

Cloud-Based Secure File Sharing with Access Control

Heer Sagaria, Divesh Kumawat, Mahipriya Boppudi, and Bhavya B

Department of Computer Science & Engineering, Parul Institute of Engineering and Technology, Parul University,
Vadodara, India

Guide: Prof. Shah Harshal Anilkumar, CSE, PIET, Parul University

Abstract—The systems used by the people to the purpose of file transfers like email attachments, cloud storage without any security features, FTP, are all built for convenience rather than security. It leaves many avenues open for attack as more information that is very highly sensitive data is being accessible through cloud storage. This paper presents a cloud based file transfer solution named SecureVault with additional security features. The characteristics of the system include AES-256 encryption, Role-based Access Control and Zero-Trust security model over simple functions of uploading and downloading files. The focus is not only on the encryption, but also on the fact of SecureVault preventing screenshots, unauthorised file downloads, share links expire automatically and all the actions taken by a user are recorded into the audit trail for monitoring dashboard base. All the data in the system is protected by a secure VPN that guarantees the safety of data in transit. The design of the platform is based on a Node.js/Express backend, metadata database MongoDB, Electron desktop app, and AWS services (S3, Cognito, KMS, CloudWatch). SecureVault passed eight tests of its ability to log in, apply roles, encrypt data, delete share links automatically after a while, detect duplicates, log actions, and succeeded in all of them. Comparing with plain file sharing using only passwords and HTTPS in regard to fine-grained access, transparency, and leak prevention capabilities, we can see an evident improvement in those. Finally, we analyze the relevance of our solution in the context of existing research related to cloud security, particularly RBAC/ABAC, hybrid/attribute-based encryption, auditing by using blockchain technology, Zero Trust security guidelines of NIST, and suggest directions for further study.

Index Terms—Security in cloud computing, secure file transfer, AES encryption, access control based on role, Zero Trust, VPN tunneling, auditing/logging, monitoring, temporary access link, AWS.

I. INTRODUCTION

File sharing has turned out to be the worst problem as a whole in terms of security in the digital operations in modern times. The files are shared from one platform to the other within employees, departments or even customers, but with an application which does not offer any security feature. The emails stored in the inbox stay there until people forget about them. The links created using the consumer cloud storage stay there until people remove them. FTP transfers happen without any encryption. These services were never built for the threat model.

It will eventually take place anyway. The papers will find their way into the wrong hands, somebody will make a copy without knowing about it, and there will not be any evidence of the whole procedure. All these facts are pretty much common knowledge for the hackers. Cracking the encrypted files may not be easy, but uncovering that the access permissions are too broad or a certain link is dead may not be easy as well. Moreover, the expectations from information management keep rising, the right file sharing should have all the three elements—encryption, access management and the paper trail.

The focus will be shifted to SecureVault, which has been created by us as part of our final year project at Parul University. As clearly mentioned, the intention behind creating this program was never to come up with a new competitor in the existing cloud storage programs. Instead, the goal behind the project was to show the feasibility of adding basic security features in a user friendly manner. These security features are: Content encryption; A robust system for identity authentication (none is ever considered secure until it proves to be so); Role/File permissions; An anti-copying mechanism for the secured content; Expiration of the share link; Monitoring of all the visible activities.

A. What This Paper Contributes

In summary, instead of the work that was supposed to be done, the following work was actually done:

- Combining various security measures like AES-256 encryption for securing files, JWT for authentication, RBAC for roles, and Zero Trust Request validation in a single step.
- The leak protection methods such as blocking of screenshots, download, and sharing of links on time are provided by the SaaS applications which a password-based HTTPS system cannot provide.
- Part of the auditing process that involves logs for each action taken and real-time monitoring technology to ensure that any suspicious activity is detected immediately, rather than after the fact.
- The existing AWS implementation (S3, Cognito, IAM, KMS, CloudWatch), deployed in the Node.js/Express backend and Electron frontend environment through unit, integration, system, security and usability testing.

II. LITERATURE REVIEW

However, before starting the work on the development of the system, we had to study twenty scientific sources on the problem of security in cloud file sharing—scientific research papers, normative documents and even blog posts by Amazon engineers. Four main aspects kept repeating: access control, encryption, accountability and reliability of the system. All the mentioned sources, along with their weaknesses, are listed in Table I. Some of these weaknesses became the foundation for our decisions.

A. Access Control: RBAC, ABAC and Hybrid Models

In all papers on cloud file sharing, Role Based Access Control appears since it simplifies permission assignment due to the

use of roles (such as uploader or administrator) instead of users. Inflexibility is another problem that is mentioned in many papers, and it is not surprising since when some permission is assigned to a certain static role, it becomes impossible to implement a permission of a contractor with limited access to a project for two weeks and having need to access just one file. Attribute Based Access Control was designed to solve this issue since it checks the attributes (such as identity, resource sensitivity, or device posture) prior to accessing the file; however, in this case, policies had to be written and maintained, which is a complex process. Several papers suggest combining these approaches (RBAC for the permissions and ABAC for exceptions), which is what we did in SecureVault—the default permission is based on the role, but it can be overwritten by per-file permission.

B. Encryption Strategies

On the contrary, AES-256 appeared like a default cipher, apart from the situations where RSA is applied jointly with AES-256, solving the problem of the key exchange. It is evident why such combination is applied, as such combination of methods allows finding the balance between complexity of computations and security. In its turn, Attribute-Based Encryption approach goes further and combines access control policy into the encryption key in such a way that the decryption can be performed only by those users who possess necessary attributes under such policy. As we see, access control mechanisms become unnecessary, while complexity of computations depends on the number of attributes. We have taken the usual way: our solution includes AES and keys managed by AWS KMS. Such solution is recommended in many resources concerning AWS and is known as the optimal compromise between encryption and access control mechanism.

C. Zero Trust and Continuous Verification

The formulation of the problem statement of the Zero Trust model adopted by NIST will be “Can I validate this specific request?” instead of “Is this request coming from the internal network?” due to the adoption of least privilege, microsegmentation as well as continuous validation. Some of the articles written by AWS blogs which we have studied rely on these concepts as well. Such an architectural model can be very useful in mitigating insider threats and credential replay attacks, but the cost associated with such an approach is certainly not low, since verification of every single request and not only the login makes such an architecture quite complicated and hard to monitor. That is the way we have designed SecureVault—token and role validation for every single file access.

D. Accountability: Audit Logging and Monitoring

It seems that there is a certain common element amongst some of the literature in the sense that both encryption and access control are not enough alone as there should be an extensive record about who did what and when. Logging is always involved in any kind of security framework and is implemented through services such as AWS CloudTrail but since the log is immutable, it can only be considered a retrospective record for which reason some pieces of literature advocate a combination of centralized logging with

real-time monitoring. It is argued in some research that the implementation of blockchain technology as an alternative to centralized logging is mandatory as it will add overhead, but will ensure certainty about the impossibility of alteration of the log by a single administrator. In this version of SecureVault, we have chosen to implement centralized, time-stamped logging with real-time monitoring—blockchain logging can be considered a future addition.

E. Where SecureVault Fits

None of these sources describe any specific method, like RBAC, ABAC, ABE, Zero Trust or blockchain logging, that could be utilized on its own. The majority of the time, a system applies more than one method due to the prevention of the single point of failure. This was our approach—we used RBAC with access overrides at file level as an access control method, AES-256 with managed keys from KMS as an encryption scheme, Zero-Trust approach to validate all requests and audit and monitoring of them all through a secure channel created via Virtual Private Network (VPN). The rest of the paper will provide detailed information on our approach and its effectiveness during the test period.

III. PROBLEM STATEMENT

There is an ongoing exchange of information between various individuals: among team members, among different companies, the company and their clients, and mostly for the most part the services were developed more for convenience than for security purposes. E-mails, cloud services and FTP only takes on single password along with encryption offered by the browser or the client program. Besides that, the system has several weaknesses, there are no limitations to who will be accessing the file, links will remain active until they are removed, there will be no trace of the person who sent the file after sending it, and there isn't anything stopping the authorized person from taking a screenshot of a read-only file.

The vulnerabilities of this sort are even more pertinent today than ever before since the hackers have ceased trying to decrypt the encrypted files and have started on the soft targets, such as outdated links, extended role and download without being watched over. Once the files have left their specific channels, there isn't much that could be done about them, and the lack of logging and permissions doesn't help in stopping the attack at any point. The solution to this problem in our opinion should be the development of a file transmission system that combines all four of these factors at once without making the entire process unnecessarily complicated for the user. And this is what SecureVault does.

IV. PROPOSED SYSTEM: SECUREVAULT

In the SecureVault, there were seven modules designed to function within the normal cloud storage. The reason behind the segmentation of the system into these seven modules is to make sure that the security is achieved in the whole process of log in, the processing of the file, even viewing of the file, since if the first step fails, it means the entire system is jeopardized.

TABLE I
 SUMMARY OF REVIEWED LITERATURE ON CLOUD-BASED SECURE FILE SHARING

Study / Source	Focus	Core Technique	Key Limitation Noted
Cloud-Based Secure File Sharing Using Access Control, IJSRCEIT (2022)	Access control for cloud storage	AES-256 with Role-Based Access Control (RBAC)	RBAC is rigid when permissions change frequently
Secure Remote Cloud File Sharing With ABAC, IEEE (2021)	Context-aware access	Attribute-Based Access Control (ABAC)	Policy design and maintenance overhead
Secure File Sharing Platform Using AWS S3, IJFMR (2025)	Hybrid encryption + OTP	AES-256, RSA key exchange, OTP + RBAC	RBAC lacks flexibility in dynamic settings
Attribute-Based Encryption for Secure Cloud Storage, IEEE TCC (2020)	Encryption-enforced policy	Attribute-Based Encryption (ABE)	Key-management complexity, performance overhead
Role-Based Access Control in Cloud Environments, ACM Surveys (2019)	Survey of RBAC in the cloud	RBAC role hierarchies	Struggles with temporary or context-specific access
Hybrid Encryption Models for Cloud File Sharing, Springer (2021)	Encryption performance	AES for bulk data + RSA for key exchange	Requires periodic key rotation for long-term resilience
Zero Trust Architecture, NIST SP 800-207 (2020)	Trust model	Continuous verification, least privilege, micro-segmentation	Raises implementation and monitoring complexity
Blockchain-Based Access Control for Cloud Storage, FGCS (2022)	Tamper-evident logging	Blockchain + ABAC	Scalability and infrastructure cost
Secure File Sharing in Multi-Tenant Cloud Systems, Elsevier (2021)	Tenant isolation	RBAC + ABAC with isolation controls	Misconfiguration risk across tenants
Audit Logging and Accountability in Cloud File Sharing, IEEE S&P (2020)	Accountability	Centralized logging (CloudTrail, IAM)	Static logs alone cannot stop an attack in progress
AWS security and architecture blogs (2021–2025)	Practical AWS controls	S3 Access Points, private API Gateway, CloudFront signed URLs, KMS, Transfer Family, CloudTrail/CloudWatch	Guidance is service-specific and needs integration effort

A. Design Principles

- **Never underestimate the value of trust.** Every single request is verified and authenticated regardless of anything that happened in previous activities within the session—this is what Zero-Trust is all about.
- **Data has to be encrypted before being stored.** Files are encrypted using AES-256 encryption and then stored in the storage layer in order to ensure that there is no plain text available in the server.
- **Provide only what is required.** A role that is defined within an RBAC environment grants the maximum privilege to a particular user, whereas a privilege based on a file can only bring down the ceiling; it cannot raise the ceiling.
- **Log what matters.** Uploads, downloads, permissions, and links are all logged with enough context—device, certificate, IP—for these events to be understood.
- **Keep it short.** Give out links or high permissions with default expiration dates.

B. System Architecture

The architecture consisted of ten levels, starting with the level which was visible to the user and going down to the storage and monitoring levels:

- **User Layer** – Individuals interact with SecureVault via desktop application.
- **Frontend Layer** – an Electron app handling login, upload/download and sharing screens.
- **API Gateway Layer** – Single entry point through which all the authenticated requests can be routed.
- **Back-end Processing Layer** – The backend processing layer

is carried out by Node.js and AWS Lambda.

- **Authentication Layer** – verification of identity using JWT together with AWS Cognito.
- **Authorization Layer** – The RBAC and IAM policies are applied before allowing any file operation to proceed further.
- **Storage Layer** – The AWS S3 contains encrypted file objects with the durability it provides.
- **Encryption Layer** – AES encrypts the contents of the file while AWS KMS handles the keys.
- **Database Layer** – The MongoDB stores the details about users, files, and permission.
- **Monitoring Layer** – AWS CloudWatch aggregates events for the dashboard and alerting.

That is how the request traverses the layers in such an order: the frontend sends the request through the API Gateway, the backend identifies the requester, authorization makes sure the requestor has access depending on his capabilities and depending on any rule related to the file, file gets fetched/written to S3 encrypted, metadata is updated in MongoDB, and everything is logged so that the monitoring layer can pick it up. Since authentication and authorization take place before any storage operation occurs, a failed one will prevent the request from getting anywhere near S3.

V. SYSTEM DESIGN

A. Entity–Relationship Model

The data model has been designed considering four different entities. The User (UserID, Role) entity has the capability of uploading files (FileID, Name, OwnerID) and has the access policy (PolicyID, Permissions), which leads to the creation of

TABLE II
 SECUREVAULT FUNCTIONAL MODULES

Module	Function
Authentication Module	Verifies the credentials and then provides a JSON Web Token (JWT) for the session. We do not consider the login to be valid for the entire session—we verify the identity every time a request is made, following the Zero-Trust policy.
File Handling Module	The file is encrypted using AES before being sent out from the system and stored in encrypted form in AWS S3, and is decrypted when the request is authorized. There is no plaintext in the storage layer.
Access Control Module	Uses RBAC with three roles — Admin, Editor, Viewer — and enables the rule per file to further restrict the permissions that would be allowed by the role. The rule cannot increase the role's permissions; it can only reduce them.
Audit Logging Module	Writes entries for all uploads, downloads, links, permissions changes and failed accesses, identified with a timestamp, device identifier, certificate ID and IP address.
Monitoring Module	Feeds the active sessions, recent notifications and average response times into a live dashboard so that an administrator can see what happened while it was happening.
Temporary Link Module	Shares create links which expire after a certain period and within a particular context. Once the period elapses or the task is accomplished, the link ceases to function, eliminating the need for manual deletion.
Secure Transmission Module	All communication between client and server is channeled through a VPN tunnel to ensure security of data and metadata even over a network that we cannot wholly trust.

activity logs (LogID, Timestamp) due to any interaction between the user or the access policy and the file. It is necessary to maintain these four different entities separate and not merge them in order to support the single file audit query.

B. Use Case View

The two actors in our system are User and Admin. Both can log in, upload, download and share a file; the Admin alone can manage access—create, modify or delete another user's access. “Sharing a file” and “managing access” have been purposely separated in the model, as one should be able to invite for a limited period of time to an object he/she already possesses but not be allowed to manipulate the role based access system. This is the privilege of an Admin alone.

C. Sequence of a File Request

In both upload and download, the six participants will remain the same and are made up of the following: User, Frontend, API Gateway, Backend, Database, Cloud Storage. In this case, the request is made by the user on the frontend, whereby the request could either be for logging in or taking any kind of action, and is sent to the API Gateway, which then triggers authentication while the backend takes care of the access check on the database, and only when the authentication process is successful will the backend trigger the process of requesting upload or download to the cloud storage. This is because the access check always comes before anything goes to the cloud storage.

D. Data Flow

At the highest level, User sends an input to the Application process, which interacts with the Authentication Service and makes contact with the Database and Cloud Storage whenever required, which in turn ends up in Monitoring Logs. The further decomposition of the Application process leads to four subprocesses—Access Control and File Process, both of which are followed by logging activities and the latter is only made available after the approval by the former.

TABLE III
 TECHNOLOGY STACK

Layer	Technology
Frontend	Electron (cross-platform desktop client)
Backend	Node.js with the Express framework
Database	MongoDB (user records, file metadata, permissions)
Object storage	AWS S3
Compute / hosting	AWS EC2, AWS Lambda for serverless logic
Identity & access	AWS Cognito for authentication, AWS IAM for authorization policy
Key management	AWS KMS, AES for file-level encryption
Session security	JSON Web Tokens (JWT), bcrypt for password hashing
Monitoring	AWS CloudWatch
Process management / CI	PM2, Jenkins pipelines

VI. IMPLEMENTATION

A. Technology Stack

B. Development Process

Waterfall was selected for the process since, in this case, security requirements were identified at an early stage, and the following stages were highly dependent on the preceding ones, meaning that there was little sense in changing requirements once they were set. The requirements analysis stage defined what security measures should be designed and implemented (encryption and access control). Design produced an ER diagram and use-case and sequence models (described in Section V). Implementation involved developing the back-end, integrating it with S3 and the client made with Electron.js; testing was continuous and not only happened at the end of the cycle (see Section VII); deployment was done on AWS EC2 through the VPN tunnel and was built with Jenkins; maintenance is currently happening and involves monitoring if something goes wrong.

Fig. 6 shows the same request path as a flowchart, from the moment a user opens the desktop client through encryption and storage, and back again on the retrieval side once a role check clears.



Fig. 1. SecureVault's ten-layer architecture, from the user-facing client down to storage and observability.

C. What the Screens Actually Look Like

Within admin part after authenticated the user, the administrator will be dropped to dashboard which will show the total of files, files shared at particular day, encryption and pending tasks. Within the user management part, all the users having the associated role will be shown and admin will have an option to change the role without moving to other screens. Within the auditor screen, there will be a log of search—by certificate, filename or IP address—where it will show the date, action performed, file name, certificate ID, device name, IP address and result of that particular action. Along with audit screen, there will be a monitor part which will show total of sessions, alerts in that day, average time and threat level. Some of the alerts like brute force attack, certificate being used after expiration will be displayed below it. The same case with the user part: login screen, file dashboard showing total/encrypted file count, a secure viewer blocking screenshots, downloads and copy-paste option in restricted documents. user's own activity.

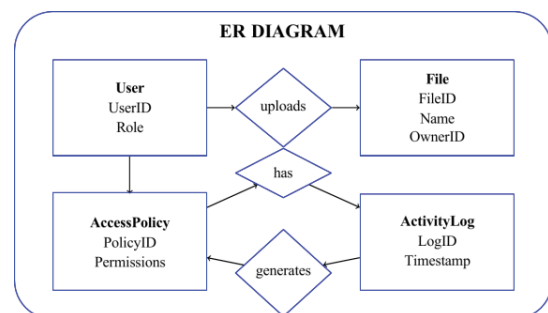


Fig. 2. Entity-Relationship diagram showing User, File, AccessPolicy and ActivityLog entities.

VII. TESTING AND VALIDATION

A. Levels of Testing

Testing has been done through four phases and the first one is a unit testing where different modules were tested individually such as encryption module and login module prior their inte-

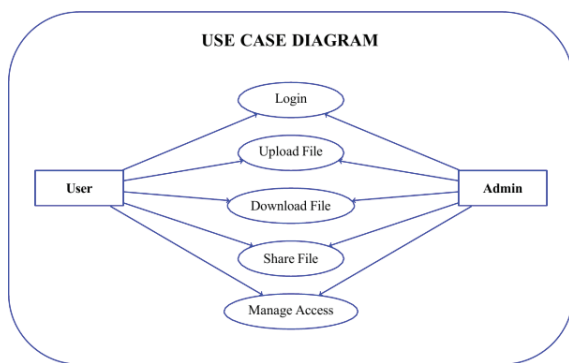


Fig. 3. Use case diagram for the User and Admin actors.

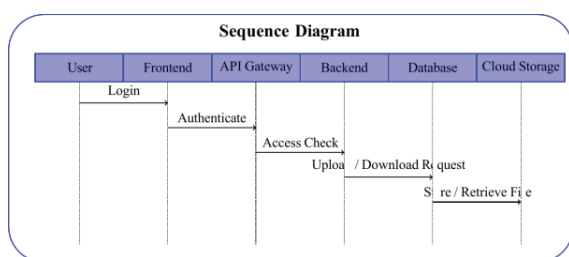


Fig. 4. Sequence diagram for an authenticated upload/download request.

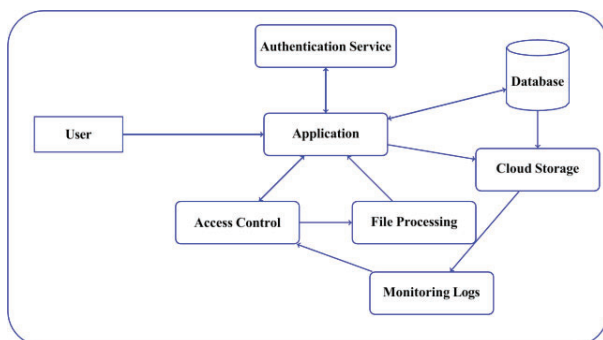


Fig. 5. Level-1 data flow diagram for the Application process.

gration together. The next step is an integration testing where the integration of different modules, for instance, authentication, role-checking and activity logging modules was tested due to the fact that there were many bugs which appeared when the modules began to work with each other by using real data instead of test data. Then came the system testing that was conducted after the integration testing where the workflow process including uploading a file to file sharing and viewing of the log file was examined. In addition to the system testing there was also a security testing which assured that the system was safe from any kind of access and encrypted data was processed properly by implementing the Zero-Trust login model. Finally, we have some non-project team members performing a usability testing on the system.

B. Validation Criteria

In addition to the test cases of each individual test, the following six other requirements were successfully tested: functioning of all the core modules—uploading, downloading, encryption, and sharing, not causing any failures afterwards; information security in both resting and transition state, without any unauthorized access; role-based restrictions preventing any higher clearance access; review of audit trail for all the operations; responsiveness of the system while multiple users work at once; and user interface with clear instructions and error messages enabling users to complete their tasks without any additional assistance. All the test cases described in Table IV have been successfully conducted, and no modifications other than those made in the user interface mentioned above were required.

VIII. RESULTS AND DISCUSSION

Table V compares SecureVault with the file-sharing techniques described in Section III from the viewpoint of the features that have been commonly regarded as the weaknesses in the related literature.

The observations that can be made based on the findings of the tests and their results in Section VII are the following. First of all, the integration of RBAC into the request verification procedure in the Zero Trust Architecture solved the problem of lack of access control without the need to formulate policies in case of full implementation of ABAC, which was quite an effective solution for the present project, even though it would have been required to implement ABAC in multi-tenant architecture. Second, the use of the time-limited and context-sensitive connections along with the restrictions on screenshots and downloads addresses the “once it’s shared, it’s uncontrollable” problem usually stated in the literature, since now it is limited not only by the time period but also by the type of actions that may be performed, and not at the moment of giving permission. Third, the approach of treating audit logs as monitoring data and not just background information helped to prove the fact via testing that the anomalies like multiple failed login attempts are immediately reported to the administrator, as it is stated in the IEEE Security and Privacy paper.

A. What Worked Well

- Deciphering the data that has been collected will not be possible without the key, and that means that although the data is available, it will not be decipherable.
- Permission settings are flexible enough to give file owners the power to specify the permissions applicable to certain users when accessing the file.
- The audit trail assists in retrospection, which is good for trust creation and even has some purpose in compliance.
- Shorter is the time frame in which the joint document stays available, thus wrong forwarding of the link won’t stay bad forever.
- Real-time monitoring refers to a case where the problem is identified during the process itself and not at any other time.
- VPNs provide additional security when transferring data over networks which are beyond our control.

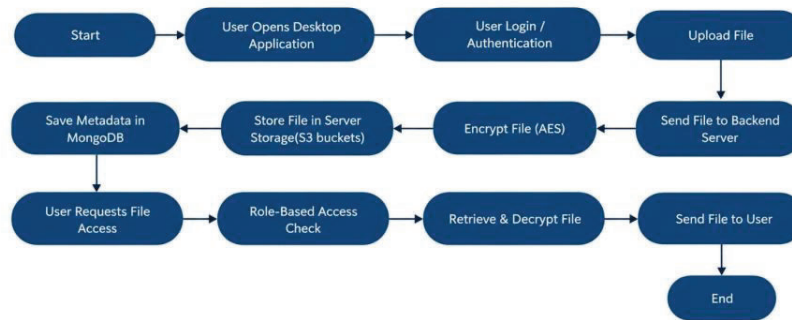


Fig. 6. End-to-end upload and retrieval workflow implemented in SecureVault.

TABLE IV
 REPRESENTATIVE TEST CASES AND RESULTS

#	Test Case	Input	Expected Output	Result
1	Valid user login	Valid credentials, role=Admin, correct e-mail	JWT generated, role=Admin, HTTP 200	Pass
2	User login with incorrect role claim	Correct credentials, incorrect role claim	HTTP 401 Unauthorized	Pass
3	RBAC implementation check	Viewer user JWT making call to /api/admin	HTTP 403 Forbidden	Pass
4	File upload and file encryption	Valid Admin JWT, file selected	File is encrypted and stored along with the metadata, HTTP 201	Pass
5	Temporary link creation	Valid Editor JWT, expiration=5 min	Link valid up till the expiration time	Pass
6	Audit trail	User download file	Action logged with timestamp and IP	Pass
7	Duplicate file upload	Upload of the same file twice	HTTP 409 Conflict	Pass
8	Monitoring dashboard	Multiple simultaneous users	Live session monitoring	Pass

B. Where It Falls Short

- Preparation of all things is something that requires some technical skills that may not be available in a smaller group of people.
- Encryption and surveillance become increasingly necessary, which do take up some computational resources that are quite noticeable as the file sizes increase.
- Although such limitations, although restrictive by nature and intended to offer protection, may be quite frustrating at times to genuine users.
- The use of the AWS environment and also the virtual private network (VPN) comes with expenses.
- The network needs to be upgraded regularly in order to address all these emerging challenges, which means that this is an ongoing process and not a one-off deal.

IX. CONCLUSION AND FUTURE WORK

Some of the things that we have learned from the course of the project include the following: Although all solutions offered in Section II (RBAC, AES Encryption, Zero-Trust Validation, centralized audit logs) are extremely common on their own, it took engineering effort to combine them together into one solution that works. The system that we have implemented with the help of Node.js/Express on the backend side, Electron on the frontend side, and AWS services for storage, identity man-

agement, and monitoring, has performed exactly as expected in all eight test cases that we have generated for testing purposes, including authentication, authorization, encryption, generation of temporary links, duplicate detection, and dashboard availability. We can conclude from the comparison that we have done in Section VIII that there is a great potential for improvement over password/HTTPS-based solutions in terms of security, access granularity, transparency, and prevention of data leakage—although we do not think that our solution is perfect, after all.

The following are some other considerations that should be in connection with the constraints and hints in the literature. For instance, the use of Attribute Based Access Control instead of the existing one will enable us to get permissions on the basis of context (e.g., specific device, specific time slot, specific set of project tags). This is so because we will not need to assign the role again when there is any change in them. The use of a tamper-proof audit log on the basis of lightweight blockchain technology that has been discussed in our FGCS paper will provide us with a tamper-proof audit log. The machine learning-based approach to anomaly detection will allow us to identify any anomaly that is not consistent with the system's usual behavior, such as the excessive number of downloads or unauthorized access attempts outside of work hours, rather than using a threshold. As far as authentication is concerned, we need to rely on multi-factor authentication and regularly review our AES key rotation to prevent any compromising of our credentials. None of these

TABLE V
 SECUREVAULT COMPARED WITH CONVENTIONAL FILE-SHARING TOOLS

Dimension	Conventional File-Sharing Tools	SecureVault
Security model	Password authentication and generic HTTPS	AES-256 encryption, Zero Trust authentication, and VPN tunneling
Access control	User role separation with coarse control (user/admin)	Fine control via RBAC — Admin, Editor, Viewer — and file-based access
Transparency	Login information only, if at all	Audit logs plus live dashboard
Shared links	Permanent	Temporary, certificate-based
Leak prevention	None available	Screenshot prevention and download blocking

solutions are necessary now, but they represent an interesting development for the future.

ACKNOWLEDGMENT

We would like to thank Prof. Shah Harshal Anilkumar for his guidance and patience in every phase of design and development of SecureVault and Dr. Shailendra Mishra, Head of Department of Computer Science & Engineering, PIET, Parul University, for backing this project at the departmental level.

REFERENCES

- [1] Cloud Based Secure File Sharing Using Access Control, IJSRCEIT, 2022. <https://ijsrceit.com/paper/CSEIT228458.pdf>
- [2] Secure Remote Cloud File Sharing With Attribute-Based Access Control, IEEE Xplore, 2021. <https://ieeexplore.ieee.org/document/9512426>
- [3] Secure File Sharing Platform Using Public Cloud (AWS S3), IJFMR, 2025. <https://www.ijfmr.com/papers/2025/6/61191.pdf>
- [4] Attribute-Based Encryption for Secure Cloud Storage, IEEE Transactions on Cloud Computing, 2020. <https://ieeexplore.ieee.org/document/9090142>
- [5] Role-Based Access Control in Cloud Environments, ACM Computing Surveys, 2019. <https://dl.acm.org/doi/10.1145/3338842>
- [6] Hybrid Encryption Models for Cloud File Sharing, Springer Journal of Cloud Computing, 2021. <https://link.springer.com/article/10.1186/s13677-021-00259-0>
- [7] NIST, Zero Trust Architecture, SP 800-207, 2020. <https://csrc.nist.gov/publications/detail/sp/800-207/final>
- [8] Blockchain-Based Access Control for Cloud Storage, ScienceDirect – Future Generation Computer Systems, 2022. <https://www.sciencedirect.com/science/article/pii/S0167739X21004123>
- [9] Secure File Sharing in Multi-Tenant Cloud Systems, Elsevier Computers & Security, 2021. <https://www.sciencedirect.com/science/article/pii/S0167404821001234>
- [10] Audit Logging and Accountability in Cloud File Sharing, IEEE Security & Privacy, 2020. <https://ieeexplore.ieee.org/document/9090145>
- [11] Secure File Sharing Solutions in AWS: A Security and Cost Analysis (Part 2), AWS Blog, 2025. <https://aws.amazon.com/blogs/security/secure-file-sharing-solutions-in-aws-a-security-and-cost-analysis-guide-part-2/>
- [12] Secure File Sharing on AWS Without Public Access, IAMOps Blog, 2024. <https://iamops.io/aws-secure-file-sharing-guide/>
- [13] Best Practices for Managing Access to Amazon S3, AWS Blog, 2023. <https://aws.amazon.com/blogs/security/best-practices-for-managing-access-to-amazon-s3/>
- [14] How to Use S3 Access Points for Secure Data Sharing, AWS Blog, 2022. <https://aws.amazon.com/blogs/storage/how-to-use-amazon-s3-access-points/>
- [15] Implementing Attribute-Based Access Control in AWS IAM, AWS Blog, 2021. <https://aws.amazon.com/blogs/security/attribute-based-access-control-abac-for-aws/>
- [16] Using AWS KMS for Secure File Encryption and Sharing, AWS Blog, 2022. <https://aws.amazon.com/blogs/security/how-to-use-aws-kms-for-encryption/>
- [17] Designing Secure File Transfer Workflows with AWS Transfer Family, AWS Blog, 2023. <https://aws.amazon.com/blogs/architecture/designing-secure-file-transfer-workflows-with-aws-transfer-family/>
- [18] CloudFront Signed URLs and Cookies for Secure File Distribution, AWS Blog, 2022. <https://aws.amazon.com/blogs/networking-and-content-delivery/serving-private-content-with-amazon-cloudfront/>
- [19] Private API Gateway Integrations for Secure File Sharing, AWS Blog, 2023. <https://aws.amazon.com/blogs/compute/using-private-api-gateway/>
- [20] Monitoring and Auditing File Access with AWS CloudTrail and CloudWatch, AWS Blog, 2023. <https://aws.amazon.com/blogs/security/cloudtrail-and-cloudwatch-for-auditing/>