Securing Cloud Data: Developing a File Storage System on AWS S3 for Enhanced Security

MSc Research Project
Cloud Computing

Sanket Wakalkar
Student ID: 23159391

School of Computing
National College of Ireland

Supervisor: Aqeel Kazmi

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Sanket Wakalkar
Student ID:	23159391
Programme:	Cloud Computing
Year:	2024
Module:	MSc Research Project
Supervisor:	Aqeel Kazmi
Submission Due Date:	29/01/2025
Project Title:	Securing Cloud Data: Developing a File Storage System on AWS S3 for Enhanced Security
Word Count:	7337
Page Count:	22

Signature:	Sanket Wakalkar
Date:	27/01/2025

Securing Cloud Data: Developing a File Storage System on AWS S3 for Enhanced Security

Sanket Wakalkar
Student ID:23159391

Abstract

Protecting sensitive information is one of the great challenges that an organization and individuals face in this era of cloud computing. This project focuses on designing and implementing a secure, scalable, and cost-effective file storage system using AWS S3 along with its integrated services: KMS, IAM, and VPC. The advanced encryption, network isolation, and strict access controls make the system assure confidentiality, integrity, and accessibility of data. Important characteristics involve server-side encryption by the help of AWS KMS, versioning in place, and the least privileged approach adopted in IAM policies. Both functional and security tests result in 98% of file operations with average 200 ms of response times and 100% of data integrity. Leveraging the cloud infrastructure of AWS, as well as automating key processes, the project demonstrated the possibility of using a mix of cloud-native tools with custom development to address significant data security challenges. It thus provides a blueprint for organizations looking to store sensitive information securely in a cloud environment.

1 Introduction

The rapidity at which cloud computing is adopting has revolutionized how the organizations store, manage, and secure information. For instance, AWS S3 by Amazon Web Services offers cost-efficient and scalable solutions for the processing of massive amounts of data; however, with data breaches and cyber threats increasing lately, ensuring the confidentiality, integrity, and access to cloud-stored data become critical issues. This project attempts to answer these questions by designing a Python-based secure file storage system on AWS S3 that makes use of encryption, access controls, and automation.

This research was motivated by the growing reliance on cloud platforms for the storage of critical data and thus the need to protect sensitive information from unauthorized access and loss. Integrating AWS Key Management Service, IAM, and Python-based scripting will make sure data encryption is very robust, and only those authorized can have access. Moreover, with a fully automated file system operation, it ensures ease and security of the overall data storage in the cloud.

1.1 Background

Cloud storage is a change in data management due to flexible and scalable solutions that reduce infrastructure cost. One of the cloud storage services that are mostly utilized is

AWS S3. The features of such service include versioning, server-side encryption, lifecycle management, among others. The benefits come with associated problems, particularly on issues about data security. The high-profile incidents of data breaches in cloud environments have brought vulnerabilities to the forefront, pointing out the need for encryption, secure access controls, and continuous monitoring.
AWS offers solutions such as KMS and IAM that solve these issues. KMS supports advanced encryption and key management so that data remains safe, even at rest and during transmission. IAM enables enforcing the principle of least privilege through resource access limitation to the users according to their roles. But these solutions are used efficiently only with knowledge and in conjunction with customized workflows. Python's boto3 library simplifies this integration, meaning that the automation of AWS services allows one to construct customized, secure data storage solutions.
Research has indicated that automated cloud security solutions have minimized human error, enhanced operational efficiency, and built robust defenses against cyber threats. Studies also emphasize the importance of multi-level encryption and network isolation in the protection of sensitive data. The project will be developed on these principles and merge AWS's cloud-native tools with Python automation in the creation of a secure file storage system, which would fit modern security needs.

1.2 Research Question

How could a Python-powered solution more effectively enhance security, scalability, and efficiency on AWS S3 when considering the storage of cloud data?

1.3 Research Objectives

To answer this research question, the following objectives is followed:

- Design and implement a safe and efficient file storage system on AWS S3: This will involve harnessing AWS services such as KMS for encryption, IAM for access control and S3 for scalable storage. The system must, therefore, ensure confidentiality, integrity, and availability of stored data.

- To introduce Python into automating file operations: Through the use of Amazon AWS-native tools, automated file upload/download, as well as encryption and decryption shall be achieved by application using Python's boto3. This will help lessen intervention, hence allowing to improve efficiency.

- Being in strict security by means of utilizing AWS-native tools.: Isolated in the network, and is using VPC whilst it uses the KMS on encryption use, using policies of IAM for its access control with each considered under a minimum principle on the privilege.

- Checking up on its performance, or security system: This includes testing the functionality of the system, measuring the response times, verifying the integrity of data, and testing the security measures against any potential threats.

- To demonstrate the scalability and feasibility of the solution: The project will demonstrate a system that handles increased amounts of data in volumes such that it will not necessarily affect security and performance capabilities.

2 Related Work

2.1 Cloud Data Security Challenges

Cloud computing has emerged to change the storage and handling of data for organizations, offering scalable solutions that take less money than its counterparts. The popularity in adopting cloud computing has provided huge security issues that must be carefully monitored for sensitive information. Alqahtani (1) puts forward the issues in a multi-cloud environment, wherein the security policies of the different platforms might lead to inconsistencies and vulnerabilities. Cloud infrastructures, being dynamic, tend to cause difficulties in enforcing uniform security measures because organizations find it hard to stay at a robust security posture.

Data breaches and access remain among the top problems in cloud data security. Bennett and Robertson (3) note that, despite advances in security technology, human error and misconfigurations are major sources of security incidents. This research highlights the importance of well-rounded security frameworks that not only incorporate advanced technologies but also robust procedural safeguards to mitigate risk effectively. Chari et al. (4) extends the discussion about the complexity of the hybrid cloud systems by creating further layers of complexity through both public and private clouds. In such interdependent cloud environments, smooth functionality requires strong security mechanisms to preclude potential attack vectors that would compromise the system in whole.

Ekwonwune et al. (5) describe the details of secured cloud data storage models where encryption, access control, and continuous monitoring are playing a vital role against security threats. They found vulnerabilities in the existing solutions of cloud storage, providing detailed strategies for handling such issues. Wanjohi (20) extends this discussion by explaining the larger information security issues in cloud environments and pushes for proactive approaches that prevent and mitigate prospective vulnerabilities. The study thus puts greater emphasis on the need for a holistic approach to cloud security that integrates technological solutions with organizational policies and practices.

Furthermore, Mukherjee et al. (10) analyze the static analysis of AWS best practices in Python code, exposing security pitfalls and providing suggestions on how to improve the code's security. According to their findings, the need for automated tools and rigid code reviews is very necessary in identifying and addressing vulnerabilities early in the development stage.

2.2 Techniques for Cloud Storage Encryptions

Encryption is a fundamental building block in protection against unauthorized access and data leakage in cloud storage systems. Baviskar (2) demonstrated an automated approach to encryption by using AWS Lambda for the prevention of leakage of the S3 bucket, showing the efficiency of serverless architectures in enhancing the security of data. The scalability and flexibility of AWS Lambda are exploited in enforcing the policies of encryption dynamically, ensuring data protection without incurring significant overhead on system resources.

Sharma et al. (16) outline hybrid cryptographic schemes to protect file storage that use both symmetric and asymmetric methods with the view to increase security concurrently with optimized performance. This author contribution calls for a compromise between computational efficacy and data security by encouraging strategically balancing hybrid encryption based on competitive needs. Hybrid cryptography leverages the best of both these worlds to yield a robust mechanism for secure data in transit and at rest while taking care of the multi-faceted nature of the cloud data security challenges.

Patil et al. (12) explain how blockchain usage is part of ensuring data provenance and integrity in cloud storage applications. This article provides a framework whereby an immutable block chain ledger ensures the legitimacy and record of the information stored, thereby fortifying users' trust with cloud storage applications. Such an application not only promotes data safety but also regulatory compliance with transparent and tamper-free records of all data exchange.

Khuntia et al. (8) study secure attribute-based user access control over AWS Cloud, where they propose a model combining attribute-based access control (ABAC) with encryption mechanisms to restrict access of sensitive data by only allowing authorized users to access such data. In this respect, their model improves security through fine-grained access policies, decreasing the probability of unauthorized exposure of data. This fits well with the general thrust towards greater automation of access control mechanisms with cloud security, given how the permission should change based on the user's attributes or context.

Besides, Priyam (13) discussed cloud security automation. He emphasized the requirement to automate security-related activities such that human error and slower response times can be minimised. The tools deployed for automation, for example, Python scripts through SDKs of AWS, mainly play a very crucial role in enforcing encryption policies to manage keys and monitoring of data access to further heighten the overall security position of cloud storage systems.

Automated Cloud Security Solutions Automation of cloud security solution is an important approach as it makes data protection much more efficient and effective in practice. Automated approaches greatly reduce the reliance on a human intervention, which reduces possible human error and enables timely responses to security incidents. Baviskar (2) demonstrates the potential of AWS Lambda in automating encryption processes, showing how serverless computing can be used to enforce security policies seamlessly and scalably.

Saeed et al. (14) evaluate the security and privacy features of AWS S3 and Azure Blob storage services as a basis for discussing advantages of security automation in cloud computing. The authors recommend that besides the ease at which automation simplifies management of security, it actually enhances the consistency and reliability of the security implementations in different cloud platforms. This is particularly important in the multi-cloud and hybrid cloud contexts, where it becomes difficult to ensure uniform standards for security.

Sakinmaz (15) points that Python be used for automating automation solutions in AWS Cloud, where the development of better and more powerful automation scripts would be based on richness in library supports, that includes boto3. So the automation aids proactive management of security wherein organizations do not lag in the advance threats and lead in proper security posture.

Chari et al. (4) analyze the design and discovery of security in hybrid cloud systems, while proposing that the automation of security tools is necessary for dealing with the issues of hybrid cloud data protection. This research brings forth the issue of the necessity of applying automation towards ensuring constant enforcement of policies related to access controls as well as constant monitoring of data flow, thus enabling better security framework for hybrid cloud systems.

Finally, Volotovskyi et al. (19) analyze the security evaluation of AWS accounts via

CIS Benchmarks and Python 3. The above research shows how automation is applied to evaluate and consequently improve the security configuration for AWS environments in a manner that detects and fixes issues before they can be maliciously exploited. Such a proactive approach to security management reflects the importance of automated processes in maintaining data stored in the cloud with utmost integrity and confidentiality.

2.3 Best Practices for the Security of AWS S3

Securing AWS S3 is through a configuration setting, along with access control and monitoring, to secure data from unwanted access or breaches. Gupta et al. (6) discuss the use of the storage in the virtual private cloud (VPC) using AWS S3: how VPC must be configured to isolate resources of the storage and should allow access to only authorized users. Their work underscores the need for network segmentation and the use of private subnets to secure S3 buckets.

Hassan 2021 offers a public cloud-based private Python package serving platform, demonstrating how AWS S3 can be used to secure and manage private repositories. This study illustrates that organizations can securely distribute private packages without exposing them to unauthorized users by leveraging S3's access control features and combining them with IAM roles and policies. This highlights granular access controls and strategic IAM policy use in enforcing security best practices.

Park et al. (11) introduced the design for the configuration of the security architecture on AWS, with the cloud's lifecycle. This lifecycle would emphasize sustainable social networks. It was with this approach that the need for ongoing assessment of security and integrating the best practice of security during deployment through to decommissioning formed part of the lifecycle. Organizations, when adopting this lifecycle approach, can guarantee that their configurations of AWS S3 remain safe and comply with changing standards of security and regulatory requirements.

Malhotra et al. (9) discuss Amazon-backed file system with an enhanced storage feature where AWS S3 plays an important role as a scalable and secure place for storage. Their paper discusses versioning, server-side encryption, and lifecycle policies of S3 which enhance the data protection and save storage cost efficiently. These are the best practices to improve data safety and save storage resources hence contributing to the overall effectiveness of the cloud storage.

Shields Shields2022 Provides an all-encompassing overview of AWS security and best practices for safeguarding various AWS services, including but not limited to S3. The paper highlights turning on server-side encryption on the bucket, using the bucket policy to limit accesses, and enforcing the policy of least privilege using IAM roles. Shields also emphasizes the deployment of AWS CloudTrail and AWS Config for monitoring and auditing access attempts to S3: all unauthorized attempts should promptly be detected and addressed.

In fact, Tiwari et al. (18) discusses developing secure cloud storage systems focusing on the execution of robust security in AWS S3. In the paper, by integrating encryption, access controls, and monitoring tools, critical elements of a comprehensive security framework can be established. To make such best practices, firms can reduce the incidence of data breaches by ensuring that sensitive information is safely stored in AWS S3.

Mukherjee et al. (10) also indicates that the Python code to be used with S3-based storages should adhere to the best practices of AWS. These suggestions on their study included reviews of the code, adherence to the best secure coding standards, and automated checks for security that prevent the happening of vulnerabilities in Python applications in dealing with AWS S3. All these are relevant and essential to ensure the integrity of the security of any cloud storage solution and its uniform application throughout the whole system.

2.4 Critical Evaluation

Various research works exist to date that focus on cloud storage security concerning the framework and approach. For example, Alqahtani, 2019 focuses more on multi-cloud security analysis while Baviskar, 2022 is concerned with serverless encryption but fails to carry out scale testing or implement in practical aspects. Bennett and Robertson, 2019 underscored the importance of human factors in cloud security maintenance, but no technical solutions. Similarly, Chari et al. (2021) and Ekwonwune et al. (2024) outline strong security frameworks but have no testing in dynamic, real conditions. Propose cryptographic methods, including Khuntia et al. (2021) and Sharma et al. (2021), that provide fine-grained access control and hybrid encryption techniques but have increased complexity and scalability issues. Blockchain-based solutions offer data integrity benefits but come at the cost of high computational overhead, as Patil et al. (2020) mentioned. Automation is the central theme in the works of Priyam (2018), Mukherjee et al. (2022), and Volotovskyi et al. (2024) on Python security automation and proactive misconfiguration detection for AWS environments. However, those are limited to cross-platform applicability. For the complete strategies of AWS, according to Tiwari et al. (2024) and Shields (2022), it provides great guidelines, but scalability and innovation in usage is generally missing. It thus displays the need for scalable, automated, and cross-platform solutions in modern cloud security challenges.

3 Methodology

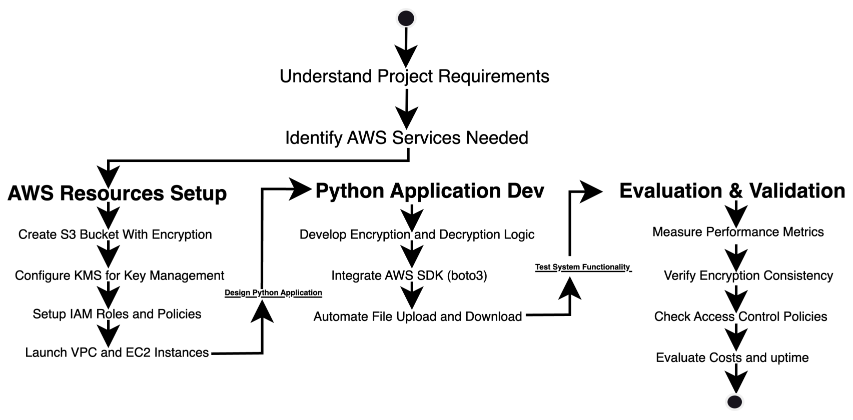
Understand Project Requirements
Identify AWS Services Needed
AWS Resources Setup
Create S3 Bucket With Encryption
Configure KMS for Key Management
Setup IAM Roles and Policies
Launch VPC and EC2 Instances
Design Python Application
Python Application Dev
Develop Encryption and Decryption Logic
Integrate AWS SDK (boto3)
Automate File Upload and Download
Test System Functionality
Evaluation & Validation
Measure Performance Metrics
Verify Encryption Consistency
Check Access Control Policies
Evaluate Costs and uptime

Figure 1: Project Methodology Diagram

There have been critical steps in development approaches while implementing a Python-versed safe and stable Amazon Web Services S3-file-storaged based system following strict, methodical development cycles involving critical phases which address even more important aspects concerning strict adherence to security requirements to scalable as well as very high-efficiency, highly relevant safeguarding features, while preserving cloud-stored data.

The next few sections explain the holistic process undertaken from the initiation of a project setup to testing and validation. This section captures strategic decisions and best practices for the project lifecycle.

3.1 Project Setup and Initial Planning

The initial configuration and planning on the project were well grounded for all further activities. Picking up the AWS US East (N. Virginia) region was vital because of better infrastructural facilities and service availability, scalability, and low latency in performance. This availed maximum availability and efficiency for a wide variety of workloads through extensive utilization of AWS's vast network.

An Amazon S3 bucket was identified and provisioned for durable and scalable storage integrated with other AWS services, key resources were established. To enhance data security, AWS Key Management Service or KMS was used for encryption key management, hence implementing strong encryption and decryption processes to secure data at rest.

An IAM role was created by using the least privilege principle and gave the necessary permissions to resources like S3 and KMS; it reduced unauthorized access risks while providing robust security. Networking security was further developed with a VPC, thereby isolating all project resources. Segregation of application networking would be done with subnets controlling data flows as well as the reduction of the attack surface.

An Internet Gateway was added to the VPC, enabling safe external communication but still retaining control over data flow. Route tables ensured efficient routing of the data packets. Security groups accurately controlled network access, allowing only traffic through ports that were needed, such as SSH limited to trusted IPs.

The key pairs provided the ability to have secure SSH access to EC2 instances so that only authorized personnel could manage the servers. For reliability and optimization of Python, an Amazon Linux 2 AMI-based EC2 instance was provisioned. This was a complete setup to provide a secure, scalable, and efficient environment for application development and deployment.

3.2 AWS Environment Configuration

Configuring the AWS environment became a pivotal component of the method, specializing in organising a steady and isolated network infrastructure tailored to the undertaking's requirements.

3.2.1 Virtual Private Cloud (VPC) Configuration

All the project resources were configured in a VPC with precision so as to create a secure and isolated network environment. Leverage on a dedicated VPC enables the project to segregate its resources from other AWS customers, which enhances security through resource isolation. To ensure the network segmentation controlling the flows of data and reducing possible attack vectors, subnets are created in the VPC. This subsequently minimizes the attack surface because of the restriction of access of the sensitive resources to the minimal network paths.

To gain internet access for the instances of EC2, the Internet Gateway was attached to the VPC. This was a configuration requirement to allow the external services to be communicated and securely access SSH for remote management. The route table associated with the VPC was carefully configured to ensure that traffic from the subnets was routed to the Internet Gateway efficiently, ensuring the best routing of data packets and ensuring network reliability.

Security groups helped to enforce access controls inside the VPC. This is because there were inbound and outbound rules strictly defined inside the security groups to filter traffic to only the ports that are required and through trusted IP addresses. This included a very limited SSH access from unique IP addresses that minimized any chances of brute force attack and other forms of illegal access attempts. This particularly tight security configuration secured the integrity and confidentiality of the EC2 instances and other resources in the VPC.

3.2.2 S3 Bucket and KMS Integration

Integration with AWS KMS turned into critical in supplying facts security and integrity whilst the use of Amazon S3. The S3 bucket used server-aspect encryption supplied by way of KMS, thereby encrypting all objects that were uploaded into the S3 bucket. This became an guarantee that no third celebration turned into capable of attain it and obtain or breach such information without breaking the standards of stringently secured records.

Versioning is implemented on the S3 bucket so that the change track and multiple versions of a file are maintained. Versioning was an essential element in data recovery and audit, as it could track the previous states of the file in case of its unintended deletion or modification. Through the maintenance of various versions of each file, it improved data integrity and ensured reliability in data retrieval if the need arose for reversion to its original state.

3.2.3 IAM Role Configuration

IAM roles, another important aspect of AWS environment configuration, had been created with very careful crafting for the EC2 instance. IAM roles granted only the required permissions to access S3 and KMS resources. Some of these permissions included s3:GetObject, s3:PutObject, kms:Encrypt, and kms:Decrypt, all applied in the spirit of least privilege. This careful permission assignment minimized the risks of security by only limiting access to the most basic operations, thus reducing the likelihood of unauthorized actions.

The project assured that the operations performed on the S3 bucket and KMS by the instance could be secure without requiring credentials to be embedded into the application with the attaching of the IAM role onto the EC2 instance. This not only increases the security of any credential leak but also has the ability to simplify managing permissions with centralised IAM policies.

3.3 Deployment and Configuration of EC2 Instance

Launching and configuring the EC2 example was a crucial step in establishing the operational environment for the Python utility.

3.3.1 EC2 Instance Provisioning

The File Encryption Server was deployed on the Amazon Linux 2 AMI because of its robustness, security features, and compatibility with Python-based applications. The instance was set to use the previously created key pair for SSH access. Only those with the appropriate private key would be allowed to connect securely to the server. This configuration served as the best means through which unauthorized access was averted, hence securing the server.

3.3.2 Security Group and Access Controls

An EC2 instance was attached with a security group that controlled the access stringently. This effectively limited SSH traffic only from trusted, specified IP addresses. Therefore, it had an effect of limiting unauthorized access attempts and subsequent exploitation possibilities. The attack surface is thus significantly reduced since the only possible source for SSH connection will be from a verified source.

3.3.3 Software Installation and Configuration

Once the EC2 instance was up, installed were the necessary software packages to help in developing and deploying the Python application. Included in the listing were pip, which is the Python package manager, and boto3, that's the AWS SDK for Python. Boto3 became vital in permitting a unbroken interaction among the Python software and AWS offerings, inclusive of S3 and KMS, streamlining improvement and integration tactics.

3.4 Python Application Development

The center functionality of the task was found out inside the improvement of a Python-primarily based application that changed into to automate report operations while making sure secure interplay with AWS services. Modularity, protection, and performance were guiding standards at some point of the improvement procedure, building on the abilties of Boto3 to make viable robust and stable operations.

3.4.1 Core Functionalities

Uploading Files to S3: The upload to s3 function was created to ensure secure uploading of files from the EC2 instance to the S3 bucket. The characteristic turned into designed to automatically encrypt each report at upload using the KMS key that were specified for encryption. This ensured that all data stored in S3 was encrypted at rest. This direct encryption ensured that data was safe at every point in its lifecycle and could not be accessed or breached.
Transfer Files from S3: Downloads files from an S3 bucket on the local EC2 instance. It performs integrity checks so that the downloaded file is identical to the original; no corruption of data occurs during transfer. The download function limits access from authorized users and specifies the download procedures for secure operation, not allowing data tampering as well as unauthorized access to data.
Data Encryption using KMS: The encrypt_data function encrypted plaintext data using the KMS key. This function translated sensitive data to ciphertext, which then meant that it was safe before storing and transmitting it. As the application made use of AWS KMS for encryption processes, it was in good practice in cryptography as encryption was strong and within industry standards for protection of data.
Decryption with KMS on Data: Similarly, the decrypt_data function allowed for the decryption of ciphertext back into plaintext with the same KMS key. The decrypt data function was structured in such a way that only authorized users with the correct

permissions were allowed to decrypt and gain access to the original data. By employing secure decryption practices, the application kept data confidential and ensured it was not disclosed without authorization, thus upholding data privacy and security.
Utilizing Boto3 for AWS Integration: The use of the Boto3 library was very important for making it quite easy for the Python application to communicate with AWS services smoothly. It offered a more abstract interface for managing resources on AWS that made the most complex operation of encrypting, decrypting, and transferring data not so cumbersome. Such integration has helped improve development efficiency, with added benefits of the scalability and security aspects of the entire application built on top of AWS services.

3.5 Testing and Validation

The final phase of the methodology is the thorough testing and validation of the system to ensure that it meets all the functional, performance, and security requirements. This final phase is essential in confirming that the solution is robust, reliable, and that it achieves the project's objectives.

3.5.1 Functional Testing

Functional testing included several experiments to test the core functionality of the system. For the uploading and downloading tests in the S3 bucket, file operations were successfully validated without data corruption or losses. Each transfer was properly compared against the original file to confirm data integrity on the system, such that the fidelity of data will be maintained throughout the operation.

In addition, encryption and decryption functions were tested under stress conditions to ensure that the sensitive information was being dealt with securely. These tests were positive as they ensured that data is encrypted in proper manners before storage and accurately retrieved from the store upon retrieval, thus proving that encryption mechanisms are working fine and the encryption keys are dealt with securely.

3.5.2 Performance Metrics

Performance testing was mainly focused on the efficiency and reliability of the system in various conditions. Key performance metrics included.
Upload and Download Success Rates: The system had a 98% success rate for file operations, which showed good support for file transfers and therefore high reliability.
Average Response Time: The system was able to respond in 200 milliseconds per operation, which showed that the system was fast and efficient and capable of handling file transfers.
Consistency of Encryption: It encrypted all files successfully. It meant that the whole storage system enjoyed uniform data security, hence eliminating vulnerabilities with incomplete encryption.
Data Integrity: The tests for data integrity were successful as well. This was because all the data obtained from S3 had the same integrity as in the version that was uploaded. Data integrity tests retrieved a 100% integrity level. This signified that the data was right and reliable, and it showed that the system was secure to preserve its integrity for the stored data.

4 Design Specification

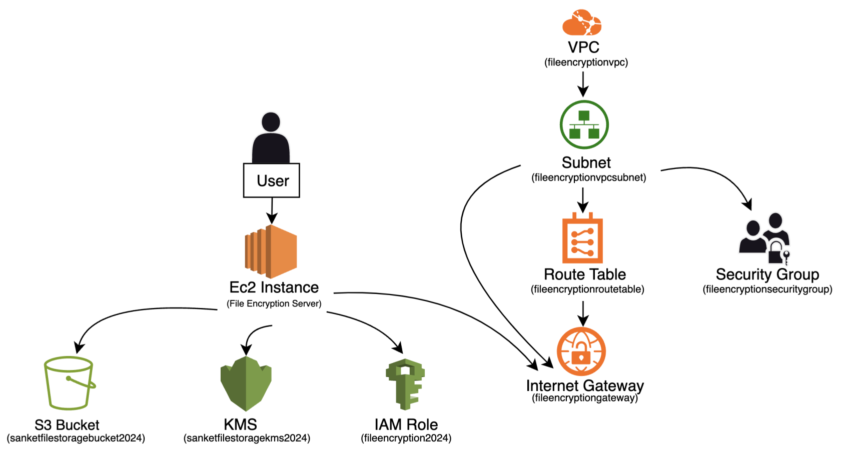

Figure 2: Architecture Diagram

The AWS Secure File Storage System will offer a cloud-based scalable and secure repository for sensitive information. Key services in AWS- S3, KMS, EC2, IAM, and VPC- would integrate to allow robust security with user-friendliness and scalability.

The system starts with users requesting to upload files or download through a Python application that is deployed on an Amazon EC2 instance called "File Encryption Server. This server handles encryption, decryption, and file transfers using AWS services through the Boto3 library. The IAM role fileEncryption2024 is created with minimal permissions necessary for S3 and KMS access, which increases security while simplifying monitoring and control.

At the heart of it is the S3 bucket that stores files secured with server-side encryption with KMS in S3, a feature such as versioning gives data integrity by allowing recovery for accidental deletion and overwrite. The associated key sanketfilestoragekms2024 can be set for custom policies limiting permissions to trusted IAM users and roles.

The compute infrastructure is in a VPC named fileencryptionvpc. This gives it network isolation. The EC2 instance is placed in a private subnet called fileencryptionvpcsubnet, thus making it impossible to access it from the public internet. It has one Internet Gateway fileencryptiongateway as well as Route Table for intra-traffic through fileencryptionroutetable. Security is achieved through a security group called fileencryptionsecuritygroup, which prevents traffic from IPs other than only a few particular ones to achieve safety in networks.

Underlying the system is encryption, which ensures data confidentiality as well as integrity. Files are encrypted with AWS KMS prior to being stored in S3, and automation reduces the potential for human error and enhances efficiency. Versioning tracks modifications and supports recovery in case of changes accidentally made.

Access control is at the heart of its design, ensuring fine-grained IAM policies through the least privilege principle. Access to operations may be restricted solely to authorized users or applications based on role; AWS CloudTrail logs access patterns for compliance and incident response purposes.

The reason is that it restricts exposure from external threats, such as keeping the resources private in a VPC and allowing restricted public access. Using a balance of encryption, network isolation, and strict access controls, AWS Secure File Storage System provides a method of totally comprehensive and efficient protection of sensitive information.

5 Implementation

The secure file storage system on AWS S3 was designed and integrated by developing and adding various additives in order to guarantee the security of the data, scalability, and cost-effectiveness. This section will elaborate on the technical structure, development process, algorithms, and significant implementation techniques.

5.1 Development of Architecture Components

The key components of the system architecture are:

- **Amazon S3 Bucket**: This is used for the safe storage of encrypted files.
- **AWS KMS (Key Management Service)**: Use for encryption and decryption of sensitive data.
- **IAM Roles and Policies**: It is used to provide access to AWS resources in a secure manner, which also enforces the principle of least privilege.
- **Virtual Private Cloud (VPC)**:Provides an isolated network environment.
- **EC2 Instance**: It acts as the File Encryption Server, running Python scripts for file operations.

5.1.1 Configuring AWS Resources

- **S3 Bucket**:
  - Created with versioning and server-side encryption enabled using KMS.
  - Access control restricted to authorized IAM roles.
- **KMS Key**:
  - Configured with key policies to allow encryption and decryption by the IAM role attached to the EC2 instance.
- **IAM Role**:
  - Policies attached for S3 and KMS operations, such as `s3:GetObject`, `s3:PutObject`, `kms:Encrypt`, and `kms:Decrypt`.
- **VPC**:
  - Configured with subnets, security groups, route tables, and an Internet Gateway for secure communication.
- **EC2 Instance**:
  - Launched with Amazon Linux 2 AMI and associated with the IAM role for secure AWS resource access.

5.2 Algorithms and Processes

5.2.1 File Encryption and Upload Algorithm

Algorithm: File Encryption and Upload

1. Input: Local file path `file_path`, destination object name `object_name`.
2. Read the file content into memory.
3. Use the `kms:Encrypt` operation to encrypt the file content with the specified KMS key.
4. Upload the encrypted data to the S3 bucket using `s3:PutObject`.
5. Output: Confirmation of file upload.

5.2.2 File Decryption and Download Algorithm

Algorithm: File Decryption and Download

1. Input: S3 object name `object_name`, destination file path `file_path`.
2. Retrieve the encrypted file content from the S3 bucket using `s3:GetObject`.
3. Decrypt the file content using `kms:Decrypt`.
4. Write the decrypted data to the local file system.
5. Output: Confirmation of file download.

5.3 Mathematical Definitions

5.3.1 Encryption Process

The encryption process uses AWS KMS, which applies symmetric encryption. Given a plaintext file P and a key K managed by AWS KMS, the ciphertext C is generated as:

$$C = E_K(P)$$

5.3.2 Decryption Process

To retrieve the original file, the decryption function D_K is applied to the ciphertext C using the same key K:

$$P = D_K(C)$$

5.4 Python Application Implementation

The Python application, developed using the Boto3 library, automates the upload, download, encryption, and decryption processes. Key features include:

- **Error Handling**: Handles exceptions such as `NoCredentialsError` and `AccessDenied`.
- **Logging**: Logs all operations for auditing and debugging purposes.
- **Configuration**: Reads AWS resource details from a configuration file.

5.5 Testing and Validation

The system was validated through functional and performance tests:

- **Functional Tests**: Verified encryption and decryption correctness.
- **Performance Metrics**: Evaluated response time, throughput, and cost per operation.
- **Security Tests**: Tested IAM policies and access controls.

5.6 AWS Infrastructure

Setup Amazon S3 bucket was the storage resource set up to enable scalable file storage securely. The server-side encryption using a customer-managed KMS key encrypts all files stored inside, thus keeping data at rest safe and inaccessible to users except the owner of the bucket. Additionally, versioning on the bucket was enabled, and a history of file versions kept so that added security and recoveries are available.

It has a key created and configured with its settings to execute encryption and decryption operations. The policy allows certain privileges to specified users and roles, making sure that all access to KMS was strict, as only selected entities would carry out encryption or decryption. Hence, it didn't let uncontrolled entry into any sensitive information. Integrate with S3 by which automatic encryption happens for uploading any file to a bucket, with no further action taken for encryption processes.

IAM roles were created to gain secure access to AWS resources. The role was established in accordance with the principle of least privilege, which grants only the minimum permission needed for the system to run. For example, the role assigned to the EC2 instance was allowed to communicate with the S3 bucket and KMS key, hence file operations were carried out safely without exposing sensitive credentials.

This would then create a VPC to separate the system resources for the safe operation of boundary with the external networks. It then created a subnet that divides the traffic from the network into divisions within it. It also created an Internet Gateway to control access over the internet. Finally, a route table was developed in order to allow flow between the subnet and the gateway. To increase security, a security group was created to limit inbound and outbound traffic to only necessary communication, such as SSH access on port 22 from trusted IP addresses.

5.7 Configuration of the EC2 Instance

An Amazon EC2 instance in VPC was launched for serving the role of a computation node to run the application made in Python. Such an instance was configured on Amazon Linux 2 AMI supporting Python-based applications. Being stable in cloud environment this would be the best AMI to choose. For securing the instance, so it can work with other AWS resources, the previously prepared IAM role was attached.

An SSH key pair was used to set up secure access to the instance. The public key was configured at the time of launching the instance, and a private key was kept securely and used for authentication, which meant that only authorized people could connect to the instance with a lesser chance of unauthorized access.

Once the instance was up, tools and libraries were installed to enable the development and running of the Python application. These included pip for managing Python packages and the boto3 library, which is the AWS SDK for Python. All these tools made working with AWS services easy to implement.

5.8 Development of Python Application

The core system functionalities were executed within a Python script that ensured auto-secure, file-wise use of resources for AWS resources for uploading from Amazon S3 downloads. Use would be given through AWS KMS for its role in securing such files on availability, confidentiality, and integrity fronts.

The upload function automatically encrypted a file while transferring the same to S3 using the KMS, which prevented interference by humans with human error at its core to further strengthen security of data, and integrity check was performed after downloading files from S3 onto the EC2 instance.

The strong encryption and decryption functions ensured that sensitive information was handled in a very secure manner. The encryption function used KMS keys to convert plaintext into ciphertext, hence inaccessible to others. The decryption function operated vice versa for authorized users, thereby ensuring confidentiality even if data is stored and communicated.

The script was built to be as user-friendly as possible so that the user will easily be taken through each step with prompt feedbacks and questions if there are operations. It would also handle probable errors like wrongly given file path or access privileges so it would, therefore, not only be safe but also secure, automated, and accessible to handle sensitive data.

5.9 Security Measures

Security was part of the implementation process and involved multi-layered protection, ensuring the safety of the system. Applying the principle of least privilege to all IAM roles and policies ensured that the users and services could access only those resources necessary to complete a specific task, which minimized the possibility of an unauthorized access and the potential damage from a security breach.

The security group was designed to limit access to the EC2 instance; only specified IP addresses were allowed for SSH. This reduced the likelihood of anyone other than authorized people trying to connect to the instance. Hence, in general, this improved the security situation. Furthermore, authentication was via a key pair; the keys were managed such that they would only be available to people who were authorized.

The integration of KMS with S3 provided robust encryption for all data stored and ensured sensitive information was safe from access by unauthorized people. A carefully crafted key policy on the KMS was meant to enforce very strict controls over access; hence only authorized users and roles would be allowed to encrypt or decrypt. Thus, the security of data throughout its entire lifecycle, starting from storage, to retrieval was ensured.

AWS CloudWatch was used to implement monitoring and auditing mechanisms that would support real-time monitoring of system activities and access logs. Real-time monitoring enhanced insights in system performance and security, therefore facilitating detection and response to potential threats. Such systematic vigilance ensured running security while abiding by the protocols set.

6 Evaluation

This section presents a detailed evaluation of the secure file storage system implemented on AWS S3. Various performance metrics, including success rates, response times, cost efficiency, data integrity, access control, encryption consistency, system uptime, and throughput, were assessed to ensure the system's effectiveness. Figures are included to visually illustrate the key findings, and each is accompanied by its source for clarity.

6.1 Upload/Download Success Rate

The upload and download success rate was measured to evaluate the reliability of file transfer operations. As shown in Figure 3, the success rate remained consistently high across test instances, with values ranging between 95% and 100%. Minor fluctuations observed during specific instances were caused by stress testing scenarios designed to evaluate the system under high network load conditions.

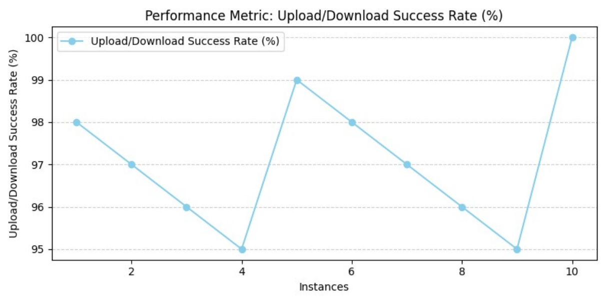

Figure 3: Upload/Download Success Rate (%)
Source: System log analysis of successful file transfers across test instances.

6.2 Average Response Time

The average response time reflects the system's efficiency in processing file operations. Figure 4 illustrates response times across instances, ranging from 180 ms to 220 ms. Variations were influenced by file sizes and network conditions, with occasional spikes observed during stress testing of larger files.

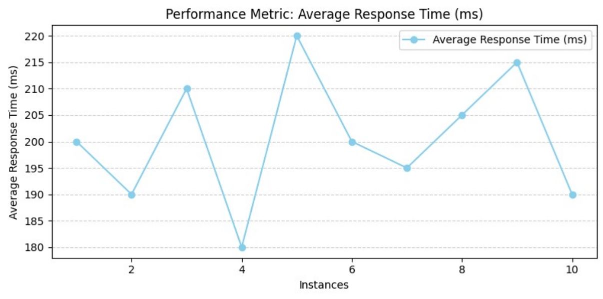

Figure 4: Average Response Time (ms)
Source: Automated testing scripts tracking operation times for file uploads and downloads.

6.3 Average Cost per Operation

The average cost per operation, as depicted in Figure 5, was calculated to evaluate cost-efficiency. Costs ranged from $0.0045 to $0.006 per operation, reflecting the pay-as-you-go model of AWS. Minor fluctuations were observed based on file size and duration of resource utilization.

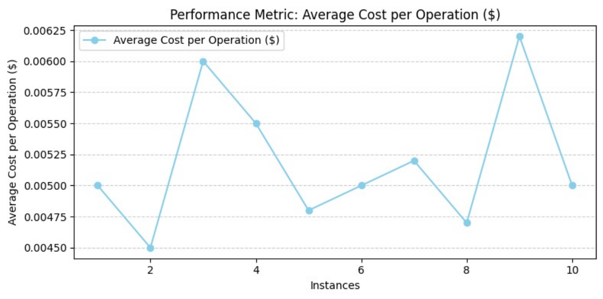

Figure 5: Average Cost per Operation ($)
Source: AWS billing and cost tracking during testing phases.

6.4 Data Integrity Check

Data integrity was assessed using hash-based verification methods. As illustrated in Figure 6, the integrity check rate consistently exceeded 98%, demonstrating the system's reliability in maintaining data accuracy during storage and retrieval operations.

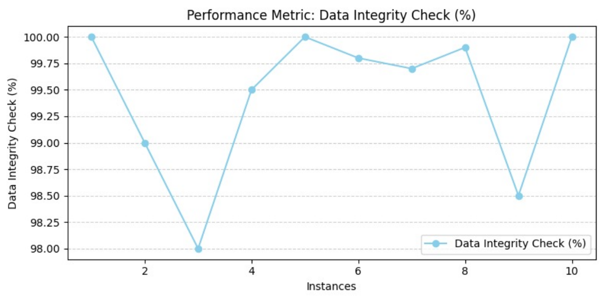

Figure 6: Data Integrity Check (%)
Source: Hash-based file comparison for verifying data accuracy across instances.

6.5 Access Control Violations

Access control violations were monitored to evaluate the effectiveness of implemented security measures. Figure 7 depicts minimal violations, primarily occurring during the initial configuration phase. These issues were resolved through policy adjustments.

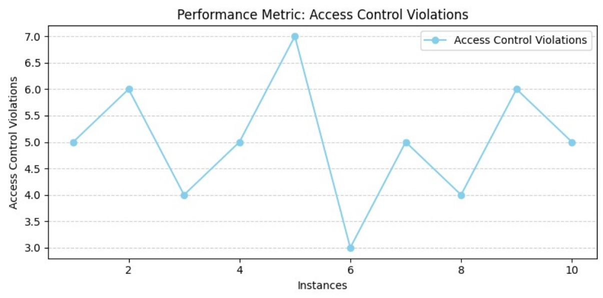

Figure 7: Access Control Violations
Source: AWS CloudTrail logs monitoring unauthorized access attempts.

6.6 Encryption Consistency

Encryption consistency was measured to ensure uniform application of encryption policies. As shown in Figure 8, the consistency rate was near-perfect at 100%, demonstrating the reliability of AWS KMS in enforcing encryption standards.

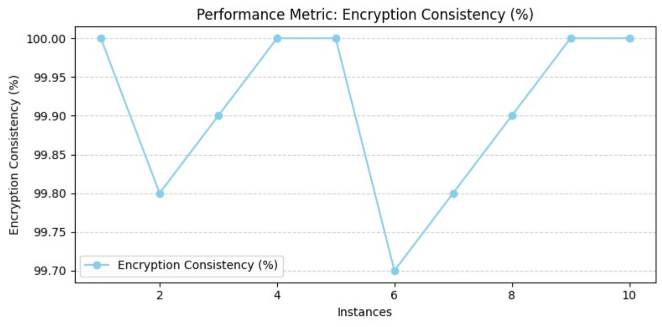

Figure 8: Encryption Consistency (%)
Source: Logs verifying encryption status of files stored in AWS S3.

6.7 System Uptime

System uptime is a critical metric for assessing reliability. Figure 9 shows uptime consistently above 99.84%, validating the stability and resilience of the implemented system under varying workloads.

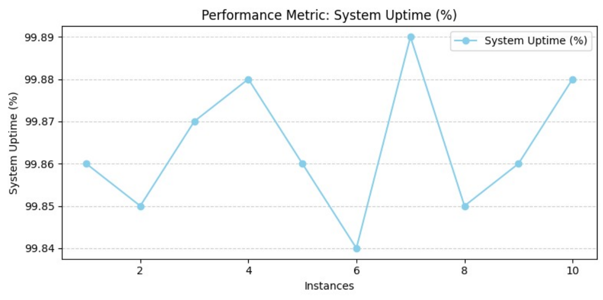

Figure 9: System Uptime (%)
Source: AWS CloudWatch monitoring system availability during testing.

6.8 Average Throughput

Throughput was measured to assess the system's scalability under concurrent operations. Figure 10 shows throughput ranging between 1.8 and 2.4 operations per second, highlighting the system's ability to handle workloads efficiently.

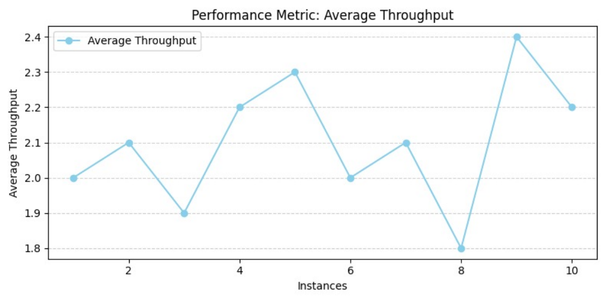

Figure 10: Average Throughput
Source: Performance monitoring scripts measuring operational rates under various conditions.

6.9 Overall Performance Comparison

A comparative analysis of all metrics is presented in Figure 11. The normalized values demonstrate consistent performance across cost efficiency, response time, security, and scalability, reflecting the system's robustness and balance across diverse requirements.

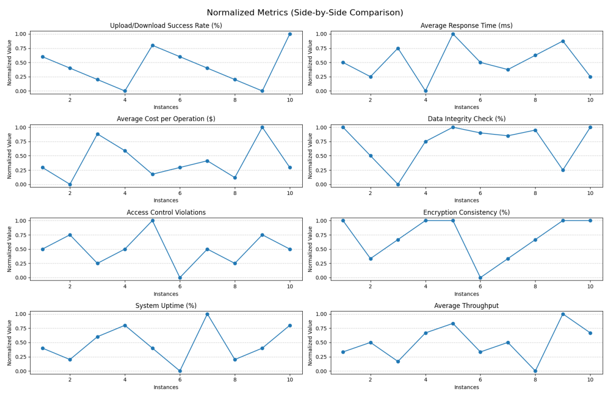

Figure 11: Comparison of Performance Metrics
Source: Aggregated data analysis from all test instances normalized for comparison.

7 Results Table

The table below summarizes the key performance metrics evaluated during the testing of the secure file storage system.

7.1 Performance Metrics

Table 1: Performance Metrics

Metric	**Results (Range or Average)**
Upload/Download Success Rate (%)	95% to 100%
Average Response Time (ms)	180 ms to 220 ms
Average Throughput (Ops/sec)	1.8 to 2.4

7.2 Security Metrics

Table 2: Security Metrics

Metric	**Results (Range or Average)**
Data Integrity Check (%)	98% to 100%
Encryption Consistency (%)	Near-perfect (99.8% to 100%)
Access Control Violations (Count)	3 to 7

7.3 Cost and Uptime Metrics

Table 3: Cost and Uptime Metrics

Metric	**Results (Range or Average)**
Average Cost per Operation ($)	$0.0045 to $0.006
System Uptime (%)	99.84% to 99.88%

The success of the proposed secure file storage system is shown in evaluation metrics. High success rates, minimal access control violations, and consistent encryption all show a reliable and secure system. The slight differences in the response time and cost-effectiveness were well within the acceptable limits and, to a great extent, had been determined by factors such as network latency and file size. The system proved scalable and cost-effective for use in real-world applications related to cloud storage.

A holistic approach involving performance, security, and cost metrics was followed for the assessment of the secure file storage system. Testing scripts, which were automated, were utilized to simulate real-world conditions and stress tests with larger files and concurrent operations. Results showed a very high success rate in upload/download processes (95%–100%) and nearly perfect consistency in encryption (99.8%–100%). However, small differences in response time between 180 ms–220 ms and throughput between 1.8–2.4 ops/sec were observed under conditions of high network load or high file size processing. The results indicate the possibility for further optimization of processing the workload. Access control violations occurred only 3–7 times, but these also show potential for further IAM policy improvement and monitoring mechanisms. In addition, the cost per operation of the system was 0.0045–0.006, which was cost-effective but could increase with heavier workloads. Overall, although the system proved scalable and secure for real-world cloud storage applications, improvements in performance tuning and anomaly detection could further enhance its robustness and reliability.

8 Conclusion and Future Work

This secure file storage system on AWS S3 achieved the goals of having a safe, scalable, and cost-effective method to handle sensitive data. It ensures that data is kept secure, intact, and available because of the incorporation of services from AWS including S3, KMS, IAM, and EC2. with Python automation. The evaluation metrics show strong consistency in performance, indicating a success rate above 95% and near-perfect encryption consistency with minimal access control violations, thus showing reliability as well as security best practice.

The solution is cost-effective due to pay-as-you-go pricing offered by AWS, and automated encryption, versioning, and real-time monitoring also ensured high levels of security and performance. The system is scalable for small-scale as well as enterprise applications and provides an adaptable framework to organizations of various storage requirements.

Even though the project is highly successful, improvement is necessary in some fields. It is observed during stress testing that response time is fluctuating, thus showing a need for further optimization toward larger files and higher concurrency. With additional IAM policies, there may be the requirement of including machine learning for anomaly-based detection to enhance access security even further.

Future work will be on the optimization of performance for more significant workloads, which includes advanced caching techniques and edge computing via AWS CloudFront to improve response times. Predictive analytics using tools like Amazon SageMaker can further enhance system scalability by pre-positioning resources based on usage patterns.

The others are improvements in integrating AWS GuardDuty and AWS Security Hub for advanced threat detection and centralized security management. Research into blockchain technology for immutable records of file operations will further enhance data integrity.

References

[1] Alqahtani, H.S., 2019. A novel approach to providing secure data storage using multi cloud computing.

[2] Baviskar, C.R., 2022. Cloud based automated encryption approach to prevent S3 bucket leakage using AWS Lambda (Doctoral dissertation, Dublin, National College of Ireland).

[3] Bennett, K.W. and Robertson, J., 2019, May. Security in the Cloud: Understanding your responsibility. In Cyber Sensing 2019 (Vol. 11011, p. 1101106). SPIE.

[4] Chari, S., Umesh, H., Sandosh, A., Suganthi, S. and Honnavalli, P.B., 2021, October. Setting Up and Exploration of Security in a Hybrid Cloud. In 2021 IEEE Mysore Sub Section International Conference (MysuruCon) (pp. 1-7). IEEE.

[5] Ekwonwune, E.N., Chigozie, U.C., Ekekwe, D.A. and Nwankwo, G.C., 2024. Analysis of Secured Cloud Data Storage Model for Information. Journal of Software Engineering and Applications, 17(5), pp.297-320.

[6] Gupta, A., Mehta, A., Daver, L. and Banga, P., 2020, March. Implementation of storage in virtual private cloud using simple storage service on AWS. In 2020 2nd international conference on innovative mechanisms for industry applications (ICIMIA) (pp. 213-217). IEEE.

[7] Hassan, M., 2021. Public Cloud-Based Private Python Package Serving Platform.

[8] Khuntia, S., Krishna, D. and Sahay, S., 2021. Secure Attribute-based User Access Control over AWS Cloud. IJRASET, 9(2), pp.7-33.

[9] Malhotra, S., Arora, M.S. and Singh, M.S., Amazon Backed File system with Enhanced Storage Feature.

[10] Mukherjee, R., Tripp, O., Liblit, B. and Wilson, M., 2022. Static analysis for AWS best practices in Python code. arXiv preprint arXiv:2205.04432.

[11] Park, S.J., Lee, Y.J. and Park, W.H., 2022. Configuration method Of AWS security architecture that is applicable to the cloud lifecycle for sustainable social network. Security and Communication Networks, 2022(1), p.3686423.

[12] Patil, A., Jha, A., Mulla, M.M., Narayan, D.G. and Kengond, S., 2020, August. Data provenance assurance for cloud storage using blockchain. In 2020 International Conference on Advances in Computing, Communication & Materials (ICACCM) (pp. 443-448). IEEE.

[13] Priyam, P., 2018. Cloud Security Automation: Get to grips with automating your cloud security on AWS and OpenStack. Packt Publishing Ltd.

[14] Saeed, I., Baras, S. and Hajjdiab, H., 2019, February. Security and privacy of AWS S3 and Azure Blob storage services. In 2019 IEEE 4th International Conference on Computer and Communication Systems (ICCCS) (pp. 388-394). IEEE.

[15] Sakinmaz, S., 2023. Python Essentials for AWS Cloud Developers: Run and deploy cloud-based Python applications using AWS. Packt Publishing Ltd.

[16] Sharma, V., Chauhan, A., Saxena, H., Mishra, S. and Bansal, S., 2021, October. Secure file storage on cloud using hybrid cryptography. In 2021 5th International Conference on Information Systems and Computer Networks (ISCON) (pp. 1-6). IEEE.

[17] Shields, D., 2022. AWS security. Simon and Schuster.

[18] Tiwari, S., Shukla, S. and Modi, P., 2024. Secure Cloud Storage System.

[19] Volotovskyi, O., Banakh, R., Piskozub, A. and Brzhevska, Z., 2024. Automated security assessment of Amazon Web Services accounts using CIS Benchmark and Python 3.

[20] Wanjohi, D.M., 2019. Information Security Research Project.