ENCRYPTED COLLABORATIVE SYSTEM FOR THE CLOUD

Sanjul Sharma

Thesis Prepared for the Degree of

Bachelor of Computer Science

University of North Texas

April 2021

Approved:

Kirill Morozov, Instructor

TABLE OF CONTENTS

Chapter 1

INTRODUCTION

1.1 Background

Cloud computing and cloud-based user-facing applications have gained popularity over the past few years, and for a good reason. These applications allow users to avoid having to download software on their own machines and even allow for collaborative uses of these applications, allowing multiple users to work on a document collaboratively. Another pro for cloud user applications can be their accessibility globally and even be easily accessed on mobile devices, giving them a considerable advantage for scalability. With more services switching over to the cloud, the question of securing the data on cloud-based services has now come to the forefront of the industry.

The most popular collaborative software currently used is the Google suite of services offering docs, excel sheets, and even presentation software. The issue presents itself when you start having to trust cloud providers to secure sensitive and essential data. With giants like Google and Amazon, the market for cloud security is very heavily favored to their favor, and third-party security becomes rare.

The encryption and decryption for these cloud services for the giants are entirely proprietary, and the information about their methods is mainly unknown. Even if you believe that these services are fully secure and the warehouses storing the data, it is hard to feel the same about the minor cloud services. The smaller services offering cloud hosting and encryption security rely on the user's machine to keep the data secure; while some do it on their host, it is hard to rely on the protection entirely.

These can be a dealbreaker for any sensitive data put on these servers, so finding an alternative can be extremely necessary. Some possible solutions can be storing only encrypted data on the cloud services. While this system can be quite effective in securing the data, this can cause issues with collaboration and rely on one user being available to encrypt and decrypt whenever needed.

Keeping up simultaneous editing and changing of these services between multiple users can be hard to execute in an all entailing service. This process requires complete protection on the user end as well as protection from the server itself. Any data stored is completely encrypted and only to be seen when being worked on by the users who have access to this data. Giving the power of encryption to the user instead of leaving it to the discretion of the service itself can be the difference in an added layer of security. This layer can protect the user from any wrongdoing from an employee working for the service itself. Saving the user from every possible data breach is of extreme importance. The significant services currently have access to the data even if they intend to keep it secure.

Another minor thing to consider is how the data stored on these systems will be government intrusion. Although this is not something every individual might have to worry about, it is a concern for some. As leaked in the media, "…government entities and technology companies in the U.S. and elsewhere may be inspecting your data as it is transmitted or where it resides on the Internet, including within clouds" [1]. So, if your data is at the mercy of the company storing it, can it ultimately provide you with security satisfaction, or is there an alternate solution somewhere that can overtake the current system in play?

1.2 Motivation

There is a lot to consider when you chose to let your data be stored online. You must ask tough questions about who you are willing to trust with this data and if they can do enough to protect you from outside attacks and guarantee that they will also protect your data from any inside breaches. While there are plenty of reasons cloud storage has surged in the past few years, it is not a system that is safe from all attacks. Anytime you chose to side with convenience, there will be sacrifices in the level of security possible. Answering these questions and finding a balance between security and convenience is of extreme importance as we move forward. The answer might seem relatively easy when considering storage for a typical task or even a company's data. Still, when you start to think of data that needs to be protected, whether it is the government, personal information, sensitive company data, the answer can be pretty complex. Finding a solution that can take advantage of the cloud storage convenience while still providing the top level of security possible is extremely important.

While services like Google suits and AWS can offer protection and collaboration, they do not focus on securing the data from within the server itself. This practice means that technically these companies can still have access to your data. Major corporations like these tend to be able to hide certain exceptions under their terms and conditions; under Google Terms of Services document, it states "…The rights that you grant in this license are for the limited purpose of operating, promoting, and improving our Services, and to develop new ones. This license continues even if you stop using our Services…" [2]. That last part in the statement should sound an alarm for anyone storing data on these services. While they do not explicitly take ownership of the data, they can use it to improve their services, which includes any data stored on their system when you sign their terms of service document.

Fighting against these significant corporations will never really be easy, and as it stands currently. The four major cloud storage companies, Google, Amazon, Microsoft, and Facebook, have an estimated "1.2 million terabytes (one terabyte is 1,000 gigabytes). And that figure excludes other big providers like Dropbox, Barracuda, and SugarSync, to say nothing of massive servers in industry and academia" [3]. With a number as large as that, a drastic movement in the cloud storage system would require a product so drastically improved that it almost forces people to change. Since such a product does not currently exist, figuring out the requirements that would call for a significant shakeup is essential.

What makes it harder is sensitive data that we want to protect can be highly sought after, and is thus, the security protocols need to be tested more often. The need for the solution software to also be collaborative can make it more complicated than simply double encrypting the data from the user and the host. The best solution would require shielding the data from the outside and inside while also providing multiple users access to the data through a secret key interface.

Starting on that system was the main aim for this project, finding methods to improve the system that currently exists and to see if there is a plausible solution to data storage that does not leverage security or convenience in any aspect. Focusing on the added layer of encryption between the software and the server and providing an easy collaborative process for the data stored, in this case, on a rich text editor accessible by anyone without the need for additional software.

1.3 Structure of the Thesis

The thesis will focus on five significant aspects of this process. Chapter 2 will focus on the preliminary process for this project. Chapter 3 will focus on the survey of existing services currently available for commercial or industrial use. Chapter 4 will focus on the proposed encrypted collaborative system and the results we could achieve with our system. Chapter 5 will focus on the developer experience on this project. Finally, Chapter 6 will be the conclusion and suggested future work on this topic.

Chapter 2

PRELIMINARIES

In this section, we discuss the tools and notions we approached for our product. We also discuss the reasons for choosing each option.

2.1 Python and NodeJS

The two main coding languages that were approached for this project were Python and NodeJS.

NodeJS is a unique language and is an open-source project. It is essentially an "…asynchronous event-driven JavaScript runtime, Node.js is designed to build scalable network applications" [4]. Implementing a framework on top of an adequate language as JavaScript provides a lot of functionality and can be great for scalable products and callback functions to provide runtime construct that can avoid server blockings. It is also a language that works well on the three major operating systems and is a lightweight option. The word editor of choice with NodeJS would be QuillJS.

Python is another lightweight coding language that has gained popularity recently. It is straightforward to use and learn while still providing a lot of functionality for a multitude of projects. Python also comes with a mature library of resources, many frameworks, and a community for debugging resources.

A significant advantage of Python is also its efficiency, reliability, and speed. It can be used in most environments and without losing any performance, unlike other coding languages. Python also does not restrict you in any way from building out functionalities for any app in question. Python also provides resources for automation of their apps, and integrations of any kind of bot service to run something like autosave can be made quite easily. With Python, the word editor we would use would be TinyMCE.

2.2 JavaScript, HyperText Markup Language, and Cascading Style Sheets

JavaScript is a potent language that offers excellent client-side scripting while being able to complement other languages. While JavaScript is one of the older languages, it is still widely used to provide an extremely responsive setup and frontend capabilities. Tools offered in JavaScript make integrating other aspects of the design extremely easy.

HyperText Markup Language (HTML) is the standard markup language for creating web pages. The method of an HTML page is done through Cascading Style Sheets (CSS). Pairing JS with HTML and CSS code gives any frontend design a complete look and control. While the frontend for this project just needs to be easy to use and minimalistic, using these three languages in unison gives us that result.

2.3 Django

Django is another open-source framework that deals explicitly with the backend of python projects. It offers high flexibility and scalability while remaining relatively simple to implement. The framework provides a fast and easy-to-navigate structure in integrating apps, especially for web use.

Django also offers added security benefits, with up-to-date security patches and significant out-of-the-box measures. Other great features with Django are its cross-platform capabilities, ability to integrate most major databases, and its ability to handle heavy traffic and large bits of information.

Django's Model-View-Template (MVT) offers a unique structure to allow developers a form of freedom from making changes on different aspects of the project. With the conjunction of MVT and Django's powerful template engine and markup language, the flexibility offered is hard to match.

2.4 QuillJS and TinyMCE

QuillJS is a rich text editor with a NodeJS framework. Quill handles operational transformation, offers many functional plugins, can integrate databases, and is well regarded for its efficiency and clean code. It's an open-source, cross-platform product built for developers to integrate easily into any project. It is an API-driven design that tries to offer a lot of functionality in a text editor alone.

Quill is easy to set up, and its bar is exceptionally high in terms of its cross-platform designation. Working on multiple platforms without giving up functionality is especially important. Any collaborative project without cross-platform access would be meaningless in this space. It also offers substantive API support on top of DOM, which is rare for most open-source text editors. Quill was built with content creation in mind and would work well for a creative project.

TinyMCE is a great rich text editor and provides an excellent open-source program to be integrated into the solution. Quill is easy to integrate, provides operation transformation, and is primarily a plugin that can be modified as needed. It does require to be hosted on a server, and this process would be done through Amazon Web Services.

The biggest advantage for TinyMCE was its use of Python as the coding language, and that was the preferred language when researching all options. QuillJS was not used for this primary reason. TinyMCE, similarly to Quill, requires the use of a database that was once again provided through AWS and its DynamoDB service. TinyMCE also provides an autosave feature. This feature was modified and extremely important for our version of the system to have aspects of operation transformation. While TinyMCE also provides a real-time collaboration system, this was not used due to complications with the auto encryption and decryption process implemented later in the project.

While Quill has a lot of functionality and offers more formatting sugar coat than TinyMCE, we went with TinyMCE due to its coding language and automation system that it provides.

2.5 Operational Transformation

Operational Transformation (OT) is an algorithm for the transformation of operations. OT is often applied to collaborative software to allow for real-time concurrent work on a document by multiple parties. Many of the major online collaborative products use this technique currently. The method provides for changes to be tracked from numerous machines concurrently, allowing for simultaneous editing on various formats. In simple text, the method uses "…the transform function that takes two operations that have been applied to the same document state (but on different clients) and computes a new operation that can be applied after the second operation and that preserves the first operation's intended change" [5].

The transformation function within this algorithm builds upon a server-client relationship to handle any clients' collaboration. Collaboration is a significant component of this project, and OT allows for the most seamless solution.

Ideally, we would use the OT method to allow multiple users to collaborate on a rich text editor while finding a way to secure the data on all machines and the server-side. This process would allow for the best solution to our problem discussed.

At the onset of this project, we had hoped to use this approach to build our solution with a working operational transformation. However, we had to later scrap the idea in this iteration due to the auto encryption and decryption process issues. We discuss future solutions to this issue in Chapter 6.

2.6 Amazon Web Services and Microsoft Azure

A significant component of our process was making sure our solution was accessible to multiple users while also making sure we did not require them to download additional software on their machines. This process would need firstly for the app to be hosted on a cloud platform.

Microsoft Azure would be a great cloud platform for this project. At the base of Microsoft's collaboration software in junction with their office suite, it would allow for all services needed to carry out any needs for our project. Azure provides its cloud hosting services through its virtual machines (VM). While it is still a preferred form of cloud hosting services amongst C-level executives, a lot of that is attributed to companies being able to keep up their "…long-standing relationships with the vendor and know that they can consume a great deal of their enterprise computing needs all in one place, from productivity and enterprise software all the way down to flexible cloud computing resources for their developers, with one hand to shake" [6].

On the other hand, Amazon Web Services has taken the lead in cloud computing and is widely used by the varying level of users. Its primary offering is the EC2 instances that allow for easy to modify platforms for all user needs. Additionally, what set AWS apart for our project needs was the vast array of additional services AWS provides, such as DynamoDb, Simple Storage (S3), built in lambda functions, and API functionality.

2.6.1 Amazon Elastic Compute Cloud

The specific service name for Amazon's cloud computing solution is the Elastic Compute Cloud or EC2. This service provides a powerful tool and offers complete administrative control over the server. "Compute instances are easily managed through the Amazon EC2 web interface, which allows users to scale up or down, boot instances, and configure processor settings with a few clicks of a mouse. Additionally, virtual servers on EC2 can be managed automatically via an application program interface (API) that can be set up by downloading a software development kit (SDK) from AWS in a choice of programming languages" [7].

EC2 also allows you to choose any operating system you selected, allowing significant flexibility and freedom of choice. Security is also a primary built-in tool offered by the EC2 servers, as the instances run on a private cloud. EC2 also has a virtual firewall setup to control incoming traffic and allows you to make custom rules for security as well.

2.7 ShareDB and DynamoDB

While the rich text editors and web hosting services provide us with almost everything we need, we have to store all that data used and changed on some database platform.

ShareDB is an open-source project that allows for backend functionality. The platform is on NodeJS and provides a complete platform for our needs that would be highly compatible with our needs, especially with Quill. Ideally, the database is stored on localhost and needs to be called upon for any data export and import.

DynamoDB is Amazon's database hosted on their platform and allows for easy and direct integration to any AWS product. It is great for web apps and is built for scaling in mind with incredible speeds. For a collaborative project, this is an excellent resource as it can handle more than our needs in terms of requests made to the database. With the need for continuous saving of the data, this would be perfect for a web app that does not require any additional download.

2.8 PostgreSQL

PostgreSQL is another robust database that is an open-source project. It is an object-relational model that has been around for 30 years and is still strongly supported. PostgreSQL offers many features for developers looking to build applications, as well as data protection and integrity. It provides a lightweight database without sacrificing speed. This feature is an essential component as we need the data to be saved and be easily accessible once stored.

The main draw for PostgreSQL is its access to extensions, built-in ones, and even those offered through third-party applications. AWS provides an easy pathing to integrate any PostgreSQL database onto a wide array of AWS services, including EC2.

2.9 Encryption

Encryption is the process of encoding information, making sure the data is safe and can only be accessed through a key. The process usually is done by "encrypting" or changing a set of plaintext data and rewriting it into an alternate form known as ciphertext. To decipher the text, the user would require a key and use the key to "decrypt" the data back into its original plaintext form.

This process is the added security we were looking to add to our project to protect our data from the server itself. This process would allow only the document's creator to have the key and use it to decipher the data they chose.

2.9.1 Advanced Encryption Standard

Advanced Encryption Standard (AES) is the current NIST standard [8] for block ciphers. This algorithm allows up to 256-bit keys and it features a fixed block size of 128 bits. The system is well respected as it has gone through years of testing and analytical spotlight and held up firmly against various cryptanalytic attempts. This cipher is based on the design principle "substitution-permutation network" and is equally effective for software and hardware.

The wide-spread industrial use of this cipher and the fact that it is a standard were the main reasons for us to adopt it as an encryption method in our implementation.

2.10 JQuery

JQuery is a fast, compact, and feature-rich JavaScript library offering a lot of support and user tools. These tools are helpful in the handling of the document itself as well as the key generation process. The language itself uses JavaScript a lot simpler as it tends to wrap typical JS code into methods that can be called through single lines of code.

JQuery also offers DOM manipulation, which would be a valuable tool in our solution, allowing us to maneuver around the DOM with ease and provide a structure. Using the JQuery engine can make the process of handling the keys, encrypting and saving the data, and decrypting and loading the data from the ciphertext.

Chapter 3

SURVEY OF EXISTING SERVICES

3.1 Microsoft Office 365

Microsoft Word is part of the Office 365 service system and is packed with a lot of features. Firstly, Microsoft Word is one of the most potent document editing software currently available. However, the service is not free and requires one of the multiple office plans currently available for purchase through direct investment or a continuous subscription model. The service requires you to sign up for a Microsoft account for any Microsoft service offered through office 365.

To collaborate on a project, this would require all users to have a subscription model of the office 365 suite and does require a download that is available on most major mobile and computer operating systems. Security options included in the system for collaboration include password protection using the RC4 level advanced encryption. Other safety features allow you to lock certain aspects of the project with layered passwords restricting different editing features ordinarily available to all users.

Microsoft Word also does not currently offer any encryption of the text inside the document currently. The documents are saved on the Microsoft OneDrive platform but do not have any safety features on the server side to protect your data from any system administrators.

3.2 Google Docs

Google Docs is Google's rich text editor hosted on the google workspace environment that offers many other suite-level services similar to Microsoft office 365. The service is free but does require users to sign up for a google account. Google doc provides multiple security features and is easily accessible on various platforms, including major mobile and computer operating systems. Each document starts as a private document only accessible by the owner of the document, and without being granted permission by the owner, other users can not access the document. Access can only be given to other google account users, and changes made to any parts of the document are tracked in an extensive log that acts as a version control system for the document.

While Google does also allow file encryption on their platform, there is currently no way to decrypt these documents on the platform itself. Users have to download the file on their operating system, edit them on software on their machines, and then these documents can be encrypted on google suits and stored. Encrypting a file currently means losing collaboration and editing services google docs typically offers.

3.3 SafePad

SafePad is a free text editor that lets you encrypt your documents with an AES encryption algorithm. The app itself allows a user to set up two different passwords and both iterations of the password encrypt the original document individually. Passwords are used as the private key in this instance. In the case of one password being cracked, the perpetrator would only receive back encrypted data.

SafePad does require a download and does not include any formatting or editing features on its plaintext editor. It is ideally used for encrypting minor data such as passwords, bank details, and other personal information. SafePad is not cross-platform enabled and is only functional on windows OS.

3.4 Gobby

Gobby is an open course project created by a group of freelance software engineers. The project code can be attained directly on GitHub and runs on the user's machine. The app allows collaborative access to documents with encrypted connections and offers configurable dedicated servers to host the document. Other features include group chat, distinct colors for all users connected to the document, and cross-platform capabilities.

The missing part of the software lies in its computer operating system's needs and its inability to offer any mobile solution to its users. The document itself is also not encrypted, but the connection to the document offers encryption and safety for the user. The software also requires download and set up for several components, including multiple libraries. There is also a lack of editing and formatting features for the document, and it is indeed more of a plaintext format.

3.5 CipherCloud

Cipher Cloud is another encryption system powered by cloud computing, offering various features to ensure the safety of user data. The features included on the platform include but are not limited to deep visibility, advanced threat protection, and end-to-end data protection.

CipherCloud is structured to offer these services for businesses and does not focus on individual user data protection. While not a focus, a user can use any of the services CipherCloud provides to encrypt their data with several levels of security. Some of the other features that can be useful for individual owners are data loss prevention, offline data access, and native device management.

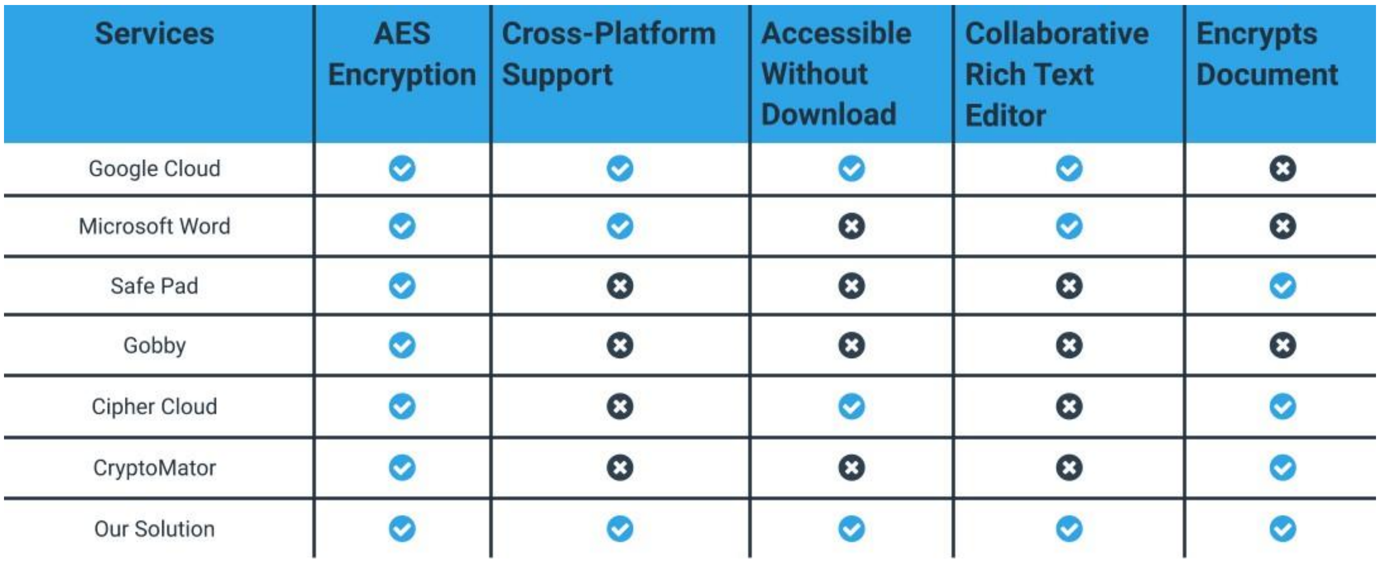
However, the lack of online collaboration features and any rich text editor capabilities are the major missing components here.

3.6 CryptoMator

CryptoMator is an open-source cloud encryption service that is fast and reliable. The software is accessible through download and works on most major mobile and computer operating systems. Data can be secured and hosted on their servers, and the user-specific folder is given the name of "vault". User filenames and files are secured under the AES encryption system and accessible from any user device.

CryptoMator states that it goes through continuous independent security audits as well as public testing regularly. Only one key is present for any data stored in a "vault", and this is only supplied to the owner of the vault at creation but can be accessed through the user's password on the system.

The lacking feature is seamless collaboration as well as any rich text editor capabilities.



Services	AES Encryption	Cross-Platform Support	Accessible Without Download	Collaborative Rich Text Editor	Encrypts Document
Google Cloud	✓	✓	✓	✓	✗
Microsoft Word	✓	✓	✗	✓	✗
Safe Pad	✓	✗	✗	✗	✓
Gobby	✓	✗	✗	✗	✗
Cipher Cloud	✓	✗	✓	✗	✓
CryptoMator	✓	✗	✗	✗	✓
Our Solution	✓	✓	✓	✓	✓

Figure 3.1 Survey of Cloud Encryption Services

Chapter 4

PROPOSED ENCRYPTED COLLABORATIVE SYSTEM

4.1 Overview

This thesis aims to propose a solution that offers a fully encrypted collaborative system for the cloud that encompasses some of the missing features from the popular systems on the market currently. The significant features we look to include in the proposed solution were AES encrypted data, support for mobile operating systems, available without any software downloads, a fully functioning collaborative rich text editor, and the ability to encrypt the document with a secret key.

Our solution was created with open-source code, with ample freedom to modify and mold the software to our needs. This solution gave us a good starting place and template to add any additional features we needed down the road. The solution code frontend is written in JavaScript, JQuery, HTML, and CSS, while the backend is built on the Django-Python architecture. The database engine is run on PostgreSQL and handles all kinds of transactions to create and update the content of the rich text editor.

Currently, our project has a working model and offers the features we intended. The data encryption uses AES cipher. Users can create a key at any time and use it to encrypt the data. Once the data is encrypted, only that key can then decrypt the data. The system also saves any key inserted for up to 10 minutes. Once the 10 minutes are up, the key does expire, and a new key needs to be created. The key can be shared via any platform with another user, allowing the solution to be collaborative. The solution can also be reached from any mobile device and does not require any external downloads to be accessed or used. Making the solution cross-platform makes it easy to access for a wide variety of users and gives the collaboration process a nice edge on the go. This process is done by hosting the app on an EC2 server using the AWS cloud services.

The solution also features a fully functioning rich text editor built upon the TinyMCE open-source project discussed in Chapter 2. This rich text editor allows for various editing and formatting options and can be used freely by any user given the secret key.

The Major component of this project is the added encryption that keeps the data secure and only accessible by the original user. Any data stored on the server is encrypted and can only be decrypted by the secret key the original user creates within the system. This feature is essential for our solution as this a key feature missing from the major collaborative editors like Google Docs and Microsoft Word.

4.1.1 The Frontend

The frontend code for the solution is written in JavaScript, JQuery, HTML, and CSS. JavaScript bootstrap engine is in command of rendering the HTML DOM for the solution. JQuery engine is in charge of handling all UI commands on the frontend controller. These JQuery operations include editing any content on the rich text editor, generating the secret key button, encrypt & save button, decrypt & load button, and the import secret key file button. In sync with the JQuery engine, the JS bootstrap engine loads the modified rich text editor from TinyMCE. The JS bootstrap engine also prepares and renders the trigger for the generate new key function, creating an AES protocol encryption-based secret key for hashing the encryption algorithm. This secret key can be stored on the user's device. The import key function allows the user to select the saved key to load into the system itself, where the key is saved for 10 minutes. Once the key is loaded into the system, the user can then use the two main functions for the encryption on the solution, the "encrypt & save" and the "decrypt & load".

The "encrypt & save" function first checks for the key to be loaded into the system. Once it has found the key, the system starts to generate the salt and starts to map the contents of the documents using the AES encryption algorithm. Once the full content of the editor is encrypted, the data is sent to the backend of the system to be handled and saved into the database.

The "decrypt & load" function is the inverse of the previous function and almost works in reverse order. The first step remains the same, and the system checks if a key is presently loaded into the software. Once the check comes back positive, a salt key starts to generate based on the secret key loaded into the system. This request is then sent to the backend, which then returns the solved text to the frontend to display onto the editor, and the data is then editable by the user.

4.1.2 The Backend

The backend code is written in the Django-Python architecture and is in charge of handling all the database requests and any of the transactions of creating and updating. Any request made by the frontend is sent through this code and is checked for validity, including the validity of the current key inserted into the system by the user.

Upon the initial load of the frontend form for the rich text editor, the current form of the data is initially displayed onto the system. This data is likely to be the encrypted data saved by the user and requires the secret key insertion and decryption functions to be used again for editing and future formatting.

The backend controller handles these requests seamlessly. Once the key is loaded into the system, and the frontend controller requests a decrypt of the data, the code first checks for the validity of the AES-based encryption key. This process results in two potential outcomes; if the key does not match the currently saved data, a "404 bad request, the key is not valid" form is sent back to the frontend to display to the user. This message alerts the user that the key being used is not the same as the secret key used to save the data initially. The other outcome for the secret key validity takes us to the next step of the decrypt process. The backend uses the salt generated to begin decoding the mapped characters saved in the database. Once this process is finished, the data can be sent back to the frontend to display for the user in an unencrypted format. This step allows the user to make any new edits or formatting changes.

The user can then once again save the data using the "encrypt & save" function. On the backend, this function takes over once the frontend sends the encrypted data. The process begins by reading the encryption key that the user has selected for the encryption process and generates the salt needed for mapping the data. Once this process is complete, the data can be written into the PostgreSQL database for future use.

While there is only one layer of encryption present on this solution, this process takes care of protecting the data from all parties and only giving access to the user that created the data originally with the secret key system. Once the initial data is written and encrypted, all future loads of the data will display the encrypted data and require the secret key insertion and decrypt functions to be validated for the data to be accessible.

4.2 The Code

In this section, we talk about the specific code for both frontend and backend solutions for this project. We will go over the significant functions that help create the optimal solution we looked to find at the beginning of this research.

4.2.1 Frontend Code

This section highlights some of the major components of the frontend code itself used for our solution. This is going to contain most of the major sections of our JS and JQuery code.

```
function checkKeyValidity() {
    let $input_check_validity = $("input[id*='check-key-validity']");
    let $rich_text = $("div[id*='rich-text']");
    let pk_ = $rich_text.data("editor-pk");
    let key_ = $("form[name*='import-key-file']").data("secret-key");
    $.ajax({
        type: $input_check_validity.data("request-method"),
        url: $input_check_validity.data("url"),
        dataType: "json",
        data: {"pk": pk_ !== undefined || pk_ !== null || pk !== 0 || pk !== "0" ? pk_ : null, "key":
key_},
        success: function (result) {
            if (result["data"]) {
                window.location.reload(true);
            }
        },
```

Here the system checks for the key validity and if the data does not match the info provided by the secret key that has been inserted the system sends back a notification letting the user know that the key is not valid and a new key must be inserted. This is an important feature to make sure the secret key has to be the one that encrypted the data.

```
        function initiateAjaxToEncryptAndSave($rich_text, pk_, key_, key_file_name,
encryptedAES) {
                $.ajax({
                    type: $rich_text.data("method"),
                    url: $rich_text.data("action"),
                    dataType: "json",
                    data: {
                        "editor_text": encryptedAES,
                        "pk": pk_ !== undefined || pk_ !== null || pk !== 0 || pk !== "0" ? pk_ : null,
                        "key": key_, "key_file_name": key_file_name
                    },
```

This bit of code helps with the initializing of the encrypt and save function, where the system takes in the current key inserted by the user and encrypts the text on the rich text editor accordingly. Once this goes through, the only way to decrypt the text is to use the same secret key used for the encryption.

```
        function initiateAjaxToDecrypt($rich_text, pk_, csrf_token, key_) {
```

```
$.ajax({
  type: $rich_text.data("method"),
  url: $rich_text.data("action") + "?key_validation=1",
  dataType: "json",
  data: {
    "key": key_,
    "pk": pk_ !== undefined || pk_ !== null || pk !== 0 || pk !== "0" ? pk_ : null
  },
  beforeSend: function (xhr, settings) {
    xhr.setRequestHeader("X-CSRFToken", csrf_token);
  },
```

Like the previous code, this set of codes initializes the decrypt feature and must be done in conjunction with the validate key from earlier.

4.2.1 Backend Code

For the backend code, there are some simple functions implemented to complement the frontend server. It is starting with the call to the database that is essential in any backend functions.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'smarteditor',
        'USER': 'postgres',
        'PASSWORD': 'postgres',
        'HOST': 'localhost',
        'PORT': '5432',
    }
}
```

This database call is to the PostgreSQL server and the setup table to store and call the data for the rest of the backend functions.

```
def post(self, request, *args, **kwargs):
    try:
        key_validation = request.GET.get("key_validation")
        if int(key_validation) == 1:
            pk_ = request.POST.get("pk", None)
            key_ = request.POST.get("key", None)
            if int(pk_) and key_:
                rich_text_inst = RichText.objects.filter(pk=pk_, key=key_).first()
                if rich_text_inst:
                    messages.success(request, "Decrypted successfully.")
                    return self.render_json_response(context={"result": "Success", "data": {
                        "msg": "Decrypted successfully."}})
        return self.render_to_response(context={"result": "Error", "message": "Key is 
invalid!!!"})
    except Exception as exp:
        pass
    pk_ = request.POST.get("pk", None)
    inst = RichText.objects.get(pk=pk_) if pk_ else None
    form = self.form_class(instance=inst)
    form.is_valid()
    instance_ = form.save(instance=inst, **request.POST)
    try:
        return self.render_json_response(context={"result": "Success", "data": {
            "msg": "Encrypted successfully.", "editor_text": instance_.editor_text, "pk": 
instance_.pk,
            "key_expired": instance_.key_expired}})
    except Exception as exp:
        return self.form_invalid(form)
```

This code set is the significant component of the backend, as the secret key validation check is met, and both requests for success encryption and decryptions are met, and any unsuccessful attempts are processed.

```
sys.stdout.write("Starting auto expiring process >>>>>>>")
    rt = RichText.objects.first()
    txt_ = rt.editor_text
    key_ = rt.key
    if txt_ is not None and key_ is not None:
        sys.stdout.write("Start auto expiring and decrypting >>>>>>>")
        decrypted_txt = decrypt(txt_, key_.encode())
        rt.editor_text = decrypted_txt.decode()
        rt.key_expired = True
        rt.key = None
        rt.key_file_name = None
        rt.save()
        sys.stdout.write("<<<<<<< End auto expiring and decrypting")
    sys.stdout.write("<<<<<<< Ending auto expiring process")
```

The key expiration part is also a vital part of our solution as it offers not only a convenience factor but allows the user to continue to work on their edit and additions to the document without having to reinsert the key multiple times.

Chapter 5

DEVELOPER EXPERIENCE

5.1 Introduction

Trust and security have been a significant discussion for tech over the last decade. With the super growth the industry has seen and with the rise in cloud storage, the topic has continued to gain attention. The security offered needs to match the data being stored on these systems. This information can be hard to attain as the major cloud storage solutions tend to be proprietary and give no insight into how secure the data is on their end of the servers. These systems offer complete security for your data from any outside attackers but tend not to include any information about the server-side protection.

As security measures tend to get more complex, the system protecting all the data should also keep improving and adding an added layer of optional security if the user desires. Encryption algorithms make it possible for any data to be secured well and hard to make it hard for hackers to attain any useable data without attaining secret keys mapping the encrypted data.

Our solution employees these security measures in a tangible way, and this is our performance impact from the advised changes to an encrypted collaborative system for the cloud.

5.2 Definitions

The rich text editor is a system interface that allows the entrance of text, allows appearance changes, formatting of the data. It is often called a WYSIWYG editor, short for what you see is what you get. Our rich text editor is running on a server instance of EC2 and can be accessed from any machine outside the localhost.

Within the rich text editor, there are other terms to consider. Changes within this editor are called edit commits, and these consist of any changes made to the document. The user that makes an Edit is called the sender. Once edits are made, the sender can choose to encrypt and save any new changes made to the document as long as a key is inserted previously into the system. If no key exists, the sender can choose to create a new key. The software then creates an AES encryption key, and this allows a map to be created with any data that is to be encrypted and saved onto the database. The saved data is then displayed as ciphertext to any user trying to access the server. To see the data as formatted plain text, the user must have a key inserted into the system and "decrypt & load" the data. As long as the correct key is inserted into the system, the server will decrypt the ciphertext and display the saved text.

For convenience, a key will stay stored on the system, once inserted, for up to 10 minutes. A message stating "Key has already expired!! Please generate a new key" will be displayed once the key storage time expires. The user can then reuse the same secret key generated previously or generate a brand-new key to be used on the system. The system will only recognize the new key once the text is encrypted and saved. Once the new key is used to encrypt the data, any previous keys will not be valid on the system.

This chain of events assures the security of the data from both outside perpetrators and anyone with access to the server. The only point of access to the data is through the secret key created by the original user.

5.3 Current Performance

In terms of performance, there are two issues to keep in mind and check for in this project: latency and a working operational transformation system.

For latency, let us talk about the two most common types we can infer for our system:

1. The latency between the sender and the server. Our rich text editor server currently does not cause any significant latency between the sender and the server, as there is no encryption occurring and the only action happening is within the rich text editor where edits are being made. As we are not deploying a complex version of the rich text editor by TinyMCE, we may expect some latency issues with large texts being edited.
2. The encryption latency is where we may see some varying latency issues pop up. As the system communicates between the frontend JavaScript and the backend python components, a few operations occur depending on the functions being called. An AES key generation must occur to generate the secret key file, and a file must be sent to the sender. Once the key is inserted into the system, the data encryption and decryption happen relatively quickly. The system only

needs to validate the key and then either map the text currently in the system or use the map to decrypt the data from the database.

The other main issue to tackle is a functioning operational transformation system. At the start of the project, our goal was to implement this into our solution; however, we could not create a working model that incorporated a true operational transformation into the collaborative environment. The issues we faced were causing the system to erase data when multiple users were working as senders. Adding this functionality to our solution would create the most optimal system for an encrypted collaborative system for the cloud that is secure and does not have any issues with multiple users.

Currently, however, our system still allows for collaboration in what we named the sea-saw method. This method requires one user to create the secret key file and edit the data as needed. Upon saving and encrypting the data, the user then must share the key file with any collaborators, and the users must take turns editing the data as needed. This method does require prior communication as well as coordinated effort during any edits made to the document.

There is also a need for user tracking that does not exist in the system, and any changes made are not tracked retroactively by the system itself. Future work on this project may take this into consideration.

5.4 Analysis

In either case, the latency is relatively low and insignificant and does not show up for small text entries. The same can be said of any large data entries in the rich text editor as well. Our proposed solution tackles the latency issue exceptionally well, and the various causes of latency experiences in other systems can be negated.

While the latency will still be higher than a document not being encrypted, this is more caused by the number of steps taken to reach an edited document rather than latency issues being caused by the encryption process itself.

Chapter 6

Conclusion and Future Work

The goal of this project was to create an advancement on the system that currently exists in terms of the security of collaborative systems for the cloud. As the amount of data stored on cloud storage systems continues to grow, there is an adamant need to secure this data to be protected from any possible source. While the major collaborative systems tend to not include any included encryption for the data stored on their systems, our solution was to tackle this and give the user a way to save only encrypted data while holding the secret key to access the data in the future.

Other cloud storage systems allow for encrypted data but are either not set up as a rich text editor or require downloading external software onto the user's device. Some cloud storage systems are not cross-platform accessible, and that is also an issue we tackled during this project.

Our current prototype is an open-source code software that can handle extensive data, formatting, and editing, is cross-platform enabled, offers a user-controlled encryption system. The missing component for our system includes the lack of a genuinely operational transformation system allowing seamless collaboration. Another missing component is user and edit tracking within the system that can be used as a version and access control system beyond the sharing of the secret key.

6.1 Experiences with Services Used

The overall experience with the languages implemented throughout this process was quite positive. Python provides an easy to configure language that is clean to look at and comes with many resources, allowing for debugging to be an easy process. The Django architecture takes this a step further and allows for easy-to-use system calls and a structured template for building backend systems. JavaScript and JQuery were similarly easy to use for the frontend configuration. Both have many resources for troubleshooting any problems and offer a significant source of libraries that can be implemented to create simple code to run functions and system calls.

In terms of server and database, Amazon's EC2 is a great resource to deploy the rich text editor server. It is easy to use and offers a free tier of service. There is an easy payment system also implemented if the deployment requires more bandwidth than initially expected. The EC2 system runs on a pay-as-you-go system, as you are only paying for the cloud computing power you use, charged monthly. For the database, PostgreSQL offers a simple, lightweight system that is easy to call and access. The most important part of the SQL database was its extensibility, making it very easy to integrate into the AWS ecosystem and in tune with the EC2 deployment of the service.

6.2 Limitations

As we conclude this thesis, we must acknowledge that this is just a prototype server deployment and is in no way a finished product. While the deployment is accessible by any user outside the local host at the time of this thesis competition, the server will be shut down in the future.

As discussed previously, operational transformation is the most significant missing component in this prototype. The server could also be subject to a distributed denial-of-service attack as the instance is not deployed on a secure network with advanced security features currently. However, this attack should not have any issues with leaking data that is not ciphertext.

Other missing security checks would need to be performed for a deployed web server of this kind as an actual product. This setup would allow many unforeseen attacks to bypass the system and access the server easily. There are some security measures in place due to the instance being deployed on the EC2 cloud computing servers.

One major flaw for the prototype is the encryption system and its requirement for the server to be refreshed after an "encrypt & save" is initialized. This issue can cause some latency and slowdown in seamless collaboration but is required for the user to see ciphertext appear once the text entry is saved on the system. Currently, the "encrypt & save" function is the only call that requires the server to be refreshed by the user.

6.3 Future Work

As mentioned previously, the next major step in the progress of this prototype would be creating a functional operational transformation system. This step would allow for seamless collaboration. Achieving this with the encryption system we currently have in place would make a completely secure cloud encryption environment. Once this is achieved, creating an organized structure for multiple encryptions saves, and decryption loads needs to be created, allowing the server to handle multiple requests simultaneously.

Another functionality that needs to be added is functioning version control for the system, allowing user tracking and ad edit tracking, allowing the users to go back to previous versions. This functionality would also require creating backups in the database, thus creating a new layer of security for the data. Having a backup system for data is good practice as it allows any failure of the system not to cause any user to lose the data they have stored on the cloud system.

Creating a secret key even to access the server would be another layer of security that would make it much harder to access the data or even the server itself. This implementation would not directly require an extra key to be created by the user; instead, it would require anyone accessing the server to have the key and ask for the key to see the rich text editor web application deployed on the server. This implementation would be a way to keep even the ciphertext secure from any attackers, as well as the server code itself.

BIBLIOGRAPHY

[1] "8 Reasons to Fear Cloud Computing." *Business News Daily*, 27 Dec. 2015, www.businessnewsdaily.com/5215-dangers-cloud-computing.html.

[2] "Google Terms of Service – Privacy & Terms." *Google*, Google, 31 Mar. 2020, policies.google.com/terms?gl=sk.

[3] Mitchell, Gareth. "How Much Data Is on the Internet?" *BBC Science Focus Magazine*, 2018, www.sciencefocus.com/future-technology/how-much-data-is-on-the-internet/.

[4] “What Is Node.js Used For? 5 Top Implementations.” *Brainhub*, 26 Feb. 2021, brainhub.eu/library/what-is-nodejs-used-for/.

[5] Agarwal, Srijan. "Building a Real-Time Collaborative Editor Using Operational Transformation." *Medium*, Medium, 15 Mar. 2019, medium.com/@srijancse/how-real-time-collaborative-editing-work-operational-transformation-ac4902d75682.

[6] Carey, Scott. "AWS vs Azure vs Google Cloud: What's the Best Cloud Platform for Enterprise?" *Computerworld*, Computerworld, 23 Jan. 2020, www.computerworld.com/article/3429365/aws-vs-azure-vs-google-whats-the-best-cloud-platform-for-enterprise.html.

[7] Underwood, Nick. *3 Benefits of Amazon EC2 Virtual Server Hosting*, 25 Feb. 2019, www.privoit.com/resources/3-benefits-of-amazon-ec2.

[8] Computer Security Division, Information Technology Laboratory. “AES Development - Cryptographic Standards and Guidelines: CSRC.” CSRC, csrc.nist.gov/projects/cryptographic-standards-and-guidelines/archived-crypto-projects/aes-development.