-178.

9. Ronald L. Rivest Len Adleman Michael L. Dertouzos: On Data Banks And Privacy Homomorphism. // In Foundations of Secure Computation, pp. 169-180, 1978.

10. Sherman S.M. Chow Jie-Han Lee Lakshminarayanan Subramanian: Two-Party Computation Model for Privacy-Preserving Queries over Distributed Databases. / SSM Chow 2009.

11. Valentina Ciriani, Sabrina De Capitani di Vimercati, Sara Foresti, Sushil Jajodia, Stefano Paraboschi, and Pierangela Samarati: Keep a Few: Outsourcing Datawhile Maintaining Condentiality. / In Proceedings of the 14th European Symposium on Research in Computer Security, September 2009.

12. E. Damiani, S. D. C. di Vimercati, S. Jajodia, S. Paraboschi, and P. Samarati: Balancing confidentiality and efficiency in untrusted relational DBMSs. // In Proceedings of the 10th ACM Conference on Computer and Communications Security, Washington, DC, October 2003.

13. В. А. Бойко "Архитектура защищённой базы данных как сервиса в облаке"/ тезисы МНСК 2014, Новосибирск

14. К. А. Шатилов "Защищённое облачное хранилище данных"/ тезисы МНСК 2014, Новосибирск

15. В. В. Егорова "Схема адаптивного шифрования для эффективного хранения зашифрованных данных в защищенной базе данных"/ тезисы МНСК 2014, Новосибирск

16. SQLite, описание доступно на www.sqlite.org

17. Boost, описание доступно на www.boost.org

18. Гарди Буч, Объектно-ориентированный анализ и проектирование с примерами приложений / третье издание, Москва, 2008 С. 177-279