CALIFORNIA STATE UNIVERSITY, NORTHRIDGE

Multi-user Search-able Attribute Based Encryption for Outsourced Big Data

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Software Engineering

by

Kyle Astudillo

December 2022

The thesis of Kyle Astudillo is approved:

Dr. Robert D Mcllhenny	Date
Dr. Adam B Kaplan	Date
Dr. Mahdi Ebrahimi, Chair	Date

California State University, Northridge

Table of Contents

Signature page	ii
Abstract	vi
Chapter 1: Introduction	1
Chapter 2: Related Work	3
Chapter 3: System Models	7
Chapter 4: Proposed Methods	17
Chapter 5: Experiments	24
Data Collection	27
Chapter 6: Results	38
Chapter 7: Conclusion	40

List of Tables

5.1 Data Summary for abe-base-enc 30
5.2 Data Summary for abe-base-dec 30
5.3 Data Summary for abe-bin-enc 31
5.4 Data Summary for abe-bin-dec 31
5.5 Data Summary for abe-lorem-enc 31
5.6 Data Summary for abe-lorem-dec 31
5.7 Data Summary for aes-base-enc 31
5.8 Data Summary for aes-base-dec 33
5.9 Data Summary for webserver-create 35
5.10 Data Summary for webserver-read 36

List of Figures

2.1 OpenABE Library Architecture from openABE Design Document [17]. . 3
2.2 KP-ABE Key Generation Process 4
2.3 KP-ABE Encryption Process 5
2.4 KP-ABE Decryption Process 5
2.5 Morales-Sandoval's System Model for Document Sharing and Retrieval in the Cloud. [18]. 6

3.1 Generic Stakeholders with Relationship Graph 7
3.2 Technology Stack for Multi-User KP-ABSE CURD system 8
3.3 Deployment Node Map 12
3.4 Use Case Diagram 13
3.5 Model Diagram 15

4.1 Create Sequence Diagram 19
4.2 Delete Sequence Diagram 20
4.3 Read Sequence Diagram 21
4.4 Read ID Sequence Diagram 22
4.5 Update Sequence Diagram 23

5.1 Encryption KP-ABE Real Time Graph 27
5.2 Encryption KP-ABE User Time Graph 27
5.3 Encryption KP-ABE System Time Graph 28
5.4 Decryption KP-ABE Real Time Graph 28
5.5 Decryption KP-ABE User Time Graph 29
5.6 Decryption KP-ABE System Time Graph 29
5.7 Encryption KP-ABE Max Memory graph 30
5.8 Decryption KP-ABE Max Memory graph 30
5.9 Encryption AES Real Time Graph 32
5.10 Encryption AES User Time Graph 32
5.11 Encryption AES System Time Graph 33
5.12 Encryption AES Max Memorygraph 33
5.13 Decryption AES Real Time Graph 34
5.14 Decryption AES User Time Graph 34
5.15 Decryption AES System Time Graph 35
5.16 Decryption AES Max Memorygraph 35
5.17 Create request Count Graph 36
5.18 Create process Time Graph 36
5.19 Read request Count Graph 37
5.20 Read process Time Graph 37

Abstract

Multi-user Search-able Attribute Based Encryption for Outsourced Big Data

By

Kyle Astudillo

Master of Science in Software Engineering

Large amounts of data is generated every day as well as the need to ingest, process, store, and maintain all that data. The scale of data is so large that normal file access and processing present different challenges. Data owners can attempt to store and share data on premise but then lose out on savings from the cloud. Data owners can store all their data on the cloud to save money but then run into a myriad of security challenges. Attribute based encryption is the key to tackling these challenges. A method leveraging key policy attribute-based encryption (KP-ABE) to allow multiple users to ingest data into a system encrypted then have other users search over its contents to receive files while using KP-ABE as a distributed authorization method will be presented, tested against AES, and system metrics will be presented. The solution presented will allow users to use a representational state transfer application programming interface to interact with the system while not having to deal with auxiliary files or encryption libraries to upload documents into the cloud and be encrypted at rest.

Chapter 1: Introduction

Organizations now have clouds as an option when deciding how to deploy their infrastructure and file storage. When [the cloud is combined with] consumption based billing, service abstraction, and scalability, cloud computing can offer compelling value to organizations deploying services [19]. Sedayao highlights major incentives that organizations have when choosing a cloud infrastructure. A major barrier blocking enterprise adoption of cloud computing, as organizations are concerned about retaining the confidentiality of their data [19]. Sedayao brings to attention problems that are inherent with cloud solutions and even brings up a problem when data is not encrypted at rest, encrypting data when sitting in storage, a cloud computing service administrator can simply just read the data stored in the service[19]. This problem is compounded when organizations want to share files with business partners, accounting firms, or law firms when encrypted files leave a trusted system and have to go into an untrusted system. With organizations now incentivized to reap the benefits of cloud pricing, there are added risks of introducing new security concerns since all their data is now in reach of public access if there is a hole somewhere files could get compromised.

Attribute based encryption(ABE) is actually a perfect fit for this type of situation by not only handling file encryption allowing files to hide in plain sight in the event of a security breach. When sharing data with another organization, ABE enforces an authorization method within the decryption key itself allowing only users with the correct key with attributes satisfying its policies to actually decrypt the file [17]. An example straight from openABE, an open source implementation of attribute based encryption types, command line interfaces documentation describes a Key Policy Attribute Based Encryption(KP-ABE) where attributes of an emails meaning who the emails was from, to whom the email was to, and the date it was sent are all encoded as attributes [13]. Then, as a data owner who

wants to share files, maybe 10000 emails, can generate a key to only decode emails where it was between certain users and even between a certain date ranges[13]. However this introduces a problem where users have all the encrypted data but now have to decrypt each file to see which files they have access to. Morales and his team proposed a solution using ABE and show how to encrypt data into the system leveraging an extraction of keywords from documents so that users can get encrypted documents they have access to with an encrypted search query that they call Attribute Based Searchable Encryption(ABSE) [18]. With a system that Morales and his team proposed it would be possible for sharing data with many users with KP-ABE keys.

Contributions to existing literature is proposing an extension to the system such that data owners can allow multiple users to ingest data into the system making it a many to many relationship with data producers to data consumers. Describing a REST API to interact with the system to upload and download files so that users do not need to have any type of encryption software or encryption keys on their system making this system cross compatible with multiple operating systems, even though openABE is a cross compatible library already. The system still allows for users to maintain the encryption software on their system so that they may store files into the system encrypted with their own extracted keywords, and receive data encrypted out of the system making it encrypted from end to end.

Chapter 2: Related Work

Attribute-based encryption (ABE) is paramount to the project, and everything is built on top of this technology. ABE first introduced in Ciphertext-Policy Attribute-Based Encryption by J. Bethencourtand, A. Sahai and B. Waters provide the first construction of a ciphertext-policy attribute-based encryption (CP-ABE) [14]. ABE differentiates itself from other encryption techniques by embedding access control into cipher-text [17]. This ability to embed access control information into the file itself allows for the ability of checking permissions client side during the decryption phase without the need for a centralized identity and access management system. Leveraging these properties will allow a use case where the storage, retrieval of encrypted data in the cloud is cost-effective while maintaining data security to its data owner at scale.

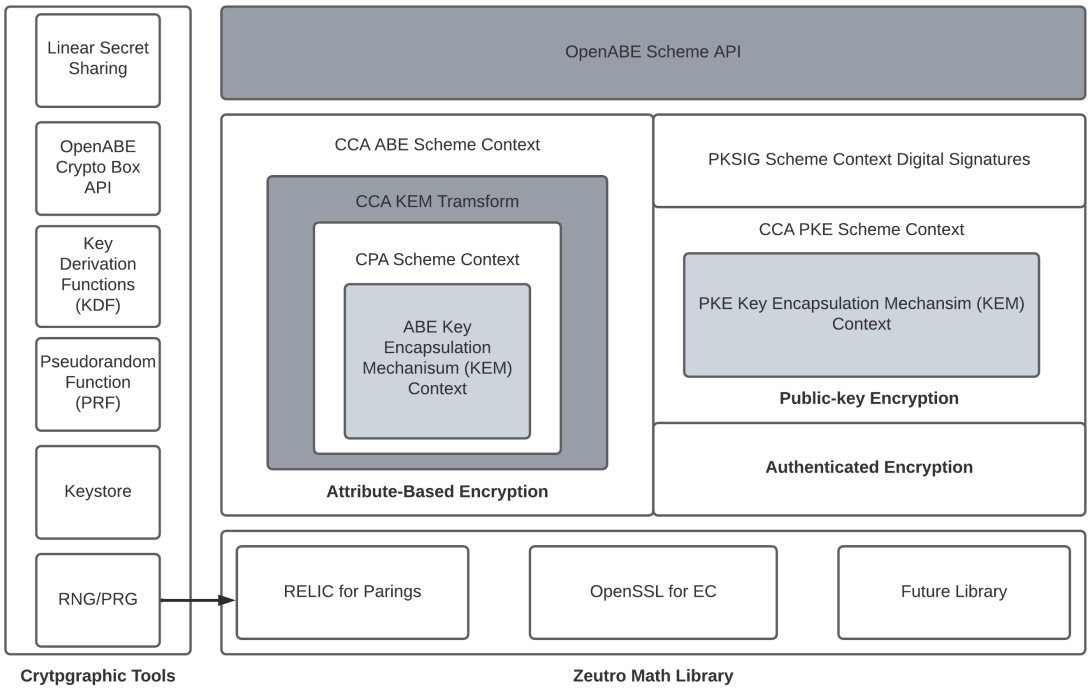

Figure 2.1: OpenABE Library Architecture from openABE Design Document [17].

The below Figure 2.1 is an architectural diagram of how OpenABE works in order

to give the ease of use but conveys the complexities of the internal components at work to put ABE in the hands of the developer. OpenABE provides three types of policies of encryption public-key (PKE), ciphertext-policy (CP-ABE), and key-policy (KP-ABE)[13]. CP-ABE and KP-ABE are both ABE but handle how they store policies and attributes differently which give both methods different behaviors when used. CP-ABE stores policy trees in the ciphertext and attributes with the users [14], while KP-ABE stores policy trees with the users and attributes with ciphertext [16]. An example of a behavior of CP-ABE is describing a user by name, job title, age, or rank then using a file to describe who can gain access to the content by describing a type of job title or how high of a rank [13]. An example of a behavior of KP-ABE is the inverse where attributes describing the file are used and then the user will be granted a policy of what files they can gain access to by describing a date or content type [13]. This project will make use of KP-ABE however an extension to CP-ABE can be made relatively easily. The process of a KP-ABE to share data is as follows. First is a setup phase in which a master secret key(MSK) and public parameter store(MPK) are generated[13]. Second using a MSK and MPK a key can be generated[13]. The key will have a policy tree that describes what kind of content may be accessed, then simply needs to be given to anyone the data owner wants to share data with[13].

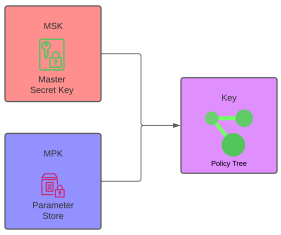

Figure 2.2: KP-ABE Key Generation Process

To encrypt data in a KP-ABE system a MPK, a random number, attributes, and a content file are needed to create an encrypted file[13].

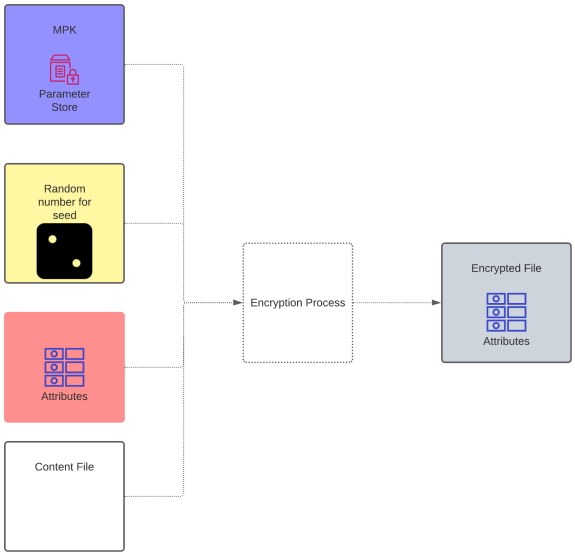

Figure 2.3: KP-ABE Encryption Process

To decrypt data in a KP-ABE system a MPK, a key, and encrypted file are used if the policy, attributes, and parameter store align then the content will be decrypted if not the decryption process will not succeed[13].

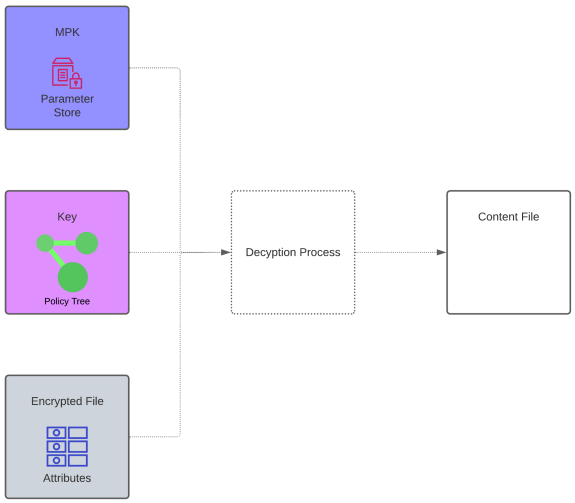

Figure 2.4: KP-ABE Decryption Process

Morales-Sandoval and his team then extended these ABE processes so that a data owner could not only encrypt files but store them in the cloud through a cloud service

provider (CSP) which indexed them by extracting words so a data consumer could search over them as well through a process called key-policy attribute based searchable encryption (KP-ABSE) shown in Figure 2.5. Figure 2.5 shows the processes of how a data owner

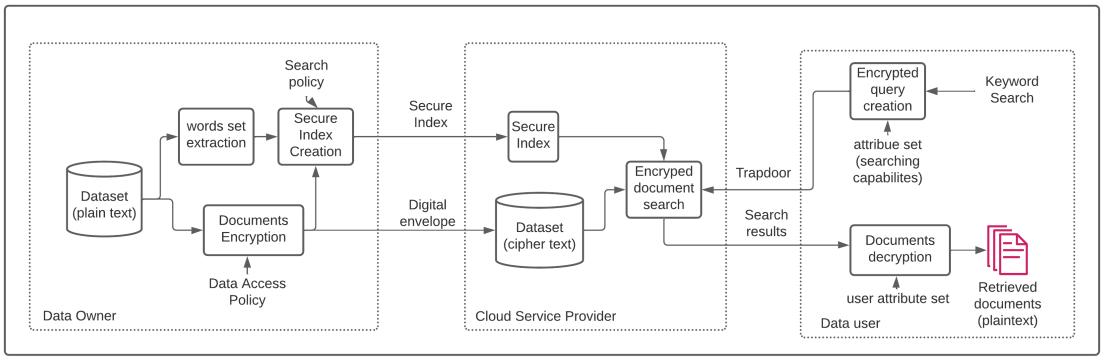

Figure 2.5: Morales-Sandoval's System Model for Document Sharing and Retrieval in the Cloud. [18].

would take their content of plain text data in a database then extract words to create a secure index to be used by an encrypted document search node, while those same documents are encrypted into cipher text and placed into a database hosted by a CSP. Then data users would be able to create an encrypted query to search through documents in which they could decrypt to get plain text documents. The project presented in the paper is inspired by Morales-Sandoval and his teams system to an ABE approach for storage, sharing and retrieval of encrypted data and will add contributions that extend this system model to add an idea of data producers to help ingest content into the system, data editors to help maintain the content, and will rework the system to accommodate an API for creating, updating, reading, and deleting content from the system.

Chapter 3: System Models

This project can solve use cases where big data from data owners need to share data with many users and wants to leverage cloud service providers to reduce cost to store data compared to on premise solutions while maintaining security of access to private data. The proposed solution allows data owners to grant permissions to data consumers and producers. The use of Attribute based encryption allows for the penalty of checking Access Control Lists also known as ACL to the client at read time which grants the benefits of reducing the time to search through data and reducing server compute time, when compared to systems where bulk files are encrypted and then decrypted in data processing to find files that were not indexed. Stakeholders that would benefit from this technology would be large organizations such as accounting and legal firms [17].

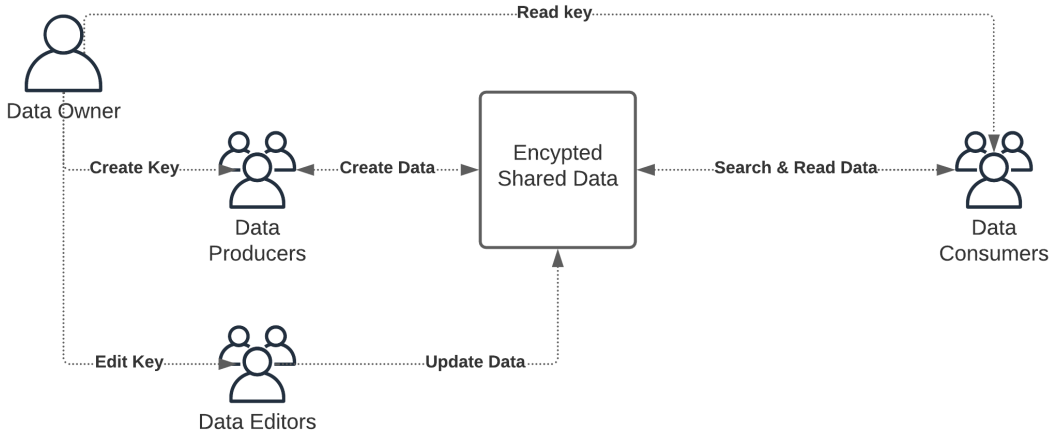

Figure 3.1: Generic Stakeholders with Relationship Graph

Figure 3.1 shows a very high level idea of a data owner giving access to different stakeholders however key generation is more complicated and will be described later.

This project is made of many components and services that work together to make up the entire solution. The technology stack used in this project include AWS EC2, Aws RDS,

AWS S3, AWS Fargate, OpenLDAP, AWS Route 53, AWS CloudFront, AWS SQS, python, OpenABE, and CentOS.

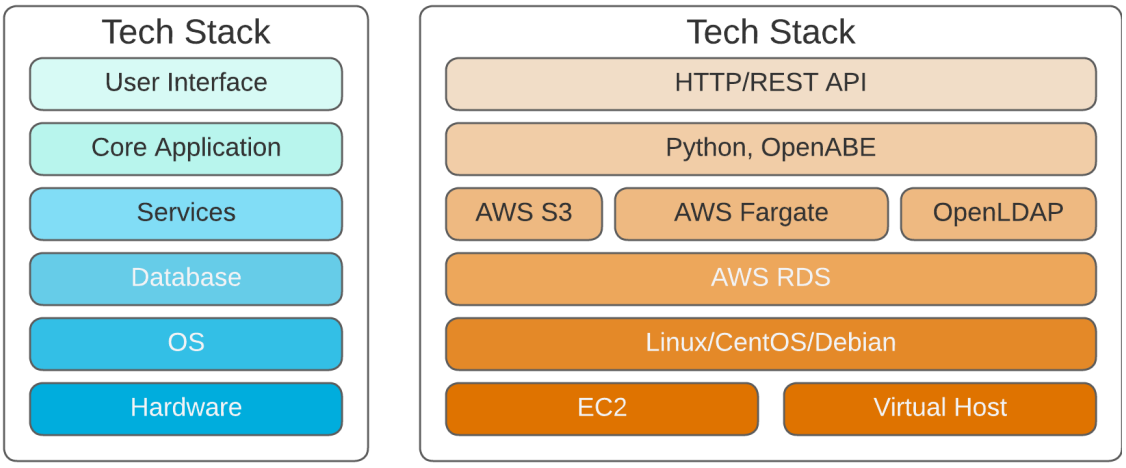

Figure 3.2: Technology Stack for Multi-User KP-ABSE CURD system

AWS EC2: Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides secure, resizable compute capacity in the cloud [2]. Within the context of this project EC2 is used for development, building, deployment, testing and node management. The ability of EC2 to quickly be resized and any easy source of compute capacity allows for development and testing of new features to be added to the project quickly. EC2 is also used for building under the scope of generating container images to use in AWS Fargate which will be described later. Another use for EC2 is a node management which means it is used almost as a jump box, a node on one network used to access another node on a restricted networks [4], to help manage nodes within the system.

AWS RDS: Amazon RDS for SQL Server is a Relational Database Service (RDS) that makes it easy to set up, operate, and scale SQL Server deployments in the cloud [5]. RDS is used as the project's model for storing information that the core application needs. There are two RDS within the project the first being secured in the data owner subnet which is used for indexing critical information for the data owner and critical information for more

intimate operations for the project. The second RDS service is hosted on the csp subnet which holds information such as the secure encrypted index and REST responses to users. This second RDS is meant to be a more public resource and is to be replicated to multiple zones for quicker read access to the rest of the system.

AWS S3: Amazon Simple Storage Service (Amazon S3) is an object storage service that offers industry-leading scalability, data availability, security, and performance [7]. S3's role in the project is to save documents used in the core application to the cloud and allow users easy access to documents anywhere. The project benefits greatly from using the S3 service in two major areas for the project. The first S3 service is hosted in the data owner subnet and is responsible for saving plaintext documents for the data owner. The second S3 service is hosted in the csp subnet as is meant to be a more public resource which is responsible for saving cipher-text documents, and its content are cached on edge nodes in CloudFront to be described later.

AWS Fargate: AWS Fargate is a serverless compute engine for containers that works with both Amazon Elastic Container Service (ECS) and Amazon Elastic Kubernetes Service (EKS) [3]. The project takes advantage of Fargate so that only the core application webserver needs to be deployed as a container. Fargates allows the project to take advantage of the cloud's elasticity so that the application auto scales when needed and will deallocate resources when not needed. This in conjunction with other components in the system such as S3 hosting ciphertext documents being in multiple zones while being cached on CloudFront edge nodes reduce the time for end users to access data in the system.

OpenLDAP: is an open source implementation of the Lightweight Directory Access Protocol [11]. The service is used to generate groups, add users, and handle basic authentication for protected API endpoints.

AWS Route 53: Amazon Route 53 provides highly available and scalable Domain

Name System (DNS), domain name registration, and health-checking web services[6]. The scope of Route 53 in this project is simply to route requests from the domain to the correct resources.

AWS CloudFront: Amazon CloudFront is a web service that gives businesses and web application developers an easy and cost effective way to distribute content with low latency and high data transfer speeds [1]. CloudFront has a very close relationship with the second S3 Bucket which stores ciphertext content. Within this project CloudFront is used to cache ciphertext to edge nodes that will reduce the time it takes to read access content to data consumers in the system.

AWS SQS: Amazon Simple Queue Service (SQS) is a fully managed message queuing service that enables you to decouple and scale microservices, distributed systems, and serverless applications [8]. A SQS is placed between Fargate webservers and is meant to buffer excess requests to a non-scaled data-owner environment in the case that the key operations server get overloaded with requests before the services can be processed.

OpenABE: a cryptographic library that incorporates a variety of attribute-based encryption (ABE) algorithms, industry standard cryptographic functions and tools, and an intuitive application programming interface (API) [10].

Centos 8.0: Community-driven free software effort focused around the goal of providing a rich base platform for open source communities to build upon [9].

A deployment graph below in figure 3.3 illustrates how this project can be deployed between multiple organizations such that they can share data between multiple networks. From the bottom right you can see multiple nodes representing client computers with different operating systems such as windows, os, and linux. These clients within an organization can connect to a webserver, which can store mpk files, is used to access the webserver located top middle of the figure 3.3 this webserver is used for interacting with

requests and handles the proposed methods later described in the following chapter. The top left describes how the private side of the system will be connected, while the top right describes how the public facing side will be connected. On the private side a S3 Bucket will be responsible for saving all plain text data ingested into the system. On the private side AWS RDS is used with tables later described. An additional aws ec2 server running centos is used to run one more webserver called key-operations which is responsible for openABE for encryption and decryption processes,as a openABE wrapper, since openABE is difficult to deploy and managing all the extra files such as msk and mpk files would be difficult to manage in a more distributed solution, therefore a more centralized solution is being proposed. On the top right is anothxer AWS RDS which is where Read requests will be processed to get s3 object keys for files in a search query. The top right node aws s3 bucket is responsible for storing all cipher text files for read requests to download.

Users break down into four groups derived from Morales-Sandoval and his team are the idea of the data owner, and data consumers [18]. This project extends the functionality by moving a role that was once in data owner and made into a new role of data producers, and a new role data editor. Each of these roles interact with different use cases within the system such as data consumer and a read action. Dark filled in circles are wrappers for openABE while white circles are actions that have been added on top of these to full-fill use cases that users will leverage to help fulfill their role. Arrows in the graph below are the relationships between use cases within the system.

The following is an explanation for use cases.

- Generate-mpk: A wrapper for openABE and is used to generate mpk files in the system.
- Generate-msk: A wrapper for openABE and is used to generate msk files in the system.

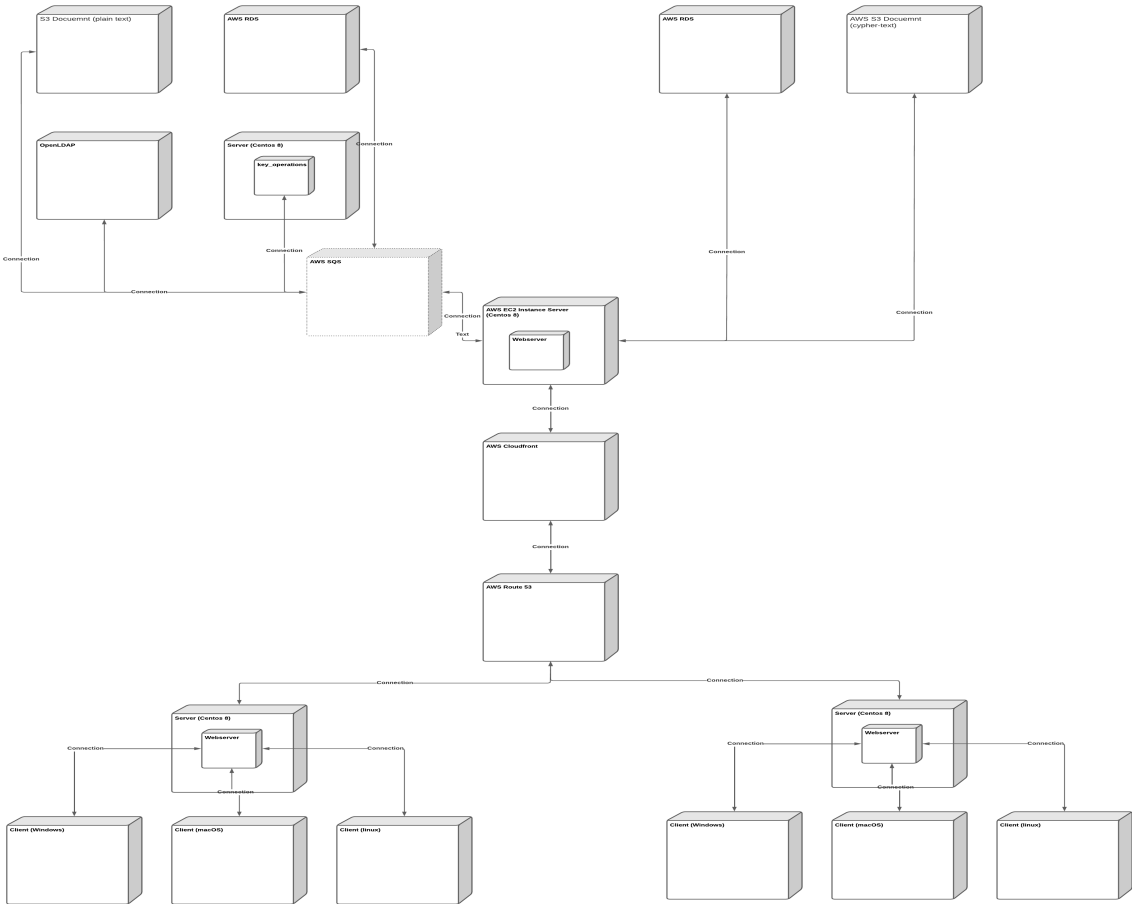

Figure 3.3: Deployment Node Map

- Generate-key: A wrapper for openABE and is used to generate a key file for the user and is stored in the private server and returned to the data owner to distribute to other users.
- Encrypt: A wrapper for openABE and is used to encrypt files on data producers and data editors requests.
- Decrypt: A wrapper for openABE and is used to decrypt files on data editors and data consumers requests.
- Create-org: An action that data owners have exclusive access to which is used to create an organization such as “TestMovingOrg” and that will use the wrappers generate-mpk and generate-msk.

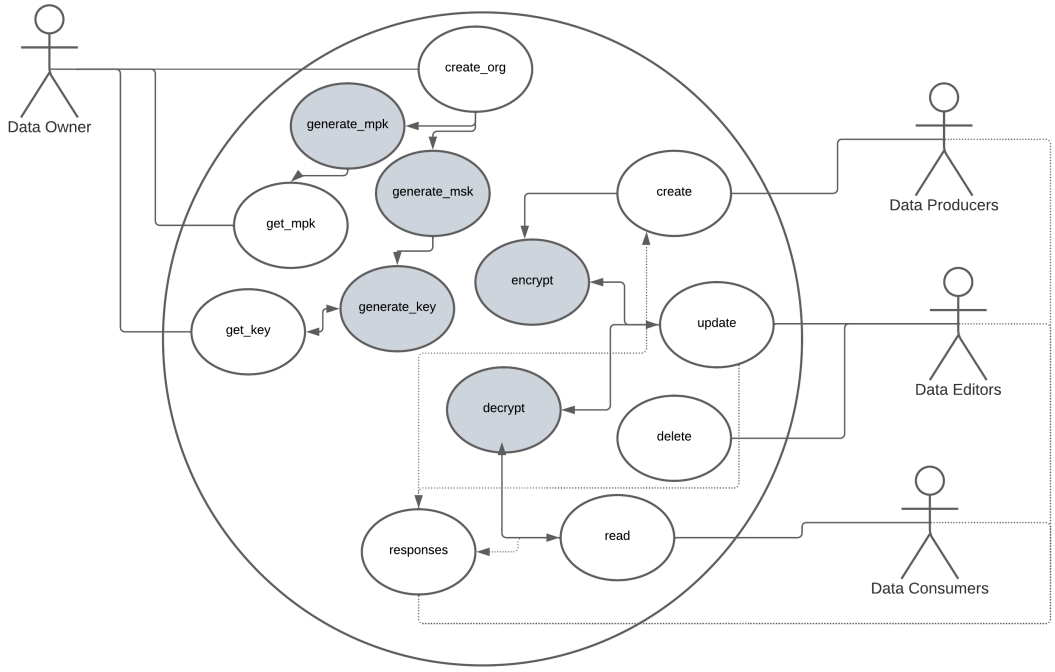

Figure 3.4: Use Case Diagram

- Get-mpk: An action that data owners have exclusive access to which is used to generate an mpk file to distribute to webserver nodes.
- Get-key: An action that data owners have exclusive access to which is used to generate a user key that is stored on a private server and returned to the data owner to distribute to data producers, editors, and consumers.
- Create: An action that is exclusive to data producers which is how content is ingested into the system.
- Update: An action that is exclusive to data editors and is used to update existing information in the system.
- Delete: An action that is exclusive to data editors and is used to remove existing information from the system.
- Read: An action that is exclusive to data consumers and is used to pull out information from the system.

- Responses: is just here to show which use cases generate a response with additional information such as a read search query will return a read response with a list of urls to do reads on in a separate request.

The models used to implement the system are broken down into two groups. One is colored in red for public access, and blue is for private access used within the system. Derived from the original work Morales-Sandoval and his team is the idea of the data owner that stores plain text data and generates a set of word set extraction that are representative of documents for searching through the encrypted documents [18]. From the previous work retrieving data of interest is possible only if the attributes used to create the trapdoor satisfy the access policy used during keywords encryption (index creation) [18]. The system being proposed is different from previous work such that extracted word set in this system are on content and not the attributes being used from the policies within trees. Therefore this system being proposed does not contain the property of a search query being an Access Control Aware Search[15]. The system proposed will search based on words within the content of files and index them for search queries so it is possible to obtain files that may not be decrypted to reflect a system such as windows where a user may see a file but not open its contents. This allows a user to query data owners for access. If a user wishes to take extra precaution they may encrypt the file within their system and upload a secure extracted keyword set that will be used to query upon.

The following are data stored within the system.

- siid: Secure Index Identifiers, generated when data is put into the system this is used to distinguish between files within the entire system.
- index: Just like data is used to index items in each of the existing tables.
- s3-plain-text-key: AWS S3 keys that contain data stored as plain text in aws, which stores the original file that was uploaded, has not been encrypted, and only available

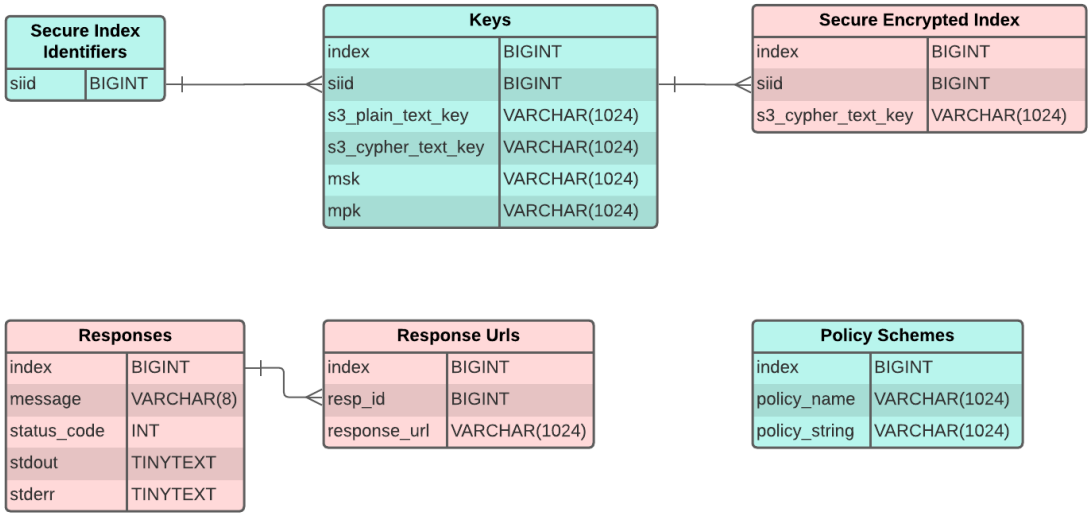

Figure 3.5: Model Diagram

to data owners.

- s3-cypher-text-key: AWS S3 keys that contain cipher text data stored in aws. This is a file that has been encrypted and is ready to download for data consumers.
- msk: The name of the msk file that was used on the data put request.
- mpk: The name of the mpk file that was used to decrypt or encrypt the data.
- message: A message back to the user as to the status of the request such as error, not complete, and complete.
- status-code: The HTTP Status code that users have received from their responses to requests into the system, such as 200 successful.
- stdout: Text response back from the server is usually an empty string on a successful request.
- stderr: Text response back from the server that something went wrong.
- resp-id: ID used for an additional request back for the results to the search query.
- response-url: If the query took less than the allotted time the results to the query will be returned here.
- policy-name: The name of the policy to be used for encryption and decryption

methods.

- policy-string: The policy to be used for the encryption and decryption methods.

The following are tables stored within the system and have private access.

- Secure Index Identifiers: Table that stores all the siids in the system
- Keys: Table that stores all information on how to interact with a file within the system.
- Policy Schemes: Table that stores policies for what attributes to generate mpk and msk files with.

The following are tables that are stored within the system and have public access.

- Secure Encrypted Index: Table that is used on public request to link the search queries from the private side to cipher-text files that data consumers can use to download.
- Responses: Table used for storing all responses back to any requests that passed through the system.
- Response Url: Table used to full-fill search query results to requests made within the system.

Chapter 4: Proposed Methods

Based off of Morales-Sandoval and his team's project the implementation for the entire system methods are the following. Word set extraction a process on the webserver that when a plain text file comes in words are extracted so that search queries can be made against the words and part of this process is storing the file within the private database, as an optional this process can be bypassed if the request provides keywords in the word-set-extraction json payload to use instead. Secure index ID creation is the process in which a public webserver requests a SIID from the private side webserver for a plain text file to be saved within the system. Secure Index ID to file is the process in which a webserver sends an siid to the public database to receive a s3-cipher-text-key. Event Return Code URL is a response with information such as a status-code, message, stdout, stderr and an optional text data within the content payload so that users may poll for results on a separate url if the query takes a while to compile the requested file list. Document Encryption is the process in which the webserver makes a call to key-operations wrapper with information like which organization, and key to encrypt with and will return a cipher text file. The encryption process can be bypassed by users if they encrypted the files before ingesting into the system using a kpabe-encrypted is true parameter. Upload Document is a process in which both the plain text file is uploaded to the private s3 bucket and cipher text file is uploaded to the public s3 bucket. Update Event is the process in which a webserver will update all of the databases with how the response was handled which will store all the s3 keys on the private db and s3-cipher-text-key, status-code, and message on the public db. Delete Document is a process on the webserver that will take an siid and will remove the siid from the system which means users will no longer be able to obtain files through the webserver. Encrypted document search the process in which a webserver will query the private db for a list of siids that match the search query. Get Document is the process in which a web server

retrieves a document from s3 to return to the user. Document Decryption is a process where a request is made with information like which organization, and key to decrypt with and a plain text file will be returned.

The create operation is one of the most resource intensive operations that take a content file from a data producer and stores the file into the cloud so that data consumers may access the file. The process involves a client making a request to the web server with a file with content, a mpk file to use for encryption, and a config file in json with parameters such as organization, and policy name. On the webserver once the request is received the word set extraction process begins where the file is stored for search queries. Next the webserver then makes a request to the CSP database for a secure index identifier. Then a response is returned to the user with a response-id that they can query for to get the status of there upload. A separate internal request is made to the key operations server to encrypt the content file which will return an encrypted cipher text file back to the webserver. With the cipher text file the webserver can now start the upload process which then stores the plain text file into a s3 bucket which returns a s3-plain-text-key and stores the encrypted file into a s3 bucket which returns a s3-cipher-text-key. With both files saved an update event takes place which stores s3-plain-text-key, s3-cipher-text-key, ssid, msk, mpk, status-code. Finally the system is complete and the user has the option to poll the status of their previous request and once a data producer sees status-code complete they know everything is successful.

The delete operation is reserved for data owners. Data owners need to send a response with the secure index identifier to the webserver. The webserver then uses the secure index to get the s3-cipher-text-key that is then used in the delete document process. The delete document process uses the s3-cipher-text-key to delete the cipher text file off the s3 bucket. Finally siid and s3-cipher-text-key are removed from the secure encrypted index table. Finally a response is returned back to the user with a status code. Data consumers

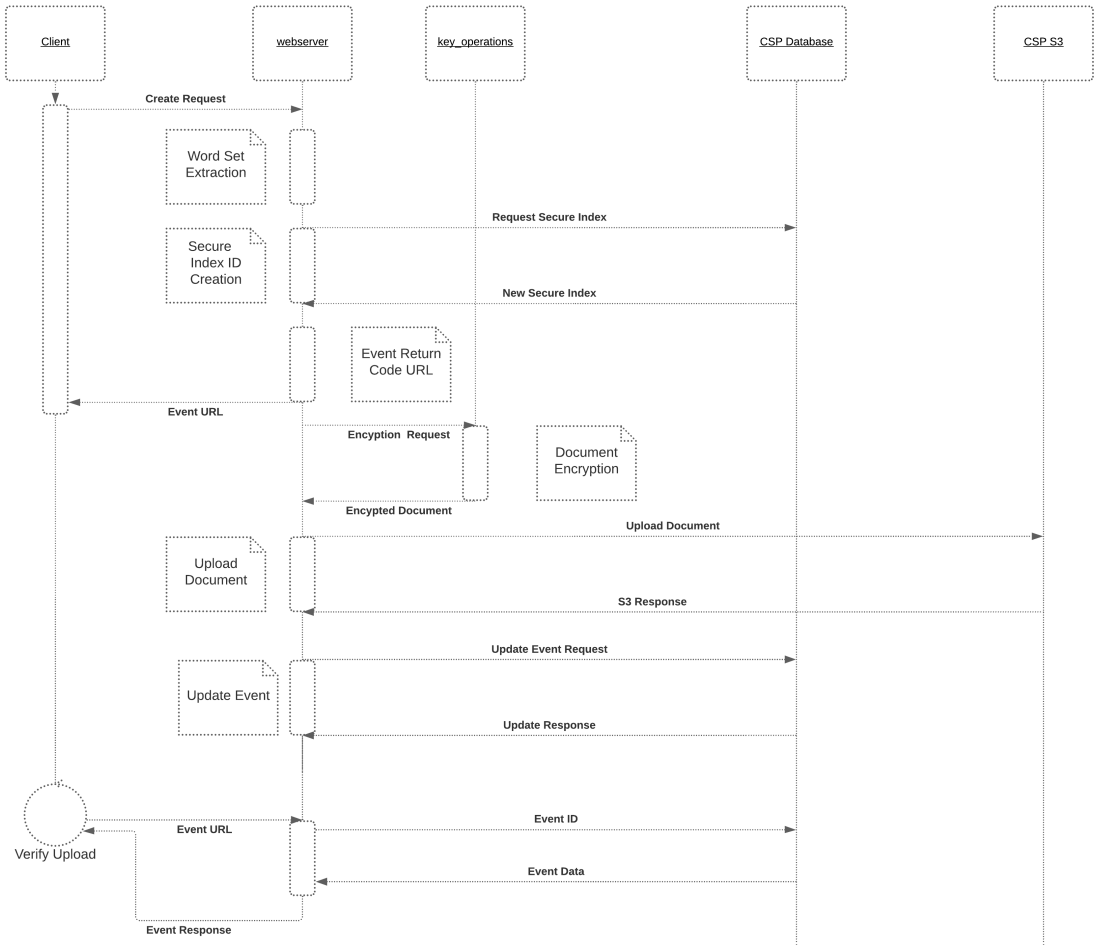

Figure 4.1: Create Sequence Diagram

will no longer have access to querying for data, and will no longer be able to download files from the system. Even though data consumers can no longer access data the plain text is still stored on the private bucket so that the data owner has a record of the content and can delete it manually from the system.

Read operations are one of the most time consuming operations within the system. Data consumers have access to this operation which consists of taking a search query and returning a list of files for the client to download. The read is more of a search operation while the read id is a more traditional read operation. The read request begins from a client to a webserver with a search query as an attribute in a json payload. The webserver then

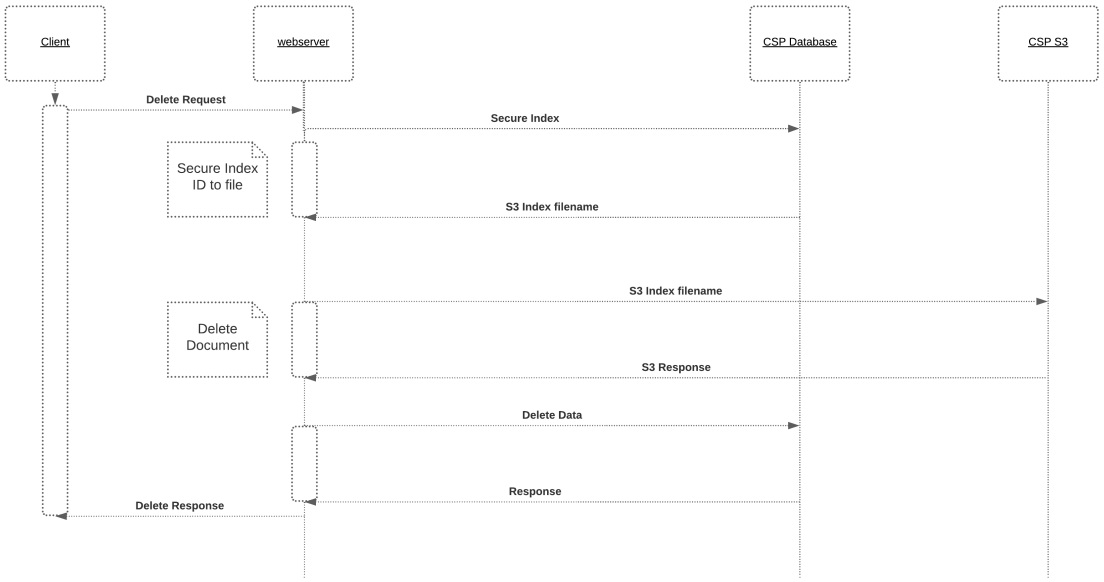

Figure 4.2: Delete Sequence Diagram

does an encrypted document search which will return a list of siids for the client to use inside a read ID request.

The read id operation is the most time consuming operation within the system due to having to decrypt the files before the return. Data consumers begin with sending a request with a ssid as a parameter in the url and mpk file. The webserver makes a request to the CSP database to obtain the s3-cipher-text-key in the response. The webserver then uses an s3 client with the s3-cipher-text-key to get the cipher text file from an aws s3 bucket. With the cipher text file returned it is sent to key operations which then decrypts the file and returns a plain text file to the webserver to return to the client.

The Update operation is the most complex of the operations. Data producers have access to modifying existing files within the system. Data producers begin with a response much like the create operation such that they need a content file, mpk file, and config file with settings such as organization, policy name, ssid, and optional keywords. The webserver then takes the response and does a word set extraction on the content file, siid

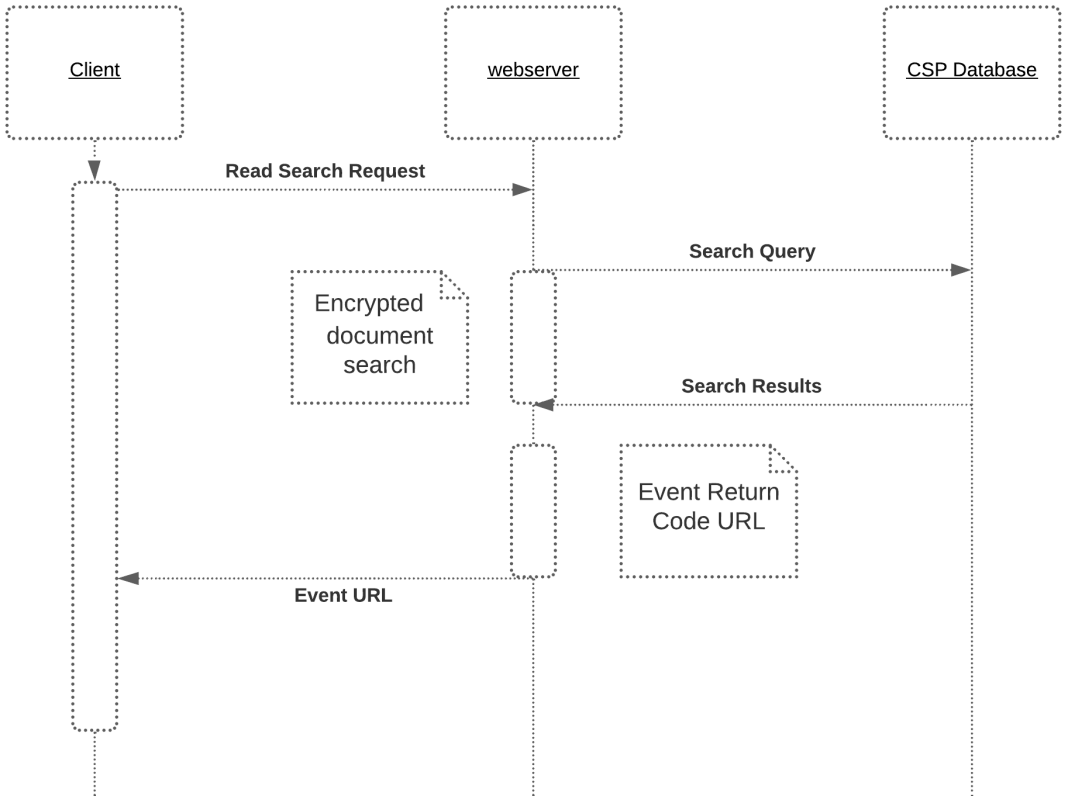

Figure 4.3: Read Sequence Diagram

is then generated and then a return code url is returned as a response to the data producer. The data producer may poll the url to see the status of the update operation in progress. The webserver then takes the content file and sends it to key operations with the mpk file to encrypt the document. Next key operations sends back an encrypted cipher text file to the web server. Then the web server begins the upload process where the plain text and cipher text files are uploaded to the respective s3 buckets. Next an update event is triggered to update and store the new siid and s3-cipher-text-key, s3-public-text-key, ssid, msk, mpk, status-code. Finally the user if requested with the event URL will show a successful status.

All create, update, read, delete operations have a very similar error message process when something goes wrong. If something goes wrong before the event URL is sent a

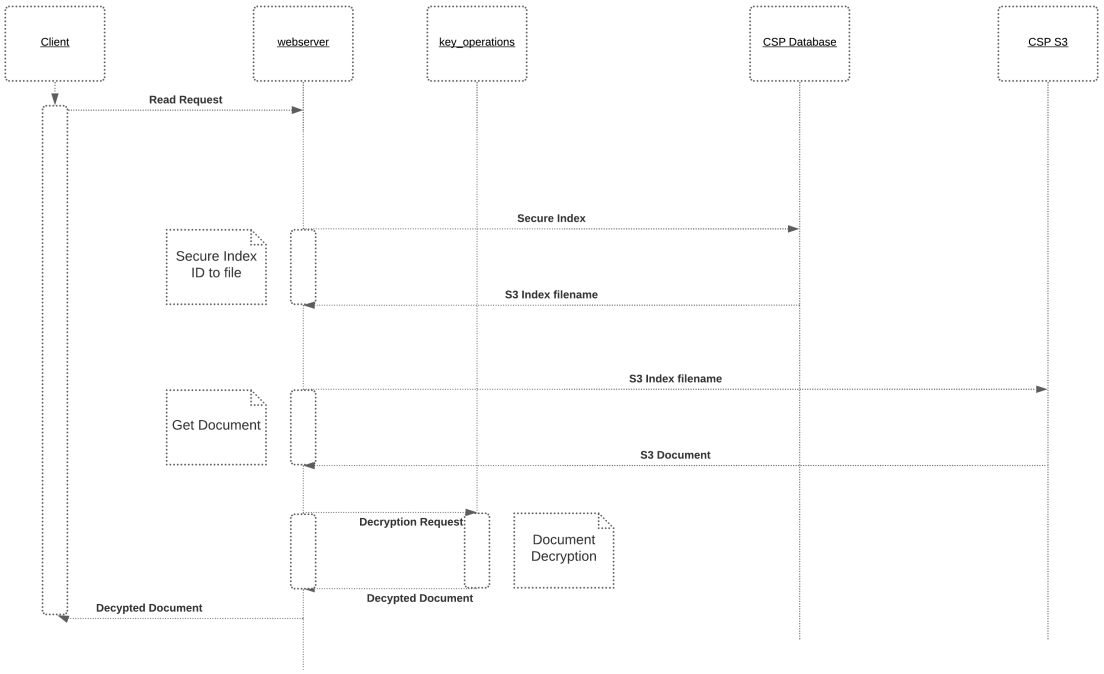

Figure 4.4: Read ID Sequence Diagram

HTTP error will be returned directly to the user, however if the event url had already been sent to the user the error message will be saved into the response table and when the user requests the event url they will be shown the error message of where things went wrong.

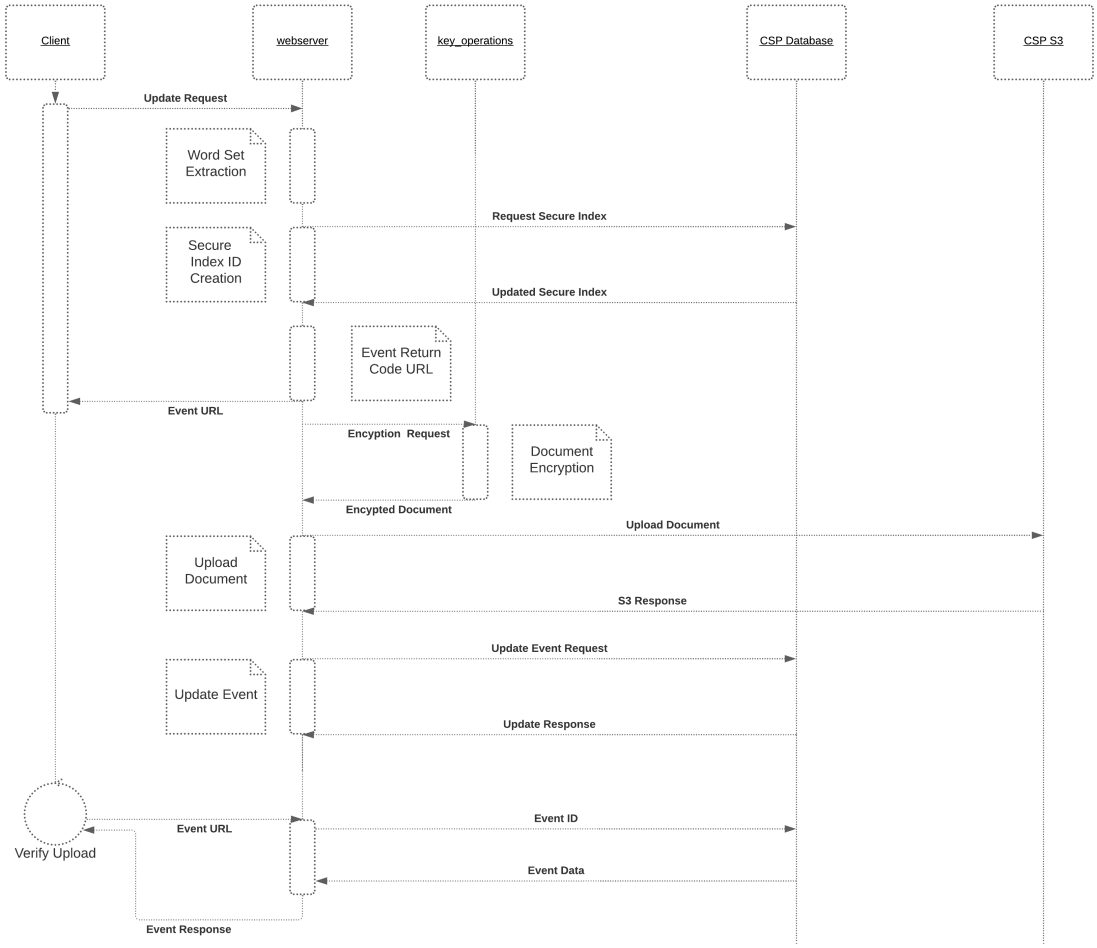

Figure 4.5: Update Sequence Diagram

Chapter 5: Experiments

OpenABE was deployed on a virtual machine running CentOS-8-Vagrant-9.3.2011-20201204.2.x8664, 1 Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz, 16MB of ram, 16 GB of SSD Memory. This virtual machine was used to run all the experiments natively without any network connections to reduce any variation between test executions. Each test ran over a 1000 iterations to get a decent sample size for the experiments. All commands were measured with the time utility. The following are the experiments abe-base-enc, abe-base-dec, aes-base-enc, aes-base-dec, abe-bin-enc, abe-bin-dec, abe-lorem-enc, abe-lorem-dec.

The first set of tests target how ABE encrypts a fully utilized text file with random characters from the start of the file to end of the file fully utilizing all 16MB with characters with no spaces. The first set of data labeled abe-base-enc are the measures of each instance of encrypting the previous files and abe-base-dec are the measures of each instance of encrypting the previous files. This first test would start by generating text files with random characters with no spaces. Once a file was generated openABE was used to encrypt the file with KP-ABE and the time for encryption was measured. Once completed openABE KP-ABE decryption was used on the file and the time for decryption was measured. Finally the original MD5SUM was checked on the original file against the newly decrypted file to verify no data loss.

The second set of tests target how KP-ABE encrypts 16MB binary files. The first set of data labeled abe-base-enc are the measures of each instance of encrypting the previous files and abe-base-dec are the measures of each instance of encrypting the previous files. This first test would start by generating text files with random characters with no spaces. Once a file was generated openABE was used to encrypt the file with KP-ABE and the time for encryption was measured. Once completed openABE KP-ABE decryption was used on the file and the time for deception was measured. Finally the original MD5SUM

was checked on the original file against the newly decrypted file to verify no data loss.

The third set of tests target how KP-ABE encrypts a fully utilized text file with random words from the start of the file to the end of the file fully utilizing all 16MB with spaces. The first set of data labeled abe-base-enc are the measures of each instance of encrypting the previous files and abe-base-dec are the measures of each instance of encrypting the previous files. This first test would start by generating text files with random characters. Once a file was generated openABE was used to encrypt the file with KP-ABE and the time for encryption was measured. Once completed openABE KP-ABE decryption was used on the file and the time for decrypted was measured. Finally the original MD5SUM was checked on the original file against the newly decrypted file to verify no data loss.

The fourth set of tests target how aes encrypts a fully utilized text file with random character from the start of the file to the end of the file fully utilizing all 16MB with characters with no spaces. The first set of data labeled aes-base-enc are the measures of each instance of encrypting the previous files and aes-base-dec are the measures of each instance of encrypting the previous files. This first test would start by generating text files with random characters with no spaces. Once a file was generated, openssl was used to encrypt the file with aes and the time for encryption was measured. Once completed openssl aes decryption was used on the file and the time for decryption was measured. Finally the original MD5SUM was checked on the original file against the newly decrypted file to verify no data loss.

The next set of tests introduce network calls which will add variation but each time a call outside of the system was made a measure was taken before and after so that when the data is shown, each network call will be shown as part of the total showing which process took how much time. Another variation to the next set of tests are these tests no longer take place on a closed virtual machine with no network connection. These tests

instead use the previously mentioned centos-8 virtual machine, a sql server version 8.0.31-0ubuntu0.20.04.1 for linux on x86-64, and a Ubuntu virtual machine, 1 Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz, 4GB of ram, 32 GB of SSD Memory. These tests also no longer use the time command line utility and instead use the python time module for measuring time. These tests still ran a 1000 times to get a decent sample size.

The fifth set of tests target how the overall system inputs data and returns data back to users. The first test with the data labeled webserver-create are the measures of the create operation and the processes behind them such as the encryption process, upload to s3 public and upload to s3 private buckets, and the rest of the processes such as generate siid are grouped into RestProcess. The next test with data labeled webserver-read are the measures of the read operation and the processes behind them such as the download for the public s3 bucket, and the decryption process, and the rest of the processes such as get s3-cihper-text-key from siid are grouped together under AVGProcessTime.

Data Collection

This chapter is dedicated to graphs obtained through experiments.

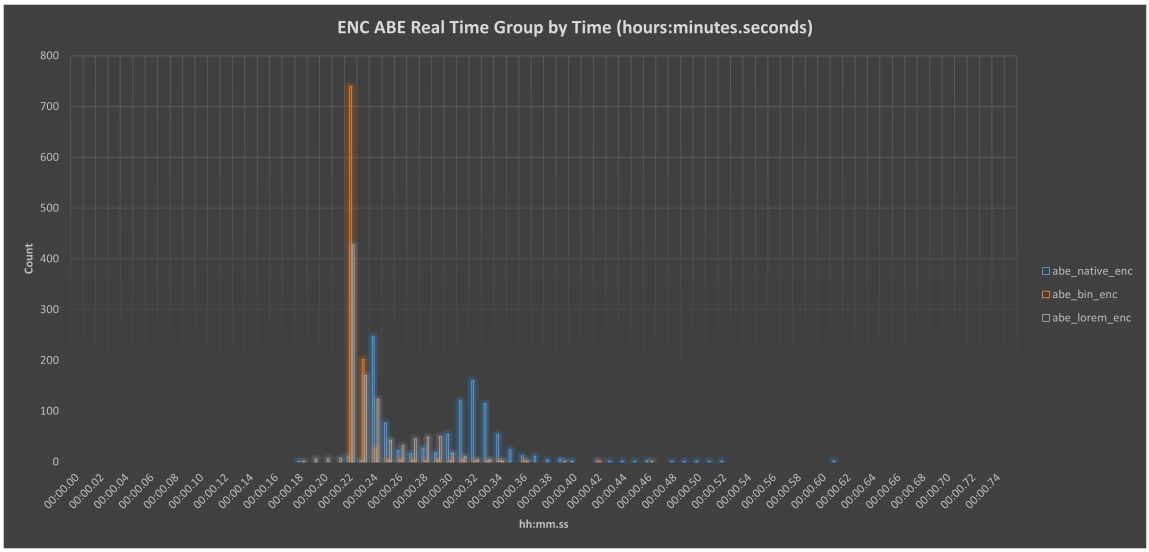

Figure 5.1: Encryption KP-ABE Real Time Graph

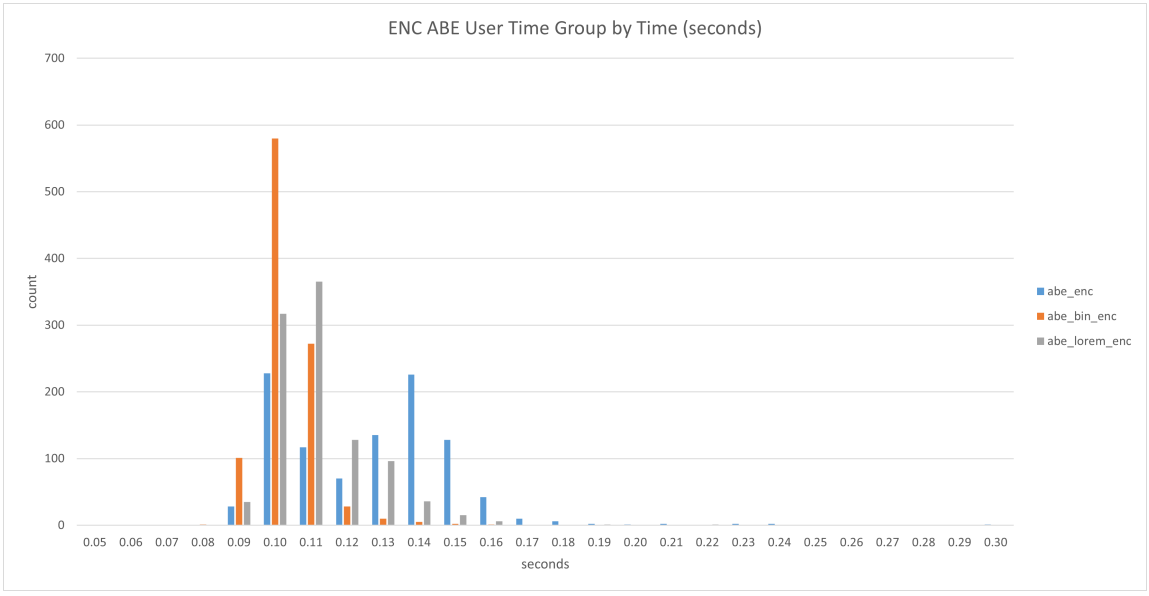

Figure 5.2: Encryption KP-ABE User Time Graph

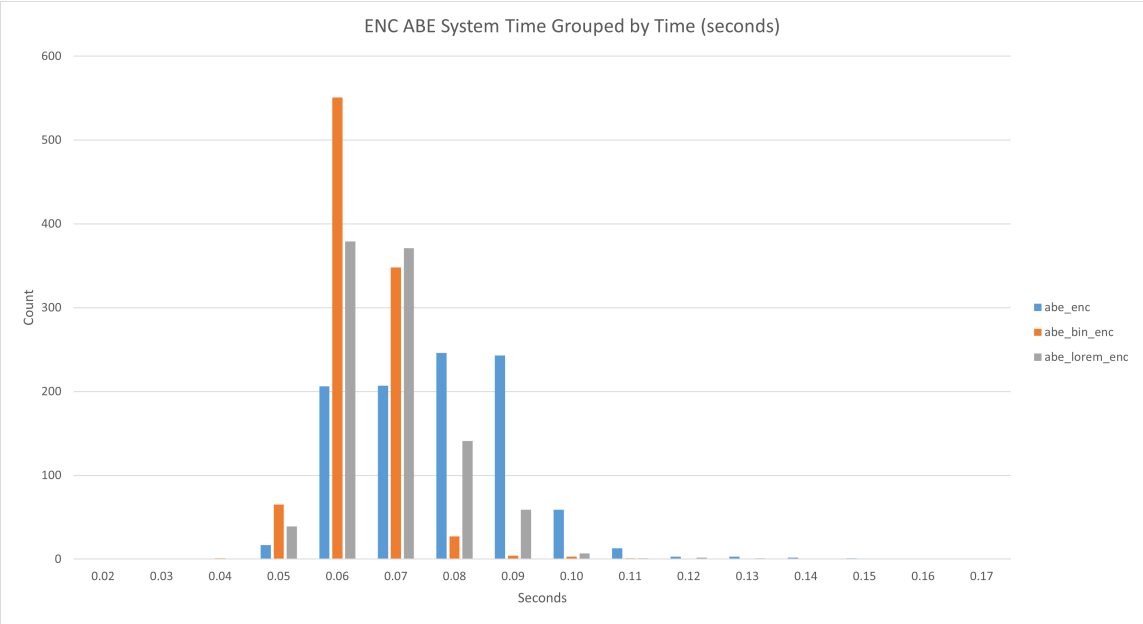

Figure 5.3: Encryption KP-ABE System Time Graph

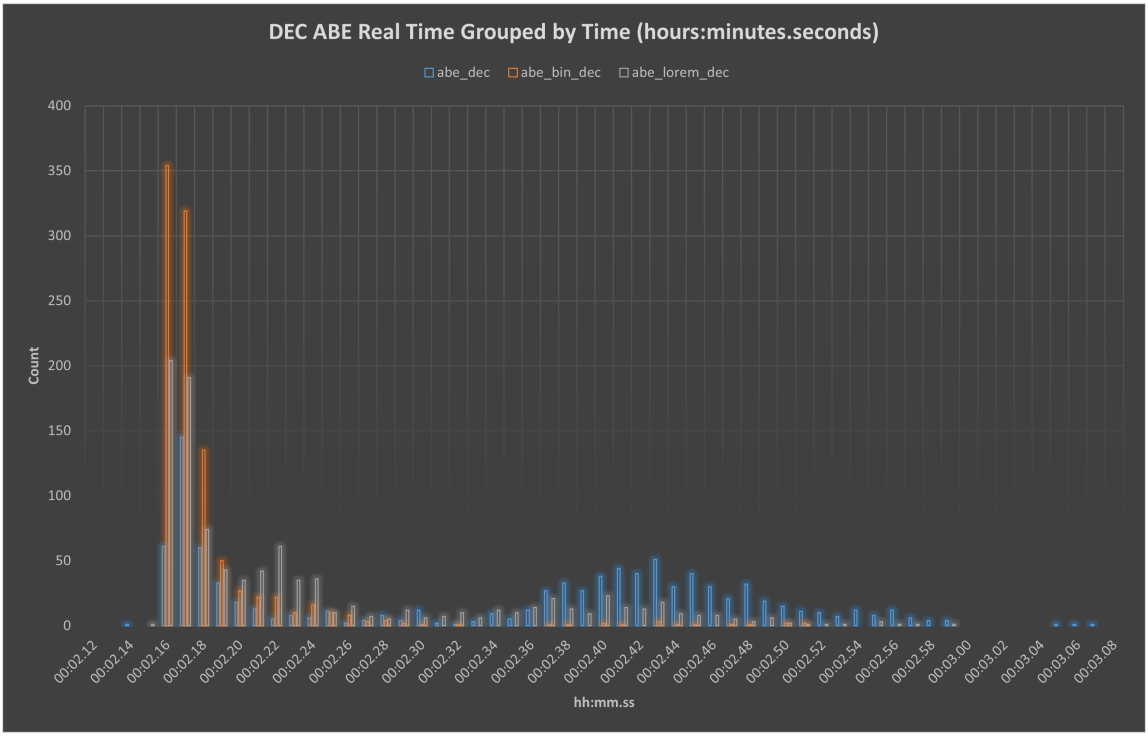

Figure 5.4: Decryption KP-ABE Real Time Graph

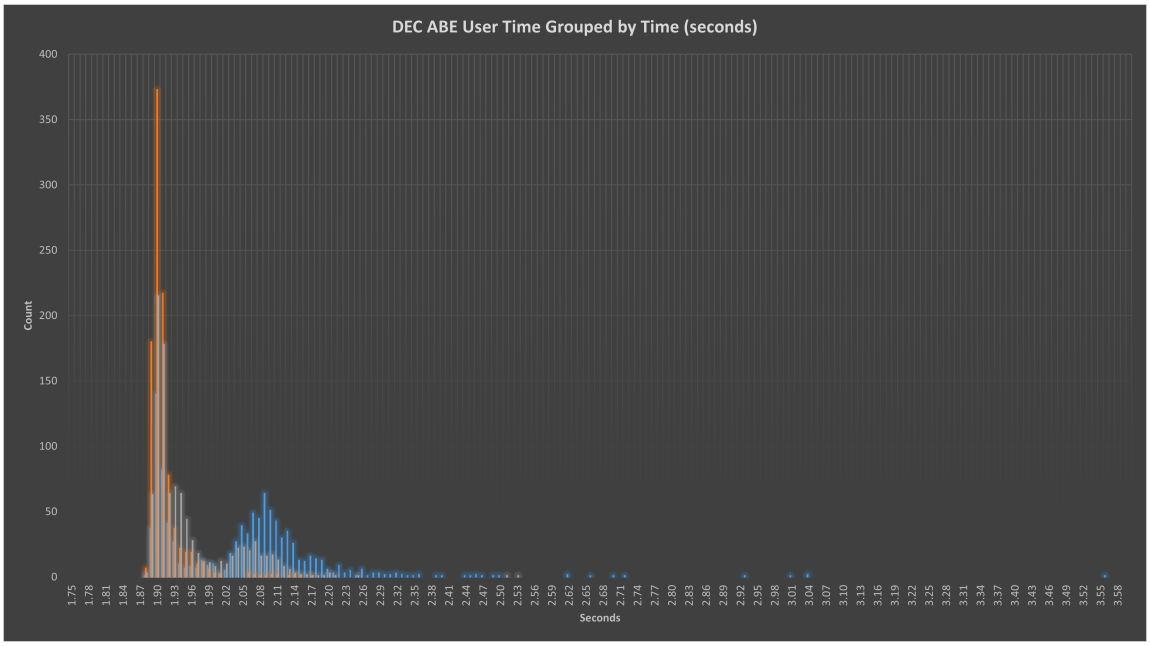

Figure 5.5: Decryption KP-ABE User Time Graph

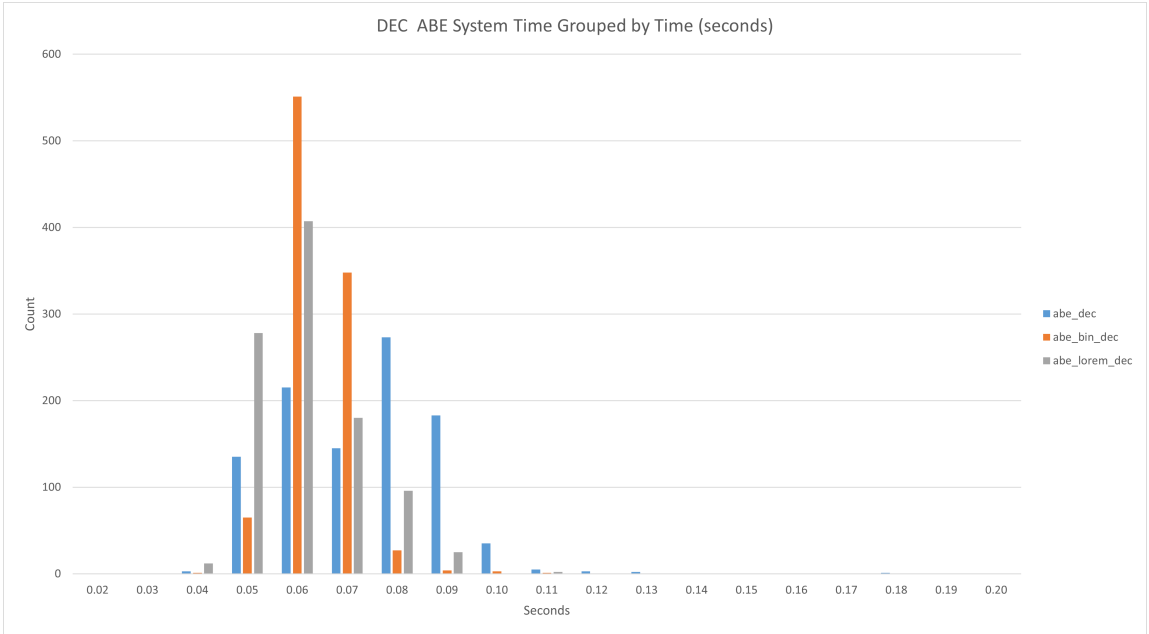

Figure 5.6: Decryption KP-ABE System Time Graph

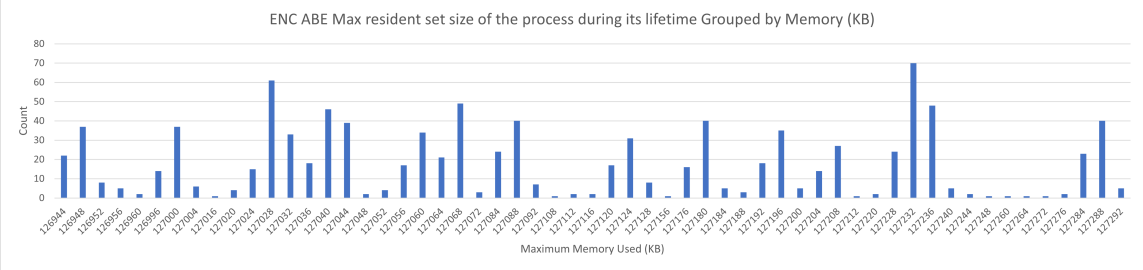

Figure 5.7: Encryption KP-ABE Max Memory graph

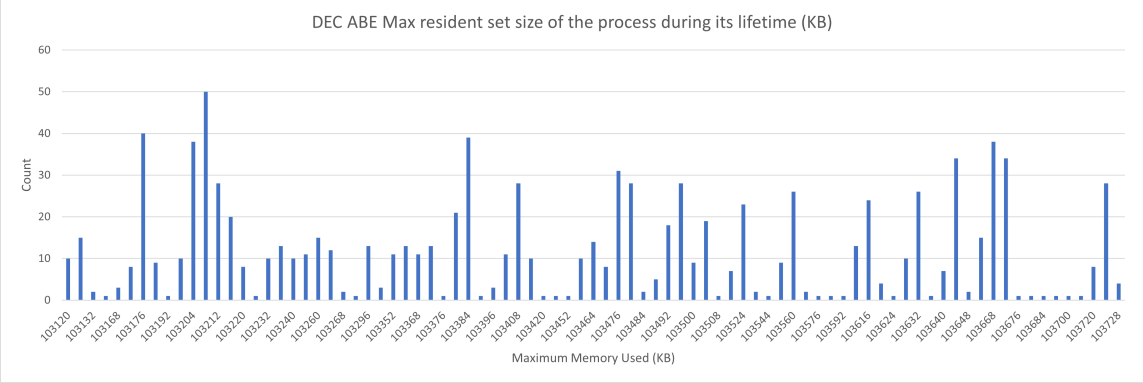

Figure 5.8: Decryption KP-ABE Max Memory graph

	USR Time	SYS TIME	Memory KB
Mean	0.13	0.08	127113.32
Median	0.13	0.08	127084.00
Mode	0.10	0.08	127232.00
Max	0.30	0.15	127292.00
Min	0.09	0.05	126944.00

Table 5.1: Data Summary for abe-base-enc

	USR Time	SYS TIME	Memory KB
Mean	2.05	0.07	103424.99
Median	2.06	0.08	103460.00
Mode	1.90	0.08	103208.00
Max	3.56	0.18	103728.00
Min	1.88	0.04	103120.00

Table 5.2: Data Summary for abe-base-dec

	USR Time	SYS TIME	Memory KB
Mean	0.10	0.06	127083.26
Median	0.10	0.06	127060.00
Mode	0.10	0.06	127020.00
Max	0.16	0.11	127292.00
Min	0.08	0.04	124264.00

Table 5.3: Data Summary for abe-bin-enc

	USR Time	SYS TIME	Memory KB
Mean	1.91	0.06	103405.66
Median	1.90	0.06	103440.00
Mode	1.90	0.06	103476.00
Max	2.17	0.10	103724.00
Min	1.88	0.04	103052.00

Table 5.4: Data Summary for abe-bin-dec

	USR Time	SYS TIME	Memory KB
Mean	0.11	0.07	127110.56
Median	0.11	0.07	127064.00
Mode	0.11	0.06	127036.00
Max	0.22	0.13	127292.00
Min	0.09	0.05	126940.00

Table 5.5: Data Summary for abe-lorem-enc

	USR Time	SYS TIME	Memory KB
Mean	1.95	0.06	103422.17
Median	1.92	0.06	103460.00
Mode	1.90	0.06	103208.00
Max	2.53	0.11	103728.00
Min	1.88	0.04	103116.00

Table 5.6: Data Summary for abe-lorem-dec

	USR Time	SYS Time	Memory KB
MEAN	0.02	0.01	4916
MEDIAN	0.03	0.01	4920
Mode	0.03	0.01	4976
MAX	0.4	0.02	8176
MIN	0	0	4760

Table 5.7: Data Summary for aes-base-enc

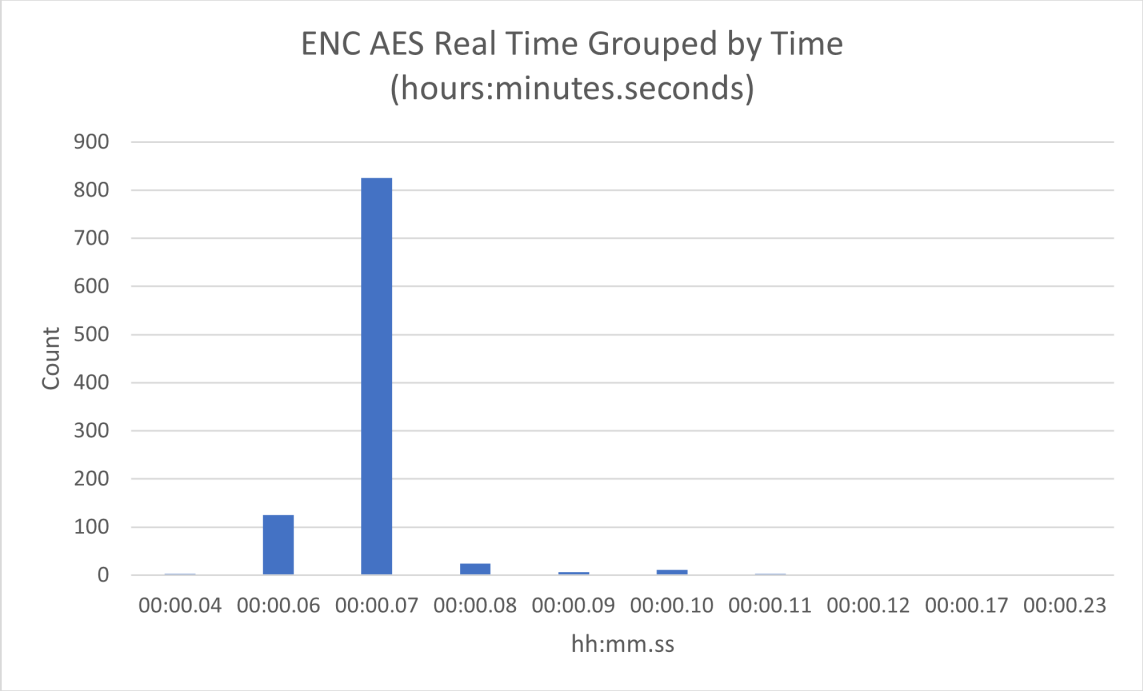

Figure 5.9: Encryption AES Real Time Graph

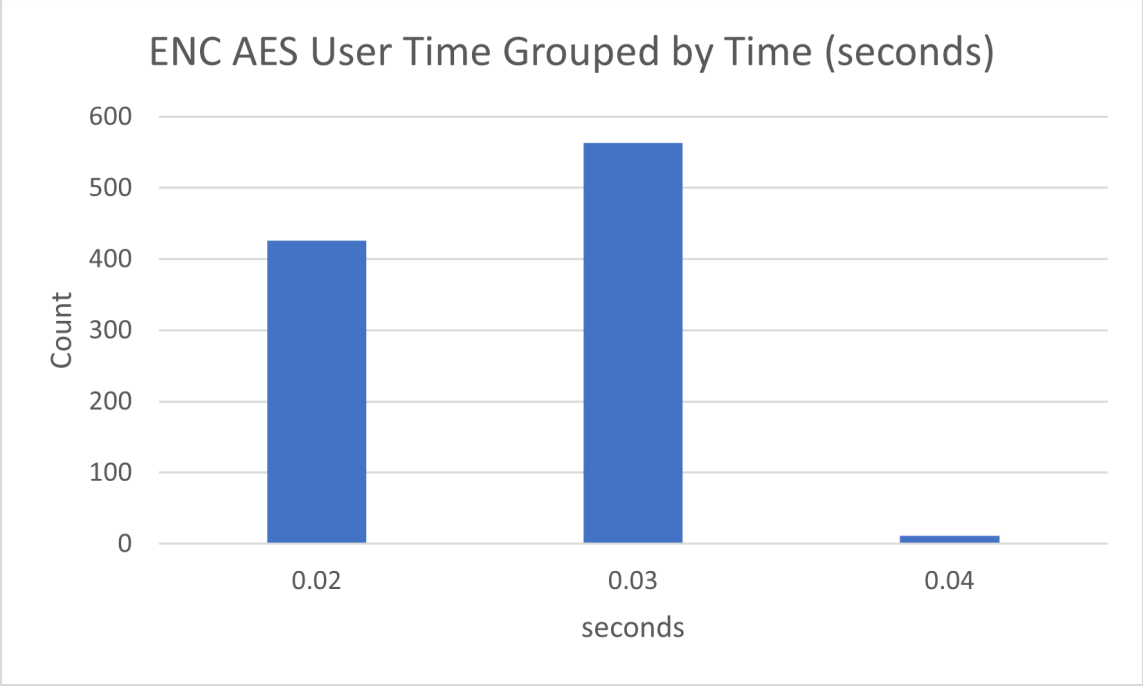

Figure 5.10: Encryption AES User Time Graph

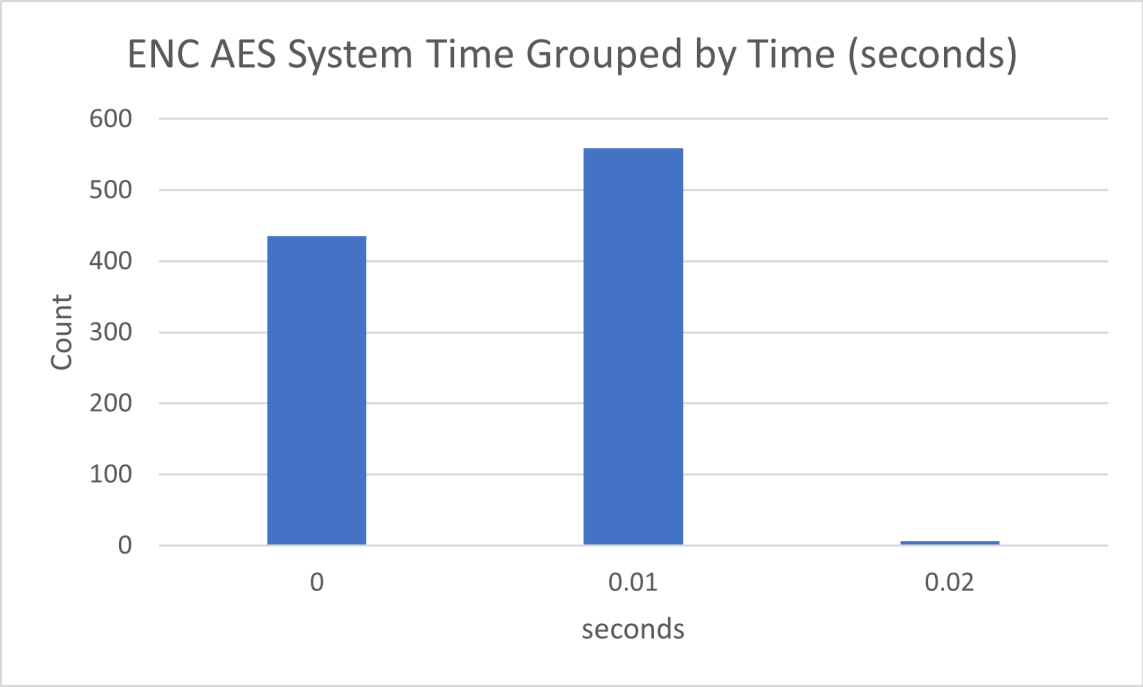

Figure 5.11: Encryption AES System Time Graph

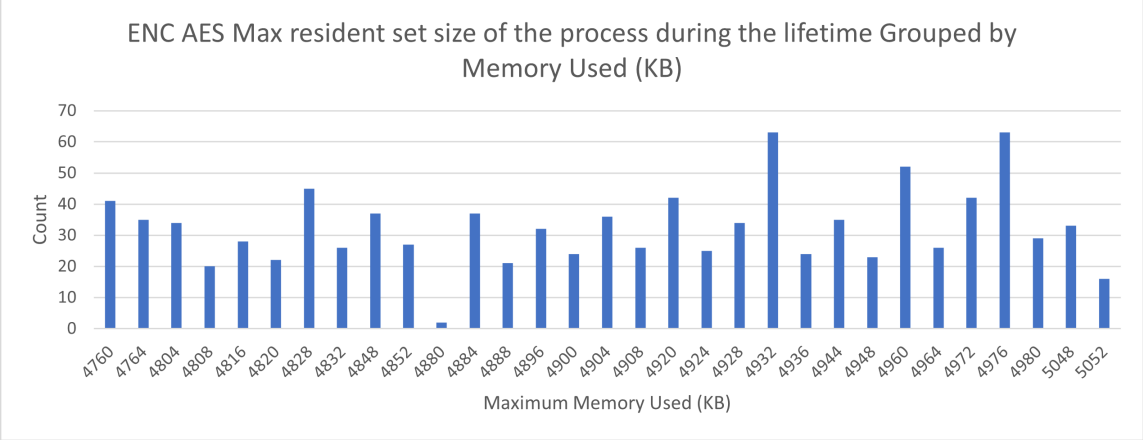

Figure 5.12: Encryption AES Max Memorygraph

	USR Time	SYS TIME	Memory KB
MEAN	0.02	0.01	4922
MEDIAN	0.02	0.01	4912
Mode	0.02	0.01	4932
MAX	0.3	0.03	8384
MIN	0	0	4760

Table 5.8: Data Summary for aes-base-dec

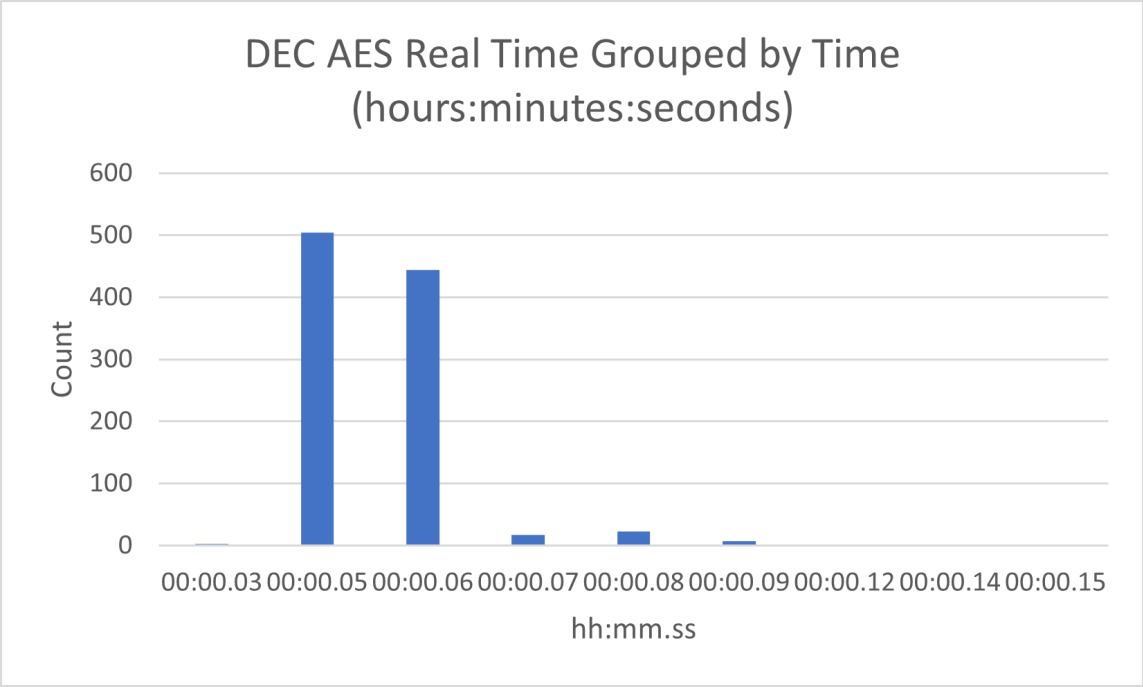

Figure 5.13: Decryption AES Real Time Graph

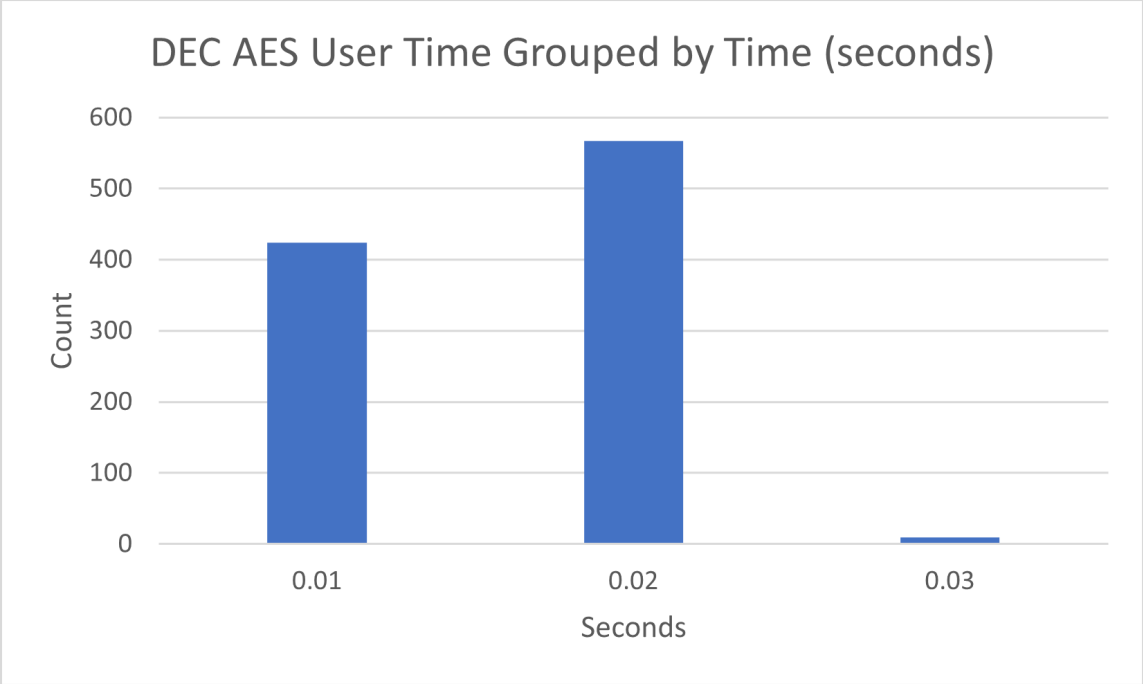

Figure 5.14: Decryption AES User Time Graph

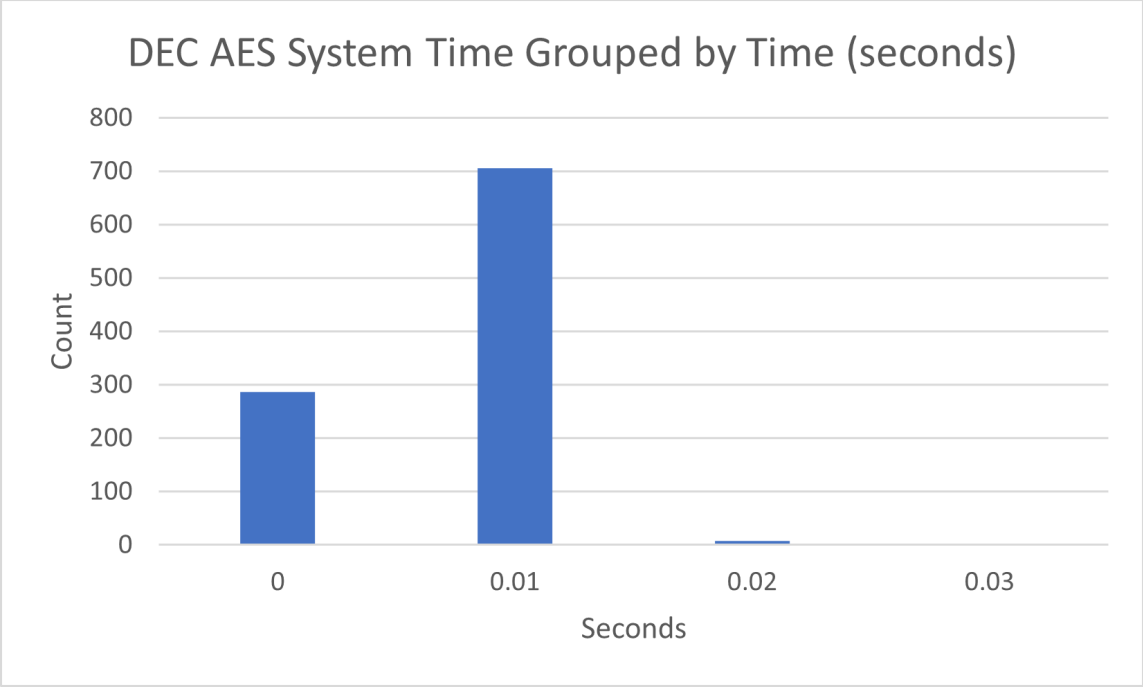

Figure 5.15: Decryption AES System Time Graph

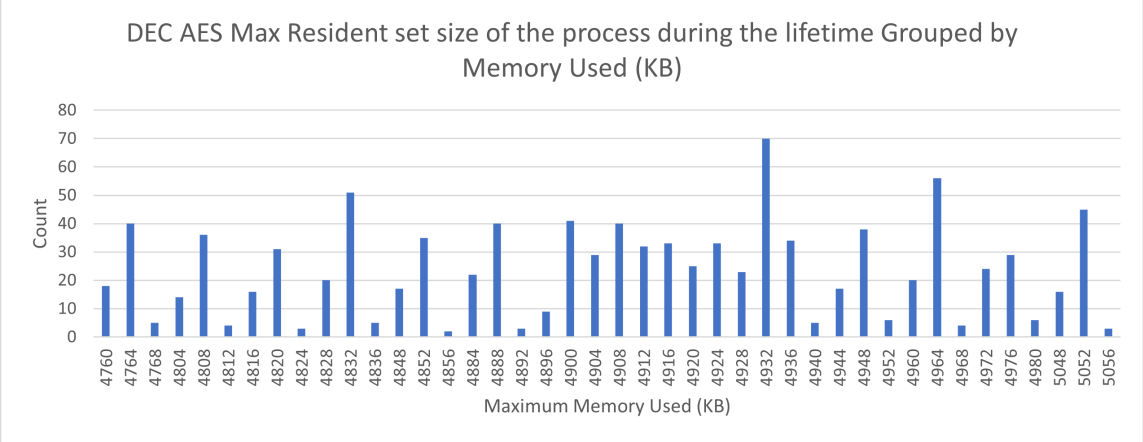

Figure 5.16: Decryption AES Max Memorygraph

	Total Time	Process Time
Mean	17.79	0.76
Median	14.92	0.29
Mode	14.14	0.28
Max	73.22	7.14
Min	13.62	0.14

Table 5.9: Data Summary for webserver-create

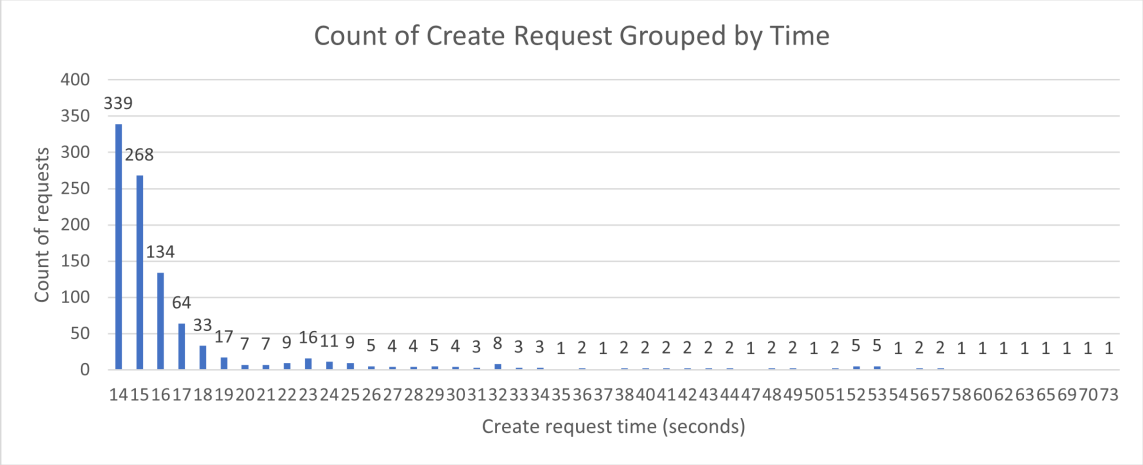

Figure 5.17: Create request Count Graph

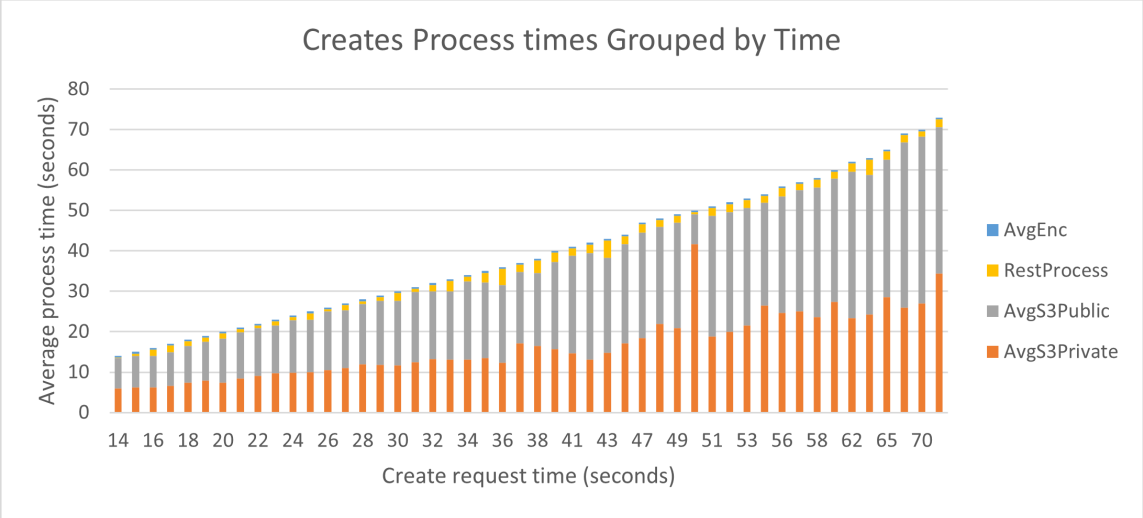

Figure 5.18: Create process Time Graph

	Total Time	Process Time	S3 Download
MEAN	8.10	4.48	0.16
MEDIAN	8.30	4.69	0.15
MODE	7.75	4.11	0.14
MAX	15.11	11.53	2.13
MIN	3.88	0.36	0.10

Table 5.10: Data Summary for webserver-read

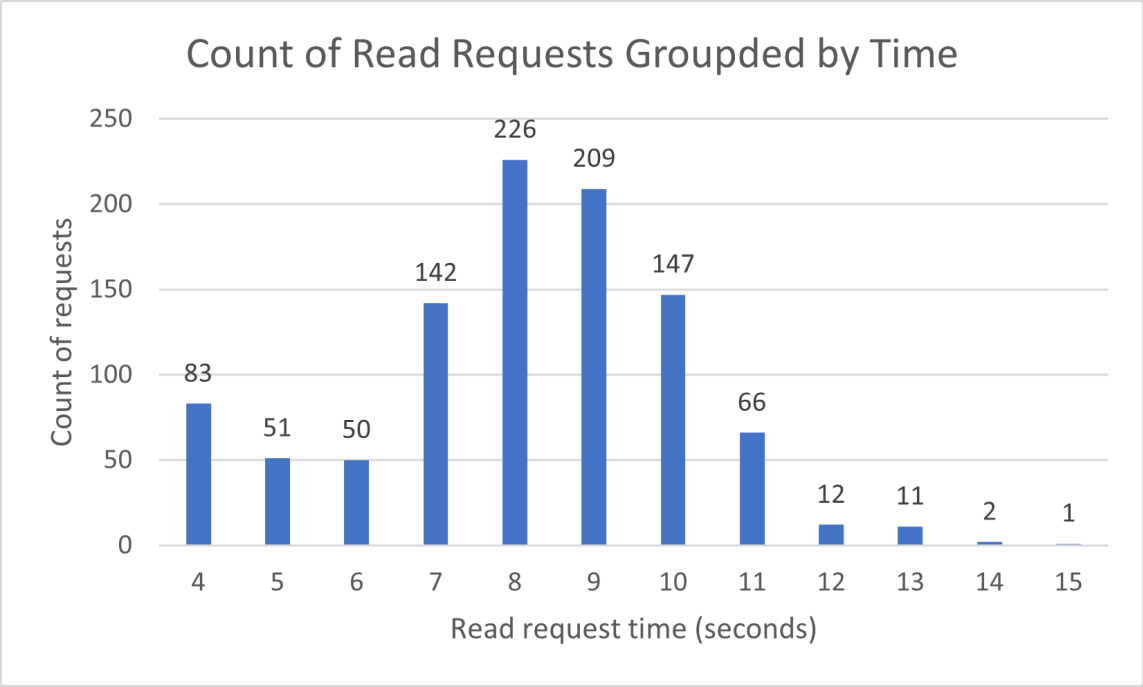

Figure 5.19: Read request Count Graph

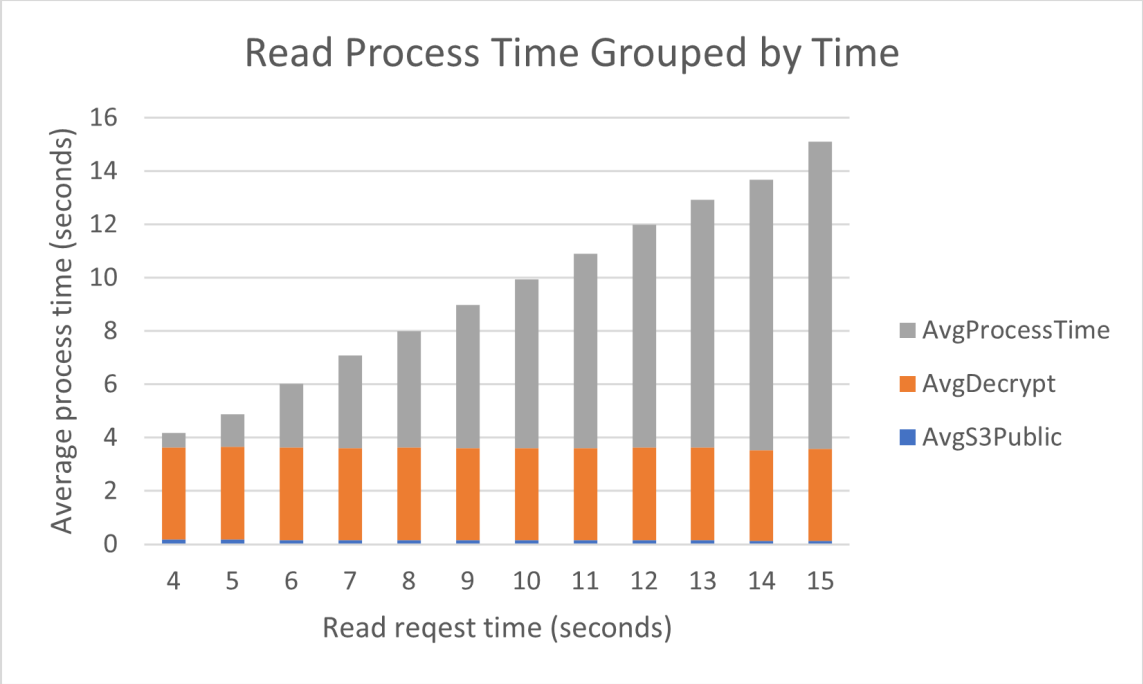

Figure 5.20: Read process Time Graph

Chapter 6: Results

Results from the first three sets of experiments can be seen in Figure 5.1 and show the processing in real time between abe-base-enc, abe-bin-enc, and abe-lorem-enc. This graph is meant to show how openABE KP-ABE does against different text types. Most of the encryption time is between 0.24 seconds between all three. However there is some variation between the three files. With the fastest minimum time of 0.8 USR seconds binary text files are the fastest for encryption processing, and also has a quicker of all the functions such as mean, mode and max. The next best file type in terms of speed was the abe-lorem-enc which had a quicker mean, median and max than abe-base-enc but not as quick as binary files. One of the main contributions to why abe-bin-enc is not up to par with the other two tests is that there were no spaces between all the characters in the 16MB file. Between Figure 5.2 and 5.3 KP-ABE uses more time with User time than with system time which is to be expected. Figure 5.4 shows the decryption metrics between KP-ABE real time between abe-base-dec, abe-bin-dec, and abe-lorem-dec. Again is the trend of abe-bin-dec files being the fastest, followed by abe-lorem-dec, then by abe-base-dec. What the decryption real Time Graph shows is that encryption is faster by a factor of 19.5, when the mean from Table 5.5 is compared to the mean in Table 5.6. However the decryption process for KP-ABE is still a lengthy process such that using abe-lorem-dec the best case scenario is a minimum time of 1.88 user seconds while a worst case might be 2.53 user seconds. In terms of memory between KP-ABE encryption and decryption between all the tests the encryption uses more memory than the decryption process using the mean from abe-lorem-enc as 127110 and the mean from abe-lorem-dec as 103422 there is a difference of 23,688 KB.

The fourth experiment uses openssl aes to use one of the most popular most used encryption methods to compare against openABE KP-ABE. The results are 0.02 seconds

as an average for encryption and 0.02 seconds for decryption. when compared against abe-base-enc the mean for encryption is 0.13 seconds and for decryption is 2.05 seconds. This results in aes being 6.5 times faster for encryption and 102.5 times faster for decryption. The difference between memory usage is aes encryption is 26 times more efficient and decryption is 21 times more efficient. It should be noted that aes is an asymmetric encryption algorithm while ABE is attribute based encryption with additional layers built into it, and even contains aes within its key encapsulation mechanism [12].

The fifth experiment results describe how the system performs when saving files into the system, and retrieving files. When saving a file into the system it can take at least 3.88 seconds or at most 15.11 seconds with 15MB files. The create is overall longer than a read with 17 seconds for a single 15MB file upload, however the process time within the request is much faster with 0.80 seconds, meaning most of the time is spent uploading the files to the s3 buckets. As figure 5.17 shows 80 percent of create requests land between 14 and 17 seconds. Figure 5.18 conveys that most of the create time is used up by the upload process to s3, while encryption takes up 0.11 seconds on average which makes up 0.6 percent of the total time, the processes that are not s3 uploads such as generate siid take 0.76 seconds on average which makes up for 4.2 percent of the overall create time request. Read requests have a more bell curve distribution shifted to the left. With a median of 8.30 seconds, a minimum time of 3.88 seconds, and a maximum time of 15.11 seconds. The overall time of the read requests take 8.10 seconds on average but the process time takes 4.47 seconds which makes up 55 percent of the requests total time. While the read decryption time takes 1.95 seconds 24 percent of the overall request total time. The portion of the read request is only 1 percent using only 0.16 seconds on average with a minimum of 0.10 seconds and a maximum of 2.13 seconds.

Chapter 7: Conclusion

In conclusion the original goal of this project was to tackle how to keep data at scale secure while eliminating the time penalty and single point of failure of authorizing users for data access using a distributed approach using KP-ABE and the properties it has such as authorizing users that decrypt content with the attributes that satisfy the user policy tree within the decryption key. Using the system that was theorized by Morales-Sandoval and his team this would allow multiple data processing clients to pull large amounts of data they were searching for with extracted keywords and the penalty for authorizing would fall onto each of the clients instead of the server. The following experiments suggest that openABE has other strengths that molded the project into how organizations or communities with data can share data between them.

One of the problems faced was deployment. OpenABE was a major pillar of this project, getting it to build was difficult, and once built getting it installed was challenging. This lead to a design change within the system proposed that users instead upload content and download content through an api and deployed nodes with the necessary mpk, msk, and key files are stores within the system so that users only need to worry about there content they want and instead have a config file they upload with a request. This also allowed the system to work with all operating systems. The second problem faced was the search query a lot of time was spent trying to figure out a way to use the users existing mpk file and key file to get a partially encrypted search query that could find another partially encrypted key word within the system however more problems kept emerging such as how would multiple users from different organizations with different msk and mpk files generate encrypted keywords that others could use encrypted searches on, and if the keywords were encrypted then search queries would have to decrypt all the keywords on each file and the time to do so would eliminate the original goal.

Future work would be solving how to limit the faults of keys such that you can not easily authenticate a user so someone could have stolen or copied a key.There is also if a user has been fired or rejected from the system they may still access to data with their key and their is no way to remove the key from the system. How this could be solved is with modifying openABE to include a use limit attribute that increments every decryption and once completed will need to request a new key from the system reducing the issue along with implementing an authentication method so that on requests if the user has been kicked out may no longer receive files making there key useless in their hands.

References

[1] Amazon cloudfront faqs, . URL https://aws.amazon.com/cloudfront/faqs/.

[2] Amazon ec2, . URL https://aws.amazon.com/ec2/.

[3] Aws fargate faqs, . URL https://aws.amazon.com/fargate/faqs/.

[4] Accessing jumpbox, . URL https://docs.aws.amazon.com/solutions/latest/centralized-logging/accessing-jumpbox.html.

[5] Amazon rds for sql server faqs, . URL https://aws.amazon.com/rds/sqlserver/faqs/.

[6] Amazon route 53 faqs, . URL https://aws.amazon.com/route53/faqs/.

[7] Amazon s3, . URL https://aws.amazon.com/pm/serv-s3/.

[8] Amazon sqs faqs, . URL https://aws.amazon.com/sqs/faqs/.

[9] The centos project. URL https://www.centos.org/.

[10] Openabe, . URL https://github.com/zeutro/openabe.

[11] Openldap, main page, . URL https://www.openldap.org/.

[12] J. Ayo Akinyele. Openabe, . URL https://github.com/zeutro/openabe/blob/master/docs/libopenabe-v1.0.0-api-doc.pdf.

[13] J. Ayo Akinyele. Openabe, . URL https://github.com/zeutro/openabe/blob/master/docs/libopenabe-v1.0.0-cli-doc.pdf.

[14] J. Bethencourtand, A. Sahai, and B. Waters. Ciphertext-policy attribute-based encryption. page 321–334, 2018. doi: ff10.1109/SP.2007.11ff.ffhal-01788815.

[15] Thouraya Bouabana-Tebibel and Abdellah Kaci. Parallel search over encrypted data under attribute based encryption on the cloud computing. *Computers and Security*, 54:77–91, 2015. ISSN 0167-4048. doi: https://doi.org/10.1016/j.cose.2015.04.007. URL https://www.sciencedirect.com/science/article/pii/S0167404815000577. Secure Information Reuse and Integration and Availability, Reliability and Security 2014.

[16] V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. page 89–98, 2006.

[17] Matthew Green, Waters, Brent Waters, Susan H., and J. A. Akinyele. The openabe design document. pages 1–29, 2018.

[18] Miguel Morales-Sandoval, Melissa H. Cabello, Heidy M. Marin-Castro, and Jose L. G. Compean. Attribute-based encryption approach for storage, sharing and retrieval of encrypted data in the cloud. pages 1–16, 2020.

[19] Jeff Sedayao, Steven Su, Xiaohao Ma, Minghao Jiang, and Kai Miao. A simple technique for securing data at rest stored in a computing cloud. volume 5931, pages 553–558, 12 2009. ISBN 978-3-642-10664-4. doi: 10.1007/978-3-642-10665-1_51.