ОГЛАВЛЕНИЕ

[ВВЕДЕНИЕ 1](#_Toc408260226)

[ГЛАВА 1. ОБЗОР СУЩЕСТВУЮЩИХ ТЕХНОЛОГИЙ 4](#_Toc408260227)

[1.1. Облачные вычисления. Базы данных в облаке 4](#_Toc408260228)

[1.2. Введение в NoSQL системы управления базами данных 5](#_Toc408260229)

[1.2.1. СУБД MongoDB 9](#_Toc408260230)

[1.2.2. СУБД Redis 12](#_Toc408260231)

[1.2.3. СУБД HBase 14](#_Toc408260232)

[1.2.4. Вывод 16](#_Toc408260233)

[1.3. Обзор угроз конфиденциальности баз данных 17](#_Toc408260234)

[1.4. Обзор криптографических стратегий защиты баз данных 18](#_Toc408260235)

[1.4.1. Стратегия защиты уровня запоминающего устройства 19](#_Toc408260236)

[1.4.2. Стратегия защиты уровня базы данных 20](#_Toc408260237)

[1.4.3. Стратегия защиты уровня приложения 22](#_Toc408260238)

[1.4.4. Использование внешнего модуля защиты 24](#_Toc408260239)

[1.4.5. Вывод 25](#_Toc408260240)

[1.5. Обзор алгоритмов шифрования 26](#_Toc408260241)

[1.5.1. Гомоморфные шифрования. Схема полностью гомоморфного шифрования 26](#_Toc408260242)

[1.5.2. Шифрование, сохраняющее порядок 27](#_Toc408260243)

[1.5.3. Симметричные схемы шифрования 28](#_Toc408260244)

[1.5.4. Вывод 30](#_Toc408260245)

[1.6. Механизмы двухфакторной аутентификации [49] 31](#_Toc408260246)

[1.6.1. Использование карт EMV для двухфакторной аутентификации 32](#_Toc408260247)

[1.7. Обзор существующих готовых решений. CryptDB (SQL) 34](#_Toc408260248)

[1.8. Вывод 37](#_Toc408260249)

[ГЛАВА 2. ОПИСАНИЕ ВЫБРАННЫХ ТЕХНОЛОГИЙ 38](#_Toc408260250)

[2.1. Блочное шифрование AES [56] 38](#_Toc408260251)

[2.2. Шифрование, сохраняющее порядок 44](#_Toc408260252)

[2.2.1. Тривиальный алгоритм OPE [40] 44](#_Toc408260253)

[2.2.2. Алгоритм Symmetric OPE (OPES) [39] 46](#_Toc408260254)

[2.3. Гомоморфное шифрование 54](#_Toc408260255)

[2.3.1. Криптосистема Пэйе [57] 54](#_Toc408260256)

[2.4. Двухфакторная аутентификация EMV [58] 56](#_Toc408260257)

[2.4.1. Смарт-карты EMV [59] 56](#_Toc408260258)

[2.4.2. Протокол ISO/IEC 7816-3 [60] 57](#_Toc408260259)

[2.4.3. Структура смарт-карты EMV 58](#_Toc408260260)

[2.4.4. Алгоритмы аутентификации SDA и DDA [61] 59](#_Toc408260261)

[2.4.5. Упрощенный алгоритм аутентификации 61](#_Toc408260262)

[ГЛАВА 3. РЕАЛИЗАЦИЯ СИСТЕМЫ 63](#_Toc408260263)

[3.1. Проект 63](#_Toc408260264)

[3.2. Используемые технологии 63](#_Toc408260265)

[3.3. Архитектура решения 66](#_Toc408260266)

[3.4. Описание реализации 68](#_Toc408260267)

[3.1. Команды Redis и эмуляция их выполнения 70](#_Toc408260268)

[3.2. Реализация гомоморфного свойства в сервере Redis 74](#_Toc408260269)

[3.1. Реализация двухфакторной аутентификации EMV 75](#_Toc408260270)

[3.2. Производительность Cryptis 77](#_Toc408260271)

[3.3. Безопасность Cryptis 85](#_Toc408260272)

[3.4. Развитие проекта Cryptis 88](#_Toc408260273)

[ЗАКЛЮЧЕНИЕ 89](#_Toc408260274)

[СПИСОК ЛИТЕРАТУРЫ 90](#_Toc408260275)

ВВЕДЕНИЕ

За довольно непродолжительный период времени информационная сфера совершила, представлявшийся ранее невероятным, технологический рывок. Возросшая производительность персональной техники, возможность сохранять большие объемы данных, а также повсеместное подключение к высокоскоростной сети привели к распространению хранения важной информации в интернете.

Онлайн приложения уязвимы к краже ценной информации, поскольку злоумышленники могут использовать ошибки в программном обеспечении и настройке ЭВМ для получения доступа к приватным данным.

В то же время присутствует существенная экономическая выгода от размещения данных в интернете, например, облачных сервисах:

* на покупке лицензионного ПО;
* на затратах на оборудование;
* на внедрении и обслуживании;
* на времени персонала.

Однако при этом совершается большое количество нарушений режима конфиденциальности информации [1; 2]:

1. Около 43% баз данных содержат чувствительную информацию.
2. 46% опрошенных организаций потеряли важную часть данных на протяжении 2008 года.
3. На 69% возросло количество краж конфиденциальной информации за период с 2007 по 2008 год.
4. $6300000 составляет средний ущерб на каждую кражу ценной информации.

Спектр данных, которые требуют защиты, довольно широк: данные о сотрудниках и клиентах, номера банковских карт, информация о здоровье, финансовые документы, секреты производства.

Итак, размещать данные в облаке очень выгодно, и в то же время существует риск их безопасности. В связи с этим требуется выработка решений, сочетающая возможность использовать сторонние центры обработки данных и защищенность важных данных.

Одним из решений стала разработка проекта CryptDB – криптографически защищенной SQL базы данных. CryptDB использует многослойное шифрование и позволяет хранить важные данные без опасения за их раскрытие, при этом поддерживает широкое подмножество операторов SQL. Ценой этого является снижение производительности вплоть до 39% [3] на некоторых операциях, что существенно ограничивает практическое применение.

Выигрыш в производительности по сравнению с SQL БД могут дать NoSQL БД за счет отсутствия проверок целостности, полного кеширования в оперативной памяти, упрощенного набора команд, специальных структур. На настоящий момент криптографически защищенных NoSQL БД для использования в облаке не существует.

В данной работе рассматривается создание криптографически защищенной NoSQL базы данных, позволяющей работать совместно с CryptDB или незащищенной реляционной СУБД. Подобная гибридная система, размещенная в облачных сервисах, позволит повысить общую производительность базы данных и сократить затраты на инфраструктуру.

В работе также рассматриваются вопросы двухфакторной аутентификации пользователей с использованием смарт-карт формата EMV. Данный способ позволяет сократить затраты на стойкую аутентификацию, поскольку часть необходимого оборудования уже присутствует у сотрудников организаций.

Цель дипломной работы: Разработка прототипа криптографически защищенной NoSQL базы данных, способного работать в облаке.

Задачи дипломной работы:

1. Изучение существующих технологий и обзор готовых решений.
2. Изучение особенностей работы NoSQL БД и их размещения в облачных сервисах.
3. Выбор и обоснование архитектуры и родительской NoSQL базы данных для реализации.
4. Выбор и обоснование криптографических примитивов для шифрования.
5. Выбор и обоснование методов и средств реализации прототипа.
6. Реализация прототипа системы управления базами данных.
7. Тестирование и испытание системы, замер производительности и безопасности.

*Предмет исследования:*

Изучение облачных решений и проверка эффективности NoSQL подхода к криптографически защищенным базам данных.

ГЛАВА 1. ОБЗОР СУЩЕСТВУЮЩИХ ТЕХНОЛОГИЙ

### ****Облачные вычисления. Базы данных в облаке****

**Облачные вычисления (ОВ)** – комплексное решение, предоставляющее ИТ-ресурсы в виде сервиса [4]. Идея облачных вычислений состоит в том, что компании, использующие ИТ-услуги, вместо приобретения ЭВМ, закупки лицензий на программное обеспечение, использует сервис.

Преимущества облачных вычислений [4]:

1. **Уменьшение расходов**. ОВ снижают как капитальные затраты, так и текущие расходы, поскольку ресурсы приобретаются только по необходимости и оплачиваются по использованию.
2. **Оптимальное использование персонала**. Высвобождение ценных сотрудников, позволяя им больше концентрироваться на увеличении прибыли.
3. **Надежная масштабируемость**. ОВ обеспечивают мгновенное масштабирование без долгосрочных обязательств.

Облачные вычисления предоставляются, в основном, в трех моделях: инфраструктура как сервис (IaaS), платформа как сервис (PaaS), программное обеспечение как сервис (SaaS).

Инфраструктура как сервис – предоставляет эластичную инфраструктуру, основанную на виртуализации ЭВМ [5]. В любой момент для сложных вычислений в кластер могут быть добавлены ресурсы. IaaS заменяет парадигму «инфраструктура как актив».

Платформа как сервис – предоставляет инфраструктуру для разработки приложений, платформу [6]. PaaS потребляет инфраструктуру облака, заменяет парадигму «приобретение лицензий».

Программное обеспечение как сервис – предусматривает предоставление доступа по сети к коммерческому ПО [7]. SaaS предоставляет дешевый способ использования программного обеспечения – использование по требованию вместо покупки лицензии на каждый компьютер. Заменяет парадигму «ПО бизнеса и потребителя как актив».

**База данных** – представленная в объективной форме совокупность самостоятельных материалов (статей, расчётов, нормативных актов, судебных решений и иных подобных материалов), систематизированных таким образом, чтобы эти материалы могли быть найдены и обработаны с помощью электронной вычислительной машины (ЭВМ) [8].

Базы данных в облаке предоставляются по модели PaaS, в их преимущества входят:

1. Простота установки, развертывания, настройки.
2. Масштабируемость.
3. Безопасность.
4. Управляемость.
5. Высокий уровень доступности.
6. Производительность.

Распространенными SQL БД в облаке являются – Amazon Relational Database Service, IBM SmartCloud, Microsoft SQL Database, Oracle Datrabase cloud; NoSQL решения: Amazon DynamoDB, Cloudant, Microsoft Azure Table Storage, MongoHQ [9].

Однако БД в облаке не позволяет размещать ценные данные, вследствие того, что является недоверенной средой.

### Введение в NoSQL системы управления базами данных

**NoSQL** (Not only SQL) – ряд подходов, методов, проектов, направленных на реализацию моделей баз данных, имеющих существенные отличия от традиционных реляционных СУБД с поддержкой SQL [10].

**NoSQL –** следующее поколение баз данных, ориентированных на:

* Нереляционность (schema-free);
* Распределенность (distribution);
* Открытый исходный код (open source);
* Горизонтальную масштабируемость (horizontally scalability);
* Легкую репликацию (easy replication);
* Простые методы доступа (simple API);
* Уход от парадигмы ACID (not ACID – atomicity, consistency, isolation, durability);
* Большие данные (big data) [11].

Отказ от основных парадигм традиционных СУБД позволяет добиться существенного прироста скорости разработки системы и ее производительности. Основная цель подхода – расширить возможности СУБД там, где язык SQL недостаточно гибок, и не вытеснять его там, где он успешно справляется со своими задачами.

Исследователи отмечают весомые преимущества использования NoSQL баз данных [10; 12].

**Высокая скорость разработки приложений**. Много времени разработчики тратят на перенос данных между структурами данных, расположенных в оперативной памяти и реляционной СУБД. NoSQL может предоставлять модель данных, которая лучше подходит для нужд приложения. Таким образом, упрощается процесс конвертирования данных, что приводит к меньшему количеству кода для написания, отладки, внедрения.

**Производительность на больших объемах данных**. Организации хранят все больше данных и заинтересованы в их быстрой обработке. При этом реляционный подход зачастую не удовлетворяет эти нужды, поскольку проектировался для запуска на одной ЭВМ. В то же время NoSQL использует много низкопроизводительных и дешевых компьютеров, что позволяет уменьшить расходы на аппаратную часть. Множество NoSQL БД спроектированы специально для запуска в кластерах, и они реализуют более совершенные подходы для обработки больших массивов данных.

**Узкоспециализированность**. NoSQL реализует модели данных, близкие к предметной области. При проектировании СУБД весомую роль играют характер решаемых задач и особенности предметной области. Также возможно применение умеренной денормализации данных для повышения производительности.

В основном в NoSQL используются следующие модели данных [10]:

* Ключ-значение (Key-Value – Redis, BerkleyDB, Oracle NoSQL);
* Документная (Document – CouchDB, MongoDB);
* Колоночная (Column-Family – Amazon SimpleDB, Cassandra);
* Графовая (Graph – FlockDB, Infinite Graph).

Часто разработчики используют гибридную архитектуру хранения данных, совмещающую реляционные и NoSQL базы данных [13]. Martin Fowler и Pramod Sadalage вводят термин «Polyglot Persistence» [14] - использование различных технологий СУБД, выбранных на основе того, как данные будут использоваться в клиентских приложениях. Создатели ставят вопрос, зачем использовать реляционные СУБД для хранения бинарных изображений, в то время как существуют средства, специализированные на этом.



Рисунок 1 – Схема использования СУБД

Подход подразумевает использование NoSQL в случаях, когда он лучше реализует функции управления. Поэтому для хранения данных могут использоваться сразу несколько СУБД. Преимуществом гибридных решений является заметный выигрыш в производительности и масштабируемости. К недостаткам можно отнести проблемы выбора СУБД для данных, сложности организации взаимодействия многих СУБД, синхронизацию данных.

Использование гибридных решений в сфере защищенных баз данных увеличит производительность и масштабируемость систем. И если в качестве SQL СУБД возможно применение CryptDB, то реализаций криптографических NoSQL СУБД авторами не было найдено, что делает невозможным эксплуатацию гибридного подхода. Реализации криптографически защищенной NoSQL СУБД, способной быть размещенной в облаке посвящена данная работа.

Опрос агентства DZone выявил самые распространенные NoSQL СУБД по состоянию на 2012 год [15]: MongoDB, Redis, CouchDB, HBase, Cassandra, Riak (в порядке убывания популярности).

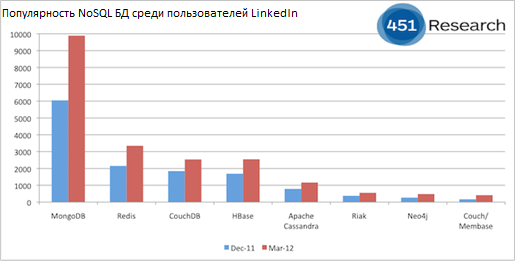


Рисунок 2 – Распространенность NoSQL СУБД

Каждая из перечисленных СУБД может работать в облаке по парадигме IaaS. Существуют также NoSQL СУБД, предоставляемые облачными хостингами (Amazon SimpleDB, Azure Table Storage), однако их исходный код закрыт, что затрудняет их использование в данной работе.

В настоящее время ведется разработка языка UnQL [16], приводящего API существующих NoSQL к единому знаменателю, но на момент составления работы реализаций UnQL не существовало, поэтому авторы не рассматривают использование этого языка.

Рассмотрим популярные СУБД в качестве базы для реализации криптографически защищенной. Выберем различные модели данных: документ-ориентированная (MongoDB), ключ-значение (Redis) и колоночная (HBase).

### СУБД MongoDB

**MongoDB** (от hu*mong*ous) – документ-ориентированная NoSQL база данных с открытым исходным кодом [17]. MongoDB – зрелый проект, версия 1.0.0 проверена в промышленной эксплуатации на протяжении более трех лет [18]. MongoDB способна работать в облаке по принципу IaaS.

Особенностями MongoDB являются:

* Документно-ориентированное хранилище. Используется JSON кодирование документа с динамической схемой полей;
* Полная поддержка индексации. Индексация по любому атрибуту;
* Развитые средства автоматической репликации;
* Автоматическая горизонтальная масштабируемость;
* Богатый язык запросов;
* Атомарные операции;
* Поддержка парадигмы Map/Reduce;
* Высокая надежность, с версии 1.8 появилось журналирование операций [19], их откат.

В MongoDB существенно упрощены модели документа и транзакций, что позволяет легко проводить горизонтальное масштабирование. СУБД, по мнению разработчиков, должна заполнить разрыв между простыми БД модели ключ-значение и реляционными СУБД. MongoDB может рассматриваться как прямая альтернатива традиционным базам данных, как центральный репозиторий данных [19].

Концептуально MongoDB представляет схожую с реляционной модель представления данных. Внутри СУБД может быть несколько баз данных, каждая из которых является контейнером для прочих сущностей.

База данных содержит коллекции – аналог традиционных таблиц. В каждой коллекции содержится несколько документов – аналог реляционных строк. Документ состоит из множества полей – подобных колонкам в реляционной модели.

Отличие MongoDB от реляционной СУБД заключается в том, что реляционные СУБД определяют колонки на уровне таблицы, а MongoDB определяет поля на уровне документа. Таким образом, документы в коллекции могут представлять собой различные по структуре сущности.

Документы кодируются в JSON-подобный вид, хранимый в двоичном формате BSON.

В отличие от традиционных БД, в MongoDB отсутствует оператор join, его функциональность реализуется через частичную денормализацию данных, выполнение нескольких запросов в клиентском ПО.

Поддержка транзакций сокращена - атомарность гарантируется только на уровне ограниченного множества операций, например, $inc, $set, $findAndModify. Клиентское приложение может эмулировать транзакции с помощью алгоритмов двухфазного коммита [19].

Отсутствует понятие изоляции. Любые данные, которые считываются одним клиентом, могут параллельно изменяться другим [20].

Взамен ограничений MongoDB предоставляет повышенную скорость разработки, исполнения запросов и развертывания. Синтаксис языка MongoDB достаточно богат.

Бесструктурность документа удобна и сводит к минимуму расхождения с объектно-ориентированной парадигмой программирования, отпадает необходимость задания схемы БД.

MongoDB способна выполнять асинхронную запись, т.е. не обязательно ожидать завершение операции записи для продолжения работы.

MongoDB разработана скорее как замена SQL СУБД, чем БД для гибридного применения в связке с реляционными БД.

### СУБД Redis

**Redis** (*RE*mote *DI*ctionary *S*erver) – современное хранилище модели ключ-значение с открытым исходным кодом [21]. Redis размещен в облаке по парадигме PaaS проектом «Redis Cloud» [22].

Отличительными особенностями Redis являются:

* Фундаментальные типы данных без дополнительных слоев абстракции;
* Использование оперативной памяти в качестве хранилища;
* Объем хранимых данных не может превышать объема оперативной памяти;
* Горизонтальное масштабирование, работа в кластере;
* Простота изучения, близость способов хранения к программным методам.

Redis спроектирован для того, чтобы дополнять традиционные СУБД, улучшать их работу на специфических данных [23]. Таким образом, Redis хорошо подходит для дополнения в том числе защищенных СУБД, например, CryptDB.

Redis – хранилище модели ключ-значение, одному ключу соответствует одно значение, с помощью ключа можно адресовать значение; и ключ, и значение представляют собой строки.

Строка является базовой структурой данных Redis, именно она хранит любое данное в БД. Строки в Redis могут представлять любые бинарные данные, например, строка может содержать изображение или сериализованный объект. Строка может занимать максимум 512 Мб.

Для строк используются встроенные атомарные операции : incr, decr, incrby, также неатомарные конкатенация (append), произвольного доступа (getrange, setrange), бинарного доступа (getbit, setbit).

Строки могут объединяться в структуры: списки, множества, хеши, сортированные множества. Доступные структуры данных часто используются в промышленном программировании, и схожесть модели представления данных БД и программ позволяет существенно ускорить разработку.

Redis – надежное хранилище данных, поддерживает автоматическую репликацию и журналирование операций [21]. В СУБД доступны четыре стратегии сохранения полученных данных.

* *Стратегия хранения в памяти* подразумевает использование БД как кеширующего сервера, данные не сохраняются на внешнем носителе;
* *Стратегия периодического сохранения* *данных* периодически сбрасывает копию данных на жесткий диск;
* *Стратегия лога транзакций* записывает каждую транзакцию на диск в режиме добавления в конец файла. Восстановление БД происходит через исполнение всех транзакций из лога;
* *Стратегия репликации* предполагает синхронизацию изменений на подчиненных серверах в кластере, количество копий неограниченно. При этом операции на запись принимает только мастер-сервер, а чтение возможно проводить и с подчиненных [24].

Redis поддерживает работу в режиме кеша – возможно удаление малоиспользуемых ключей при недостатке оперативной памяти.

Redis поддерживает работу в кластере – используется горизонтальное масштабирование, в этом же кластере могут работать узлы репликации.

Кроме того, у Redis есть поддержка транзакций – используется блокировка командой multi и встраиваемых процедур на языке Lua.

Еще одной интересной особенностью СУБД является режим очереди сообщений – клиентское приложение может подписываться на сообщения по определенному ключу и рассылать их.

В настоящее время Redis используют Twitter, Instagram, Blizzard, Stackoverflow, Guardian, flickr, Groupon [21]. Разработчики планируют дальше развивать СУБД: сейчас проект спонсирует компания VMware [25].

Redis предоставляет простой язык запросов и специально спроектирована для работы с реляционными СУБД – в целом отлично подходит для исследования возможности гибридной схемы работы защищенных БД.

### СУБД HBase

**HBase** – нереляционная распределенная база данных с открытым исходным кодом, является аналогом Google BigTable. HBase – NoSQL колоночная СУБД, созданная для поддержки параллельных вычислений над огромными (петабайты) массивами данных в кластере. Особенности HBase [26]:

* Линейная и модульная масштабируемость;
* Целостные операции чтения и записи;
* Атомарное и конфигурируемое распределение таблиц;
* Встроенный блочный кеш;
* Поддержка сжатия и фильтров для каждого столбца.

HBase не является прямой заменой реляционных СУБД. Она спроектирована для хранения разряженных данных. HBase не поддерживает SQL, по факту HBase не является реляционной СУБД.

HBase хранит набор таблиц. Каждая таблица содержит строки и столбцы. Каждая таблица должна иметь первичный ключ и все запросы к БД должны содержать этот ключ. Столбец представляет собой атрибут объекта, например, время или цена. HBase позволяет группировать атрибуты так, что они хранятся совместно, при этом группы могут располагаться отдельно. В этом состоит отличие от реляционных баз данных, в которых все столбцы таблицы хранятся вместе [27]. Кроме того, HBase не хранит пустые ячейки, что позволяет существенно экономить дисковое пространство по сравнению с реляционными БД [28].

HBase, в отличие от документ-ориентированных БД, требует создания схемы БД и определения колоночных групп до начала манипуляций с данными.

В основном, HBase используют для приложений парадигмы MapReduce. MapReduce – модель распределенных вычислений, представленная Google [29]. Модель состоит из двух шагов: Map и Reduce. На шаге Map проходит предварительная обработка входных данных. На шаге Reduce происходит свертка предварительно обработанных данных, полученных от рабочих узлов. Инновационность технологии заключается в возможности параллельного выполнения обоих шагов.

Разработчики советуют использовать HBase при выполнении условий [30]:

* Существует огромное количество данных, порядка сотни миллионов или миллиардов записей. Если же количество данных меньше, то традиционные БД более эффективны;
* Возможен отказ от возможностей реляционных БД: типизированные столбцы, вторичные ключи, индексы, транзакции, SQL языки;
* Достаточное количество аппаратных ресурсов. Необходимо минимум пять ЭВМ для запуска только HDFS (файловая система HBase).

HBase – специфичная для огромных объемов данных СУБД, уровня Google, Youtube, Bing. В типичном использовании защищенных БД такого объема не предполагается, кроме того HBase мало приспособлен к использованию в гибридной схеме вместе с SQL СУБД.

### Вывод

В качестве базовой СУБД для защищенного NoSQL решения наиболее всего подходит Redis. Redis предоставляет простой язык запросов и специально спроектирован для гибридной работы с реляционными СУБД. Кроме того, существуют решения PaaS Redis в облаке.

### Обзор угроз конфиденциальности баз данных

Угрозы раскрытия конфиденциальных данных присутствуют на каждом этапе их обработки: во время хранения в центре обработки данных, при анализе резервных копий, во время передачи по открытым каналам связи [2].

Ralf Durkee вводит следующий спектр угроз конфиденциальности информации, хранящейся в базах данных [31]:

1. Кража или потеря хранилища информации БД.
2. Кража или потеря хранилища бэкапов БД.
3. Кража или потеря бэкапов операционной системы.
4. Эксплуатация прямого доступа к БД.
5. Компрометация системы БД.
6. Системный администратор БД или компрометация его учетной записи.
7. Компрометация аккаунта пользователя БД.
8. Эксплуатация приложения с ограниченной учетной записью.
9. Эксплуатация приложения с неограниченной учетной записью.

Данная работа реализует систему защиты от первых четырех угроз по приведенной классификации, а также седьмой за счет использования двухфакторной аутентификации EMV.

Гибридную СУБД планируется размещать в облаке – недоверенной среде. Поэтому стоит предусмотреть возможность того, что злоумышленник может считывать содержимое оперативной памяти ЭВМ с целью поиска ключей шифрования и чтения открытых данных.

### Обзор криптографических стратегий защиты баз данных

Защита базы данных от несанкционированного доступа реализуется по нескольким направлениям. Работа «Database Encryption» [32] вводит четыре стратегии использования криптографических методик для защиты баз данных: уровня запоминающего устройства, уровня базы данных, уровня приложения, аппаратного уровня.

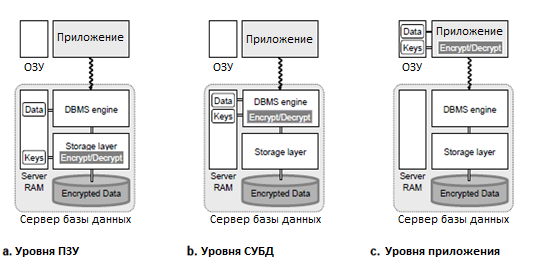


Рисунок 3 – Методики защиты баз данных

Рассмотрим подробно каждую из них в контексте размещения базы данных в облаке и укажем существующие реализации.

### Стратегия защиты уровня запоминающего устройства

**Стратегия уровня запоминающего устройства** (Storage-level encryption). На этом уровне шифрование данных происходит в структурах файловой системы: папках и файлах. Весь процесс шифрования и дешифрования выполняет файловая система, для базы данных оно прозрачно.

Преимуществами решения являются:

* Не требуется изменений в базе данных и клиентских приложениях.

Также присутствуют значимые недостатки:

* Существенная потеря производительности вследствие отсутствия гранулярного доступа;
* Невозможно распределять привилегии для разных пользователей;
* Необходим дополнительный контроль того, что важные данные не реплицируются во временных файлах, журналах.

Существует множество реализаций шифрующих файловых систем [33]: Encrypted File System (EFS) для Windows OS; EncFS, eCryptfs для семейства Linus OS. Стратегия использования специальных файловых систем реализуема для любой СУБД - Oracle Database, MsSQL, MySQL могут использовать данный подход. Стратегия не дает гарантии сохранения конфиденциальности информации, поскольку процедура расшифрования файлов СУБД производится ресурсами облака – недоверенной среды.

Злоумышленник может использовать дамп оперативной памяти сервера БД для поиска ключа расшифрования или открытых данных.

### Стратегия защиты уровня базы данных

**Стратегия уровня БД** (Database-level encryption). На этом уровне данные шифруются во время вставки или извлечения из базы, т.е. стратегия использования шифрования должна быть встроена в структуру БД.

Преимущества использования:

* Любая гранулярность шифрования. Можно защитить отдельно строку, столбец, таблицу, строку по логическому значению, например, данные о продажах стоимостью более $1000;
* В связи с гранулярностью шифрования удается удержать падение производительности на достаточном уровне.

Недостатки стратегии:

* В зависимости от реализации может потребоваться изменение пользовательских приложений;
* Требуется включение пакета безопасности в базу данных;
* Невозможность создания индексов.

СУБД Oracle Database имеет несколько реализаций данной стратегии защиты (отличаются деталями реализации и развертывания):

* Oracle DBMS Obfuscation Toolkit (DOTK);
* Oracle DBMS\_CRYPTO package;
* Oracle Transparent Data Encryption (TDE);
* Oracle Advanced Security Option;
* Oracle Database Vault.

Особый интерес представляет технология TDE – она позволяет прозрачно для приложений зашифровывать отдельные блоки данных [34]. TDE поддерживает все структуры БД: таблицы, индексы, каталоги страниц. Возможно частичное шифрование данных в таблицах (по колонкам), что позволяет существенно увеличить скорость обработки запросов и в то же время сохранять конфиденциальность ценной информации.

Данные автоматически шифруются во всех подсистемах – от хранения до журналирования. TDE также предоставляет встроенный сервис управления ключами, упрощая процесс их распределения и хранения. TDE использует аппаратное ускорение на процессорах Intel с поддержкой AES.

СУБД MsSQL также поддерживает свою реализацию TDE [35]. Решений TDE для MySQL в настоящее время нет.

Стоит отметить, что данная стратегия не обеспечивает сохранения конфиденциальности обрабатываемой информации в облаке, поскольку проводит процедуры шифрования и дешифрования в недоверенной среде. Нарушители могут использовать слежение за оперативной памятью сервера и считывать оттуда ключи шифрования или данные в открытом виде.

### Стратегия защиты уровня приложения

**Стратегия уровня приложения** (Application-level encryption).Метод перемещает процесс шифрования и дешифрования в клиентское программное обеспечение, поэтому данные всегда хранятся и передаются в защищенном виде.

Преимущества решения:

* Разнесение ключа для доступа к данным и сервера базы данных;
* Возможность реализации различных стратегий распределения ключа;
* Наилучшая защита от внешнего наблюдателя.

Недостатки решения включают:

* Требуется модификация клиентских приложений;
* Требуется реализация политик хранения ключа;
* Отсутствие гранулярности на уровне БД. В связи с этим возможно резкое увеличение количества информации, передающейся между БД и приложением;
* Падение производительности;
* Невозможность использования встроенных процедур, триггеров в БД.

Стратегия позволяет сохранять конфиденциальность информации, размещенной в облаке, поскольку процедуру шифрования и дешифрования проводит клиентское программное обеспечение. Злоумышленник, имея доступ к оперативной памяти сервера, журналам и внешним носителям БД, сможет получить лишь зашифрованные данные.

Существенным недостатком стратегии является необходимость изменения клиентских приложений, однако они могут быть сведены к минимуму через использование промежуточных серверов шифрования, размещенных в доверенной зоне. Подобным образом работает, например, CryptDB. Кроме того, CryptDB поддерживает гранулярность данных, что существенно снижает количество передаваемых данных между клиентом и сервером.

### Использование внешнего модуля защиты

**Использование внешнего модуля защиты** (Hardware security module). HSM предоставляет защищенное (в том числе физически) хранилище ключей. В зависимости от реализации аппаратной части HSM может также предоставлять функции шифрования/дешифрования данных, разграничения доступа. Часто в HSM реализуют функцию резервного копирования ключей [36].

Преимущества решения:

* Высокий уровень защиты;
* Расширенные возможности управления ключами.

Недостатки включают:

* Высокая стоимость;
* Низкая производительность, особенно при использовании встроенного сопроцессора шифрования;
* Малый выбор алгоритмов шифрования, сложность или невозможность их замены;
* Необходимость поддержки аппаратной части СУБД;
* Сложная интеграция с облачными решениями.

HSM не в полной мере подходит для применения с облачными решениями. При использовании шифрования внутри устройства требуется передача больших объемов данных, теряются функции СУБД, становится невозможным выполнение большинства SQL запросов. Если использовать HSM только как хранилище ключей, то нарушитель по-прежнему может считывать открытые данные из оперативной памяти сервера БД, поскольку сервер расшифровывает данные ключом, полученным от HSM.

### Вывод

Стратегия уровня приложения обеспечивает наилучшую защиту ценных данных, при этом некоторые ее недостатки могут нивелироваться особенностями БД. Так, например, некоторые БД изначально не имеют поддержки встроенных процедур. Существенное преимущество заключается в том, что злоумышленник даже при наличии полного доступа к серверу БД, его резервными копиями и журналами, не сможет расшифровать хранимые данные.

Кроме того, стратегия защиты уровня приложения не всегда требует модификации самого приложения: возможно создание внутреннего сервера шифрования, который выполняет функцию прокси между приложением и внешней базой данных.

С практической точки зрения стратегия шифрования на клиентском уровне интересна тем, что позволяет вынести хранение ценных данных в облако.

### Обзор алгоритмов шифрования

Стратегия шифрования уровня приложения дает возможность использовать для ценных данных БД, расположенные в недоверенной среде. Однако практическое применение этого метода затруднено в связи с необходимостью передавать огромные массивы данных между СУБД и клиентом. Также типичные для СУБД операции приходится выполнять в клиентском приложении.

Проблема выполнения операций в СУБД до недавнего времени была неразрешимой, поскольку СУБД, имея в распоряжении лишь зашифрованные данные, не может ничего узнать об исходных значениях.

В настоящее время появились классы специальных алгоритмов шифрования, решающих эту проблему.

### Гомоморфные шифрования. Схема полностью гомоморфного шифрования

**Гомоморфное шифрование** (Homomorphic Encryption, HE). Позволяет производить определенные действия с зашифрованным текстом и получать зашифрованный результат, который соответствует результату операций, выполняемых с открытым текстом. С применением гомоморфных схем шифрования становится возможным, например, сложение зашифрованных чисел с получением верного зашифрованного результата. Большинство схем гомоморфного шифрования поддерживают только одну выполняемую операцию. Приведем гомоморфные свойства для распространенных систем шифрования.

Таблица 1 – Список гомоморфизмов шифрований

|  |  |
| --- | --- |
| Криптосистема | Гомоморфизм |
| RSA |  |
| Эль-Гамаля |  |
| Гольдвассер-Микали |  |
| Пэйе |  |

**Полностью гомоморфное шифрование** (Full Homomorphic Encryption, FHE) [37] - впервые было предложено Craig Gentry в 2009 году. Криптосистема основана на использовании свойств идеальных решеток.

Система поддерживает два гомоморфизма над данными:

(1)

Свойство гомоморфизма существенно расширяет список доступных практических использований. Недостатком предложенной схемы являются существенные вычислительные затраты: увеличение сложности составляет около триллиона раз [38]. Поэтому практическое применение в настоящее время затруднено. Craig Gentry надеется внедрить схему в промышленные решения в течение 5-10 лет [39].

### Шифрование, сохраняющее порядок

**Шифрование, сохраняющее порядок** (Order Preserving Encryption, OPE) - впервые было представлено в 2004 году совместно исследователями Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, Yirong Xu [40].

Криптосистема сохраняет порядок, если выполняется условие:

(2)

Инновационная схема позволяет СУБД создавать индексы над зашифрованными данными, осуществлять выборку, в связи с чем резко сокращает объем данных, передаваемых от сервера к клиенту при использовании стратегии защиты уровня приложения.

На данный момент идея, предложенная исследователями, подверглась ряду изменений, что привело к появлению некоторых улучшений первоначального алгоритма. Alexandra Boldyreva предложила алгоритм **OPES** - симметричную схему для подобного шифрования [41]. Liangliang Xiao расширил криптосистему на объекты, отличные от чисел [42]. Racula Ada Popa – ideal security схему шифрования [43].

В настоящее время стойкость шифрования, сохраняющего порядок, была проанализирована несколькими группами исследователей, которые не выявили атак, компрометирующих надежность системы [44; 45].

### Симметричные схемы шифрования

**Симметричные криптосистемы** – способ шифрования, в котором для шифрования и расшифрования применяется один и тот же криптографический ключ [46]. Ключ шифрования должен сохраняться в секрете обеими сторонами, алгоритм выбирается до начала обмена сообщениями.

Алгоритмы симметричного шифрования характеризуются:

* Стойкостью;
* Длиной ключа;
* Числом раундов;
* Длиной блока.

В мировой практике широко используются шифры DES, AES; в российской - ГОСТ 28147-89.

**DES**[47] - шифр с 64-битовыми блоками. Использует сеть Фейстеля и 16 раундов, для шифрования использует ключ с длиной 56 бит. Известны атаки со сложностью 238, поэтому использование DES небезопасною.

Прямым развитием DES в настоящее время является 3DES. Он выполняет 3 раза алгоритм DES, а длина ключа 3DES равна 3\*56=168 бит. Скорость работы 3DES в 3 раза ниже, чем у DES, но криптостойкость намного выше — время, требуемое для криптоанализа 3DES, может быть в миллиард раз больше, чем время, нужное для вскрытия DES.

Скорость работы алгоритма на AMD процессоре с частотой 2,2 ГГц составляет 13 МБайт/с.

**AES** [47] **-** блочный шифр с переменной длиной ключа, не использующий сети Фейстеля. Размер блока составляет 128 бит. Длина ключа может быть выбрана равной 128, 192 или 256 бит. Алгоритм в использует 10, 12 или 14 раундов в зависимости от длины ключа.

26 мая 2002 года AES был объявлен стандартом шифрования. В июне 2003 года NSA США стало использовать AES для государственной тайны. Вплоть до уровня SECRET было разрешено использовать ключи длиной 128 бит, для уровня TOP SECRET требовались ключи длиной 192 и 256 бит.

Скорость работы алгоритма с ключом длины 128 бит на AMD процессоре с частотой 2,2 ГГц составляет 109 МБайт/с. В процессорах Intel встроен аппаратный модуль [48] AES, поэтому скорость AES256 составляет 2000 МБайт/с.

### Вывод

Для сервера Redis были выбраны следующие системы шифрования: AES, криптосистема Пэйе и OPES.

В качестве симметричной системы шифрования для данных, не требующих дополнительной обработки – AES, поскольку алгоритм является промышленным стандартом защиты данных.

В качестве данных, над которыми необходимо проводить операцию сложения (единственная арифметическая операция, доступная в Redis) – криптосистему Пэйе, поскольку она поддерживает гомоморфное свойство для сложения.

В качестве шифрования, сохраняющего порядок – OPES, поскольку симметричная система проще в использовании, а ее безопасность была проанализирована независимыми группами исследователей.

### Механизмы двухфакторной аутентификации [49]

Для проверки подлинности личности могут использоваться три различных типа информации:

* Субъект знает;
* Субъект имеет;
* Часть субъекта.

Для надежной аутентификации эксперты рекомендуют использование более одного типа информации от субъекта. Например, таким сочетанием может быть секретная фраза и отпечаток пальца.

В настоящее время в организациях широко используются ключи iButton, eToken, смарт-карты и различные смс-сервисы. При этом стоимость внедрения готовых решений может оказаться довольно большой из-за необходимости разворачивания инфраструктуры публичных ключей и закупки необходимого оборудования.

### Использование карт EMV для двухфакторной аутентификации

**EMV** – Europay, MasterCard, Visa – международный стандарт для обеспечения инфраструктуры банковских карт с чипом [50]. Стандарт был разработан с целью повышения безопасности проводимых финансовых операций по картам. Стандарт определяет физическое, электронное и информационное взаимодействие между банковской картой и терминалом доступа. Существуют также стандарты для бесконтактных карт.

Введение стандарта позволило обеспечить повышенный уровень безопасности транзакций и их контроля. После введения стандарта снизилось количество проверок, проводимых человеком – визуальный контроль голограммы, подписи, сверка имени. Внедренный в карту чип не подвержен воздействию магнитных полей, что было проблемой для карт предыдущих поколений.



Рисунок 4 – Чип смарт-карты формата EMV

Чип безопасности (см. рисунок 4) имеет большую степень безопасности по сравнению с магнитной полосой. Секретный ключ чипа записывается в закрытую область памяти в процессе изготовления. Благодаря этому, извлечение ключа внешними средствами становится невозможным без нарушения целостности самого чипа [51]. ПИН-код карты проверяется самим чипом, поэтому его перехват также осложнился.

Интересно также то, что стандарт полностью описывает инфраструктуру открытых ключей, применяемых для аутентификации держателя карты и проведения платежных транзакций. Инфраструктура гарантирует целостность, уникальность карты и подлинность личности владельца.

Приведенные выше факторы позволяют использовать банковскую карту в качестве средства аутентификации в сторонних системах. В дипломной работе будет рассмотрено использование публичной области данных для осуществления двухфакторной аутентификации пользователей СУБД.

Преимущества использования карт EMV для аутентификации пользователей:

* Каждый пользователь уже имеет такую карту, что снижает затраты на инфраструктуру;
* Открытые стандарты;
* Дешевизна считывателей смарт-карт;
* Готовая инфраструктура публичных ключей.

Таким образом, использование банковских карт для двухфакторной аутентификации во внебанковских приложениях представляется авторам перспективным направлением.

### Обзор существующих готовых решений. CryptDB (SQL)

CryptDB [3] – первая промышленная СУБД, использующая стратегию криптографической защиты уровня приложения. CryptDB поддерживает выполнение большинства типичных SQL запросов: поиск, сортировка, математические функции над зашифрованными данными. При этом падение производительности по сравнению с MySQL составляет в среднем 15-26%, а пиковое падение – на операции обновления с суммированием – составляет 39% [52].

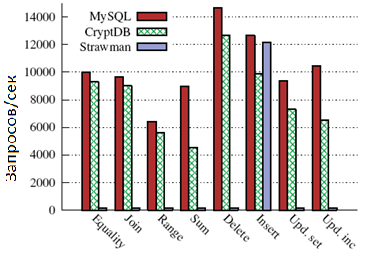


Рисунок 5 –Падение производительности CryptDB

Исследователи MIT поставили производительность и минимальные изменения клиентского приложения в качестве главного приоритета. Они использовали различные схемы гомоморфного, симметричного, сохраняющего порядок шифрований.

Инновационной моделью CryptDB является способность на лету переключаться между различными криптографическими схемами [53]. Возможность реализована за счет многоуровневого «луковичного» шифрования, когда данные зашифрованы в несколько слоев различными алгоритмами.

Каждый слой характеризуется своим ключом и списком поддерживаемых операций. На нижнем слое используются самые стойкие к атакам алгоритмы, а операции в верхних слоях возможны без расшифрования нижних.

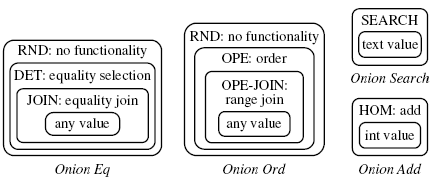
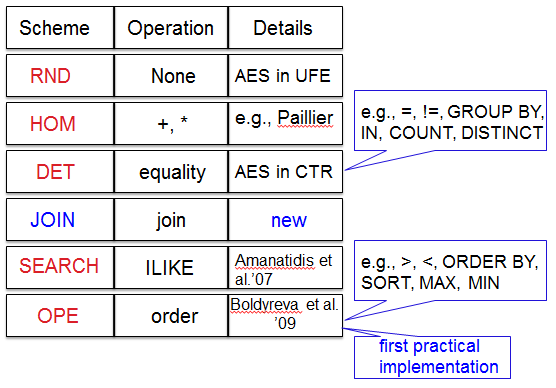


Рисунок 6 – Схема луковичного шифрования CryptDB

Список поддерживаемых запросов включает: сравнения, поиск, слияние, арифметические операции. Существует ряд ограничений на вычисления внутри запроса (например, нельзя использовать “salary > age\*2”, но возможны “salary > age”, “salary > 100”).

По оценкам исследователей, CryptDB поддерживает около 99,5% часто используемых запросов к БД [54]. Эти операции включают select, insert, update, delete для условий порядка, слияния, схожести по шаблону, сравнения, арифметических операций.

CryptDB реализован в виде прокси-сервера, а для выполнения операций на стороне СУБД используется ряд встроенных процедур. Прокси-сервер CryptDB видоизменяет запрос клиентского приложения посредством шифрования аргументов, а встроенные процедуры выполняют модификации «лука» для выполнения операций. Ответ СУБД проходит через процедуру расшифрования прокси-сервера.



Рисунок 7 – Архитектура CryptDB

CryptDB позволяет размещать СУБД-носитель в недоверенной среде, например, в облаке. Поддерживаются MySQL 5.1 и PostgreSQL 9 без модификации исходного кода СУБД. Перечисленные СУБД допускают развертывание в облаке по типу IaaS. Решение CryptDB также поддерживает политики управления ключами, т.е. разнесение БД и системы хранения ключей, что улучшает безопасность.

Недостатками CryptDB являются:

* Необходимость изменения клиентского приложения, если оно уже было написано со сложными запросами;
* Необходимость размещения прокси-сервера в доверенной зоне.

На текущей стадии развития проекта опробована работа CryptDB в качестве СУБД для защищенного форума. Падение производительности составляет в среднем 14,5% по сравнению с MySQL [52]. На тесте производительности СУБД TPC-C производительность падает на 26% [52].

В дальнейшем исследователи планируют развивать проект: изначально он финансировался Google и Citigroup, на настоящий момент агентство оборонных проектов DARPA (Defense Advanced Research Projects Agency) выделило грант на дальнейшие исследования в области защищенных баз данных [55].

### Вывод

На текущий момент не существует NoSQL криптографической СУБД. При этом подобные СУБД могут существенно увеличить скорость существующих SQL решений в гибридной архитектуре.

Увеличение скорости достигается за счет упрощения изначальной структуры БД, упрощения криптографического слоя и гибридной архитектуры, когда ценные данные располагаются в NoSQL СУБД, а обычные – в SQL.

Опыт CryptDB показал, что прикладным разработчикам удобнее работать с криптографической СУБД как с обычной. Возможность появляется благодаря использованию прокси архитектуры – криптографическое ПО встраивается в канал связи между СУБД и клиентскими приложениями. Разумно использовать этот подход и при создании NoSQL базы данных.

ГЛАВА 2. ОПИСАНИЕ ВЫБРАННЫХ ТЕХНОЛОГИЙ

### ****Блочное шифрование AES**** [56]

Алгоритм AES представляет собой блочный шифр с переменной длиной ключа, не использующий сети Фейстеля. Размер блока составляет 128 бит. Длина ключа может быть выбрана равной 128, 192 или 256 бит. Алгоритм зафиксирован в стандарте FIPS-197 «Advanced Encryption Standard» в 2001 году [56].

При описании алгоритма используется поле Галуа (конечное поле) , построенное как расширение поля по корням неприводимого многочлена

(3)

Данный многочлен выбран из соображений эффективности представления элементов поля. Элементарные операции, используемые в алгоритме, выполняются в указанном поле.

Шифр является последовательностью итераций, выполняемых над некоторой промежуточной структурой, называемой состоянием.

Приведем основные определения и вспомогательные процедуры в таблице 2.

Таблица 2 – Основные определения и вспомогательные процедуры AES

|  |  |
| --- | --- |
| **Основные определения** | |
| **Определение** | **Описание** |
| Round Key | Раундовый ключ. Получается из ключа шифрования, используя процедуру KeyExpansion. Применяется к состоянию на этапе AddRoundKey. |
| State | Состояние. Промежуточный результат шифрования, массив размером 4 строки и Nb столбцов. |
| S-box | Нелинейная таблица замен, использующаяся в нескольких трансформациях обратимой замены байт. |
| Nb | Число столбцов (32-х битных слов), составляющих State. |
| Nk | Число 32-х битных слов, составляющих ключ шифрования. |
| Nr | Число раундов, которое является функцией Nk и Nb. |
| **Вспомогательные процедуры** | |
| KeyExpansion | Процедура, используемая для генерации Round Keys из Cipher Key |
| AddRoundKey() | Трансформация, при которой Round Key складывается по модулю 2 c State. |
| MixColumns() | Трансформация, которая берет все столбцы State и смешивает их данные (независимо друг от друга) благодаря умножению на многочлен. |
| RotWord() | Функция, использующаяся в процедуре Key Expansion, которая берет четырех-байтное слово и производит над ним циклическую перестановку. |
| ShiftRows() | Трансформация, которая циклически смещает строки State на разные величины. |
| SubBytes() | Трансформация, которая обрабатывает State, используя S-box, применяя его независимо к каждому байту State. |
| SubWord() | Функция, используемая в процедуре Key Expansion, которая берет на входе четырёх-байтное слово и, применяя S-box к каждому из четырёх байтов, выдаёт выходное слово. |

Шифр является последовательностью итераций, выполняемых над некоторой промежуточной структурой, называемой состоянием. Состояние представляет собой прямоугольный массив байтов. В массиве 4 строки, а число столбцов составляет

(4)

Ключ шифрования аналогичным образом представляется в виде прямоугольного байтового массива с 4 строками, количество столбцов

(5)

Входные и выходные значения алгоритма представляются в виде одномерных байтовых массивов соответствующей длины.

Состояние и ключевой массив заполняются из входных массивов:

(6)

где , т.е. вначале по столбцам, потом по строкам. Количество итераций обозначается и зависит от в соответствии с таблицей 3.

Таблица 3 – Количество итераций AES

|  |  |  |  |
| --- | --- | --- | --- |
| **Nk** | 4 | 6 | 8 |
| **Nr** | 10 | 12 | 14 |

Итерационное преобразование состоит из четырех этапов и на псевдокоде записывается следующим образом:



Рисунок 8 – Итерационное преобразование AES

Последняя же итерация отличается от остальных:



Рисунок 9 – Последнее итерационное преобразование AES

**SubBytes.** Это операция нелинейной обратимой байтовой замены (S-box), состоящая из двух шагов. Во-первых, каждый байт заменяется на обратный к нему в поле , при этом байт отображается в себя. Во-вторых, над каждым байтом выполняется аффинное преобразование в поле , задаваемое следующим матричным уравнением:

(7)

Это преобразование может быть описано в полиномиальном виде как

(8)

Полином-множитель взаимно прост с модулем операции, поэтому она является обратимой в поле.

**ShiftRows.** Это преобразование является циклическим сдвигом влево строк массива состояния на различную величину. Строка 0 не сдвигается, строка 1 сдвигается на 1 позицию, строка 2 — на 2 и строка 3 на 3. Обратное преобразование – это сдвиг строк вправо на такое же количество позиций.

**MixColumns.** Cтолбцы массива состояния рассматриваются как полиномы над полем . Преобразование заключается в умножении каждого столбца на фиксированный полином

(9)

по модулю . Полином выбран взаимно простым с модулем, поэтому умножение обратимо – для этого достаточно умножить состояние на обратный к многочлен.

**AddRoundKey.** RoundKey каждого раунда объединяется с состоянием. Производится побитовое сложение по модулю 2 каждого байта состояния с соответствующим байтом массива ключа. По свойству операции «исключающего или», процедура обратна сама себе при повторном вызове.

**Алгоритм обработки ключа.** Состоит из двух частей: алгоритма расширения ключа и алгоритма выбора раундового ключа. В результате вычисляются раундовые ключи длиной 128 бит (размер блока) из ключа шифрования.

**Алгоритм расширения ключа**. Получает ключи для всех раундов. Расширенный ключ представляет собой линейный массив 4-байтовых слов и обозначается как . Существует две версии функции - для и для .

Для алгоритм следующий:

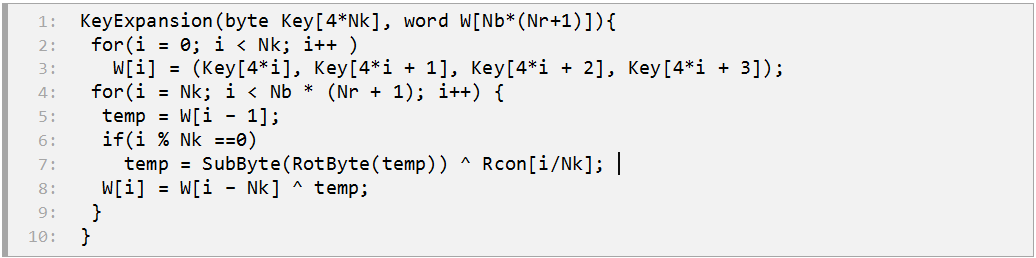


Рисунок 10 – Расширение ключа AES

В алгоритме используется функция – возвращает слово, в котором каждый байт является результатом применения S-box к байту, находящемуся на соответствующей позиции во входном слове.

Функция представляет собой циклический сдвиг байтов в слове, т.е.

(10)

Константы заданы в поле

(11)

**Выбор раундового ключа**. Раундовый ключ для на итерации выбирается из расширенного ключа, начиная с элемента и до .

**AES Rijndael**. Приведем процесс шифрования на псевдокоде:

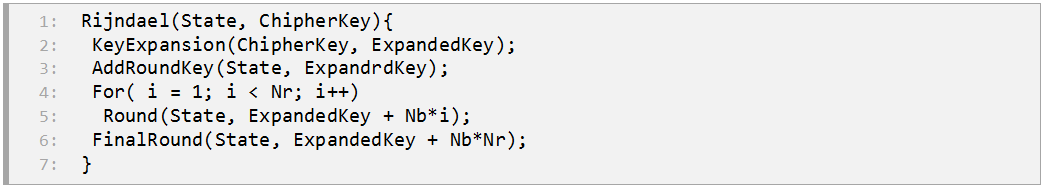


Рисунок 11 – Алгоритм шифрования AES

### ****Шифрование, сохраняющее порядок****

**Order preserving encryption**. OPE – шифрование, сохраняющее порядок зашифрованного текста таким же, как и у открытого текста. OPE – функция , такая что выполняется:

(12)

Алгоритмом OPE будем называть алгоритм, реализующий OPE.

Задача, решаемая OPE – это обеспечение эффективных «граничных» запросов (range query) над зашифрованными данными, возможность построения индексов и сортировки данных. Под эффективным подразумевается **логарифмическое время** выполнения относительно объема данных.

### ****Тривиальный алгоритм OPE**** [40]

Тривиальная функция OPE описана Rakesh Agrawal в работе «OPE for Numeric Data» в 2004 году [40]. Алгоритм использует строго возрастающую последовательность случайных чисел.

**Идея шифрования.** Пусть необходимо зашифровать множество открытых текстов – оно задано заранее и не меняется в процессе работы алгоритма. Сгенерируем уникальных значений с помощью криптографического ГПСЧ с необходимым распределением. Полученный ансамбль чисел отсортируем и сохраним в таблице . Закрытый текст получается из открытого текста результате вычисления выражения . Получаем, что i-тый открытый текст кодируется в i-тый элемент отсортированной таблицы . Таблица хранит полную информацию о шифровании и дешифровании, поэтому должна храниться в секрете – она является ключом процедуры.

**Схема шифрования.** Схема состоит из трех частей: процедуры генерации таблицы случайных величин , процедуры шифрования и процедуры дешифрования. В качестве таблицы используется структура данных массив. Для упрощения вычислений, но без потери общности, положим, что множество открытых текстов имеет структуру

(13)

**Алгоритм генерации .** Для генерации таблицы используется ключ и криптографический ГПСЧ. Основное свойство полученной в результате таблицы – возрастающий порядок уникальных значений. Опишем процедуру на псевдокоде:

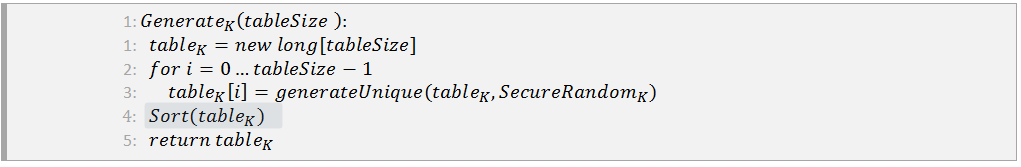


Рисунок 12 – Генерация ключевой таблицы в OPE

**Шифрование и дешифрование.** Шифрование осуществляется индексацией в таблице , т.е. вычислением выражения

(14)

Дешифрование заключается в поиске элемента в , для этого используется бинарный поиск. Запишем сказанное на псевдокоде:



Рисунок 13 – Шифрование и расшифрование OPE

**Безопасность схемы.** Приведенная схема не раскрывает никакой информации об изначальных значениях кодируемых величин, кроме их порядка, в том числе благодаря тому, что уникальные значения сгенерированы по постоянному распределению, не зависящему от исходных данных. При этом даже если злоумышленник получил доступ ко всем зашифрованным текстам, он не может получить по ним таблицу преобразований .

**Сложность схемы.** Пусть – количество уникальных значений, хранимых в базе данных. Процедура шифрования получает i-тый элемент из массива, требуя на это операций. Дешифрование осуществляется бинарным поиском по массиву, сложность составляет . Генерация таблицы состоит из генерации уникальных значений и их сортировки, сложность алгоритма .

**Недостатки схемы.** Во-первых,объем занимаемой памяти в два раза превышает количество уникальных значений, хранимых в базе данных. Во-вторых,существует сложность добавления новых открытых текстов: при добавлении значения , где необходимо перекодировать значения .

### ****Алгоритм Symmetric OPE (OPES)**** [39]

Алгоритм описан Alexandra Boldyreva в статье «Order-preserving Symmetric Encryption», опубликованной в 2009 году [39]. Авторы развили работу Agrawal et. al «Order Preserving Encryption for Numeric Data», представленную на конференции SIGMOD в 2004 году. В работе Alexandra Boldyreva et. al сконструировали схему симметричного OPE.

**Идея схемы.** Alexandra Boldyreva показывает связь между OPE схемами с гипергеометрическим HG и отрицательным гипергеометрическим NHG распределениями. Связь позволяет моделировать OPE схему через генератор HG или NHG. При этом существуют эффективные алгоритмы точной генерации случайной величины этих распределений.

**Связь с NHG**. Пусть есть множество открытых текстов – «домен» и закрытых – «шеренга» , такие что . Рассмотрим любую OPE функцию , которая может быть представлена набором чисел из упорядоченных элементов. Найдем распределение вероятностей выбора таких комбинаций из . Представим, что есть шаров в урне, из которых черные и белые. На каждом шаге мы извлекаем случайный шар из урны без замены. Случайная величина , описывающая количество шаров в выборке после того, как был взят y-тый черный шар имеет NHG распределение. Используя вышеприведенное представление OPE, можно показать, что для точки имеет NHG распределение относительно случайного выбора . Если предположить существование эффективной схемы генерации NHG, возможно создание настолько же эффективной схемы OPE.

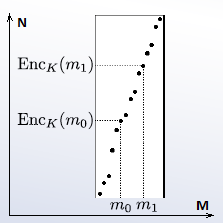


Рисунок 14 – Схема сохранения порядка OPE

**Переход от NHG к HG**. К сожалению, проблема существования эффективных алгоритмов генерации NHG до сих пор открыта. Известны аппроксимации NHG через негативное бинарное распределение, но их качество недостаточно. Однако для родственного распределения – гипергеометрического HG, существуют эффективные и точные (не аппроксимирующие) алгоритмы генерации. HG в качестве параметров принимает M черных шаров, N-M белых шаров и возвращает X – количество черных шаров в выборке y шаров. Сложность получившейся схемы составит в среднем .

Итак, существует биективная связь между – функции, сохраняющей порядок домена D после преобразования в шеренгу R и множеством всех комбинаций мощностью | из элементов.

(15)

В гипергеометрическом распределении из шаров и черных при размере выборки , вероятность того, что случайная величина составляет:

(16)

Интуитивно, эквивалентность правых частей уравнений означает, что можно построить произвольную OPE как результат эксперимента, в котором шаров, из которых черные. Далее произвольно вытаскиваем шар из урны. Если y-тый шар – черный, то - образ последней неассоциированной точки в домене.

**Обработка множества точек**. Вычисление отражения домена в шеренгу независимо для каждой точки по распределению HG еще не сохраняет порядок. Ситуацию можно исправить через сохранение всех предыдущих пар открытый-закрытый текст и корректирование параметров HG генератора для каждого нового открытого текста . Однако такой подход требует существенного объема памяти и снижает производительность.

**Альтернативный подход**. Первый шаг к уменьшению объема памяти для пар открытый-закрытый текст – хранение в более компактной структуре, где пары кодируются постоянной, но экспоненциально длинной случайной лентой. Идея заключается в том, что при шифровании открытого текста , алгоритм осуществляет бинарный поиск по ленте. Вначале вычисляется точка , далее если , то , иначе и так далее, пока не будет определен образ . Каждое определение закрытого текста делается в зависимости от значения HG, вычисленного с соответствующими параметрами и случайными величинами из ленты, индексированными открытым текстом. Алгоритм расшифрования определяется схожим образом.

Вместо непосредственной генерации и хранения длинной случайной ленты, ее можно вычислять на лету. Куски ленты генерируются из ключа шифрования и . Для обеспечения практической применимости, используемый PRF может быть основан на блочном шифре.

Поскольку параметры HG генератора и количество случайных величин с ленты, которые он использует, варьируется во время бинарного поиска, PRF должен удовлетворять свойствам вариативности длины данных по входу и выходу (variable input-length - VIL, variable output-length - VOL), т.е. быть LF-PRF (length-flexible PRF). Авторы получают данный генератор из VIL-PRF и VOL-PRG.

**Алгоритмы LazySample, LazySampleInv.** Алгоритмы - результат вышеупомянутых размышлений. Алгоритмы реализуют и соответственно, вначале они разделяют общее состояние – глобальные массивы величин и , изначально они пусты.

**Подзадача HGD.** HGD принимает на вход параметры и возвращает такое, что для каждого имеем с вероятностью

, (17)

где и .

**Подзадача GetCoins.** GetCoins генерирует монеты и принимает на вход параметры

, (18)

Функция возвращает случайные монеты . GetCoins возвращает уникальные случайные величины.

**Описание работы LazySample, LazySampleInv.** Введем обозначение, означающее что выбирается случайно и независимо из множества , используя множество случайных величин – **монет** - мощностью – необходимой для этого выбора. Положим как количество монет, необходимых для генерации HGD. Опишем алгоритмы на псевдокоде:

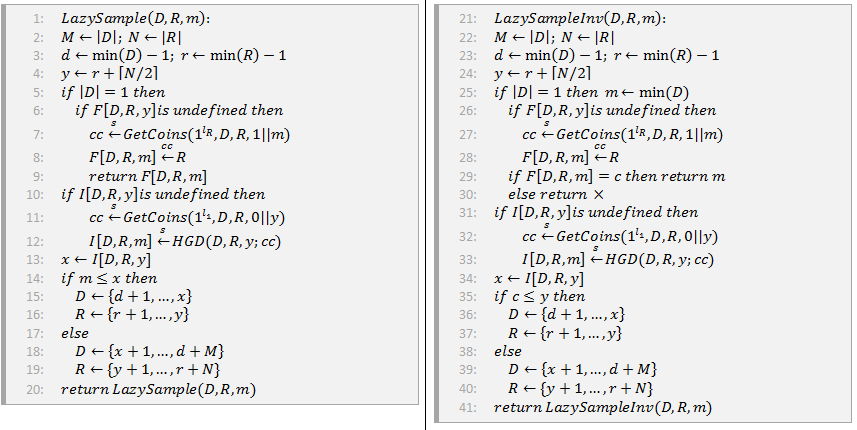


Рисунок 15 – Алгоритмы LazySample и LazySampleInv

Для вычисления образа , LazySample осуществляет стратегию ассоциирования отрезков шеренги к отрезкам домена бинарным поиском. Вначале алгоритм вычисляет значение – середину шеренги и ее прообраз в домене. Для этого в 12 строке он высчитывает значение HGD – какое количество точек в домене ассоциированы с и сохраняет это значение в массиве . Это значение – правая сторона выражения , где , вычисляемое в 13 строке. Таким образом, мы считаем, что отрезок и связаны кодирующим преобразованием.

Если открытый текст меньше левой границы найденного отрезка домена, то алгоритм запускает рекурсивный поиск в строке 20 в нижней половине диапазона. Иначе запускается поиск в верхней половине отрезка. Данный процесс продолжается до тех пор, пока отрезки по левую и правую сторону от не будут ассоциированы с .

Наконец, в строке 8 алгоритм выбирает произвольную точку из найденного отрезка , результат записывается в массив как образ , полученный функцией. Обратный алгоритм LazySampleInv действует по схожему принципу.

**Анализ LazySample и LazySampleInv**. Полученный результат еще не является схемой шифрования, поскольку функции разделяют глобальные массивы . Как было описано выше, для их упразднения вводится псевдослучайная лента. Сложность алгоритмов оценивается авторами как в худшем случае и в среднем.

**Процедура генерации ленты TapeGen.** TapeGen – это LF-PRF, т.е. псевдослучайная функция, которая принимает аргументы произвольной длины и возвращает случайные числа также произвольной длины. Alexandra Boldyreva конструирует TapeGen как композицию VIL-PRF и VOL-PRG. В качестве первого используется OMAC/CMAC на блочном шифре, например, AES. Полученные алгоритмы шифрования на псевдокоде:

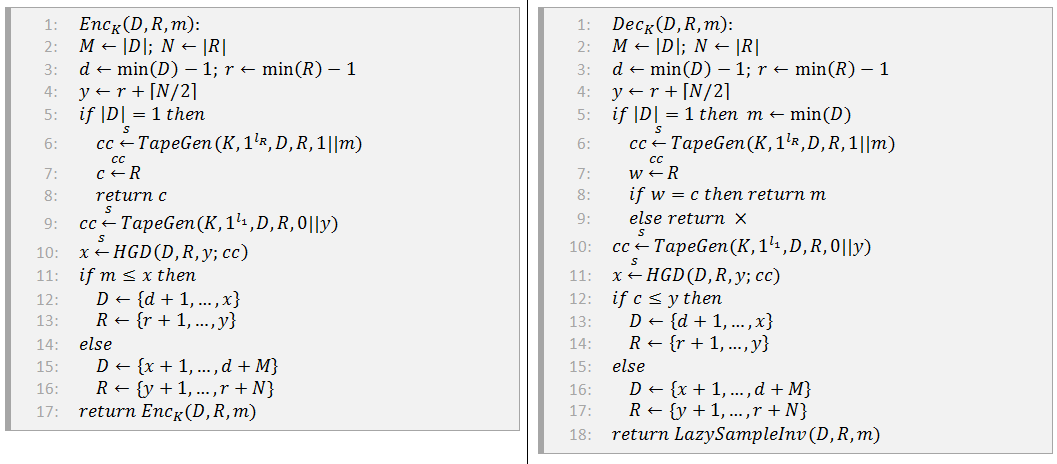


Рисунок 16 – Алгоритмы шифрования и дешифрования OPES

**Безопасность схемы**. Alexandra Boldyreva et. al начали криптографические исследования OPE, анонсированного на SIGMOD’04 для осуществления эффективных запросов над зашифрованными данными. Они показали невозможность достижения стойкости против indistinguishability against chosen-plaintext attack (IND-CPA) – атаки на основе адаптивно подобранного открытого текста, позволяющего подобрать ключ.

Авторы несколько ослабляют критерий безопасности посредством ввода pseudorandom order-preserving function against chosen-ciphertext attack (POPF-CCA) – атаки на основе подобранного шифротекста для OPE. Смысл POPF-CCA в том, что злоумышленник не может отличить результат шифрования OPE от соответствующей «идеальной» схемы OPE.

При проектировании алгоритмов LazySample и LazySampleInv Alexandra Boldyreva et. al ставили целью добиться максимально высокого уровня безопасности, который необходим для практического применения OPE. Алгоритмы удовлетворяют POPF-CCA критерию безопасности, если TapeGen является LF-PRF.

**Сложность алгоритмов**. При использовании рекомендованного алгоритма TapeGen, шифрование и дешифрование в худшем случае работают за время

(19)

В нотации O большое получаем

(20)

**Выбор множества R**. Необходимая для безопасности мощность шеренги зависит от входного домена . Один из предлагаемых авторами способов – выбирать как можно большим, например для увеличения количества OPE функций преобразования и, следовательно, повышения безопасности. Количество таких функций определяется сочетанием и максимально при . Если взять приближение

, (21)

то при . В следующей работе Alexandra Boldyreva «Order-Preserving Encryption Revisited: Improved Security Analysis and Alternative Solutions» [44] после углубленного анализа схемы OPE приходит к мнению о рекомендуемой мощности, достаточной для промышленного применения.

### ****Гомоморфное шифрование****

**Homomorphic Encryption**. **HE** – шифрование, позволяющее производить определенные действия с зашифрованным текстом и получать зашифрованный результат, который соответствует результату операций, выполняемых с открытым текстом.

Разные криптографические системы поддерживают различные гомоморфные свойства. Например, криптосистема Пэйе поддерживает следующее свойство

(22)

где – функция шифрования, а – открытые тексты. Свойство позволяет выполнить операцию сложения по модулю на сервере. При этом необходимо знание только шифротекстов . Клиент же впоследствии может получить значение через расшифровку преобразования .

### ****Криптосистема Пэйе**** [57]

**Криптосистема Пэйе** — вероятностная криптосистема с открытым ключом, изобретенная французским криптографом Паскалем Пэйе (англ. Pascal Paillier) в 1999 году. Аналогично криптосистемам RSA, Гольдвассера-Микали и Рабина, криптосистема Пэйе основана на сложности факторизации составного числа, являющегося произведением двух простых чисел.

**Генерация ключей.** Генерация ключей связана с нахождением больших простых чисел. Приведем алгоритм генерации.

1. Выбираются два больших простых числа и , такие, что

(23)

1. Вычисляется и .
2. Выбирается случайное число , такое что .
3. Вычисляется

, (24)

где , а деление – это вычисление обратного по модулю.

1. Открытым ключом является пара .
2. Закрытым ключом является пара .

От величины простых чисел и зависит криптостойкость системы, поэтому их выбирают большими.

**Алгоритм шифрования.** Предположим, что нужно зашифровать открытый текст . Приведем алгоритм шифрования.

1. Выбирается случайное число .
2. Вычисляется шифротекст

(25)

Алгоритм является вероятностным, поскольку в процессе вычисления шифротекста участвует случайная величина.

**Алгоритм расшифрования.** На входе имеем шифротекст .

1. Вычисляем исходное сообщение как

(26)

### Двухфакторная аутентификация EMV [58]

Надёжная аутентификация является одним из ключевых элементов системы информационной безопасности. Использование смарт-карт позволяет сократить расходы на внедрение двухфакторной аутентификации.

### Смарт-карты EMV [59]

**EMV** – международный стандарт банковских операций по картам с чипом, разработанный совместными усилиями компаний Europay, MasterCard и Visa. Стандарт определяет физическое и информационное воздействие между банковской картой и платежным терминалом. Открытость стандарта позволяет использовать карты и во внебанковских приложениях.

Каждая карта является интеллектуальным устройством, обладающим памятью и собственным вычислителем, что позволяет обеспечить повышенную безопасность по сравнению с традиционными банковскими картами. Карта хранит уникальный секретный ключ, который записывается в защищенную память на стадии изготовления, и его невозможно извлечь без нарушения целостности самого чипа.

В качестве считывателя можно использовать любой, совместимый с интерфейсом ISO 7816. В данной работе используется считыватель ACR38 фирмы ACS, рекомендованный для считывания УЭК (Универсальная Электронная Карта).



Рисунок 17 – Считыватель смарт-карт УЭК

### Протокол ISO/IEC 7816-3 [60]

Стандарт устанавливает характеристики протокола для обмена информацией с картой. Любая часть информационного обмена записывается в пакет APDU (Application Protocol Data Unit), это касается как запросов, так и ответов от карты.

EMV устанавливает следующий базовый набор команд:

Таблица 4 – Команды протокола EMV

|  |  |
| --- | --- |
| **Команда** | **Описание** |
| application block | Блокирование приложения |
| application unblock | Разблокирование приложения |
| card block | Блокировка карты |
| external authenticate | Внешняя аутентификация |
| generate application cryptogram | Расчёт криптограммы |
| get data | Получение данных |
| get processing options | Получение опций транзакции |
| internal authenticate | Внутренняя аутентификация |
| PIN change / unblock | Смена пин-кода |
| read record | Чтение записи файловой системы |
| select | Выбор текущего приложения |
| verify | Проверка |

Пакет запроса APDU включает следующие поля:

Таблица 5 – Пакет запроса APDU

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Header** | | | | **Trailer** | | |
| CLA | INS | P1 | P2 | Lc | Data | Le |

Приведем также формат пакета ответа APDU:

Таблица 6 – Пакет ответа APDU

|  |  |  |
| --- | --- | --- |
| **Body** | **Trailer** | |
| Data | SW1 | SW2 |

Опишем поля пакета:

Таблица 7 – Описание полей пакета APDU

|  |  |
| --- | --- |
| **Поле** | **Описание** |
| CLA | Байт класса протокола, фиксированное значение *0x00* |
| INS | Байт идентификатора инструкции, например  *0xA4* – команда select  *0xB2* – команда read record |
| P1 | Байт первого параметра инструкции |
| P2 | Байт второго параметра инструкции |
| Lc | Размер поля Data в байтах |
| Data | Байты данных |
| Le | Ожидаемый размер ответа на запрос |
| SW1 | Первый байт статуса ответа |
| SW2 | Второй байт статуса ответа |

Протокол не устанавливает стандарты сериализации данных.

### Структура смарт-карты EMV

Тип и формат карты однозначно идентифицируются специальной публичной областью памяти ATR (Answer To Reset). С помощью этой области можно определить историю изменения данных карты, производителя, эмитента, платежную систему, регион использования и прочие данные.

Каждая карта содержит защищенную область памяти и публичную.

В защищенную область на стадии изготовления записывается секретный ключ, и его невозможно извлечь без разрушения целостности самого чипа. Секретный ключ используется для подписи пакетов в алгоритмах аутентификации SDA и DDA.

В публичной области памяти хранятся данные магнитной ленты, описатели хранимых структур TLA (Tag Length Value), служебная информация о держателе карты и прочие. Для хранения данных используется внутренняя файловая система, основанная на записях статичной длины.

На карте также хранятся приложения со своими внутренними данными, чаще всего это платежные программы от MasterCard или Visa, они идентифицируются числом AID (Application Id).

Для считывания публичной области нужно сначала выбрать приложение, и далее запросить все его публичные записи в файловой системе.

### Алгоритмы аутентификации SDA и DDA [61]

**SDA** (Static Data Authentication) – алгоритм аутентификации по статичным данным. Он используется для проверки, что данные, записанные на карте, действительны и никем не изменялись.

SDA основан на вычислении цифровой подписи с помощью асимметричной криптографии. С помощью закрытого ключа эмитент подписывает важные данные на карте, а публичным ключом терминал проверяет эту подпись. Закрытый ключ эмитента сохраняется в секрете.

После проверки SDA терминал удостоверяется, что данные на карте валидны и не изменялись. Однако, злоумышленник мог скопировать такие данные, поэтому следующим шагом используется DDA.

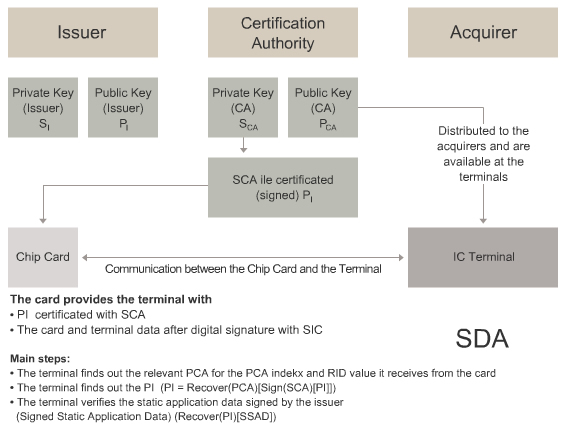


Рисунок 18 – Схема алгоритма SDA

**DDA** (Dynamic Data Authentication) – алгоритм аутентификации по динамическим данным. Он используется для проверки того, что карта является уникальной и соответствует заявленной.

При выполнении алгоритма карта подписывает изменяющиеся сессионные данные с помощью своего закрытого ключа. Сессионные данные генерируются терминалом и действуют только то время, пока карта находится внутри, что позволяет противостоять атакам копирования.

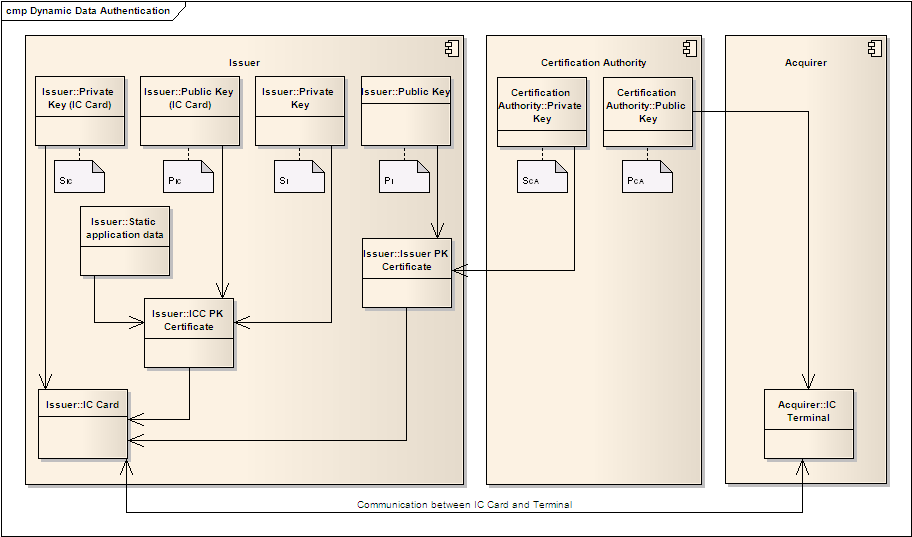


Рисунок 19 – Схема алгоритма DDA

### Упрощенный алгоритм аутентификации

Алгоритмы SDA и DDA являются надежными, но сложны в реализации, поскольку имеют различия для карт разных платежных систем.

Авторы предлагают упрощенный алгоритм аутентификации при помощи карты, в котором последняя используется только как носитель ключевого материала. Схема основана на использовании HMAC для подписи сессионных данных и используется совместно с парольной защитой.

**HMAC** (Hash-based Message Authentication Code) – алгоритм генерации хеш-кода аутентификации сообщений. Алгоритм используется для проверки подлинности информации, передаваемой между двумя сторонами.

В качестве закрытого ключа используется значение криптографической хеш-функции от всей публичной области памяти карты. Публичная область памяти уникальна для каждой карты.

Перед началом использования в базу данных сервера для определенного пользователя заносится закрытый ключ его карты.

Каждый сеанс сервер генерирует случайное число – сессионный ключ. Для прохождения аутентификации пользователю необходимо сгенерировать валидную HMAC подпись для этих данных и переслать обратно серверу.

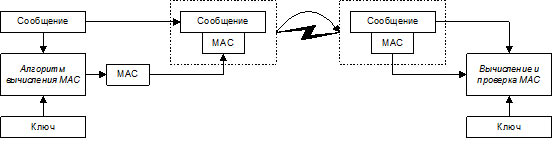


Рисунок 20 – Схема алгоритма HMAC

Если подпись, принятая от клиента, совпадает с подписью, сгенерированной сервером, то пользователь считается валидным.

ГЛАВА 3. РЕАЛИЗАЦИЯ СИСТЕМЫ

### ****Проект****

В качестве названия проекта было выбрано слово «**Cryptis**». Далее в тексте так будет именоваться облачная криптографическая NoSQL СУБД.

Результаты данной работы размещены по веб-адресу <http://cryptis.ru/> и <http://cryptis.ru/en/>. Исходный код доступен по адресу <https://github.com/pashikperm/cryptis>.

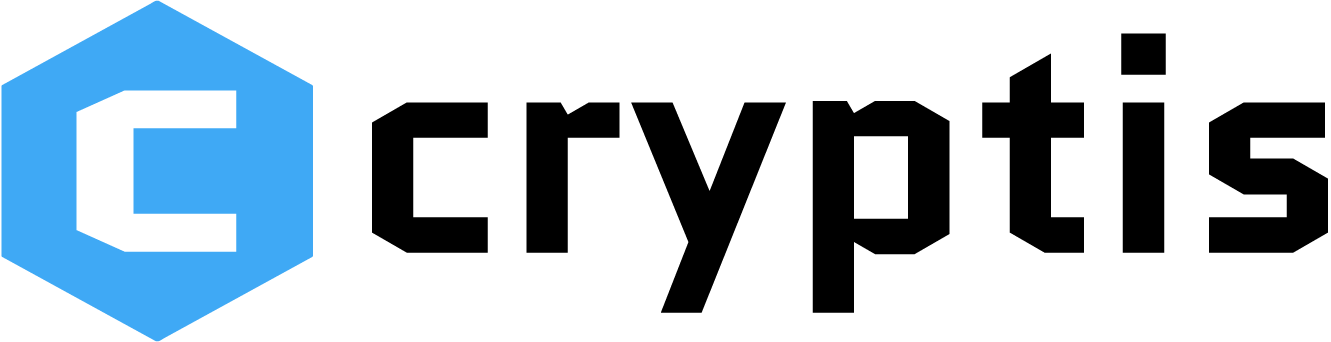


Рисунок 21 – Логотип проекта

### ****Используемые технологии****

**Платформа**. Система Cryptis используется в гетерогенных сетях, на клиентских ЭВМ и серверах, находящихся в доверенной зоне. По причине большого разнообразия современных операционных систем был выбран кросс-платформенный язык программирования Java. Среди его преимуществ по сравнению с прочими [62]:

1. Унифицированный доступ к различным СУБД. Java использует технологию JDBC – унифицированных драйверов, предназначенных для подключения и взаимодействия с СУБД.
2. Встроенные криптографические примитивы. В стандартной библиотеке реализован широкий спектр криптографических алгоритмов, существенно упрощающих и ускоряющих разработку защищенных приложений.
3. Кросс-платформенность. Java позволяет запускать приложение без перекомпиляции на основных распространённых системах, включая мобильные платформы.
4. Достаточная производительность. В последних версиях Java появился компилятор, на лету кодирующий программу в машинный код, JIT. JIT повысил производительность Java и на текущий момент код на Java лишь в 1,8 – 3.0 раза медленнее аналогичного кода на Си [63].
5. Бесплатность. Java – открытое программное обеспечение.

**Облачные вычисления**. В качестве облака выбрано Redis Cloud [22]. Это PaaS решение, позволяющее быстро создать Redis базу данных. Облако поддерживает все операции Redis, масштабируется в широких пределах, имеет удобную утилиту для настройки СУБД. Также поддерживаются репликация, архивация и восстановление по расписанию. Стоимость обслуживания зависит от количества хранимых данных – до 20 Мегабайт предлагаются по бесплатному тарифу, хранение 100 Мегабайт стоит 8$ в месяц.

В целях тестирования был зарегистрирован аккаунт с бесплатным тарифом: возможно создание только одной базы данных и максимум 10 подключенных клиентов единовременно:

Таблица 8 – Характеристики решения Redis Cloud

|  |  |
| --- | --- |
| **Характеристика** | **Значение** |
| Географическое расположение | Восточная Европа |
| Сетевой адрес | pub-redis-12800.eu-west-1-1.2.ec2.garantiadata.com |
| Порт подключения | 12800 |
| Персистентность | Включена |
| Доступная память БД | 25 Мегабайт |
| Версия Redis | 2.8.19 |

Однако Redis Cloud не подходит для тестирования гомоморфного шифрования, поскольку последнее требует модификации сервера СУБД. Поэтому дополнительно было использовано облачное решение от DigitalOcean [64], его характеристики описаны в таблице 9.

Таблица 9 – Характеристики решения DigitalOcean

|  |  |
| --- | --- |
| **Характеристика** | **Значение** |
| Географическое расположение | Амстердам |
| Сетевой адрес | 178.62.164.34 |
| Порт подключения | 12800 |
| Персистентность | Включена |
| Доступная память БД | 512 Мегабайт |
| Версия Redis | Модифицированная 2.8.19 |

Предоставленные ресурсы достаточны для проведения тестирования работы Cryptis в облаке.

**Сетевая безопасность**. Cryptis использует сеть для взаимодействия с СУБД, поэтому необходимо проверить безопасность передаваемых данных. В качестве программы-анализатора трафика выбрана Wireshark [65]. Wireshark имеет графический пользовательский интерфейс и широкие возможности по сортировке и фильтрации информации. Программа позволяет просматривать весь проходящий по сети трафик в режиме реального времени. Wireshark умеет разбирать различные сетевые протоколы, отображать значения передаваемых структур. Программа распространяется под свободной лицензией GNU GPL и бесплатна для использования.

**Версия СУБД**. Для тестирования на локальной ЭВМ используется последняя стабильная версия Redis 2.8.19 – 32-битная.

**Оболочка СУБД.** В качестве клиента выбран консольный redis-cli. Его преимущества заключаются в минималистичном интерфейсе и легковесности. Redis-cli поставляется вместе с сервером Redis.

**Клиент СУБД.** Для языка Java в качестве клиента для подключения к Redis часто используется проект Jedis [66]. Клиент единственный отмечен как рекомендуемый на официальном сайте Redis. Авторы Jedis преследовали цель максимального упрощения доступа к СУБД. Клиент совместим с версией протокола Redis 2.0, на текущий момент активно развивается. Актуальная стабильная версия Jedis 2.6.0 – используется в Cryptis.

**Библиотеки математики.** Для моделирования некоторых распределений используются библиотеки Apache «Commons Math» [67] 3.3.2 и «Statistical Distribution Library» [68] 0.9.1. В исходный код последней вносились некоторые модификации для увеличения точности моделирования. Для работы с длинными числами на сервере Redis для гомоморфного шифрования использовалась библиотека GNU GMP.

**Технология Redis Pipelining** [69]**.** Технология позволяет существенно ускорить выполнение запросов Redis. Сервер формирует очередь входных команд и выводит результат их работы не сразу, а через некоторое время. Cryptis поддерживает pipelining.

**Среда программирования.** Используется бесплатная версия IntelliJ IDEA: Community Version.

### ****Архитектура решения****

**Принцип работы.** Обычно организации физически хранят информацию на серверах внутри собственных помещений. Взаимодействие с СУБД клиентские приложения осуществляют через сеть.

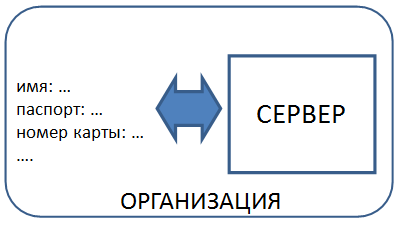


Рисунок 22 – Хранение информации в организациях

Cryptis рассматривается как прозрачная для клиента и сервера программа, применяющая криптографические средства. Последние используются для обеспечения безопасности хранения вовне организации, например, в облаке (Cryptis работает внутри организации). Специальные виды шифрования позволяют выполнять часть операций в облаке прямо над зашифрованными данными, другая часть эмулируется Cryptis.

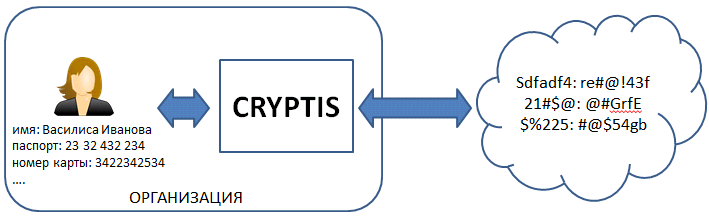


Рисунок 23 – Инфраструктура с использованием Cryptis

**Выполнение запросов в облаке.** Часть манипуляций с данными Redis выполняет в облаке самостоятельно без расшифровки данных. Метод возможен благодаря применению гомоморфных и сохраняющих порядок шифрований.

**Эмуляция команд Redis.** Некоторые команды невозможно выполнить на сервере, например, конкатенация строк требует шифрования, отличного от криптосистемы Пэйе. В случае невозможности выполнения команды на сервере, ее выполнение эмулируется Cryptis: зашифрованная информация передается по каналу связи, далее Cryptis расшифровывает ее, проводит операцию и отправляет обратно в зашифрованном виде. Режим эмуляции приводит к значительному снижению скорости обработки информации вследствие частых пересылок по сети.

**Прозрачность на уровне исходных кодов.** Для обеспечения прозрачности криптографических средств используется принцип прозрачности на уровне исходных кодов. Интерфейс для клиентских приложений выглядит точно так же, как и интерфейс клиента СУБД, например, Jedis.

**Надстройка Jedis.** Использование прозрачности на уровне исходных кодов имеет ряд существенных недостатков. Во-первых, требует перекомпиляции всех пользовательских приложений. Во-вторых, обновление криптографического ПО также требует перекомпиляции.

В качестве преимуществ можно указать высокую производительность из-за отсутствия задержек сети при передаче информации до криптографического прокси, а затем до СУБД.

Сохранение прикладного интерфейса Jedis позволяет избежать необходимости переписывать клиентское ПО, требуется только перекомпиляция. Стоит учесть, что Jedis – рекомендуемый сообществом Redis клиент для взаимодействия с БД, т.е. он широко распространен, поэтому интеграция Cryptis в большую часть ПО будет малозатратна.

При этом сохраняется прозрачность для прикладного программиста, т.к. прикладной интерфейс Jedis не изменяется.

### ****Описание реализации****

**Общее описание и использование.** Был реализован Java-пакет «cryptis», который может быть подключен к клиентским приложениям как в исходном виде, так и в виде библиотеки jar. После подключения к проекту, становится доступным класс «Cryptis». Для использования программы в прикладном коде, необходимо заменить все упоминания «Jedis» на «Сryptis».

После подключения Cryptis осуществляет взаимодействие с СУБД в зашифрованном виде. В зависимости от структур данных Cryptis использует различные виды шифрования: симметричный AES, одну из схем OPE, или криптосистему Пэйе.

**Открытая информация.** Часть информации, связанной с хранимыми данными, остается в открытом виде. В основном, это справочная информация, ее открытый вид обусловлен внутренними ограничениями Redis. В открытом виде передается информация о количестве элементов структур данных, флаги наличия элементов, коды результатов выполнения операций. Схемы OPE по своей природе раскрывают порядок следования элементов коллекций. При выборке элементов множества, параметр количества элементов передается открыто.

Для повышения общей безопасности схемы, рекомендуется использовать защищенное соединение (SSL, IPSec, PPTP) между Cryptis и СУБД.

**Ограничения тривиальной схемы OPE (DummyOPE).** DummyOPE показывает высокую производительность шифрования и дешифрования, однако множество допустимых значений ограничено. В полученной реализации поддерживаются целые числа из множества , где ограничено сверху объемом доступной памяти и пределом длины массивов в Java. Кроме того, используемая версия Redis имеет 32-битное исполнение, поэтому образ открытого текста должен вмещаться в знаковое 32-битное целое.

**Ограничения схемы OPES.** В работе используется модифицированный генератор HGD из библиотеки «Statistical Distribution Library». Модификация позволила увеличить диапазон допустимых значений вычисления HGD, стало возможным генерация распределения для 32-битных входных параметров. Как было указано выше, авторы схемы OPES рекомендуют использовать как можно большее множество образов элементов при малом количестве открытых текстов. Длина параметров схемы должна быть кратна размеру байта. Кроме того, размеры множеств открытых и закрытых текстов требуется указать до первой процедуры шифрования. Размеры не меняются в процессе работы.

**Ограничение атомарности команд.** В режиме эмуляции команд теряется их атомарность. Для решения проблемы транзакций существует два подхода:

1. Использование общего посредника для многих приложений.
2. Искусственные блокировки во время эмуляции для встроенного посредника.

Недостатком первого подхода является значительное снижение производительности из-за необходимости выделенного посредника и сетевых задержек.

Недостатком второго подхода является необходимость модификации сервера Redis для поддержки блокировок.

Cryptis потенциально может гарантировать атомарность операций, но для этого на одну родительскую СУБД должен работать строго один экземпляр Cryptis, и все операции над данными должны проходить через него. В случае нескольких экземпляров Cryptis сохранение атомарности представляется технически сложным.

### ****Команды Redis и эмуляция их выполнения****

**Классификация**. В Redis команды разделяются **по типам данных** и воздействию на них [70]. Приведем классификацию по типам данных (в нее также входят сервисные команды, изменяющие состояние сервера).

Таблица 10 – Классификация команд Redis по типам данных

|  |  |
| --- | --- |
| Keys | HyperLogLog |
| Strings | Pub/Sub |
| Hashes | Transactions |
| Lists | Scripting |
| Sets | Connection |
| Sorted sets | Server |

Приведем также классификацию **по характеру действия**:

1. **Команды чтения (R)**. Команды, считывающие данные из БД. Обычно, они выполняются с помощью двух сетевых пересылок – командной и результатной. Например, команда *get.*
2. **Команды записи (W)**. Команды, записывающие данные в БД. Также выполняются с помощью двух сетевых посылок – запрос на изменение и результат. Например, команда *set.*
3. **Команды модификации (M)**. Команды, которые меняют значение в зависимости от исходного. Например, увеличение значения на единицу – *incr*. Для незашифрованной БД она выполняется за две пересылки – команды и результата. Для зашифрованной БД с AES шифрованием – с помощью команды чтения и записи, т.е. 4 пересылки. Если же поле зашифровано гомоморфным шифрованием, то увеличение осуществляется за 2 пересылки, что существенно ускоряет время выполнения команды.
4. **Прочие (O).** Сервисные команды и команды вызова процедур.

**Часто используемые команды.** Оптимизации подвергаются в первую очередь на часто используемые и сильно нагруженные команды. Пример таких команд был опубликован в блоге разработчиков Instagram [71], приведем их выдержку.

Таблица 11 – Используемые команды Redis в Instagram

|  |  |
| --- | --- |
| **Часто используемые** | **Наиболее нагруженные** |
| SISMEMBER (R) | SISMEMBER (R) |
| HGET (R) | HGET (R) |
| HINCRBY (M) | HINCRBY (M) |
| SMEMBERS (R) | SMEMBERS (R) |
| MULTI (O) | MULTI (O) |
| EXEC (O) | SADD (W) |
| LPUSH (W) | SREM (W) |
| EXPIRE (O) | EXEC (O) |

**Эмуляция команд.** Любая команда модификации может быть эмулирована клиентом. Для этого потребуется минимум 4 пересылки данных, однако с использованием гомоморфного и сохраняющего порядок шифрований, количество пересылок сокращается в два раза.

Приведем псевдокод эмуляции команды *append*, которая добавляет значение *value* в конец строки с индексом *key*.

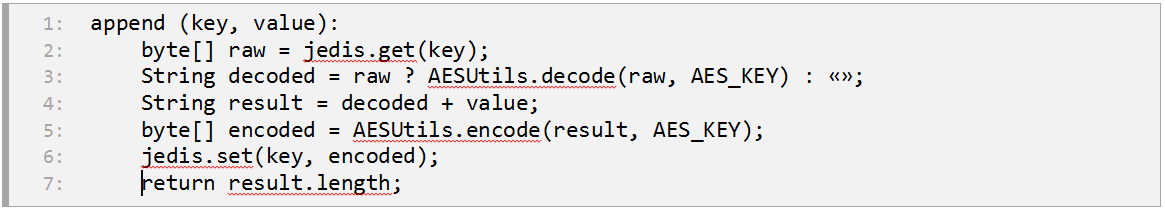


Рисунок 24 – Алгоритм эмуляции команды append

**Набор поддерживаемых команд Redis.** Количество доступных команд Redis 2.0 составляет более 120 [69], для прототипа нецелесообразно реализовывать их все. Был выбран список команд, позволяющий сравнить производительность с CryptDB. При выборе принимались во внимание тесты производительности CryptDB, в которых авторы стремились выяснить показатели работы в различных сферах практического применения [53]. Поддерживаемые команды приведены в таблице 12.

Таблица 12 – Поддерживаемые команды Redis

|  |  |
| --- | --- |
| **Команда** | **Описание** |
| auth | Аутентификация пользователей, необходима для работы в облаке |
| quit | Процедура стандартного отключения пользователя |
| set | Ассоциирование значения с ключом |
| get | Получение значения по ключу |
| hset | Установка значения в структуре «хеш» по ключу |
| hmset | Занесение многих значений одновременно в структуру «хеш» |
| hgetall | Получение всех полей (ключей и значений) структуры «хеш» |
| incr, decr | Инкремент и декремент значения |
| incrby, decrby | Инкремент и декремент значения на определенную величину |
| zadd | Добавление значения в структуру «упорядоченное множество» |
| zscore | Получение ключа сортировки элемента из упорядоченного множества |
| zrangebyscore | Возврат значений, укладывающихся в границу из упорядоченного множества |
| zrevrange | Получение нескольких первых элементов из упорядоченного множества |
| zincrby | Увеличение значения в структуре «упорядоченное множество» |
| del | Удаление ключа |
| zrem | Удаление ключа из упорядоченного множества |
| exists | Проверка существования ключа |

Часть из этих команд поддерживает Pipelining – технологию одновременного выполнения нескольких запросов, существенно увеличивающую производительность интерфейса Redis.

### ****Реализация гомоморфного свойства в сервере Redis****

Криптосистема Пэйе обладает следующим гомоморфным свойством:

(27)

Таким образом, для сложения открытых текстов сервер должен произвести умножение закрытых текстов.

На момент написания работы, актуальная версия сервера Redis 2.8.19 не может осуществлять операцию умножения. Redis поддерживает операции сложения, однако накладывает существенные ограничения на разрядность чисел – не более 32 или 64 бит для Linux версии и 32 бит для Windows.

Модификация сервера подразумевает две части:

1. Увеличение разрядности поддерживаемых чисел.
2. Операция умножения данного из БД на число.

В результате реализации были созданы дополнительные команды *multby* и *hmultby*.

**Команда multby.** Команда принимает на вход два аргумента – идентификатор, по которому находится первый операнд в БД, и непосредственно второй операнд умножения. Результат выполнения записывается в БД и возвращается в качестве результата команды.

**Команда hmultby.** Команда принимает на вход три аргумента. Первые два – идентификатор ассоциативного массива и элемента-операнда внутри него, третий – операнд, на который умножается первый.

**Длинная арифметика.** Сервер Redis написан на языке C, поэтому для реализации длинного умножения использовалась библиотека, также реализованная на C.

В качестве библиотеки авторам подошла GNU GMP, в ее преимущества входят:

1. Свободная лицензия GNU.
2. Компиляция для множества платформ.
3. Оптимизация бинарного кода для целевых платформ.
4. Протестированная корректность работы.
5. Развитое сообщество.

Для реализации длинного умножения использовалось семейство функций **mpz**.

### ****Реализация двухфакторной аутентификации EMV****

**Механизм аутентификации.** Механизм действия аутентификации устроен так, чтобы обеспечить максимальную совместимость с существующим протоколом Redis RESP, т.е. используются те же команды, но у них появляется дополнительный функционал.

В команде *AUTH* Redis требует указания общего для всех пользователей пароля. Модифицированный сервер Redis ожидает строку, состоящую из имени пользователя и пароля, разделенными двоеточием.

**Двухфакторная аутентификация.** Двухфакторная аутентификация осуществляется в два шага. Первый шаг – обычная аутентификация пользователя. В качестве кода возврата после аутентификации клиент получает сессионный ключ.

Второй шаг - клиентское ПО должно зашифровать его закрытым ключом и отправить серверу результат командой AUTH. Если клиент верно вычислил преобразование для сессионного ключа, то сервер считает, что он прошел аутентификацию успешно. В качестве хранилища закрытого ключа выступает банковская карта стандарта EMV.

Для осуществления двухфакторной аутентификации необходимо изменение клиентского ПО для того, чтобы оно поддерживало два этапа аутентификации и считывание закрытого ключа с банковской карты.

### ****Производительность Cryptis****

**Среда тестирования.** Для тестирования используется ЭВМ с установленной 64-разрядной операционной системой Windows 7. Процессор Core i3 M330 – два ядра 2,13 Гигагерц, используется технология HyperThreading. Объем оперативной памяти составляет 8 Гигабайт, постоянное запоминающее устройство выполнено по технологии SSD.

На ЭВМ установлена виртуальная машина с Linux Mint 15, ей отведено 2 процессора и 2 Гигабайта оперативной памяти. Виртуальная машина и операционная система хост-машины установлены на SSD. В качестве родительских СУБД используется Redis 2.8.19 и MySQL Community Edition 5.6.22.

**Измерение.** Производительность измеряется в количестве обрабатываемых запросов в секунду. На заполненной базе данных измеряется время вычисления множества запросов, из которого высчитывается производительность. Запросом считается черный ящик, на входе которого находятся аргументы, а на выходе – информация из базы данных. Запросы считаются эквивалентными, если при одинаковых базах данных и аргументах, возвращают одинаковые данные.

Средняя производительность считается как арифметическое среднее производительности всех видов запросов, измеряется в запросах/сек. Будем считать, что продукт эффективнее, если средняя производительность выше при равных условиях.

Сравниваются различные по системе команд СУБД – SQL и NoSQL, поэтому необходима трансляция DML с сохранением эквивалентности запросов. Трансляция осуществляется вручную.

**Данные.** Используется модель данных, представляющая клиента магазина. Хранятся его идентификатор, ФИО, номер паспорта, адрес, телефон, пароль, электронная почта, номер социального страхования и остаток на депозите. Данные каждого пользователя генерируются и вставляются автоматически как одна из частей замера производительности. Количество пользователей выбрано равным 25000.

**Запросы.** Для замеров были отобраны восемь часто используемых запросов к базам данных [53], использующихся для замеров производительности CryptDB. На заполненной базе данных измеряется скорость выполнения 3000 запросов со случайными аргументами. Запросы приведены в таблице 13.

Таблица 13 – Производимые запросы

|  |  |
| --- | --- |
| **Запрос** | **Описание** |
| insert | Последовательная вставка пользователей в БД по одному |
| insertBatch | Вставка пользователей в БД, используя механизмы пакетного выполнения, встроенные в JDBC или Jedis (Cryptis) |
| selectEqual | Получение информации о пользователе по его идентификатору |
| selectRange | Получение идентификаторов пользователей, чей депозит лежит в определенных пределах |
| selectTop | Получение идентификаторов пользователей с наибольшими депозитами |
| updateSet | Обновление имени пользователя – устанавливается значение |
| updateSum | Обновление депозита пользователя – к текущему добавляется случайная величина |
| delete | Удаление пользователя по идентификатору |

**Индексы.** В СУБД часто используются индексы для ускорения выполнения запросов. Более того, в Redis сильно затруднено выполнение различных операций без создания дополнительных структур. Для запросов selectRange, selectTop в Redis была создана структура упорядоченное множество для поля депозита. Затраты на поддержание структуры учитывались в запросах вставки и удаления.

Для того чтобы MySQL находилась в тех же условиях, в СУБД был создан индекс по полю депозита. Это дало существенный эффект ускорения – по результатам замеров скорость с 50-60 запросов/сек увеличилась до 1500-1600 запросов/сек.

**Производительность родительских СУБД.** Приведем результаты замеров производительности родительских СУБД без включения криптографических (см. рисунок 25). В диаграмме не отражены результаты для операции insertBatch – для Redis они слишком большие, что сбивает масштабирование диаграммы. Операция insertBatch в Redis выполняется со скоростью 30712 запросов/сек, в MySQL – 561 запросов/сек. Без учета insertBatch Redis в среднем в 2,03 раза быстрее MySQL на выбранных данных и запросах. С увеличением количества данных, хранимых в базе данных, отрыв будет возрастать – это особенность технологии NoSQL.

Отдельно приведем результаты замеров для Redis в облаке (рисунок 26). Два выбранных облака расположены удаленно и были выбраны бесплатные тарифные планы, поэтому они существенно уступают в производительности локальным решениям, поэтому параметры замеров были уменьшены в 10 раз: 2500 тысячи пользователей и 300 запросов.

Рисунок 25 – Производительность родительских СУБД

Рисунок 26 – Производительность Redis Cloud и DigitalOcean

**Производительность Cryptis.** Приведем результаты замеров производительности Cryptis по сравнению с Redis (рисунок 27). При этом используются два режима тестирования – с выбранным шифрованием DummyOPE или OPES. В графике не отражены результаты для операции insertBatch – для Redis – 30712 запросов/сек, Redis DummyOPE – 22312 з/сек, Redis OPES – 3151 запросов/сек. В среднем падение производительности для Redis Dummy составляет 11%, для Redis OPES – 26%.

Сильное падение производительности порядковых запросов приводит к выводу о том, что схема OPES требует оптимизации, например, кеширования результатов вычисления случайной ленты или всей функции целиком.

Рисунок 27 – Производительность Cryptis

**Производительность Cryptis HE.** Приведем результаты замера производительности updateSum при включении криптосистемы Пэйе (рисунок 28). Производительность родительской СУБД составляет 1928 запросов/сек. Производительность Cryptis OPES составляет 621 запрос/сек (эмуляция), а Cryptis HE позволяет получить приемлемые 1337 запросов/сек за счет меньшего количества пересылок по сети. Итого для схемы Cryptis HE падение производительности на операции updateSum составляет 31%.

Рисунок 28 – Производительность Cryptis HE

**Производительность Cryptis в облаке.** Для облачной схемы задержки сети и невысокая производительность родительских БД приводят к инвариантности результатов при включении и отключении шифрования. Исключение составляют эмулируемые операции – скорость updateSum упала с 7 запросов/с без эмуляции до 3 запросов/с (57%) с ней для Redis Cloud. В DigitalOcean, благодаря применению Cryptis HE, производительность упала с 10 запросов/с до 3 запросов/с (30%). Таким образом, применение гомоморфного шифрования значительно сокращает потери производительности.

**Производительность CryptDB.** Приведем замеры производительности CryptDB (рисунок 29). Общее снижение производительности составляет 18%. Показатель говорит о том, что существуют пути оптимизации используемой в Cryptis схемы OPES.

Рисунок 29 – Производительность CryptDB

**Производительность Cryptis и CryptDB.** Объединим полученные результаты в диаграмму (рисунок 30). Данные insertBatch не вносятся из-за их большой абсолютной величины, их значения см. выше.

Рисунок 30 – Производительность CryptDB и Cryptis

В среднем Cryptis быстрее CryptDB в 1,7-2,0 раз. При этом схема OPE в Cryptis реализована удачнее – у нее менее жесткие ограничения на диапазон значений и количество используемой памяти.

### ****Безопасность Cryptis****

**Отчет Wireshark.** Использование Wireshark осуществляем в операционной системе Windows 7. Кросс-платформенность Cryptis позволяет запускать его в том числе и в этой операционной системе. Подключение Cryptis произведем к облачной базе данных. Отчет Wireshark свидетельствует о том, что данные передаются в зашифрованном виде – на рисунке представлены команды zcore, zadd, del, zrem – последние две используют pipelining.

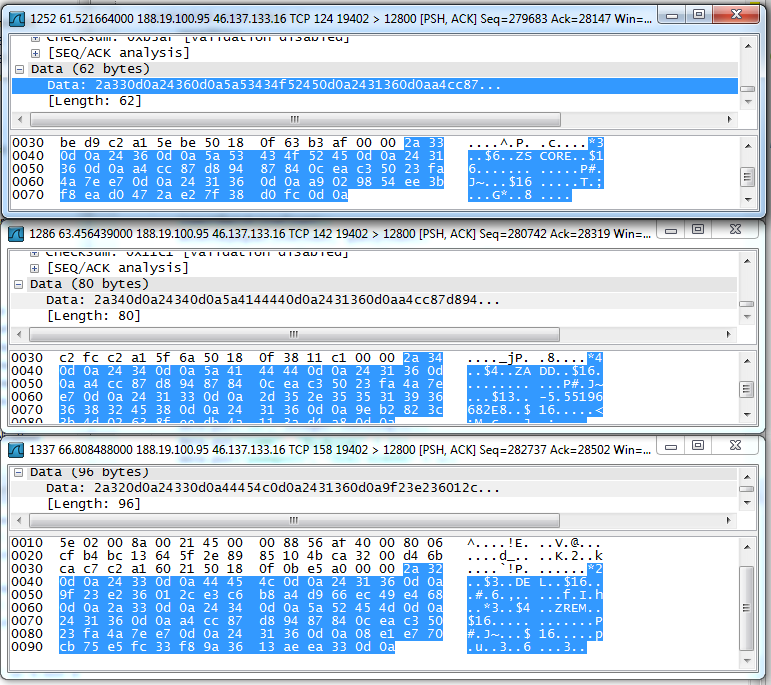


Рисунок 31 – Фрагмент отчета Wireshark

**Дамп базы данных.** Дамп базы данных, полученный redis-cli, свидетельствует о том, что все данные, отправленные во время тестовых запусков, хранятся в зашифрованном виде.

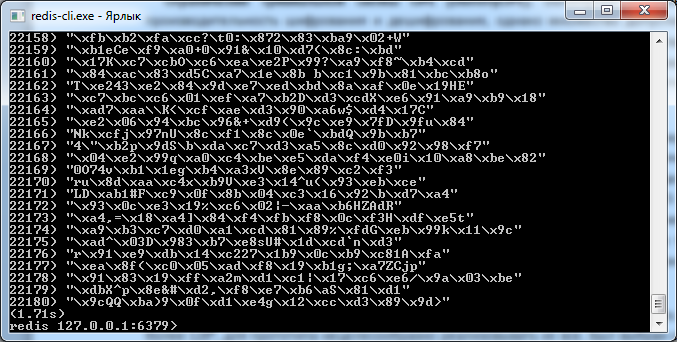


Рисунок 32 – Фрагмент дампа базы данных

**Проверка двухфакторной аутентификации EMV.** Длятестирования был применен модифицированный клиент, он считывает ключевой материал с карты (например, в них содержится фамилия держателя) и шифрует им сессионный ключ. В качестве считывателя был использован ACS УЭК, предназначенный для считывания универсальной электронной карты.

В ходе тестирования сеансы установления соединения происходили успешно.



Рисунок 33 – Считыватель ACS с банковской картой MasterCard

### ****Развитие проекта Cryptis****

**Общий вектор.** Стоит стремиться к промышленному решению: улучшение документации, упрощение интеграции.

**Модель безопасности.** Возможно создание модели безопасности, основанной на разделении доступа к данным по ключу. Модель можно реализовать с помощью регулярных выражений. Ключевая идея – создание правил вида «пользователь может читать ключи “pricing:\*”, но не может читать “users:\*”».

**Расширение списка поддерживаемых команд.** Количество команд, реализованное на данный момент, крайне мало для практического применения. Необходима поддержка большего разнообразия команд.

**Оптимизация OPE.** Результаты замеров производительности CryptDB показали, что функция OPE имеет пространство для оптимизаций.

**Интеграция других гомоморфных шифрований.** Добавление других гомоморфных шифрований позволит выполнять различные арифметические операции над числами в облаке.

ЗАКЛЮЧЕНИЕ

Все поставленные задачи решены:

1. Изучены существующие технологии и готовые решения.
2. Изучены особенности работы NoSQL баз данных и их размещения в облачных сервисах.
3. Выбрана архитектура Cryptis и родительская NoSQL база данных Redis.
4. Выбраны криптографические примитивы AES, OPES и криптосистема Пэйе (HE) для защиты данных.
5. Выбраны средства реализации прототипа – язык Java.
6. Прототип Cryptis реализован.
7. Прототип оттестирован, проведены замеры производительности и подтверждена безопасность.

В результате выполнения дипломной работы был создан прототип криптографической NoSQL базы данных – Cryptis. Были успешно проведены испытания двухфакторной аутентификации EMV.

Благодаря использованию родительской СУБД с NoSQL технологией, прототип показал более высокую эффективность выполнения запросов по сравнению с CryptDB.

В результате замеров производительности получили, что в среднем падение производительности по сравнению с Redis для Redis DummyOPE составляет 11%, для Redis OPES – 26%, для Redis HE – 31%. В среднем Cryptis быстрее CryptDB в 1,7-2,0 раз в протестированных ситуациях.

# СПИСОК ЛИТЕРАТУРЫ

1. **Application Security, Inc.** Доклад "Безопасность баз данных". [В Интернете] http://www.slideshare.net/ngsec/application-security-14372318.

2. **Software Application Solutions.** Доклад «Protecting Sensitive Data Using Encryption and Key Management». [В Интернете] https://www.owasp.org/images/c/c1/Database\_Encryption.ppt.

3. **Racula Ada Popa, Catherine M.S. Redfiels, Nickolai Zeldovich, Hari Balakrishnan.** Статья «CryptDB: Protecting Confidentiality with Encrypted Query Processing». [В Интернете] http://people.csail.mit.edu/nickolai/papers/raluca-cryptdb.pdf.

4. **IBM.** Статья "Основы облачных вычислений". [В Интернете] http://www.ibm.com/developerworks/ru/library/cl-cloudintro/index.html.

5. —. Статья «Модели сервисов облачных вычислений: Часть 1. Инфраструктура как сервис». [В Интернете] http://www.ibm.com/developerworks/ru//library/cl-cloudservices1iaas/index.html.

6. —. Статья «Модели сервисов облачных вычислений: Часть 2. Платформа как сервис». [В Интернете] http://www.ibm.com/developerworks/ru//library/cl-cloudservices2paas/.

7. —. Статья «Модели сервисов облачных вычислений: Часть 3. Программное обеспечение как сервис». [В Интернете] http://www.ibm.com/developerworks/ru/library/cl-cloudservices3saas/.

8. Гражданский кодекс РФ, ст. 1260.

9. **Habrahabr.** Статья «Облачные базы данных: кто их делает и на что они способны». [В Интернете] http://habrahabr.ru/company/cloudsnn/blog/148483/.

10. **Martin Fowler, Pramod J. Sadalage.** *NoSQL Distilled.* б.м. : Addison-Wesley, 2012.

11. Официальный сайт NoSQL движения «NoSQL». [В Интернете] http://www.nosql-database.org/.

12. **Peschka, Jeremiah.** Статья «Top Five Reasons to User NoSQL». [В Интернете] http://facility9.com/2010/09/five-reasons-to-use-nosql/.

13. **Открытые системы.** Статья «NoSQL: назад в будущее». [В Интернете] http://www.osp.ru/os/2012/02/13012856/.

14. **Martin Fowler, Pramod Sadalage.** Доклад «The Future is Polyglot Persistence». [В Интернете] http://martinfowler.com/articles/nosql-intro.pdf.

15. Статья «Most Popular NoSQL Database According to Linkedln Skillsets». [В Интернете] http://architects.dzone.com/articles/graph-nosql-database-linkedin.

16. Официальный сайт языка UnQL. [В Интернете] http://www.unqlspec.org/display/UnQL/Home.

17. Официальный сайт MongoDB. [В Интернете] http://www.mongodb.org/.

18. **nixp.** Статья «MongoDB 1.0 GA – первый релиз новой СУБД». [В Интернете] http://www.nixp.ru/news/MongoDB-1-0-GA-первый-релиз-новой-СУБД.html.

19. Книга «The Little MongoDB Book». [В Интернете] http://www.mongodb.org/.

20. Статья «Архитектура MongoDB. Часть 1. Отличие от СУБД, обработка запросов в MongoDB». [В Интернете] http://www.hi-lo.ru/article/mongodb-architecture-1.

21. Официальный сайт «Redis». [В Интернете] http://redis.io/.

22. Официальный сайт проекта «Redis Cloud». [В Интернете] http://redis-cloud.com/.

23. **Seguin, Karl.** *«The little Redis book».*

24. **Блинков, Иван.** Статья «Redis: подробный обзор». [В Интернете] http://www.insight-it.ru/tekhnologii/subd/redis-podrobnyjj-obzor/.

25. Статья «VMware: the new Redis home». [В Интернете] http://oldblog.antirez.com/post/vmware-the-new-redis-home.html.

26. Официальный сайт «HBase». [В Интернете] http://hbase.apache.org/ .

27. **IBM.** Статья «What is HBase». [В Интернете] http://www-01.ibm.com/software/data/infosphere/hadoop/hbase/.

28. —. Статья «Ищем путь к семантическому Web при помощи HBase». [В Интернете] http://www.ibm.com/developerworks/ru/library/os-hbase/.

29. **Wikipedia.** Статья «MapReduce». [В Интернете] http://ru.wikipedia.org/wiki/MapReduce.

30. **Apache, Inc.** Документация «The Apache HBase Reference Guide», глава «9. Architecture». [В Интернете] http://hbase.apache.org/book.html.

31. **Durkee, Ralph.** *Доклад «Protecting Sensitive Information with Database Encryption».* 2008.

32. **Luc Bouganim, Yanli GUO.** Публикация «Database Encryption». [В Интернете] http://www-smis.inria.fr/~bouganim/Publis/BOUGA\_B6\_ENC\_CRYPT\_2009.pdf.

33. Статья «Зашифрованные файловые системы». [В Интернете] http://e-wars.net/2012/05/16/zashifrovannye-fajlovye-sistemy/.

34. **Oracle.** Документация «Transparent Data Encryption». [В Интернете] http://www.oracle.com/technetwork/database/options/advanced-security/index-099011.html.

35. **Microsoft.** Документация MsSQL «Прозрачное шифрование данных (TDE)». [В Интернете] http://msdn.microsoft.com/ru-ru/library/bb934049.aspx.

36. **RSA Security.** *Статья «Developing a Database Encryption Strategy», RSA Security.*

37. **Gentry, Craig.** Статья «Fully homomorphic encryption using ideal lattices». [В Интернете] http://www.cs.cmu.edu/~odonnell/hits09/gentry-homomorphic-encryption.pdf.

38. **Forbes.** Статья «An MIT Magic Trick: Computing On Encrypted Databases Without Ever Decrypting Them». [В Интернете] http://www.forbes.com/sites/andygreenberg/2011/12/19/an-mit-magic-trick-computing-on-encrypted-databases-without-ever-decrypting-them/.

39. **TechnologyReview.** Статья «Homomorphic Encryption». [В Интернете] http://www2.technologyreview.com/article/423683/homomorphic-encryption/.

40. **Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, Yirong Xu.** Статья «Order Preserving Encryption for Numeric Data». [В Интернете] http://dl.acm.org/citation.cfm?id=1007632.

41. **Boldyreva, Alexandra.** Статья «Order-Preserving Symmetric Encryption». [В Интернете] http://www.cc.gatech.edu/~aboldyre/papers/bclo.pdf.

42. **Xiao, Liangliang.** Статья «A Note for the Ideal Order-Preserving Encryption Object and Generalized Order-preserving encryption». [В Интернете] http://eprint.iacr.org/2012/350.pdf.

43. **MIT.** Статья «An Ideal-Security Protocol for Order-Preserving Encoding». [В Интернете] http://web.mit.edu/ralucap/www/.

44. **Boldyreva, Alexandra.** Статья «Order-Preserving Encryption Revisited: Improved Security Analysis and Alternate Solutions». [В Интернете] http://www.cc.gatech.edu/~aboldyre/papers/operev.pdf.

45. **Xiao, Liangliang.** Статья «Security Analysis for Order Preserving Encryption Schemes». [В Интернете] http://www.utdallas.edu/~xll052000/OPEproof-TR3.pdf.

46. **Wikipedia.** Статья «Симметричные криптосистемы». [В Интернете] http://ru.wikipedia.org/wiki/Симметричные\_криптосистемы.

47. **Шнафер, Брюс.** *Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си.* Москва : б.н., 2002.

48. **THG.** Статья «Intel Core i5 (Clarkdalte): анализ аппаратного ускорения шифрования AES». [В Интернете] http://www.thg.ru/cpu/aes\_clarkdale/index.html.

49. Статья "Контроль доступа". [В Интернете] http://ru.wikipedia.org/wiki/Контроль\_доступа\_(информатика).

50. *Официальный сайт EMV.* [В Интернете] http://www.emvco.com/.

51. *Статья "Fraud and EMV".* [В Интернете] http://www.gemalto.com/emv/fraud\_emv.html.

52. **MIT.** Официальный сайт «CryptDB». [В Интернете] http://css.csail.mit.edu/cryptdb/.

53. **Popa, Racula Ada.** Доклад «CryptDB: Confidentiality for Database Applications with Encrypted Query Processing». [В Интернете] http://people.csail.mit.edu/nickolai/papers/raluca-cryptdb.pdf.

54. **MIT.** Статья «CryptDB: A Practical Encrypted Relational DBMS». [В Интернете] http://css.csail.mit.edu/cryptdb/.

55. **Engadget.** Статья «MIT's got a way of using encrypted data without decrypting it, next stop, traveling without moving». [В Интернете] http://www.engadget.com/2011/12/23/mits-got-a-way-of-using-encrypted-data-without-decrypting-it-n/.

56. Стандарт FIPS-197 «Advanced Encryption System». [В Интернете] 2001 г. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

57. Paillier's Cryptosystem. *Centro Ricerche IPPARI.* [В Интернете] http://www.ippari.unict.it/~catalano/Corsi/Tesi-Cap3-Paillier.pdf.

58. Документация "EMV". [В Интернете] http://www.openscdp.org/scripts/tutorial/emv/index.html.

59. Статья "EMV". [В Интернете] http://ru.wikipedia.org/wiki/EMV.

60. Стандарт "ISO/IEC 7816-3:2006". [В Интернете] https://www.iso.org/obp/ui/#!iso:std:iso-iec:7816:-3:ed-3:v1:en.

61. Статья "EMV Tutorial". [В Интернете] http://www.openscdp.org/scripts/tutorial/emv/.

62. **Wikipedia.** Статья "Java". [В Интернете] https://ru.wikipedia.org/wiki/Java.

63. Статья «Ubuntu: Intel Q6600 quad-core Computer Language Benchmarks Game». [В Интернете] http://benchmarksgame.alioth.debian.org/u32q/which-programs-are-fastest.php?gcc=on&javasteady=on&java=on&csharp=on&javaxint=on&calc=chart.

64. Официальный сайт DigitalOcean. *Официальный сайт DigitalOcean.* [В Интернете] https://www.digitalocean.com/.

65. Официальный сайт «Wireshark». [В Интернете] http://www.wireshark.org/.

66. Официальный сайт «Jedis». [В Интернете] https://github.com/xetorthio/jedis.

67. Официальный сайт «Commons Math». [В Интернете] http://commons.apache.org/proper/commons-math/.

68. Официальный сайт «Statistical Distribution Library». [В Интернете] http://statdistlib.sourceforge.net/.

69. Документация "Redis". [В Интернете] http://redis.io/topics/pipelining.

70. Документация команд Redis. [В Интернете] http://redis.io/commands.

71. **Instagramm, Inc.** Redis-faina: a query analysis tool for Redis. [В Интернете] http://instagram-engineering.tumblr.com/post/23132009381/redis-faina-a-query-analysis-tool-for-redis.