Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение
высшего образования

«Московский физико-технический институт

(государственный университет)»

Факультет управления и прикладной математики

Кафедра теоретической и прикладной информатики

AWS API-СОВМЕСТИМОЕ ШИФРОВАНИЕ И УПРАВЛЕНИЕ КЛЮЧАМИ ДЛЯ OBJECT STORAGE

Выпускная квалификационная работа
(бакалаврская работа)

Направление подготовки: 03.03.01 Прикладные математика и физика

Выполнил:
студент 376 группы _____ Щербатов Кирилл Алексеевич

Научный руководитель:
к.ф.-м.н. _____ Коротаев Кирилл Сергеевич

Москва, 2017

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Данная работа является исследованием, относящимся к области прикладных задач хранения больших объемов данных.

Ввиду наблюдающихся в IT-индустрии интенсивных процессов облачной интеграции приложений, количество удаленно хранимой информации неизменно растет. Программно-определяемые распределенные файловые системы – современное инженерное решение, обладающее конкурентным потенциалом для удовлетворения рыночных потребностей в обеспечении свойств масштабируемости, отказоустойчивости и быстродействия хранилищ. Обостряющаяся с увеличением обслуживаемой информации общая проблема защиты конфиденциальных данных требует от разработчиков и интеграторов комплексного подхода к своему решению.

Её актуальность определяют современные отраслевые требования: политика подавляющего большинства дата-центров согласуется с клиентским запросом доступ к данным ограничить.

Многие теоретические аспекты построения защищенных систем рассматриваются в классической работе Э. Таненбаума и М. ван Стеена «Распределенные системы. Принципы и парадигмы» [1]. В труде производится обзор важных принципов устройства протоколов пользовательского взаимодействия, архитектурных особенностей программы, а также рассмотрены типичные векторы атак. Академическая общность материалов выходит за рамки наших реалий и не дает непосредственных ответов на определяемые рынком вопросы. С другой стороны, концептуальная документация от Amazon «Protecting Data Using Server-Side Encryption..» [2], [3] оставляет без внимания технические вопросы реализации, как раз требующие педантичного учета самых разных факторов. Наконец, прикладные вопросы криптографии хорошо рассмотрены в труде T.Denis и S. Johnson «Cryptography for Developers» [4], однако безотносительно рассматриваемого нами спектра задач.

Обобщение и адаптация существующих подходов для случая объектного хранилища значимы в условиях наблюдаемого распространения SDS[1].

ПОСТАНОВКА ЗАДАЧИ

Цель работы – комплексное решение проблем безопасности пользовательских данных в объектном хранилище. Здесь и далее под безопасностью мы будем подразумевать техническую возможность хранить информацию таким образом, чтобы даже при наличии единовременного физического доступа к дискам, нельзя было бы не только получить данные, но и собрать статистику, упрощающую эвристический подбор ключа.

В рамках дипломной работы подлежат выполнению следующие задачи:

1. Изучение документации Amazon S3, выявление сценариев доступа, определение корректных поведенческих реакций для разработки процедур, совместимых с ними
2. Разработка протоколов и реализация программных модулей логики Server Side Encryption with Client provided key (SSE-C) шифрования в составе OStorage[2]
3. Проведение Unit-тестирования прототипа
4. Построение модели защищенной системы управления ключами Key Management System на основе HashiGroup Vault

Практическим результатом дипломной работы является разработанный прототип, реализующий логику SSE-C в составе Acronis Storage.

[1] Software-defined storage, программно-определяемая распределенная файловая система
[2] Object Storage, объектное хранилище

Результаты дипломной работы изложены в пояснительной записке, состоящей из трех частей:

Первая часть посвящена описанию программно-определяемых распределенных файловых систем на примере хранилища с открытым исходным кодом Ceph и обзору архитектурных особенностей Acronis OStorage, имеющих значение для решаемой нами задачи.

Вторая часть содержит описание технических вопросов реализации шифрования пользовательских данных с помощью предоставленного ключа.

В третьей части характеризуются Key Management-системы и предложена модель KMS, построенная на основе защищенного key-value хранилища Valut.

1. РАСПРЕДЕЛЕННЫЕ ПРОГРАММНО-ОПРЕДЕЛЯЕМЫЕ ФАЙЛОВЫЕ СИСТЕМЫ

1.1 Описание проблемы

Требования, предъявляемые к системам хранения данных, невероятно возросли в последние несколько лет. Согласно последним отраслевым исследованиям [5], объем хранимой информации в крупных компаниях ежегодно возрастает на 40-60 процентов. Так по прогнозу IDC к 2020 году ожидается свыше 6.6 зеттабайт данных [6].

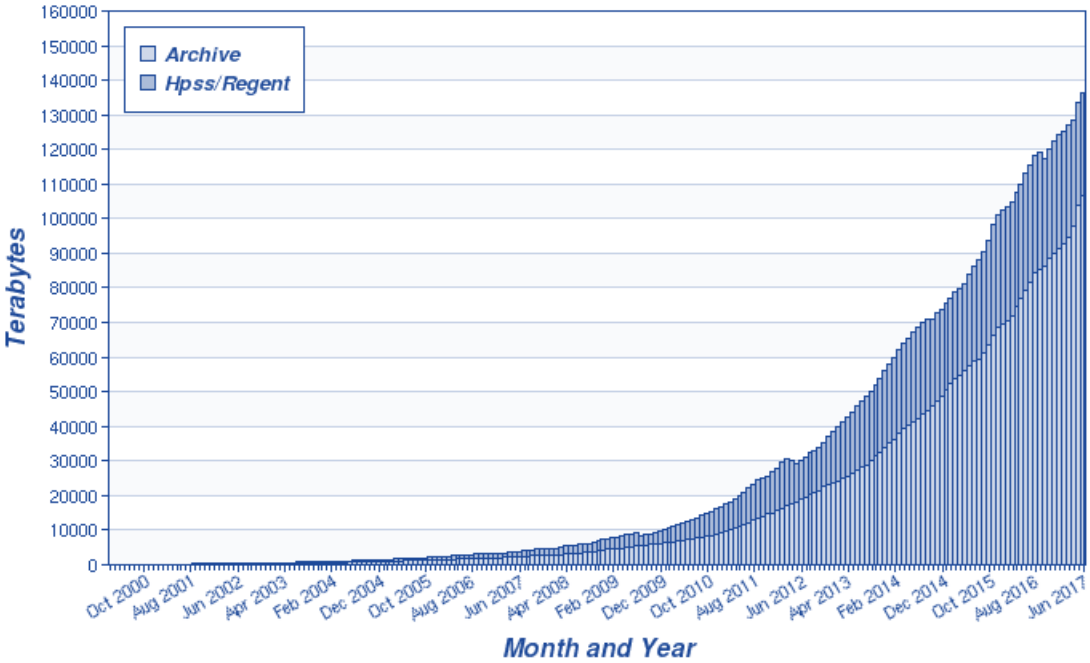

Рис. 1 - Cumulative Storage by Month and System [7]

Традиционные системы хранения обладают рядом недостатков, существенно ограничивающих перспективы их эксплуатации. Прежде всего, перестроение RAID[3], являющееся обязательной составляющей сервисного обслуживания, - многочасовой процесс, в протяжение которого оборудование недоступно. Более того, в случае одновременного выхода из строя нескольких

[3] Redundant Array of Independent Discs

дисков в одной RAID-группе, процесс восстановления не производится [8]. В то время как в прошлом вероятность такого события была пренебрежимо мала, с ростом объемов дисковых хранилищ, с ней приходится считаться.

С другой стороны, архитектурные особенности оборудования и драйверов[4], как правило, патентуются фирмой-производителем, в связи с чем аналогичные по функциональности компоненты зачастую оказываются несовместимыми. Необходимость поддержки быстро устаревающего железа ощутимо увеличивает стоимость таких решений.

Наконец, традиционные системы плохо подходят для одной из наиболее востребованных сегодня задач – виртуализации, требующей гипервизорного управления ресурсами.

Программно-определяемые хранилища (SDS) приведенными выше недостатками не обладают. Помимо ощутимого уменьшения стоимости владения инфраструктурой TCO[5], SDS необыкновенно гибки: такая архитектура способна обеспечить неограниченную масштабируемость и отвечающую задаче надежность хранения[6]. Этим и объясняется стремительный рост их популярности [9].

Acronis Storage является программным SDS-решением, позволяющим легко и быстро на основе доступного недорогого гетерогенного оборудования создать защищенное хранилище как SAN[7] или NAS[8]. Будучи оптимизированным для хранения больших объемов данных – как горячих, так и холодных – Acronis Storage обеспечивает избыточность информации (посредством репликации и кодов исправления ошибок), высокую доступность и самовосстановление[9] [10].

---

[4] field-programmable gate arrays (FPGAs), application-specific integrated circuits (ASICs)
[5] Total Costs of Ownership
[6] flexibility, scalability, reliability
[7] Serial Attached SCSI
[8] Network Attached Storage
[9] High availability, self-healing

В рамках этой главы мы произведем обзор архитектуры SDS на примере хранилища с открытым исходным кодом Ceph[10], после чего перейдем к рассмотрению объектного хранилища Acronis, с которым и будем работать далее.

1.2 Общее устройство Ceph

Экосистема Ceph может быть разделена на четыре основные части: клиенты хранилища, сервисы метаданных MDSs[11] (кэширующие и синхронизирующие распределенные метаданные), кластер объектного хранилища состоящий из OSDs[12] (содержащий как данные, так и метаданные объекта) и монитор кластера.

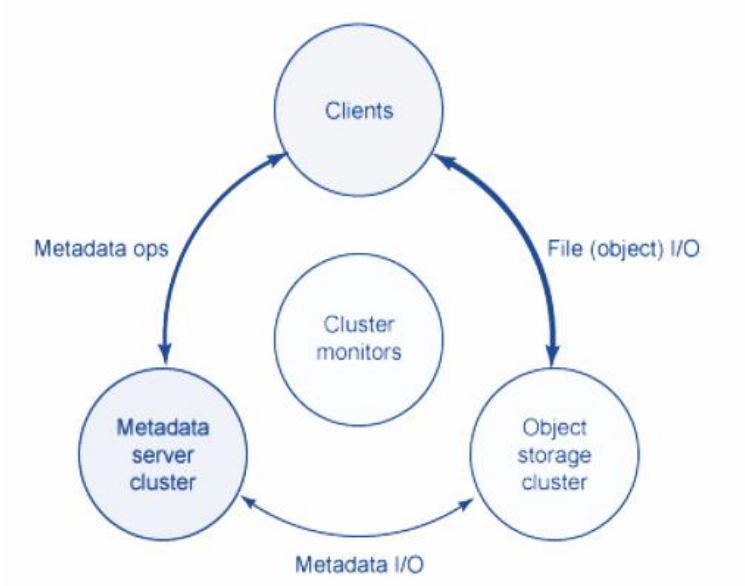

Рис. 2 - Сущности Ceph [16]

[10] http://ceph.com/
[11] MetaData Server
[12] Object Storage Daemon

На Рис. 2 проиллюстрирована общая схема взаимодействия: клиенты запрашивают информацию у MDSs, возвращающих, среди прочего, имя OSD, к которому следует обратиться с операциями ввода-вывода. Таким образом, ряд высокоуровневых POSIX[13]-функций (open, close и rename) осуществляются MDS-сервисами, в то время как ряд других (как read и write) – OSD-сервисами.

Данные, хранимые в OSD представляют собой совокупность блоков фиксированного размера (chunks). Эти блоки в автоматическом режиме реплицируются между всеми доступными серверами в кластере для обеспечения высокой доступности данных.

Так, в случае выхода из строя фиксированного числа OSD и MDS сохраняется возможность взаимодействовать с хранилищем; а динамическое перераспределение блоков после отказа узла, учитывающее текущую конфигурацию системы, обеспечит сохранение свойства отказоустойчивости и впредь (см. Рис. 3).

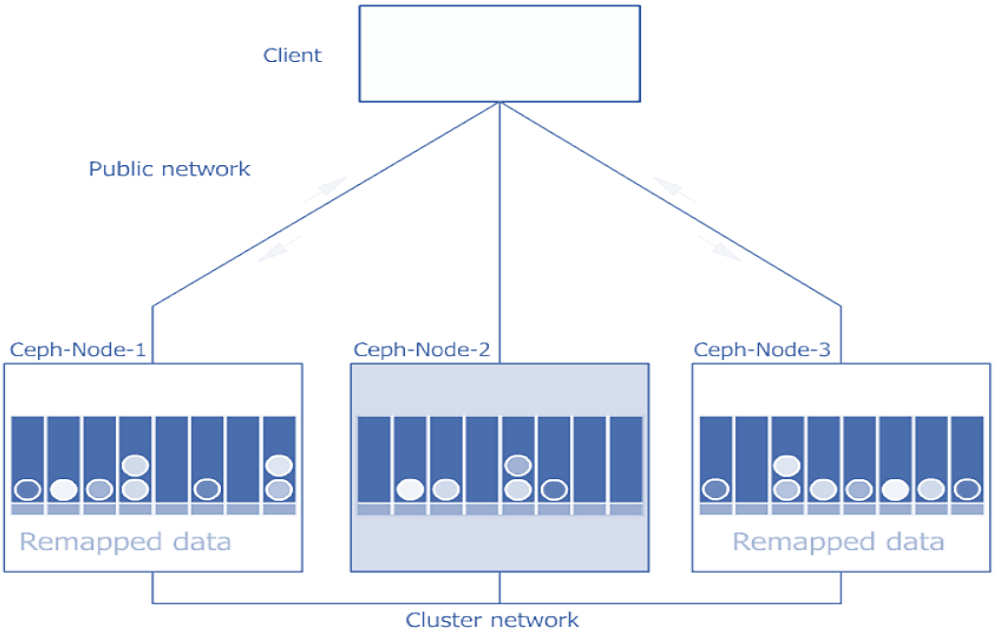

Рис. 3 - Перераспределение chunk(ов) в случае выхода из строя OSD Ceph-Node-2 [19]

[13] The Portable Operating System Interface for uni-X, семейство стандартов IEEE Computer Society для обеспечения базовой совместимости между операционными системами

В качестве виртуальных представлений пользователя (см. рис. 4) хранилище Ceph предоставляет службы:

1. блочного устройства, предоставляющего возможности изменения размеров размеры разделов, создания снимков и клонирования
2. объектного хранилища, поддерживающего RESTful[14] API, совместимое с Amazon S3 и OpenStack SWIFT
3. файловой системы Ceph (FUSE)

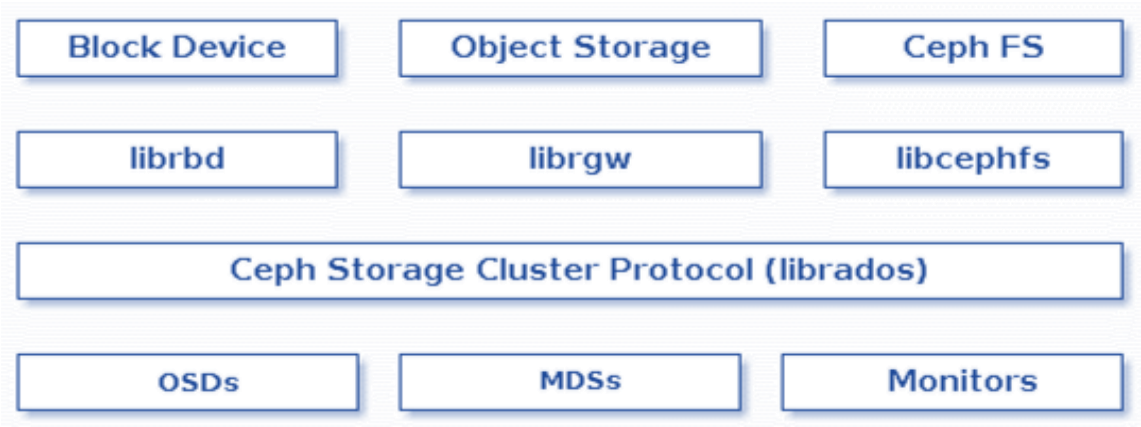

Рис. 4 - Представления Ceph [16]

Так как архитектурные особенности объектного хранилища имеют для нас большое значение в дальнейших построениях, остановимся на их детальном рассмотрении.

[14] Repreentational State Transfer (full) – способ взаимодействия частей распределённого приложения по сети

1.3 Объектные хранилища данных

1.3.1 Особенности модели доступа

В противопоставление блочным устройствам, оперирующим с данными в терминах блоков из секторов и дорожек, и файловым системам, организующим абстракцию иерархии путей, минимальная единица адресации объектного хранилища – цельный пользовательский объект, который может быть получен только по уникальному для кластера числу–идентификатору ID. Такие объекты хранятся в контейнерах (buckets) вместе с определяемыми как клиентской, так и серверной стороной метаданными.

ID	Binary Data	Metadata
1234	01010101010101001101010010 01011000010101001101010010 01011000010101001101010010	name1 value1 name2 value2 nameN valueN

Рис. 5 - Примитив информации объектного хранилища [17]

Модель хранилища позволяет естественным образом наращивать логические дисковые пространства, обеспечивая оптимальное для многих задач соотношение стоимости хранения и скорости (которая оказывается ниже, чем у традиционных файловых систем), в то время как задание произвольной метаинформации не только замещает иерархию путей для отображения взаимосвязи между объектами, но и открывает новые возможности [11] [12]. Например, хранение хэш-сигнатур, позволяет проверять аутентичность данных и, выявляя дубликаты, производить соответствующие оптимизации.

Информация, загруженная в объектное хранилище, в дальнейшем не изменяется: для клиента изменение объекта зачастую означает создание совершенно нового экземпляра. С точки зрения устройства хранилища, это

позволяет не вводить понятия блокировок на чтение; а так как объект однозначно определяется идентификатором, MDS достаточно хэша ID[15] для определения физического расположения объекта на сервере (см. Рис. 6).

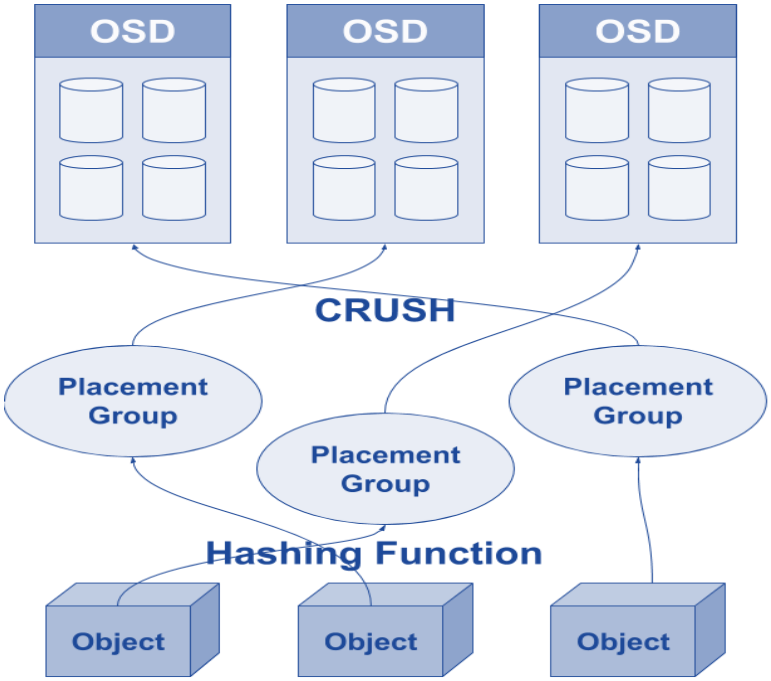

Рис. 6 - Отображение объекта в группы хранения в Ceph [20]

1.3.2 Архитектура Acronis OStorage

Acronis Storage предоставляет пользовательское представление объектного хранилища в виде S3 RESTfull API интерфейса, аналогичного широко распространенному Amazon S3. Инфраструктура OStorage представлена объектными серверами OS, серверами имен NS и s3-gateways GW. Эти сервисы запускаются на многих машинах для обеспечения свойства высокой доступности.

Модель взаимодействия компонент объектного хранилища Acronis проиллюстрирована на Рис. 7.

[15] IDentificator – уникальное значение, по которому однозначно определяется сущность (объект, демон etc.)

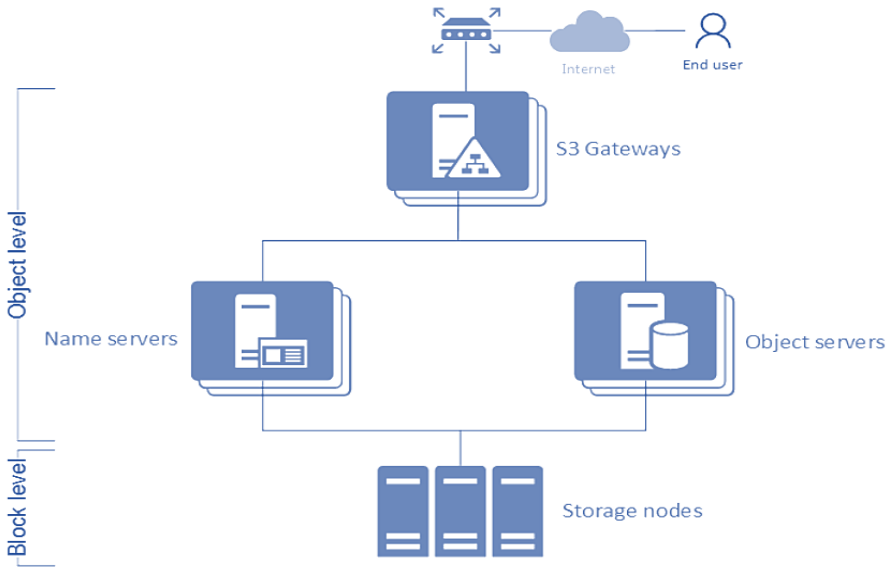

Рис. 7 - Архитектура Acronis OStorage [10]

Объектные сервера OS хранят пользовательские данные, полученные от S3 Gateway, упакованные в специальные контейнеры для лучшей производительности файловых операций в дальнейшем. Сервера имен NS хранят метаданные объекта, также полученные от GW. Метаданные включают в себя по меньшей мере имя объекта, размер, ACL[16], его расположение в кластере и владельца.

S3 Gateway выступает в роли proxy[17] между объектными серверами и пользователями хранилища. Сюда поступают S3-запросы. Для обслуживания внешних соединений используется NGINX[18] сервер. GW занимается пользовательский аутентификацией и проверкой ACL.

[16] Access Control List – проверка прав доступа к объектам
[17] посредника
[18] Engine X - HTTP и reverce-proxy сервер

1.4 Результаты раздела

Логику, связанную с первичной обработкой запросов на шифрование и обращений к KMS[19] в составе Acronis OStorage, следует реализовывать как часть GW, а сами криптографические операции следует осуществлять на объектных серверах OSs. Важно выработать корректные сценарии, в которых некорректный, или заведомо небезопасный запрос задействует наименьший объем кода. Должа также обязательно осуществлена правильная[20] настройка NGINX-сервера, обеспечивающая использование TSL[21].

[19] Key Management System
[20] Детальное рассмотрение будет осуществлено далее
[21] Transport Layer Security

2. РЕАЛИЗАЦИЯ ЛОГИКИ SSE-C В ACRONIS OSTORAGE

2.1 Описание проблемы

Возможность хранить данные зашифрованными представляет интерес как для потенциального клиента хранилища, так и для компании, его предоставляющей. Так, в случае организации шифрования всех объектов диска, упрощается процесс его утилизации и минимизируются последствия несанкционированного доступа. В свою очередь, пользователь обретает возможность хранить удобным для него образом конфиденциальную информацию[22].

В рамках этой главы мы рассмотрим механизмы серверного шифрования данных с помощью предоставленного пользователем ключа.

2.2 Анализ угроз

Э. Таненбаум и М. ван Стеен выделяют следующие угрозы защиты компьютерных систем [1]:

1. перехват (interception),
   т.е. ситуация в которой неавторизованный агент получает доступ к службам или данным
2. прерывание (interruption)
   т.е. повреждение, утрата данных в результате которых служба становится недоступной
3. модификация (modification)

[22] sensitive data

т.е. неавторизованная фальсификация данных или фальсификация служб так, чтобы они не соответствовали своему исходному предназначению

4. подделка (fabrication)

   т.е. использование дополнительных данных или осуществление деятельности, невозможной в нормальных условиях

Системные сущности, присущие архитектуре объектного хранилища были рассмотрены в первой части работы. Это службы OS, GW, логическая сущность объекта, состоящего из данных и метаданных. Клиент не взаимодействует напрямую с OS и с системными метаданными, поэтому с ними связанные сценарии, как, например, при физическом доступе злоумышленника к кластеру, выходят за рамки данной работы[23].

Требования к защите системных сущностей состоят в том, чтобы предупредить каждую из выделенных угроз. Классические механизмы, посредством которых этого можно достичь - шифрование, аутентификация, авторизация, аудит [13].

Под шифрованием подразумевают преобразование исходной информации в новую форму, лишенную начального содержания и в пределе не допускающую обратной трансформации без специального *ключа*.

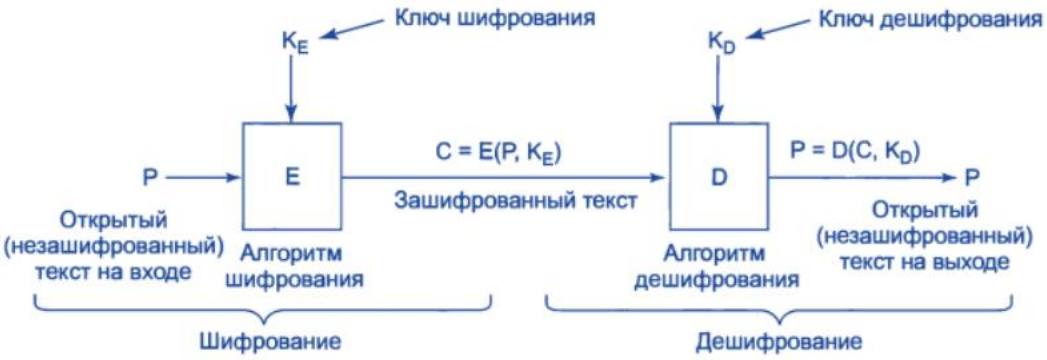

Рис. 7 - Шифрование данных [13]

[23] Рассматривается круг проблем, когда злоумышленник – в худшем случае - единоразово получает физический доступ к дискам, в противном случае можно было бы полагать, что кластер не шифрует данные в принципе, что является предположением вне плоскости данной работы.

Под аутентификацией понимают проверку подлинности заявленного свойства – имени, роли, адреса.

Авторизация позволяет проверять права на совершение действий, различающиеся между группами пользователей. Так объект пользователя А должен быть недоступен пользователю Б даже при наличии специального ID и ключей (доступа и шифрования).

Средства аудита обеспечивают слежение за тем, что пользователь делает, и, хотя это не является непосредственной защитой информации, это позволяет узнавать о совершившихся событиях, попытках атак и своевременно принимать меры.

Подходы к предупреждению выделенных угроз защиты системных сущностей в объектном хранилище приведены в Таблица 1.

Таблица 1 – Механизмы защиты системных сущностей объектного хранилища при взаимодействии с клиентом

	перехват	**прерывание**	**модификация**	**подделка**
GW	TLS, аутентификация	systemctl, restart	-//-	аудит
данные, метаданные[24]	шифрование, авторизация	репликация	HMAC[25]	модель Белла – Ла Падулы

Помимо ранее рассмотренных базовых подходов к обеспечению безопасности в таблице также приведены HMAC и модель Белла – Ла Падулы – достаточно широко использующиеся идеи.

[24] как уже было отмечено, несистемные
[25] Hash-based Message Authentication Code

Сигнатура HMAC предназначена для проверки подлинности сообщений. В отличии от обычного хэша, совпадающего для одинаковых объектов и имеющего коллизии, HMAC использует алгоритм MAC[26], требующий секретного ключа для вычисления.

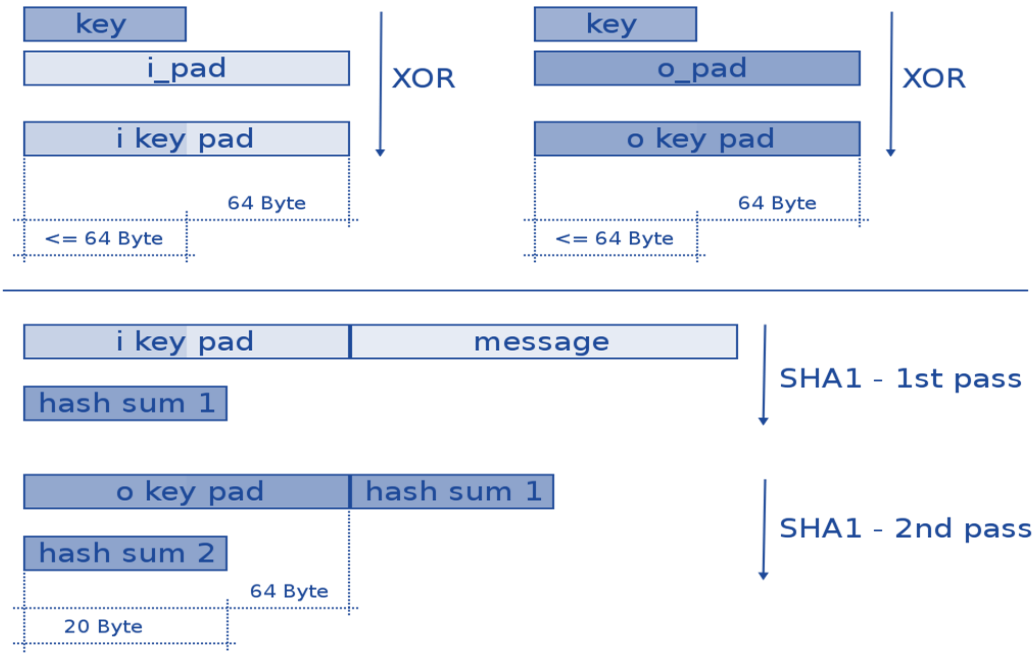

Рис. 8 - Вычисление HMAC [24]

Примечательно, что, как уже было отмечено, хранимые данные, для которых считается HMAC могут быть зашифрованы. Отсюда возникает вопрос, в каком порядке нужно осуществлять операции: вычислять HMAC исходного сообщения или уже зашифрованного. В пользу второго подхода говорит большая энтропия информации, а в пользу первого – временные затраты расшифровки данных для злоумышленника, подбирающего ключ (сейчас – являющегося авторизованным легитимным для системы пользователем) (См. [4], «Authentication Code Algorithms», «Encrypt then MAC?»). Так как это значение получается только в случае успешной расшифровки объекта, достоинства второго метода оказываются для нас более значимыми.

[26] Message Authentication Code, RFC 2140

Модель Белла – Ла Лападулы – достаточно общая концепция, разработанная изначально для обеспечения безопасности работы с документами Правительства США, состоит в следующем [13]:

1. The Simple Security property: процесс, запущенный на уровне безопасности k может проводить операцию чтения данных только своего или более низкого уровня.
2. The * property: процесс, запущенный на уровне безопасности k может проводить запись только в объекты более высокого уровня

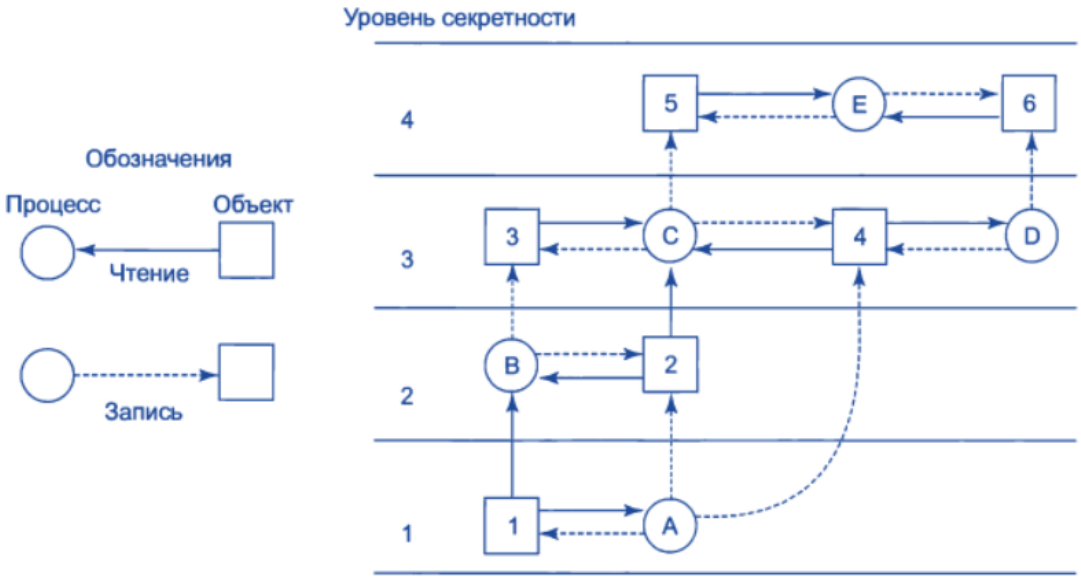

Рис. 9 - Модель Белла — Лападулы [13]

В практическом приложении к рассматриваемой задаче это определит политику передачи и использования контекста шифрования, информации и метаданных в модульной архитектуре программы.

2.3 Amazon SSE-C

Amazon Simple Storage Service – объектное хранилище, позволяющее хранить любой объем данных и извлекать их.

Server-side encryption with customer-provided keys (SSE-C) в объектном хранилище Amazon S3 предполагает отправку дополнительных заголовков x-amz-server-side-encryption-customer-key, x-amz-server-side-encryption-customer-key-MD5, x-amz-server-side-encryption-customer-algorithm для S3-запросов [2]. Пример запроса GET, использующего SSE-C, к пользовательскому хранилищу example-bucket.s3.amazonaws.com приведен на Рис. 10.

```
GET /example-object HTTP/1.1
Host: example-bucket.s3.amazonaws.com
Accept: */*
Authorization: authorization string
Date: Wed, 28 May 2014 19:24:44 +0000
x-amz-server-side-encryption-customer-key:
g0lCfA3Dv40jZz5SQJ1ZukLRFqtI5WorC/8SEKEXAMPLE
x-amz-server-side-encryption-customer-key-MD5: ZjQrne1X/iTcskbY2m3example
x-amz-server-side-encryption-customer-algorithm: AES256
```

Рис. 10 - Запрос GET, использующий SSE-C заголовки

При загрузке объекта, байты шифруются с помощью AES256, после чего экземпляр ключа удаляется из памяти. Amazon S3 не хранит предоставленного значения. Вместо этого сохраняется HMAC ключа, посчитанный со случайной солью, который используется для проверки новых обращений к хранилищу. Это значение не может быть использовано для расшифровки объекта, что означает потерю объекта в случае утраты ранее предоставленного ключа[27].

[27] рассмотрению KMS, решающего эту проблему, посвящена 3я глава

Объектное хранилище Amazon S3, при обработке SSE-C заголовков:

1. Отклоняет запросы, сделанные по HTTP.
2. В случае использования функциональности контроля версий в контейнере (bucket), каждая из версий объекта может иметь свой ключ шифрования и, в силу уже названных обстоятельств, учет того, какой ключ использовался для какой версии объекта, лежит на клиенте.

S3-операции, поддерживающие SSE-C заголовки:

1. GET, возвращающая объект и его метаданные
2. HEAD, получающая только метаданные объекта
3. PUT, загружающая объект в хранилище
4. Multipart Upload, загружающая объект по частям; каждый из запросов серии должен содержать SSE-C совпадающие заголовки
5. POST, загружающий объект с метаданными, передаваемыми не как заголовки, а как поля формы
6. Copy, создающая еще один экземпляр объекта в хранилище. Отличие этой операции от прочих состоит в том, что необходимо предоставить ключи как для исходного объекта (если он зашифрован), так и для создаваемого (если требуется осуществить его шифрование).

2.4 Технические детали реализации

2.4.1 Первичная обработка запросов

Принятый в OStorage gateway S3-запрос, прежде чем он будет обработан, должен быть проверен. Схема, иллюстрирующая логику обработки дана в приложении (Рис. 18 - Схема первичной обработки SSE-C запроса к объектному хранилищу).

Прежде всего, необходимо пресекать попытки общения с сервером не по TLS: ключи, передаваемые таким образом беспрепятственно могут быть получены слушающей трафик стороной (Eva). При промышленной установке, Acronis Storage настраивают на использование HTTPS; соответственно, обслуживающий клиентов nginx будет отклонять HTTP-обращения, а такие запросы даже не попадут в OStorage gateway. Тем не менее, в случае «неправильной» конфигурации, небезопасное соединение определяется по URI в составе тела запроса и отклоняется.

Далее проверяется MD5 предоставленного пользователем ключа (присылаемые заголовки представлены с помощью base64) и, в зависимости от типа запроса – на чтение или на создание объекта – осуществляется работа с метаданными.

Сначала вычисляется keyed-hash message authentication (HMAC) для ключа, и, в случае обращений к хранилищу, связанных с чтением существующего объекта, он сверяется со хранимым значением. Для этой операции используется криптографическая хэш-функция SHA1[28]. Ключ, для которого посчитанное значение не совпадает с сохраненным, для расшифровки данных не используется – запрос отклоняется уже на этом этапе.

[28] Несмотря на то, что SHA-1 больше не считается криптографической функцией [22] [21], она по-прежнему широко используется при подсчете HMAC в силу высокой скорости вычисления и надежности MAC независимо от используемой хэша [23].

Примечательно, что ничто не препятствует использованию пользователем одинакового ключа как для разных объектов, так и для разных версий одного и того же объекта. Система, использующая предоставленный ключ непосредственно была бы заведомо уязвимой. Поэтому, на смену предоставленному пользователем значению, следует создать новый ключ (на самом деле пару – ключ и вектор инициализации), уникальный для каждого объекта; а для вычисления HMAC – уникальную соль. Соответственно, два одинаковых объекта в хранилище, зашифрованные одним и тем же ключом, окажутся представленными разным набором байт, а HMAC, хранимые в метаданных, не совпадут.

Таким образом, связанные с криптографией и хранимые с объектом метаданные для пользовательского eKey - значения

```
ivSalt, eKeySalt, eKeyHMAC = HMAC(sha256, eKey, eKeySalt)
```

Шифрование осуществляется с помощью

```
(iv, key) = EVP_BytesToKey(sha256, eKey, ivSalt)
```

2.4.2 Отклонение запросов

В связи с необходимостью имитировать поведение Amazon S3, были исследованы оригинальные ответы сервера при отклонении клиентского запроса. Сводная таблица возвращаемых кодов ошибок и сообщений дана в приложении А (Таблица 2).

2.4.3 Шифрование объекта

Блочное шифрование данных AES256 может осуществляться в одном из нескольких режимов. В силу специфики объектного хранилища, объекты в котором создаются, но не изменяются, выбран режим сцепления блоков шифр текста CBC[29], не обладающий недостатками сохранения статистических особенностей открытого текста режима простой замены.

Использование более совершенных режимов как PCBC[30], обладающих внутренним параллелизмом неоправданно, так как блоки данных, получаемые по fastcgi приходят последовательно, а в производительность линейного режима работы CBC лучше.

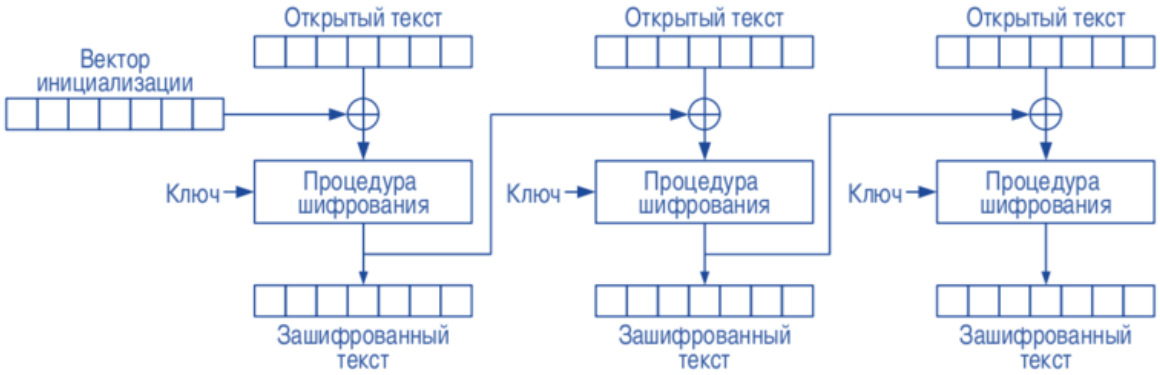

Рис. 11 - CBC режим шифрования [4]

Неизменность объектов в объектном хранилище при выборе режима шифрования существенна. S3-протокол поддерживает ranged requests – формат запросов, позволяющих считать не весь объект, а лишь его часть. Зашифрованные блоки являются зависимыми между собой: шифр-блок используется как вектор инициализации для шифрования, следующего и, если бы объект бы мог модифицироваться частями, пришлось бы переписывать все

[29] Cipher block chaining
[30] Propagating cipher block chaining – распространяющееся сцепление блоков шифра

данные, следующие за местом внесения изменений. В случае же запросов на чтение, проблема решается выравниванием запросов по границам шифр-блоков. Предшествующий левой границе блок, используется в качестве вектора инициализации. Начальный iv-вектор применяется при необходимости расшифровать самый первый шифр-блок.

Криптографические операции над объектом осуществляется в составе модуля libpcs_ostor_clt. Сначала поступает анонс производимой операции – в нашем случае это может быть запрос на запись или чтение. В его состав включена необходимая для осуществления криптографии информация – начальный вектор инициализации и ключ шифрования, сгенерированные по клиентскому ключу на этапе первичной обработки заголовков. В этот же момент, при необходимости, запрос к хранилищу выравнивается.

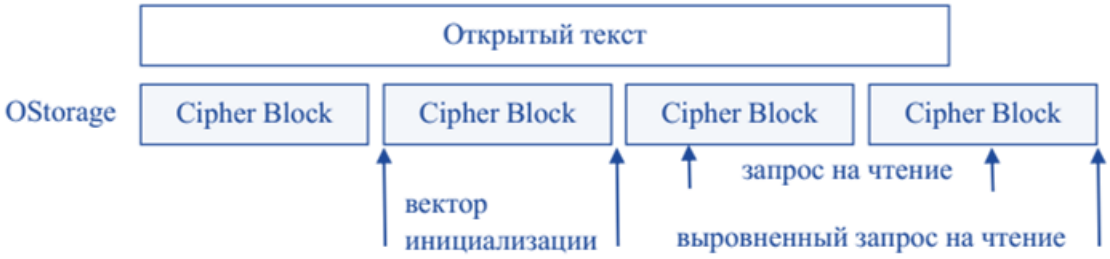

Рис. 12 - Выравнивание запросов на чтение

Далее принимается серия сообщений с данными “msg”, вообще говоря, заранее не известной длины, получаемых по fastcgi[31]. Используемые интерфейсы OpenSSL выдают блоки зашифрованных данных, из которых формируется преобразованное сообщение msg’, уже уходящее на запись в нижележащую абстракцию файловой системы. Даже если исходное сообщение было кратно

[31] FastCGI – клиент-серверный протокол взаимодействия веб-сервера и приложения

размеру блока шифрования и вся «полезная» информация уже сохранена, по завершении передачи на запись отправляется последний шифр-блок.

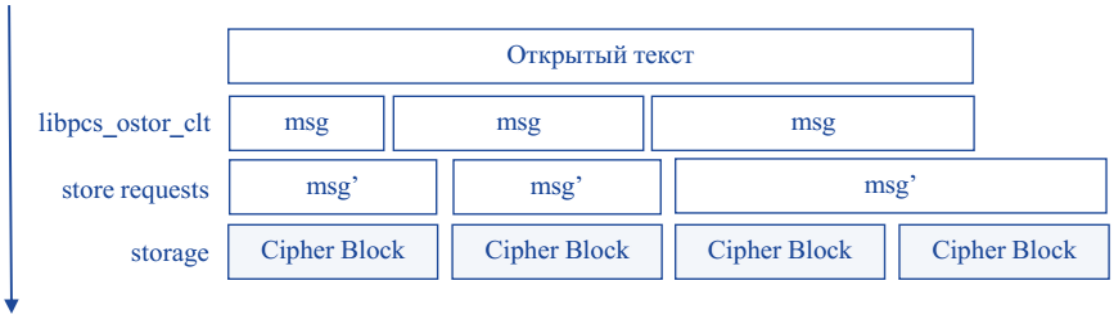

Рис. 13 - Формирование блоков для записи

2.5 Unit-тестирование прототипа

Для решения поставленных задач была осуществлена развертка кластера из виртуальных машин под управлением VZ (RHEL) – based дистрибутива Acronis Storage 2.1 с инициализированным сервисом объектного хранилища OStorage. В рамках постановки эксперимента осуществлялась работа над кодом компонента S3 Gateway daemon, запросы к которому на первом этапе формировались python-скриптом.

Затем для проверки надёжности прототипа были использованы адаптированные для SSE-C заголовков Fuzzing-тесты Ceph «S3 compatibility tests» [14]. Тестирование проблем не выявило.

2.6 Результаты раздела

Нам удалось реализовать рабочий прототип, реализующий логику Amazon SSE-C для объектного хранилища Acronis Storage. Тем не менее, ответственность за безопасность используемых ключей всецело лежит на клиентской стороне; реализация функциональности Key Management системы решает данную проблему. Этому и будет посвящена следующая часть.

3. МОДЕЛЬ KMS НА ОСНОВЕ HASHICORP VAULT

3.1 Описание проблемы

KMS-системы предоставляют простой интерфейс, позволяющий создавать ключи шифрования и управлять ими.

В таком сценарии клиентская сторона не имеет ключа, с помощью которого можно было бы хранимые на сервере данные расшифровать, а располагает лишь его идентификатором, который по требованию может быть сменен (rotation). Таким образом, например, решаются вопросы, связанные с утечкой ключей у клиента. В свою очередь облачное хранилище не запоминает ни ключей шифрования, ни соответствующих объекту идентификаторов. Общая схема работы с KMS представлена на Рис. 14 и Рис. 15.

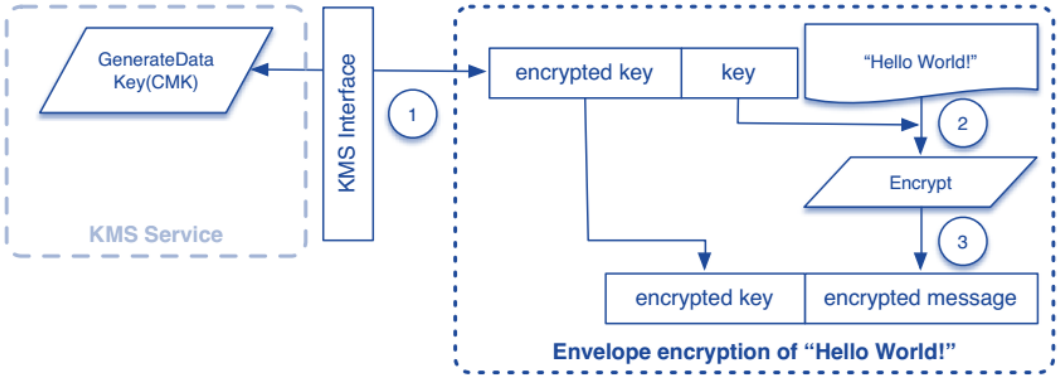

Рис. 14 - Шифрование в режиме KMS

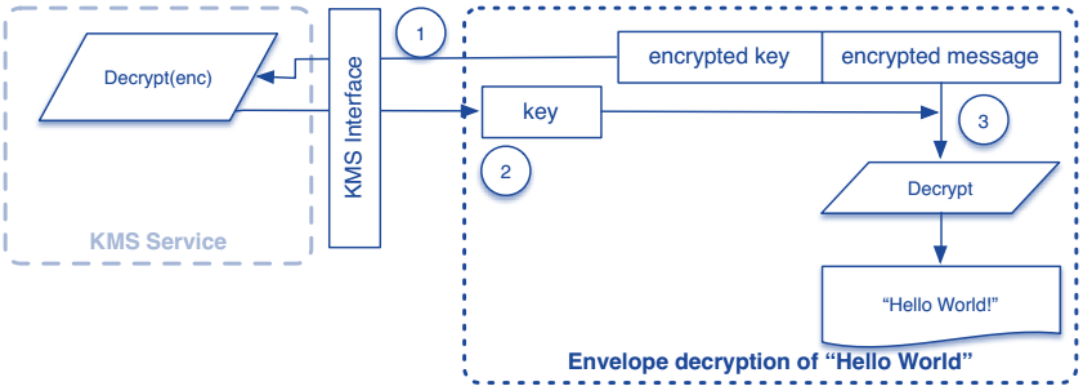

Рис. 15 - Расшифровка в режиме KMS

Из постановки задачи следуют серьезные требования к системе, хранящей соответствие ключей и их идентификаторов:

1. Надежность
2. Доступ на основе кворума
3. Контроль доступа
4. Аудит
5. Производительность

Комплексное изучение вопроса привело к нахождению подходящего решения с открытым исходным кодом под лицензией Mozilla Public License 2.0 «HashiCorp Vault» [15].

3.2 HashiCorp Vault

Vault – универсальный инструмент для безопасного управления *секретами*. Под секретом здесь и далее понимается то, доступ к чему необходимо строго контролировать. Это могут быть ключи, пароли, сертификаты. Vault обеспечивает единый интерфейс для любого типа информации, обеспечивая при этом жесткий контроль и детализированный журнал аудита. Доступ к Vault может осуществляться по RESTfull API.

Данные хранилища секретов могут быть расположены как на диске машины исполнения, так и в отдельном S3 Object Storage. Информация хранится в зашифрованном виде. Для инициализации (перевода программы в состояние готовности обслуживания клиентов) необходимо «открытие Vault» - основанный на кворуме протокол введения любых k из n существующих ключей независимыми агентами, инициирующий расшифровку хранилища.

Vault представляет из себя древообразную key-value БД (для ключей принято использовать пути как secret/cluster/my_secret). Доступ осуществляется при помощи token(ов). Первичный token появляется при создании хранилища,

далее токены лишь наследуются. Для каждого из них может быть определена политика доступа к веткам посредством создания *роли* и правил наследования, описывающих, какие действия владелец может производить. Возможно также устанавливать время жизни ttl[32]. При удалении token(a) удаляется все поддерево, им порожденное.

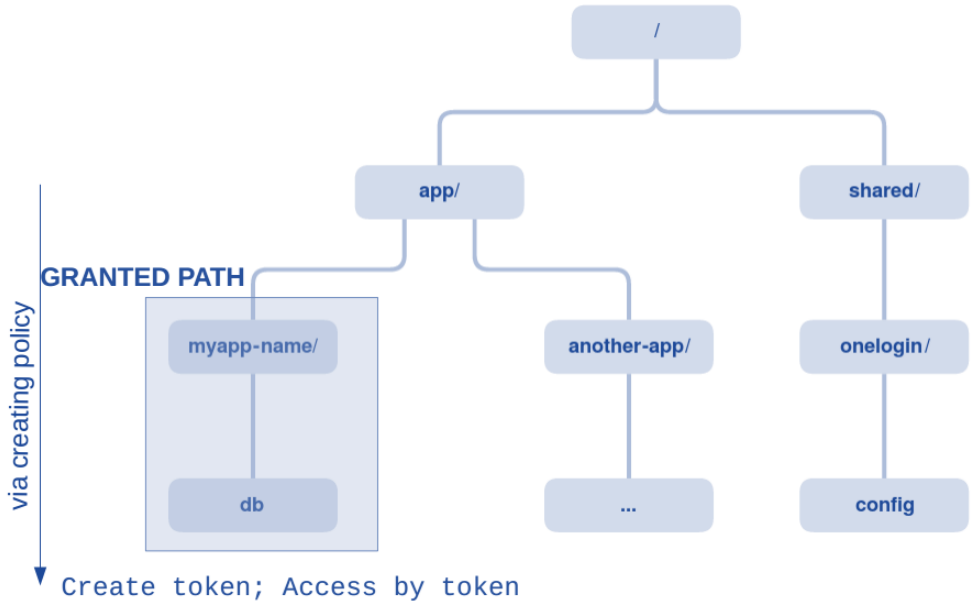

Рис. 16 - Роли доступа в Vault [18]

3.3 Vault в роли KMS

Прежде всего, необходимо ограничить пространства идентификаторов ключей разных пользователей для обслуживания ключей SSE-шифрования (Рис. 17).

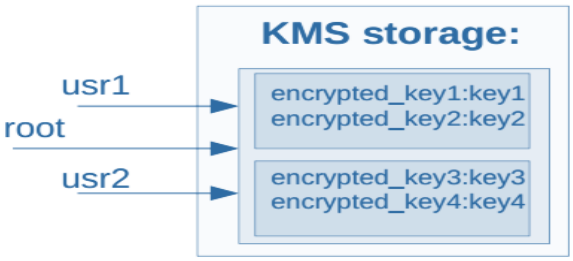

Рис. 17 - Пространства ключей пользователей

[32] Time to leave

Для этой цели для каждого bucket(a) пользователя определим эксклюзивную[33] роль чтения и записи значений поддерева вида /secret/regionID/clusterID/userID/bucketID и породим первичный пользовательский токен CMK[34] в указанном выше порядке наследования. Там и будут храниться сгенерированные по требованию ключи.

В связи с необходимостью обслуживания сценариев удаления пользователя и кластера, accessor-токены, обеспечивающие полный контроль над созданным клиентским токеном, будем хранить в системной ветке secret/path_manage/.

3.4 Постановка эксперимента

Сборки Vault представлены для подавляющего большинства современных систем. В рамках постановки эксперимента использовалась виртуальная машина под управлением CentOS 7. Vault создавался в режиме разработчика (-dev), в памяти, уже открытый.

Были написан python-скрипты: класс, выполняющий работу серверного демона выполняемого на машине с HashiCorp Valult, принимающий запросы на генерацию пары (ключ шифрования, идентификатор ключа в Vault) для токена в эксклюзивном поддереве, выдачу ключа и создание путей; класс, реализующий логику клиентской работы. Исходные коды и сценарий использования приведены в ПРИЛОЖЕНИЕ Б.

В роли клиента здесь подразумевается код OS, выполняющий SSE-логику в режиме KMS. Демон KMS слушает выделенный порт. Подключаемый TCP-сокет обслуживается отдельным потоком. Коммуникация осуществляется

[33] Т.е. чтение и запись может производиться исключительно создаваемой ролью bucket(a), но не ролью пользователя или кластера, от которых она наследуется
[34] Customer Master Key

типичным для RPC[35] образом: стаб-код клиента сериализует составные python-объекты с запросом в JSON, которые восстанавливаются на сервере.

Для генерации случайного значения ключа и идентификатора был использован источник энтропии, использующий устройство /dev/random системы.

3.5 Результаты раздела

Разработанная архитектура может быть использована для хранения пользовательских ключей, обеспечивая их сохранность посредством строгих политик доступа, шифрования и избыточности данных используемого в качестве backend(a) другого специализированного SDS Object Storage. Тем не менее, до введения ее в эксплуатацию, необходимо отдельно решить проблемы, возникающие в связи с необходимостью трансакционной работы с Vault[36].

[35] Remote Procedure Call

[36] Этого можно достичь, например, ведением и обслуживанием журнала. Так как задачей данного раздела ставилось рассмотрение Vault на предмет использования в качестве KMS и построение рабочей модели, этот отдельно технически непростой вопрос выходит за рамки работы.

ЗАКЛЮЧЕНИЕ

В данной работе были изучены проблемы безопасности пользовательских данных, хранимых в Object Storage. После исследования специфики хранения и обслуживания данных программно-определяемых распределенных файловых систем был осуществлён системный анализ угроз структурных компонент объектного хранилища. С учетом полученных результатов был реализован прототип, осуществляющий Server-Side Encryption with client provided key в составе Acronis Object Storage.

Также была разработана модель Key-Management Service, основанная на open-source решении HansiCorp Vault. Были предложены сценарии использования серверной стороной для осуществления Server-Side Encryption в режиме KMS.

Результаты дипломной работы были опубликованы в сборнике научных трудов МФТИ «Математическое моделирование информационных систем»[37].

[37] Статья «AWS API-совместимое шифрование для Object Storage SDS-решения»

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

[1] М. в. С. Эндрю Таненбаум, Распределенные системы. Принципы и парадигмы, Санкт-Петербу́рг: Питер, 2003.

[2] Amazon Web Services, «Protecting Data Using Server-Side Encryption with Customer-Provided Encryption Keys (SSE-C),» Amazon Web Services, Inc., 2017. [В Интернете]. Available: https://docs.aws.amazon.com/AmazonS3/latest/dev/ServerSideEncryptionCustomerKeys.html. [Дата обращения: 2017].

[3] Amazon Web Services, «AWS Key Management Service Cryptographic Details,» Amazon Web Services, Inc., August 2016. [В Интернете]. Available: https://d0.awsstatic.com/whitepapers/KMS-Cryptographic-Details.pdf. [Дата обращения: April 2017].

[4] S. D. T. и S. Johnson, Cryptography for Developers, Rockland, Massachusetts, US: Syngress Publishing, 2006.

[5] NERSC, «Storage Trends and Summaries,» Lawrence Berkeley National Laboratory, 03 January 2017. [В Интернете]. Available: http://www.nersc.gov/users/storage-and-file-systems/hpss/storage-statistics/storage-trends/. [Дата обращения: 16 April 2017].

[6] IDC, «The digital univerce in 2020: Big data, bigger digital shadows, and biggest growth in the fast east - United States,» IDC, February 2013. [В Интернете]. Available: https://www.emc.com/collateral/analyst-reports/idc-digital-universe-united-states.pdf. [Дата обращения: 16 April 2017].

[7] CISCO, «The Zettabyte Era: Trends and Analysis,» CISCO, 7 June 2017. [В Интернете]. Available: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/vni-hyperconnectivity-wp.html. [Дата обращения: 16 June 2017].

[8] Cleversafe, Inc, «Why RAID Dead for Big Storage,» StorageNewsletter, 14 May 2013. [В Интернете]. Available: http://www.storagenewsletter.com/2013/05/14/why-raid-dead-for-big-storage-cleversafe/. [Дата обращения: 10 May 2017].

[9] J. Sanders, «Why Ceph could be the RAID replacement the enterprise needs,» ZDNet, 29 April 2016. [В Интернете]. Available: http://www.techrepublic.com/article/why-ceph-could-be-the-raid-replacement-the-enterprise-needs/. [Дата обращения: 10 October 2016].

[10] Acronis, «Acronis Storage 2.0 Administrator's Guide,» Acronis International GmbH, 06 April 2017. [В Интернете]. Available: https://www.acronis.com/en-us/download/docs/as/adminguide/. [Дата обращения: 14 April 2017].

[11] C. Evans, «Storage gets smarter with metadata,» *Storage Magazine,* № 3, 2015.

[12] Е. Лебеденко, «Object Storage: можно ли прожить без файлов?,» 23 November 2012. [В Интернете]. Available: http://old.computerra.ru/vision/722808/. [Дата обращения: 17 June 2017].

[13] Э. Таненбаум и Х. Бос, Современные операционные системы. 4-е издание, СПб: Питер, 2015.

[14] Ceph, «Compatibility tests for S3 clones,» GitHub, Inc. , 28 March 2017. [В Интернете]. Available: https://github.com/ceph/s3-tests. [Дата обращения: 30 March 2017].

[15] HashiCorp, «Vault Documentation,» HashiCorp, [В Интернете]. Available: https://www.vaultproject.io/docs/index.html. [Дата обращения: 14 february 2017].

[16] Ceph, «Welcome to Ceph,» Red Hat, 2016. [В Интернете]. Available: http://docs.ceph.com/docs/master/. [Дата обращения: 14 April 2017].

[17] M. T. Jones, «Ceph: A Linux petabyte-scale distributed file system,» IBM Corporation, 04 June 2010. [В Интернете]. Available: https://www.ibm.com/developerworks/linux/library/l-ceph/l-ceph-pdf.pdf. [Дата обращения: 26 September 2016].

[18] C. Stokes, T. Vattathil и B. Chavis, «HashiCorp Vault on the AWS Cloud,» HashiCorp, Inc..; Amazon Web Services, Inc., November 2016. [В Интернете]. Available: https://s3.amazonaws.com/quickstart-reference/hashicorp/vault/latest/doc/hashicorp-vault-on-the-aws-cloud.pdf. [Дата обращения: 12 February 2017].

[19] AlexBin, «Знакомство с хранилищем Ceph,» TM, 3 November 2016. [В Интернете]. Available: https://habrahabr.ru/post/313644/. [Дата обращения: 16 April 2017].

[20] P. G. K. Lockwood, «Principles of Object Storage,» 13 July 2016. [В Интернете]. Available: http://www.glennklockwood.com/data-intensive/storage/object-storage.html. [Дата обращения: 17 April 2017].

[21] Y. L. Y. H. Y. Xiaoyun Wang, «Collision Search Attacks on SHA1».

[22] M. Stevens, «Cryptanalysis of MD5 & SHA-1,» CWI, Amsterdam.

[23] M. Bellare, «Advances in Cryptology,» в *New Proofs for NMAC and HMAC*, Berlin, 2006.

[24] «Hash-based message authentication code,» Wikipedia, 08 April 2017. [В Интернете]. Available: https://en.wikipedia.org/wiki/Hash-based_message_authentication_code. [Дата обращения: 16 June 2017].

ПРИЛОЖЕНИЕ А

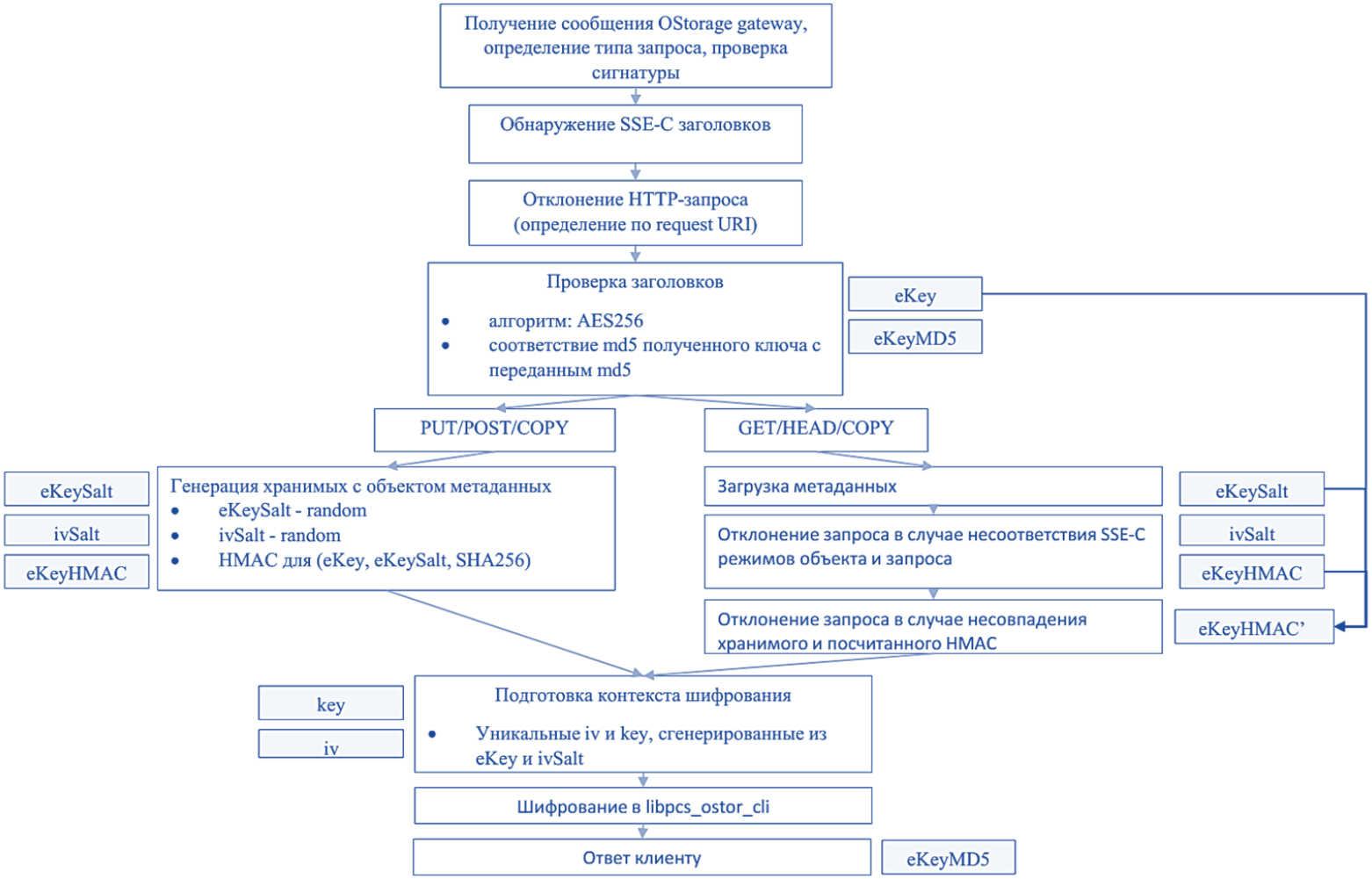

Рис. 18 - Схема первичной обработки SSE-C запроса к объектному хранилищу

Таблица 2 - Amazon S3 error response

Problem	HTTP Error	HTTP Error Description	Message
SSE_NO_HTTPS	400	InvalidArgument	Requests specifying Server Side Encryption with Customer provided keys must be made over a secure connection.
SSE_NO_EKEY_MD5_HEADER	400	InvalidArgument	Requests specifying Server Side Encryption with Customer provided keys must provide the client calculated MD5 of the secret key.
SSE_NO_EKEY_HEADER	400	InvalidArgument	Requests specifying Server Side Encryption with Customer provided keys must provide an appropriate secret key.
SSE_NO_EALGO_HEADER	400	InvalidArgument	Requests specifying Server Side Encryption with Customer provided keys must provide a valid encryption algorithm.
SSE_INVALID_EALGO	400	InvalidEncryptionAlgorithmError	The Encryption request you specified is not valid. Supported value: AES256.
SSE_BAD_EKEY_B64	400	InvalidArgument	The secret key was improperly encoded. The secret key must be Base64 encoded.
SSE_BAD_EKEY_MD5_B64	400	InvalidArgument	The MD5 hash of the secret key was improperly encoded. The MD5 hash must be Base64 encoded.
SSE_EKEY_INVALID_SIZE	400	InvalidArgument	The secret key was invalid for the specified algorithm.
SSE_EKEY_MD5_MISTMATCH	400	InvalidArgument	The calculated MD5 hash of the key did not match the hash that was provided.
SSE_EDATA_DIRECT_REQUEST	400	InvalidArgument	The object was stored using a form of Server Side Encryption. The correct parameters must be provided to retrieve the object.
SSE_NEDATA_EREQUEST	400	InvalidArgument	The encryption parameters are not applicable to this object.

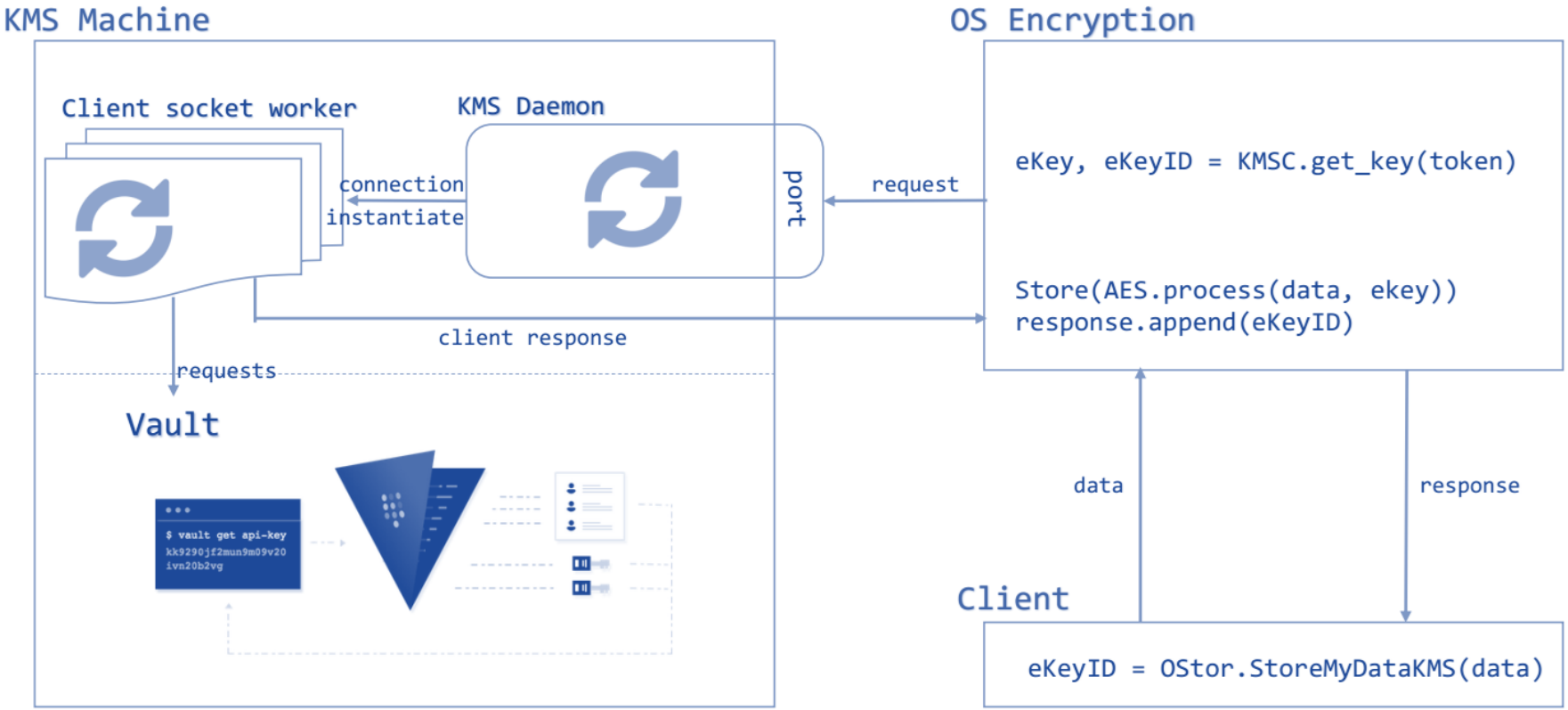

Рис. 19 - Схема KMS

ПРИЛОЖЕНИЕ Б

Подготовка окружения

Загрузка Vault с https://www.vaultproject.io/downloads.html

```
$: wget https://releases.hashicorp.com/vault/0.7.3/vault_0.7.3_linux_amd64.zip
```

Запуск в in-memory режиме разработчика

```
$: vault server –dev
Unseal Key: PDzzEOP3OujxgLaoLSmtFJA0JOvah8LkAxiIs3fLaxY=
Root Token: 4bb01006-6dde-8b47-8463-c4a56048dc6a
```

Тестирование авторизации

```
$: vault auth 4bb01006-6dde-8b47-8463-c4a56048dc6a
```

Создадим токен, используемый KMS для создания изолированных пользовательских пространств

```
$: vault token-create
Key                 Value
---                 -----
token               0f8efec9-01a0-738a-9065-a17f566dbf4b
token_accessor      d06866d5-0973-b6ca-3aa2-56cddb7e4f71
token_duration      0s
token_renewable     false
token_policies      [root]
```

Объявим переменные окружения, используемые демоном KMS

```
$: VAULT_ADDR=127.0.0.1
$: CREATE_PATH_TOKEN=0f8efec9-01a0-738a-9065-a17f566dbf4b
```

Исходные коды

KMS_Daemon.py

```
import socket, json, os, hvac
from enum import Enum

# KMS Daemon Unit
# should be launched on the same machine Vault is working on
# System vars VAULT_ADDR and CREATE_PATH_TOKEN should be declared

class ServerResponseStatus(Enum):
  FAILURE                 = 1
  SUCCEEDED               = 2


class ServerError(Enum):
  INVALID_REQUEST         = 'Invalid request'
  UNKNOWN_OPERATION             = 'Unknown operation'
  INTERNAL_SERVER_ERROR   = 'Internal server error'
  NO_ERROR                = 'No error'


class ServerOperation(Enum):
  GENERATE_KEY            = 1
  OBTAIN_KEY                    = 2
  DELETE_KEY                    = 3
  ROTATE_KEY                    = 4
  CREATE_PATH                   = 5
  DELETE_PATH                   = 6


class ClientRequestFields(Enum):
  TOKEN                   = 1
  OP                      = 2
  PARAMS                  = 3


class ServerResponseFields(Enum):
  OP_STATUS               = 1
  USR_MSG                 = 2
  RES_DICT                = 3
  # optional respDict fields
  STATUS                  = 4
  EKEY_ID                 = 5
  EKEY                          = 6
  TOKEN                   = 7


class KeyManagementServerDaemon:

  def __init__(self, serverAddress, serverPort, logger):
    from daemon import DaemonContext

    with logger as l:
      self._fragmentSize = 1024

      self._serverAddress = serverAddress
      self._serverPort = serverPort
```

```
    self._logger = l

    with DaemonContext():
      self._serve_forever()



def _serve_forever(self):
  from threading import Thread

  with self._logger as l:
    server = socket.socket()
    server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    server.bind(self._serverAddress, self._serverPort)
    l.debug("Server socket created for %s %s", self._serverAddress, self._serverPort)
    server.listen(1)

    while True:
      try:
        l.debug("Server is waiting for a new client...")
        client, address = server.accept()
        l.debug("Connected client with address %r", address)
        thread = Thread(target=self._serve_client, args=[client, address])
        thread.daemon = True
        thread.start()
      except Exception as e:
          l.error("Unexpected error: %r", e)


def _serve_client(self, client, address):
  with self._logger as l:
    fragments = []
    try:
      while True:
        chunck = client.recv(self._fragmentSize)
        if not chunck:
          break
        fragments.append(chunck)
    except Exception as e:
        l.error("%r : Unexpected error on recieve: %r", address, e)

    data = b"".join(fragments)
    status, msg, respDict = self._process_request(data)
    self._response_client(client, address, status, msg, respDict)
    client.close()


def _process_request(self, address, data):
  with self._logger as l:
    try:
      request = json.loads(data)

      token = request[ClientRequestFields.TOKEN]
      op = request[ClientRequestFields.OP]
      params = request[ClientRequestFields.PARAMS]
    except Exception as e:
      l.info("%r : Request parsing failed! %r", address, e)
      return ServerResponseStatus.FAILURE, ServerError.INVALID_REQUEST, None

    supportedOpsFuncs = {
      ServerOperation.GENERATE_KEY    : self._generate_key,
```

```
      ServerOperation.OBTAIN_KEY     : self._obtain_key,
      ServerOperation.REMOVE_KEY     : self._remove_key,
      ServerOperation.ROTATE_KEY     : self._rotate_key,
      ServerOperation.CREATE_PATH    : self._create_path
      ServerOperation.DELETE_PATH    : self._delete_path
    }
    func = supportedOpsFuncs.get(op, None)
    if func == None:
      l.info("%r : Unknown operation recieved %r", address, op)
      return ServerResponseStatus.FAILURE, ServerError.UNKNOWN_OPERATION, None

    resDict = func(token, params)
    if resDict == None:
      return ServerResponseStatus.FAILURE, ServerError.INTERNAL_SERVER_ERROR, None
    else
      return ServerResponseStatus.SUCCEEDED, ServerError.NO_ERROR, resDict


def _response_client(self, client, address, status, msg, respDict):
  with self._logger as l:
    responseData = {
      ServerResponseFields.OP_STATUS   : status,
      ServerResponseFields.USR_MSG   : msg,
      ServerResponseFields.RES_DICT   : respDict
    }
    l.debug("%r : Bulded user response %r", address, responseData)

    response = None
    try:
      response = json.dumps(responseData)
    except Exception as e:
      l.error("%r : Bulding response json failed! %r", address, e)
      return
    try:
      client.sendall(response)
    except Exception as e:
      l.error("%r : Can't send client response! %r", address, e)


def _hvac_authenticated_client(self, token):
  with self._logger as l:
    try:
      client = hvac.Client(url=os.environ['VAULT_ADDR'], token=token)
    except Exception as e:
      l.error("%r : Can't construct hvac client! %r", e)
      return None
    if not client.is_authenticated() == False:
      return None
    return client


def _generate_key(self, token, clientpath):
  try:
    key = os.urandom(256)
    keyID = os.urandom(256)
    with self._hvac_authenticated_client(token) as vclient:
      vclient.write('secret/'+clientpath+'/'+str(keyID), key=key)
  except Exception as e:
    return {ServerResponseFields.STATUS : ServerResponseStatus.FAILURE}
  return {ServerResponseFields.STATUS : ServerResponseStatus.SUCCEEDED,
    ServerResponseFields.EKEY : key,
```

```
    ServerResponseFields.EKEY_ID : keyID}


def _obtain_key(self, token, keyID):
  with self._logger as l:
    key = None
    try:
      clientpath = store[0]
      keyID = stor[1]
      with self._hvac_authenticated_client(token) as vclient:
        key = vclient.read('secret/'+clientpath+'/'+str(keyID))
    except Exception as e:
      l.error("Unexpected error %r", e)
      return {ServerResponseFields.STATUS : ServerResponseStatus.FAILURE}
    return {ServerResponseFields.STATUS : ServerResponseStatus.SUCCEEDED,
      ServerResponseFields.EKEY : key}


def _rotate_key(self, token, store):
  with self._logger as l:
    try:
      clientpath = store[0]
      keyID = stor[1]
      with self._hvac_authenticated_client(token) as vclient:
        key = vclient.read('secret/'+clientpath+'/'+str(keyID))
        keyIDNew = os.urandom(256)
        key = vclient.write('secret/'+clientpath+'/'+str(keyIDNew), key=key)
        vclient.delete('secret/'+clientpath+'/'+str(keyID))
    except Exception as e:
      l.error("Unexpected error %r", e)
      return {ServerResponseFields.STATUS : ServerResponseStatus.FAILURE}
    return {ServerResponseFields.STATUS : ServerResponseStatus.SUCCEEDED,
      ServerResponseFields.EKEY_ID : keyIDNew}


def _delete_key(self, token, store):
  with self._logger as l:
    try:
      clientpath = store[0]
      keyID = store[1]
      with self._hvac_authenticated_client(token) as vclient:
        vclient.delete('secret/'+clientpath+'/'+str(keyID))
    except Exception as e:
      l.error("Unexpected error %r", e)
      return {ServerResponseFields.STATUS : ServerResponseStatus.FAILURE}
    return {ServerResponseFields.STATUS : ServerResponseStatus.SUCCEEDED}


def _apply_policy(vclient, clientpath):
  policy = """
  path "sys" {
    policy = "deny"
  }

  path "secret/""" + clientpath + """/*" {
      policy = "write"
  }

  path "auth/token/*" {
    policy = "write"
  }
```

```
    """
    vclient.set_policy(clientpath+'-policy', policy)


  def _create_path(self, dummy, clientpath):
    with self._logger as l:
      try:
        with self._hvac_authenticated_client(
             os.environ('CREATE_PATH_TOKEN')) as vclient:
          if (vclient.get_policy(clientpath+'-policy') != None or
            vclient.list('secret/'+clientpath) != None or
            vclient.read('secret/path_manage/'+clientpath) != None):
            l.debug("Path already exists")
            raise StandardError

          self._apply_policy(vclient, clientpath)
          token = vclient.create_token(policies=[clientpath+'-policy'])
          vclient.write('secret/path_manage/'+clientpath,
accessor_token=token['auth']['accessor'])
      except Exception as e:
        l.error("Unexpected error %r", e)
        return {ServerResponseFields.STATUS : ServerResponseStatus.FAILURE}
    return {ServerResponseFields.STATUS : ServerResponseStatus.SUCCEEDED,
      ServerResponseFields.TOKEN : token}


  def _delete_path(self, dummy, clientpath):
    with self._logger as l:
      try:
        with self._hvac_authenticated_client(
             os.environ('CREATE_PATH_TOKEN')) as vclient:
          # first of all revoke all permissions removing policy
          vclient.delete_policy(clientpath+'-policy')
          # remove all tokens using own paret accessor token storing
          accessor_token = client.read('secret/path_manage/'+clientpath)
          assert accessor_token
          vclient.revoke_token(accessor_token['data']['accessor_token'], accessor=True)
          vclient.delete('secret/clusters_manage/'+clientpath)
          # remove all stored data(keys). Make this path available again ???
      except Exception as e:
        l.error("Unexpected error %r", e)
        return {ServerResponseFields.STATUS : ServerResponseStatus.FAILURE}
    return {ServerResponseFields.STATUS : ServerResponseStatus.SUCCEEDED}
```

KMS_Client.py

```
import socket, json
from KMS_Daemon import ServerResponseStatus, ServerError, ServerOperation,
ServerResponseFields, ClientRequestFields

class KeyManagementServerClient:

  def __init__(self, serverAddress, serverPort, logger):
    with logger as l:
      self._serverAddress = server_address
      self._serverPort = serverPort
      self._logger = l


  def del_key(self, token, path, keyID):
    resDict = self._make_request(token, ServerOperation.DELETE_KEY, [path, keyID])
    return self._resDictStatusTest(resDict)


  def delete_path(self, path):
    resDict = self._make_request(None, ServerOperation.DELETE_PATH, path)
    return self._resDictStatusTest(resDict)


  def create_path(self, path):
    resDict = self._make_request(None, ServerOperation.CREATE_PATH, path)
    if resDict == None:
      return None
    try:
      status = resDict[ServerResponseFields.STATUS]
      if status != ServerResponseStatus.SUCCEEDED:
        return None
      token = resDict[ServerResponseFields.TOKEN]
      return token
    except Exception as e:
      self._logger.error("Unexpected response structure - %r", e)
      return None


  def make_key(self, token, path):
    resDict = self._make_request(token, ServerOperation.GENERATE_KEY, path)
    if resDict == None:
      return None
    try:
      status = resDict[ServerResponseFields.STATUS]
      if status != ServerResponseStatus.SUCCEEDED:
        return None
      ekey = resDict[ServerResponseFields.EKEY]
      ekey_id = resDict[ServerResponseFields.EKEY_ID]
      return ekey, ekey_id
    except Exception as e:
      self._logger.error("Unexpected response structure - %r", e)
      return None


  def get_key(self, token, path, keyID):
    resDict = self._make_request(token, ServerOperation.OBTAIN_KEY, [path, keyID])
    if resDict == None:
      return None
```

```
    try:
      status = resDict[ServerResponseFields.STATUS]
      if status != ServerResponseStatus.SUCCEEDED:
        return None
      ekey = resDict[ServerResponseFields.EKEY]
      return ekey
    except Exception as e:
      self._logger.error("Unexpected response structure - %r", e)
      return None


  def rotate_key(self, token, path, keyID):
    resDict = self._make_request(token, ServerOperation.ROTATE_KEY, [path, keyID])
    if resDict == None:
      return None
    try:
      status = resDict[ServerResponseFields.STATUS]
      if status != ServerResponseStatus.SUCCEEDED:
        return None
      ekeyID = resDict[ServerResponseFields.EKEY_ID]
      return ekeyID
    except Exception as e:
      self._logger.error("Unexpected response structure - %r", e)
      return None


  def _resDictStatusTest(self, resDict):
    if resDict == None:
      return None
    try:
      status = resDict[ServerResponseFields.STATUS]
      return (status == ServerResponseStatus.SUCCEEDED)
    except Exception as e:
      self._logger.error("Unexpected response structure - %r", e)
      return None


  def _make_request(self, token, op, param):
    with self._logger as l:
      request = {
        ClientRequestFields.TOKEN   = token,
        ClientRequestFields.OP     = op,
        ClientRequestFields.PARAMS   = param
      }
      return self._request_server(request)


  def _parce_response(self, sock):
    with self._logger as l:
      fragments = []
      try:
        while True:
          chunck = sock.recv(self._fragmentSize)
          if not chunck:
            break
          fragments.append(chunck)
      except Exception as e:
        l.error("Unexpected error on recieve: %r", e)
        sock.close()
        return None
```

```
    data = b"".join(fragments)
    try:
      response = json.loads(data)

      status = request[ServerResponseFields.OP_STATUS]
      msg = request[ServerResponseFields.USR_MSG]
      res = request[ServerResponseFields.RES_DICT]
    except Exception as e:
      l.error("Response parsing failed! %r", e)
      return None

    if status != ServerResponseStatus.SUCCEEDED:
      l.error("Server failed running operation: %r", ServerError(msg))
      return None

    return res


  def _request_server(self, requestDict):
    with self._logger as l:
      try:
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.connect(self._server_address)
      except Exception as e:
        l.error("Can't establish connection with server %r : %r - %r",
          self._serverAddress, self._serverPort, e)
        return None

      try:
        request = json.dumps(requestDict)
      except Exception as e:
        l.error("Can't pack request %r - e", requestDict, e)
        sock.close()
        return None

      try:
        sock.sendall(request)
      except Exception as e:
        l.error("Can't send request to server %r - e", requestDict, e)
        sock.close()
        return None

      res = self._parce_response(sock)
      sock.close()
      return res
```

start_kms_daemon.py

```
import logging
import os
from KMS_Daemon import KeyManagementServerDaemon

logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)
KMS = KeyManagementServerDaemon(os.environ['KMS_ADDRESS'], os.environ['KMS_PORT'],
logger)
```

kms_cipher_tests.py

```
# this is just logical encrypt/decrypt use-case specific for OStorage imitation

import logging
from KMS_Client import KeyManagementServerClient
from Crypto.Cipher import AES

logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)
KMSC = KeyManagementServerClient(os.environ['KMS_ADDRESS'], os.environ['KMS_PORT'],
logger)

clientpath = 'USA_STORAGE/UID12345'

# first, register path on user creation
token = KMSC.create_path(clientpath)

# Encryption case
text = 'A really secret message. Not for your eyes'

ekey, ekey_id = KMSC.make_key(token, clientpath)
encryption_suite = AES.new(ekey[:128], AES.MODE_CBC, ekey[128:])
encrypted_text = encryption_suite.encrypt(text)
# forget ekey

# let's rotate probably-leaked value
ekey_id = KMSC.rotate_key(token, clientpath, ekey_id)

# Decryption case
ekey = KMSC.get_key(token, clientpath, ekey_id)
decryption_suite = AES.new(ekey[:64], AES.MODE_CBC, ekey[64:])
restored_text = decryption_suite.decrypt(encrypted_text)
assert restored_text == text

# Remove user case
KMSC.delete_path(self, clientpath)
```