

# **LOCKSTASH: ACCESSING DATA VIA SECURE STORAGE SYSTEM**

A major project report submitted in partial fulfilment of the requirement  
for the award of degree of

**Bachelor of Technology**

in

**Computer Science & Engineering / Information Technology**

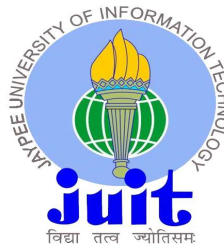
*Submitted by*

**Shivam Karn (201297)**

**Aradhya Taneja (201270)**

*Under the guidance & supervision of*

**Mr. Praveen Modi**



**Department of Computer Science & Engineering and  
Information Technology**

**Jaypee University of Information Technology,**

**Waknaghat,**

**Solan - 173234 (India)**

# CERTIFICATE

This is to certify that the work which is being presented in the project report titled “**LOCKSTASH: ACCESSING DATA VIA SECURE STORAGE SYSTEM**” in partial fulfillment of the requirements for the award of the degree of B.Tech in Computer Science And Engineering and submitted to the Department of Computer Science and Engineering, Jaypee University of Information Technology, Waknaghat is an authentic record of work carried out by **Shivam Karn (201297)** and **Aradhya Taneja (201270)** during the period from August 2023 to May 2024 under the supervision of **Mr. Praveen Modi, Department of Computer Science and Engineering, Jaypee University of Information Technology, Waknaghat.**

**Shivam Karn (201297)**

**Aradhya Taneja (201270)**

The above statement made is correct to the best of my knowledge.

**Mr. Praveen Modi**

**Assistant Professor (Grade II)**

**Computer Science & Engineering and Information Technology**

**Jaypee University of Information Technology, Waknaghat**

# CANDIDATE'S DECLARATION

We hereby declare that the work presented in this report entitled '**LOCKSTASH: ACCESSING DATA VIA SECURE STORAGE SYSTEM**' in partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology in Computer Science & Engineering / Information Technology** submitted in the Department of Computer Science & Engineering and Information Technology, Jaypee University of Information Technology, Waknaghat is an authentic record of my own work carried out over a period from August 2023 to May 2024 under the supervision of **Mr. Praveen Modi** (Assistant Professor (Grade - II) , Department of Computer Science & Engineering and Information Technology).

The matter embodied in the report has not been submitted for the award of any other degree or diploma.

(Student Signature with Date)

Student Name:

Roll No.:

(Student Signature with Date)

Student Name:

Roll No.:

This is to certify that the above statement made by the candidate is true to the best of my knowledge.

(Supervisor Signature with Date)

Supervisor Name:

Designation:

Department:

Dated:

# ACKNOWLEDGEMENT

Firstly, I express my heartiest thanks and gratefulness to almighty God for His divine blessing to make it possible to complete the project work successfully.

I am really grateful and wish my profound indebtedness to Supervisor **Mr. Praveen Modi** (Assistant Professor (Grade - II), Department of Computer Science & Engineering and Information Technology) for deep knowledge & keen interest of my supervisor in the field of “**Information Security**” to carry out this project. His endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, reading many inferior drafts and correcting them at all stages have made it possible to complete this project.

I would like to express my heartiest gratitude to **Mr. Praveen Modi**, Department of CSE, for his kind help to finish my project.

I would also generously welcome each one of those individuals who have helped me straightforwardly or in a roundabout way in making this project a win. In this unique situation, I want to thank the various staff individuals, both educating and non-instructing, which have developed their convenient help and facilitated my undertaking.

Finally, I must acknowledge with due respect the constant support and patience of my parents.

Name: Shivam Karn  
Roll No: 201297

Name: Aradhya Taneja  
Roll No: 201270

# TABLE OF CONTENTS

<b>Cover Page</b>	
<b>Certificate</b>	<b>i</b>
<b>Declaration</b>	<b>ii</b>
<b>Acknowledgement</b>	<b>iii</b>
<b>List of Figures</b>	<b>vi</b>
<b>List of Abbreviations, Symbols or Nomenclature</b>	<b>viii</b>
<b>Abstract</b>	<b>ix</b>
<b>Chapter 1: Introduction</b>	<b>1</b>
1.1 Introduction	1
1.2 Problem Statement	4
1.3 Objectives	5
1.4 Significance and Motivation of the Project Work	6
1.5 Organization of Project Report	7
<b>Chapter 2: Literature Survey</b>	<b>8</b>
2.1 Overview of Relevant Literature	8
2.2 Key Gaps in the Literature	22
<b>Chapter 3: System Development</b>	<b>24</b>
3.1 Requirements	24
3.2 Project Design and Architecture	24
3.3 Data Preparation	34
3.4 Implementation	35
3.5 Key Challenges	44
<b>Chapter 4: Testing</b>	<b>45</b>
4.1 Testing Strategy	45
4.2 Test Cases and Outcomes	45

<b>Chapter 5: Results and Evaluation</b>	<b>51</b>
5.1 Results	51
<b>Chapter 6: Conclusions and Future Scope</b>	<b>52</b>
6.1 Conclusion	52
6.2 Future Scope	52
<b>References</b>	<b>53</b>

# LIST OF FIGURES

S. No.	Title	Page No.
1	Encryption Time Comparison with AES and Proposed System	12
2	Decryption time Comparison with AES and Proposed System	13
3	Protocol Diagram for SRP - 3	15
4	Architecture of TrustStore System	17
5	PBKDF2 Working Procedure	21
6	Flowchart for signup using SRP protocol	26
7	Flowchart for login using SRP protocol	28
8	Flowchart for Upload	31
9	Flowchart for Download	32
10	MongoDB Database Schema	35
11	Code Snippet for Signup Process	36
12	Code Snippet for Login Process	37
13	Code Snippets depicting how media password is created	39
14	Code Snippets depicting how upload process works	40
15	Code Snippets depicting how download process works	44
16	User Creation	45
17	Error Generation due to incorrect login information	46
18	Media Password Creation	47
19	Debug Information showing AES key and its encrypted version	48
20	User details in Database	48
21	Uploaded Files	49
22	Encrypted Files	49
23	Stored Files in Database	49

24	Data of each user being stored separately and securely	50
25	File integrity check	50



# LIST OF ABBREVIATIONS

<b>S. No</b>	<b>Short Form</b>	<b>Full Form</b>	<b>Page No.</b>
1	AES	Advanced Encryption Algorithm	1
2	PBKDF2	Password Based key encryption Standard	2
3	SHA	Secure Hash Algorithm	2
4	SRP	Secure Remote Protocol	8
5	S-Box	Substitution Box	9
6	RC6	Rivest Cipher 6	10
7	BRA	Backward Reduction Algorithm	10
8	LSB	Least Significant Bit	10
9	3DES	Triple Data Encryption Standard	11
10	DES	Data Encryption Standard	22
11	RSA	Rivest, Shamir, Adleman	22

# ABSTRACT

Due to the increasing need for safe storage of files in the digital era, there is an equally important need for strong cryptographic systems. The design and implementation of such a platform is proposed in this study. In order to increase the security levels, the proposed system will use the method of hybrid cryptography i.e. combining both symmetric and asymmetric encryption, using the strengths of both the types of encryption methods will help us to make our system more secure.

In simpler words, we decide to use the method of hybrid cryptography algorithms to increase the security of the system. The main goal is to develop a web system where the user can upload their files in a secure and safe medium, in an encrypted format and then later retrieve it by decrypting the original file.

# CHAPTER 1: INTRODUCTION

## 1.1 INTRODUCTION

Information these days is considered to be one of the most important asset that a person can have. Since early times, people have made an effort to safely keep their data by storing it on various kinds of storages. In today's world, cloud storage is said to be one of the most famous way of storing data because it offers the users an easy to use method, moreover it is adaptable, and most importantly it is able to meet the high storage demands of the modern world through its cloud storage capabilities.

The growth of third party storage providers such as cloud significantly increases the consumption of processing and storage capacity, while simultaneously improving user convenience of data access. But the problem arises in building a trust with any independent third party storage supplier to give your important personal data. Hence to make sure that the client remains satisfied with the service proved by the provider it is important that the crucial information that is being kept into the third party storage provider must be crypted well so that no unauthorized person can get access to it and misuse the information.

In conventional signup/login methods, the details of the user are stored on the server, whereas our project suggests a method which ensures that password is not stored on the server instead only email is sent to the server, now if the email already exists we proceed further otherwise we generate a salt for our key derivation function. A key derivation function can be simply thought of as a mechanism that takes a password and turns it into a symmetric key suitable for cryptographic operations (such as AES). After this, the email, password and the salt that we have generated are passed in our key derivation function. The key derivation function that we use here is PBKDF2 (password based key derivation function 2).

In the input function of our PBKDF2 (password based key derivation function 2) we give the password, a salt, and an integer that denotes the key length that we want as output.

The following example further demonstrates the working of PBKDF2 – Password:

IL0veCrypt0!!!

Salt: D04A77B765E5CEDA3A84AA2704C4C1908A72

Iterations: 10,000

Hash Function: SHA256

Desired Key Length: 32 bytes (256 bits) Output:

D122D2A917B3EC896214FER87CAEC0FBF914D00912423EBFCEBED72ABF82C9  
45AE1

In PBKDF2, it is noted that if we increase the number of iterations, then the computational cost also increase. Although this won't stop the attacker from brute forcing the key output itself, but this method is useful because it prevents or hinders the attacker's ability to search through the original password's key space.

Another important reason to use PBKDF2 is because it can hash a password, which means that its plaintext password won't be stored in the database. By iterating through the hashing procedure many times, it becomes slower for an attacker to brute force against either your live system or a database dump. Furthermore, PBKDF2 uses salting to defend against rainbow table attacks and to shield users who have made the bad choice to repeat their password across several websites.

PBKDF2 is also used to convert the password that has been received from the user into a symmetric key that is appropriate for the AES algorithm, this is an important and a common application for the algorithm. This is dependent on the fact that PBKDF2 can generate a hash of any length that satisfies the entropy criteria, and as long as the length and the entropy requirements are met for the AES, the AES can accept that value. This feature comes in quite

helpful for products like password vaults. A provider can guarantee that its software can only decrypt sensitive data while the user is logged in by using PBKDF2 and the password entered at login time.

After the signup / login, the project aims to develop a web interface that allows users to upload their files that they want to store on our system. For this purpose we create an AES key, this key is further encrypted with a password on the client side and is saved on the server. Further we will save file after encrypting it with a normal key. Now we will save this file with a new name and encrypt its original name. This is done so as to increase the security in case the database gets compromised.

To download the file after the encryption process, decryption password is needed, decryption of encrypted key from the server and then using this key as password for AES decryption of file and replace generated name with original and download it.

## 1.2 PROBLEM STATEMENT

In a society that is becoming more and more digitally dependent with each passing day, the need for secure sensitive material storage and transfer has become essential. The project's goal is to fill that gap by creating a hybrid cryptographic system for safe file storage. The method aims to combine the advantages of both symmetric and asymmetric encryption techniques to ensure data security and integrity during the storage and retrieval process. The primary goal of the project is the design and implementation of a file storage system that guarantees the security and confidentiality of the stored files. Hybrid cryptography will be used to achieve this, hybrid cryptography combines the use of asymmetric encryption for individual file segments with symmetric encryption techniques for preserving the encryption keys.

Also when the user requests for a particular file during the file retrieval phase, the system retrieves the encrypted segments and the matched encrypted symmetric keys from the server. The user's private key is then used to decrypt the encrypted symmetric keys, providing access to the symmetric keys needed for decryption. Now, with the decrypted symmetric keys in hand, the system proceeds to decrypt the encrypted segments, ultimately reconstructing the original file.

Mixed encryption system involves a balance between the more efficient but less secure symmetric encryption and the more secure but less efficient asymmetric encryption. Any storage system efficiency depends on how effectively it store and retrieve information to the requirements of protection, confidentiality and authenticity. This gives a practical and credible result for companies and individuals sensitive on privacy of personal data within the internet world which narrows down increasingly.

## 1.3 OBJECTIVES

The main objectives of the project are –

- 1) Enhanced Security - The project aims to establish a highly secure system where only authorized users can access stored files. This gives a sense of enhanced security and protection to the users.
- 2) Secure Data Retrieval - The project seeks to create a secure way for authorized users to get their encrypted files back. The decryption process should be transparent and efficient, ensuring a smooth experience while maintaining security.
- 3) Application and demonstration of encryption algorithms- The project seeks to showcase the practical use of diverse information security algorithms.

## **1.4 SIGNIFICANCE AND MOTIVATION OF PROJECT WORK**

The creation of safe storage system using hybrid cryptography is of great importance in today's digital world. The motivation for the creation of this project lies in the increasing concerns that are surrounding data security, issues such as compromise of sensitive information and unauthorized access. With increasing number of cyber threats on sensitive information, there is an increasing need to create an advanced system so as to protect the crucial information.

The significance of using hybrid cryptography lies in the fact that the method of hybrid cryptography, uses both symmetric as well as asymmetric encryption. With the help of hybrid cryptography one can create a strong system so as to protect against any cyber threat. Our main motivation is the important need to ensure the security of stored data, at rest as well as at transmission.

Another motivating factor for us is due to the need to protect the data against any insider threat as well. With the help of asymmetric key pairs, the system can have an extra layer of security which makes unauthorized access more challenging including to those who are working within the organization.

In conclusion, the development of a secure storage system is driven by the motivation of providing data security to the users and treating safety and security of the data as the primary objective. Moreover the system tries to ensure a simple and interactive user interface, so that the users can easily interact and have a user friendly experience.



## 1.5 ORGANIZATION OF PROJECT REPORT

The report is organised as follows:

- Chapter-01 presents us with the introduction of the study, along with the problem statement, objectives, significance, and organization of the project.
- Chapter-02 give us the existing related work in the field of secure storage systems and cryptographic algorithms, it further presents the outputs of their analysis which we compare and use in our project.
- Chapter-03 puts forward the system that we have developed to counter the security issues in third party storage providers. This is the chapter that includes the requirements, analysis, the design of project and its architecture and the implementation of the project.
- Chapter-04 gives us the Testing strategy or the tools used in the project.
- Chapter-05 presents the results and evaluation of the project along with comparison to any existing solution (if applicable).
- Chapter-06 presents the conclusion of the study along with future scope.

# CHAPTER 2: LITERATURE SURVEY

## 2.1 OVERVIEW OF RELEVANT LITERATURE

In today's world, there is an increasing volume of data and its storage, along with the growing threats of cyber-attacks, due to the combination of the former two, the important need for secure storage systems that are safe and durable has also increased by many folds. This literature survey tries to provide a detailed survey of the existing research and development in the field of secure storage systems. The articles that follow, clarify the goals a of secure storage system, and tell us about its advantages and disadvantages along with different cryptographic encryption algorithms that are used in them, the method of database encryption, cloud storage and its challenges and security threats in third party storage providers, SRP Protocol, PBKDF2 function etc. are studied within the given literature survey.

In the detailed analysis that follows, security has emerged to be the primary problem or cause for concern with third-party storage providers. For this reason we try to find different cryptographic encryption algorithms and analyse the most practical encryption algorithms from them.

Cloud computing storage security issues are generally classified into two groups: security issues that are raised by cloud-based service providers (companies that offer software, platforms, or structures as a service) and security issues raised by their clients (businesses or organizations that host operations or store data on the cloud). Participation and regular explanation of responsibilities are maintained under a cloud provider's participated security responsibility model.

A variety of studies on secure storage systems are examined in this literature review. The studies include important topics such as encryption, decryption, cloud storage, data security, hybrid cryptography and database encryption. The research studies investigate cryptographic methods,

access control methods, database encryption to make the system secure.

Therefore to make a secure storage system, it is majorly concluded that one must think from the user perspective of view and not from the company' point of view. For the user it might be necessary that they get the highest level of security so as to protect their data, whereas for a company it might not be feasible to get such a technology. Now in such a case the company has to find a trade-off, which would make sure that the user is satisfied as well as the process is also beneficial to the company.

The detailed study involving the research papers is as follows –

**Title: “Secure Data Storage in Cloud using Encryption Algorithm” (2021)**

This study basically focuses on cloud computing. The authors have given us an insight as to what cloud computing is and how is it helpful to us. The writer's further put pressure on the fact that how it is not easy for the users to trust any intermediate service providers for their crucial data, and how even the major companies that are present in the cloud domain assert on the face that both their organization and client bear the responsibility for the security. As a result, when data is kept in the cloud, it needs to be properly encrypted to prevent any other user from reading it even if they manage to gain access to it.

In this research work, the security issues in cloud storage are reduced. The system that the authors have recommended uses a combination of encryption algorithms like AES algorithm combined with S-box and Fiestel algorithm. We send and store data in a system in non readable format by sending it over a network in secure manner using cryptographic algorithm.

The encryption term related to make sure about the secret data. It checks the unapproved clients to access or utilize that data. These days numerous kinds of existing encryption method are accessible and greater part of them are in utilization as of now. In the proposed research paper, the data storage using encryption process comprises of two modules: AES and Fiestel. The submitted input information is divided into sub portions, these sub portions in the encoding

calculation is known as a square. Every square is separated into key and related plain content. The fiestel method further separates the important key into minor number of squares, spread over method of shift, and rotate. In the AES calculation the code key which is used is created in this network.

The reverse procedure of encryption is known as decryption. The data of cipher is divided into two parts of 128 bits each. The left part of the segment of chunk of 128 bits is given to the AES technique to recover the plain text and right side of 128 bits is given to Fiestel technique to recover the key. After this process, key and plain text are combined and are given as the output information to demanded client.

In conclusion to this paper, the writers have given a method which can help us to increase the safety of storage that is present on the cloud, this is done with the help of encryption techniques like RSA, AES etc.

**Title: “Secure File storage in Cloud Computing using Hybrid Cryptography algorithm”  
(2015)**

In today’s world the technique used to make sure that the data is safe is cryptography and along with this the technique of steganography is also used. To achieve better security results in cloud computing it is recommended to not use only one algorithm. Cryptography and steganography technique are more popular now a day’s for data security. Using a single algorithm is not considered to be effective for high level security in cloud computing. In this paper, the writers have proposed a new security mechanism using symmetric key cryptography algorithm and steganography. In the proposed system AES, blowfish, RC6 and BRA algorithms are used to provide block wise security to data.

Further LSB steganography technique is introduced for key-information security. Key information contains which part of the file is encrypted using which algorithm and key. The file is splited into 8 parts, each and every part of file is encrypted using a different algorithm. All parts of the file are encrypted simultaneously using the multithreading technique. LSB technique

is used to insert the data encryption keys in the cover image. For decryption purposes the reverse of encryption process is applied.

The main reason behind this system is reduced storage cost. Private cloud is more secure than public cloud. To increase the security a file in cloud computing, the source file is break into different parts, then every part of the file is encrypted and stored on more than one cloud. Information about file is stored on cloud server for decryption purposes. If an attacker tries to recover an original file, then he will get only a single part of the file. AES and 3DES algorithms are merged into hybrid algorithm to accomplish confidentiality. It is harder for attacker to recover secret file of user.

In the given solution to the problems algorithms such as blowfish, RC6 and AES are used to improve the data security. The given solution is the combination of the former said algorithms. The methods used involves symmetric algorithms, symmetric algorithms take only one key for the encoding and decoding purposes of a file. After this the cover image is used to hide the information of the key with the help of LSB technique. A text file's encode and decode times are computed by contrasting the current AES and Blowfish algorithms. For AES, the file size is specified in megabytes.

The analysis of time between AES and the given solution is given as follows –

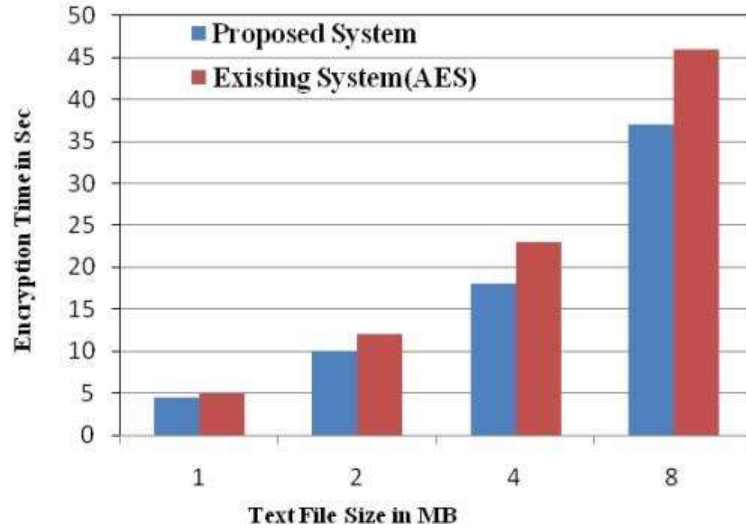


Figure 1: AES vs. given solution

From fig1, we can see that the given solution needs significantly less amount of time the encoding of file. This is so because in proposed system combination of symmetric key cryptography algorithms are used together. The hybrid form of algorithm, requires 19% to 22% less amount of time for text file as compared to the already present solution.

The analysis b/w the decryption time with AES and given solution is as follows –

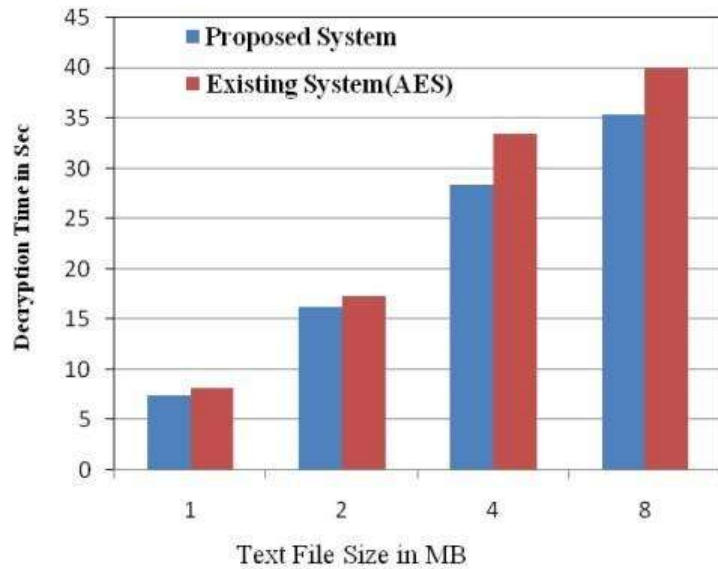


Figure 2: Decryption time Comparison with AES and Given solution

We can see in fig2 that the existing solution takes about 18% to 20% maximum amount of time needed for file decryption as compared to hybrid form of algorithm. The AES algorithm takes the least amount of time to decrypt data.

With the help of given proposed security mechanism, high security, very low delay and most importantly integrity of data is achieved. The given text file encryption method requires 19% to 22% less time as compared to AES algorithm, and 18% to 22% less time for decryption purposes.

**Title: “Formal Methods Analysis of the Secure Remote Password Protocol” (2020)**

Due SRP stands for secure remote password protocol, SRP is a secure password based authentication protocol and key exchange protocol. It is basically an Authentication protocol for password based authentication over an insecure network. This paper tells us about the Secure Remote Password. It also tells us about weaknesses using the Cryptographic protocol shapes

analyser. The challenging part in the SRP protocol is its formal analysis, because currently no existing tool can provide us with a reason to use the expression “ $v + g^b \text{ mod } q$ ”.

We can complete the study of all the sequences that are possible in SRP by modelling  $v + g^b$  as encryption. The design constraints of SRP give rise to a crucial weakness in SRP protocol. A server that has a malware or is malicious in nature has the ability to fake the session involving authentication methods with a client, without the client knowing about it. A malicious server can fake an authentication session with a client, without the client’s knowledge. Now this attack can also lead to something known as a privilege attack, if the client is given more privileges than the server.

There can be many reasons for which a protocol can fail, these include weak cryptographic algorithm used, implementation error, improper configuration etc. This study focuses on the structural weakness that is, fundamental logic errors.

SRP stands for Secure Remote Password Protocol and it is a Password Authenticated Key Exchange (PAKE) used in many products in the industry. In a PAKE protocol, there is a pre shared password between two participants which is used to authenticate both the users with each other and generate a key known as the session key. The main purpose of SRP is that it aims to avoid dictionary attacks by not storing the plaintext passwords on the server. As it does not use encryption, hence it also avoids the export controls.

The Fig3 give summary of how SRP-3 works, in this two participants Raj and Rahul create a key called the session key K, which gives rise to generation of a password P that is known to both Raj and Rahul. The SRP-3 protocol works in 3 phases: the first phase includes the registration, second phase includes the Key establishment and third phase is the key verification. The protocol will further create a new session key K, which is known to Raj and Rahul which both of them can use.

In phase-1, Raj must register her password P with Rahul before executing the protocol. Bob will then store  $(s, v)$  and index it as Alice. Here s is a random salt and  $x = h(s, p)$  is the salted hash



value of the Alice's Password.  $v = g^x$  is a verifier that is derived from P.

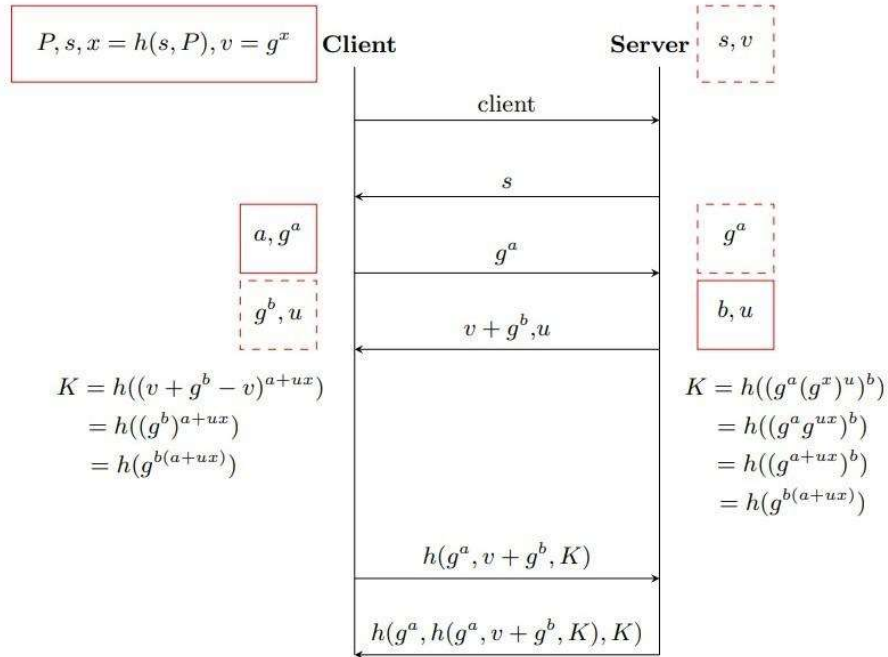


Figure 3: Protocol Diagram for SRP-3

During phase-2, Raj will send his identity as Raj to Rahul, Rahul will then find Raj's salt  $s$  and the stored verifier  $v = g^x$ , where  $x = h(s, P)$ . Rahul then sends Raj her salts, Raj then receives Rahul's salt  $s$ , calculate  $x = h(s, P)$  and generate an arbitrary secret number that can be used only once. Raj calculates and sends  $g^a$  to Rahul. Rahul then receives  $g^a$  and generates a random arbitrary key  $b$  and calculates  $v + g^b$ . In Phase-3, Alice computes  $M1$  sends it to Rahul, Rahul computes  $M2$  and sends it to Raj. Raj then verifies the value that is received by recomputing  $M2$ . This is the Basic working of the SRP protocol.

**Title: “A secure storage service in the Hybrid Cloud” (2011)**

In general, cloud storage providers are less secure because there are many dangers that a user might encounter with them, some of the issues faced by the user can be – a problem can occur where the data might be exposed in the database, that is anyone can see that data, this can generate problems for the user since there is no security of their data now. Another example of danger associated with cloud storage providers is that, an attacker can change the contents of the data of the user, or delete the entire data or maybe modify the data to something new completely. Another problem with cloud storage providers is that, if a user wants to access the data, then there is a possibility that he might not be able to do so, because the cloud storage provider can deny them the access to do so, such a case can occur during high demand.

The authors of this paper give us a simple solution so as to make the storage practises in the cloud environment more secure. In the paper the writers have shown that how we can easily store and then retrieve the files very securely using this system even in cloud environments.

There is another problem related to a storage provider that is cloud based, it is that the user will have very less authority over the security of their storage. Now this can further give rise to many problems such as the cloud service provider can fool the users and sell the important and confidential data of the users to some high paying companies in order to earn money, they can do this without the customers getting to know about this.

Now to tackle these problems, one has to make sure that the data that is being stored in the storage system is encrypted well. Hence maintaining data integrity is very necessary both when the system is running and when it is not. Another risk is data availability, also referred to as the potential for third parties or the cloud provider to restrict the access to data.

The design of the proposed system is shown in fig4, each component is described as -

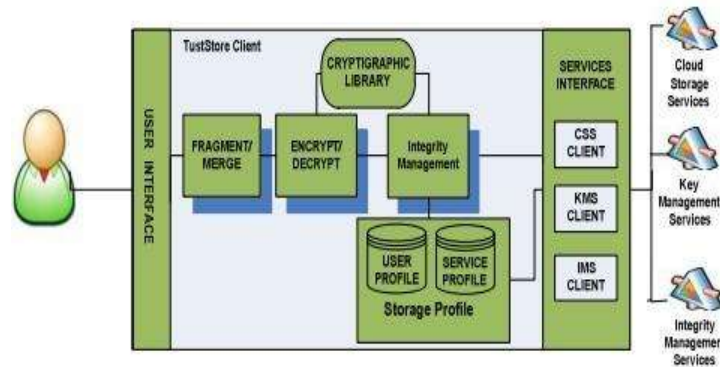


Figure 4: Architecture of proposed system

Users can work together to share and safely store the crucial data in storage environments that are cloud based and are public. The proposed system can help the users to achieve so. To do this, the data is first split up, encrypted, and signed before being uploaded to storage.

### **Title: “Generation of Random Key Using Randomness of Hash function” (2020)**

Random numbers play a huge role in modern world cryptographic systems. In today’s world most cryptographic systems depend on pseudo random number generators (PRNG). To get some random values a mode of noisy channel is used by these protocols, these noisy channels can be fan noise or disk noise etc. But the problem is that there may be some requirements in form of hardware for these channels which are noisy, but interestingly there is the pseudo random nature that is related to hash function, so we can use hash functions to replace these noisy channels. This research paper discusses some of the methods that help us to create a key and also their disadvantages.

Algorithms such as Advanced Encryption Standard, Data Encryption Standard or One Time Passwords take the help of randomness in the key expansion phase. A secure random value is needed so as to make the system secure from any foreign threat. Noisy channels can be used for

this purpose but it cannot be done in cases where disk space is low.

A function that take an unspecific length of input is called hash function. Hash function produces an output string that has a fixed length. An important property that a hash function should have is that it should have strong collision resistance and it should have the pseudo random property. The pseudo random property of the hash-functions can help us to counter the problems encountered by noisy channels. This study tells us about the key- derivation function that are based on hashes, and it also discusses how they are helpful to us.

The key derivation functions in general have many drawbacks such as –

- The keys require a significant amount of random bits in order to facilitate the security. If we generate a key directly by a random number generator then it is difficult to get these many bits.
- Password-derived keys are easily guessed or vulnerable to dictionary or birthday attacks, so they cannot provide the level of protection required by cryptographic keys.
- Key distribution function, derived key length, asymmetric key strength, and key derivation method are the major factors that determine security in a system where key has already been shared.

So, to overcome all these problems and generate secure cryptographic keys we tend to use hash functions along with the key derivation functions.

### 2.1.1 GENERATING KEYS WITH THE HELP OF HASH

A key derivation function is a function that is involved in cryptographic methods, it takes, information known as keying-material to perform the required action. A key derivation function follows the process of extract and expand, in the extraction part the calculation takes the required information and with the help of mathematics it creates a Key K. This key K is the pseudo random key and this crating of key can also be termed as extraction of key. In the expansion part, a the name itself suggests, the algorithm expands and converts the key K into more pseudo random keys.

Representation of the extract function is as follows –

<b>Options:</b>	
Hash	A hash function
<b>Input parameters:</b>	
salt	Salt value(Optional); (set to a string of zeros by default, if not passed).
IKM	Input Keying Material.
<b>Output:</b>	
prk	A specific-length pseudo-random key

Figure 5: Parameters involved in an extract function

The representation of the expand function is as follows –

<b>Options:</b>	
Hash	A hash function
<b>Input parameters:</b>	
prk	usually, the output from the extract stage) A specific-length pseudo-random key.
info	Application-specific information (can be a null string too.)(Optional)
L	Length of Output Keying Material in octets (Bytes).
<b>Output:</b>	
OKM	Output Keying Material of L octets.

Figure 6: Parameters involved in an expand function

### 2.1.2 PBKDF2 (PASSWORD BASED KEY DERIVATION METHOD)

In a general case, the password that a user chooses has less amount of randomness. Therefore is not considered to use these passwords as keys in cryptography. With the help of Key derivation functions. But a key can be created using these passwords, with a value that is secret in nature. This key can now be further used in algorithms.

The implementation of Password based key derivation function takes the help of a variable say  $i$ , this  $I$  denotes the total number of iterations for which the algorithm will run, in addition to this a random function is used, but the point to note here is that the random function must be pseudo in nature.

In password based key derivation, suppose the user chooses a password, ‘w’, then we send this

password, a salt 's' and length of the key. The number of iterations chosen should be larger so as to give higher security. Note that attacks that involve dictionary can be avoided by this algorithm and even brute force attacks can also be avoided, this is so because the original password is not recovered quickly or easily

The working procedure of PBKDF2 is as follows –

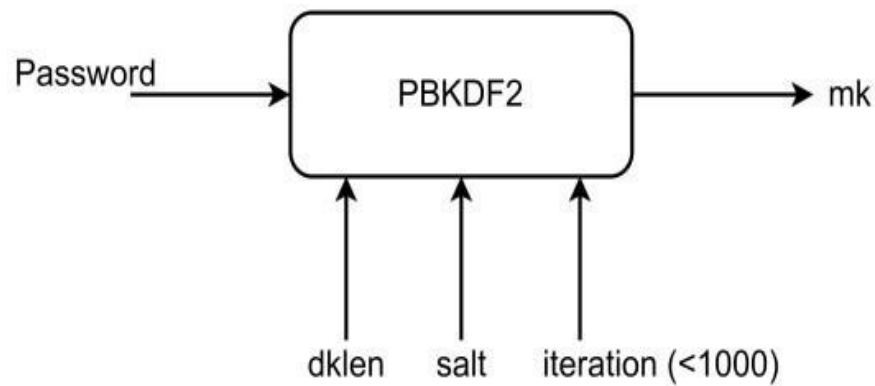


Figure 7: PBKDF2 working procedure

As it can be seen from the above figure, the PBKDF2 function takes password, salt length, and number of iterations as the input and it produces a master key.

Even though the PBKDF2 is considered to be a strong algorithm, it still has some cons, a major con is that it can be cracked using a suitable equipment since it has a low level design. Along with this the number of operations are high in PBKDF2 due to some useless operations being done during the computation of the hash function.

## **2.2 KEY GAPS IN THE LITERATURE**

The literature survey has given us some valuable information regarding secure storage systems and the technology used to build a secure storage system, moreover it has also helped us to know more about some algorithms and methods such as SRP protocol, key derivation functions, PBKDF2(password based key derivation function) . The literature survey has also exposed us to many encryption algorithms used in the modern world such as AES, DES, 3DES, RSA and more.

Despite all this information, there are certain gaps in the literature survey.

The papers mention the security challenges in cloud storages, but don't give us a detail analysis about the problems in cloud computing storages, moreover no concrete solution is provided to overcome these challenges. It is said we can use encryption algorithms such as AES with S-Box but there can be more focus on constructing a more optimised algorithm.

There can be more mention of how other techniques such as steganography can be combined with cryptography so as to increase the security of a system.

There can be more focus on different key derivation functions, including the strengths and weaknesses of PBKDF2 (password based key derivation function 2), and the cases where PBKDF2 might fail.



The papers do mention the technique of password hashing, but there can also be a mention of how human factors affect the passwords. For example – Some people may have the tendency of keeping password only related to sports or some people may use their names in the passwords.

There is not any simple android app which demonstrates and explains the working of srp protocol.

# CHAPTER 3: SYSTEM DEVELOPMENT

## 3.1 Requirements

### Software Requirements

- VS Code, Sublime Text, LunarVim
- MongoDB, PostgreSQL, etc.
- Git / Github

### Hardware Requirements

- Random Access Memory (RAM): 4 GB or above
- Central Processing Unit (CPU): 2.4 GHz Processor and above
- Operating System (OS): Windows/Linux

## 3.2 PROJECT DESIGN AND ARCHITECTURE

The primary goal of this project is to offer a highly secure storage system, where only the authorized users having the right permission can access the stored files, this involves using strong authentication methods to prevent any unauthorized entry. Moreover, the project should always allow the authorized users to get their encrypted files back. The decryption process should be as transparent as possible and also efficient, ensuring a smooth experience while maintaining security. Moreover, there must be a system to ensure that if the storage system and

its content ever gets compromised, the attacker should not be able to access the information contained in that file. So, a number of issues needed to be resolved, including:

- How will the login/signup process work?
- How will the encryption and decryption process work?
- Which encryption algorithms will be used and why?
- What will be done in case the contents of the system gets compromised?
- How will the files be retrieved?

The plan for the design of the project includes creating an encrypted system where the user can upload their files, the files will be encrypted as well as with its name, so that in future if ever the storage system gets compromised then the attacker can never get to access the contents of the original file. Moreover a process is needed so that during the signup process the password is not saved on the server, for this PBKDF2 (password based key derivation function) method is used. The protocol we have used for the functioning of this project is called Secure Remote Password (SRP) protocol.

### **3.2.1 Secure Remote Password (SRP) Protocol:**

We wanted to store the files in the server after encrypting but the key was also needed to be kept somewhere, if we saved it in the database, then there might be a possibility that if the database gets breached, then the attacker can recover the key using the password present in the database.

This protocol allows authentication without the need to save passwords anywhere. This is used almost everywhere in the industry. It is a key exchange protocol. After the successful authentication, it also exchanges secure keys which can allow the server and client to verify each other when communicating.

### 3.2.2 Sign Up:

During signup process, we take the email from the user and check it against our database. If the email is not present then we continue with the process, otherwise we throw error.

If email is not present then on the client side, we generate a salt and use email, password and the salt to create a secret key using PBKDF2 function. We use the secret key along with the SRP group to create a verifier. We send this verifier along with the email, salt, and SRP group to the server for storing it in the database.

The flowchart for signup using the SRP protocol is as follows –



Figure 8: Flowchart for signup using SRP protocol

### 3.2.1 Login:

The process of login functionality of the project is as follows -

During login process, we get the email from the client and check whether it is present in the database. If it is there then we send the salt and SRP group to the client, otherwise we throw an error (we can also send a fake salt for more security).

Now the client will generate a key using the salt, email and password and then use this key to generate a key pair using the SRP group. The client will then send the public key to the server and keep the private key.

Both will perform an operation to generate a session key using the given resources. Now client will use the session key to send encrypt and send the message to the server, and the server can further verify the genuineness of the message by decrypting it with the session key. If it fails then either side can decline the request.

We used the secure-remote password library to implement this method using Node.js. First we generate a secret key using email, password and salt and then create a verifier using provided library functions. After that we send it to the database.

At login, we create ephemeral at the client side, and ephemeral using verifier on the server side as well. We exchanged the public keys and the generated private keys using PBKDF2 function. After that the exchanged proofs are verify it.

If the verification was successful then we send an encrypted key to the client side, along with the encrypted list of files.

The flowchart for login using the SRP protocol is as follows –

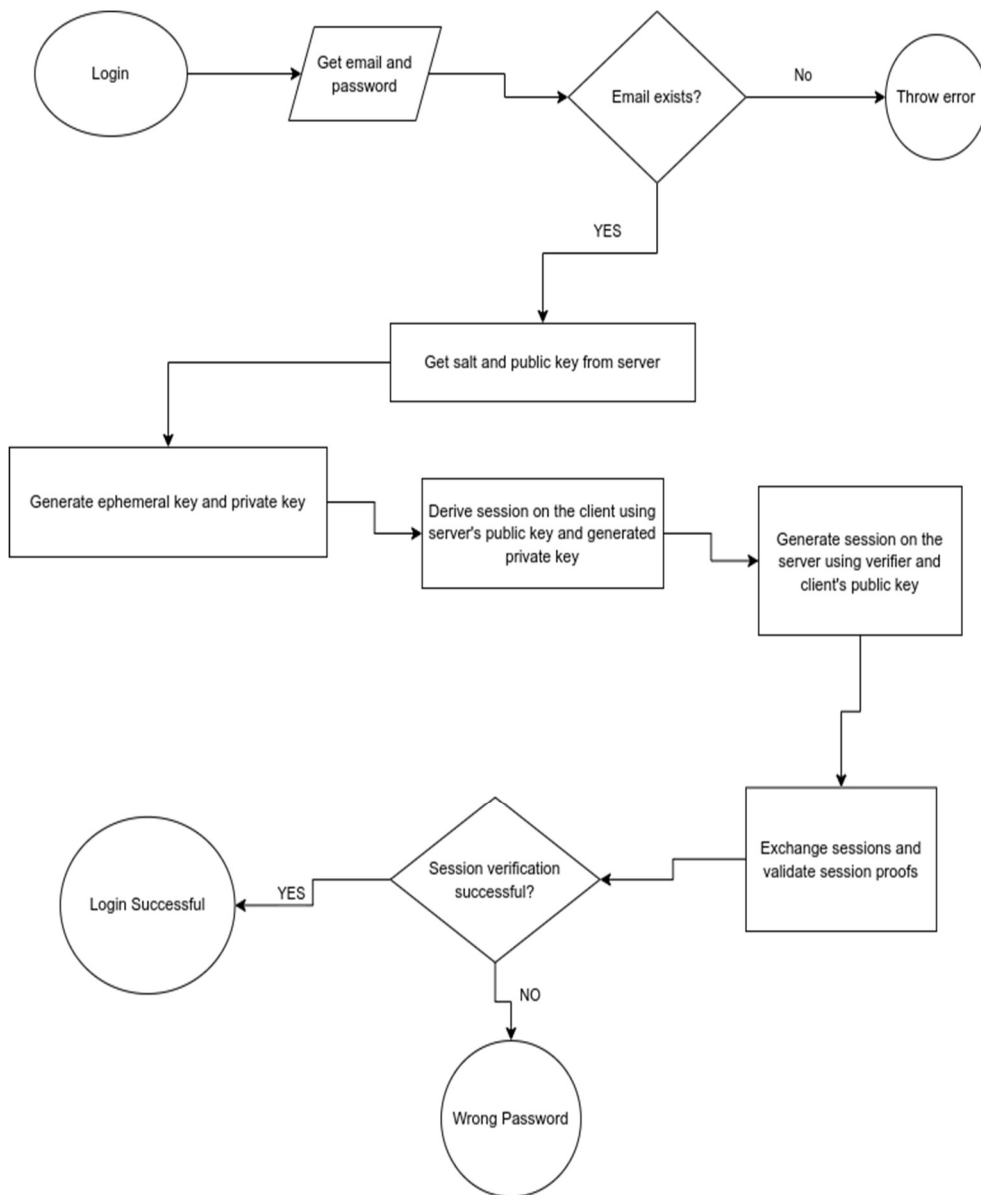


Figure 9: Flowchart for login using SRP protocol

The detailed explanation of the involved process is as follows -

The user accesses the website, after this it is checked whether the account already exists on the system or not. To check this, only the email of the user is sent to the server, now if the email exists we will proceed further and if it does not exist we will move on to the signup/registration part.

The registration part includes taking the email and password of the user and generating a salt value. A salt is nothing but a random string of characters that is added to the password to enhance them. The salt can be placed on either side of the password. Salting is necessary because the hashed passwords are not different to each other due to the nature of hash functions. When given the same input the hash function can give the same output. For example:

If Alex and Bob both choose a password, “pwndont44” then their hash would be the same –

Alex: 4420d1918bbcf7686defdf9560bb5087d20076de5f77b7cb4c3b40bf46ec428b Bob:  
4420d1918bbcf7686defdf9560bb5087d20076de5f77b7cb4c3b40bf46ec428b

As we can see, if two users have the same password, then they will have the same hash value.

The attacker can exploit this loophole, once the attacker gets access to the database, he will start looking at duplicate hashes, due to the duplicate hashes it can be concluded that there is either no salt or a very weak algorithm is used. A lot of same hashes will lead to the conclusion that every account can have a default password. The attacker will find the default password and exploit the sensitive information. The unsalted passwords can be attacked using method of dictionary attack.

So, to reduce the damage that a hash table or a dictionary attack can cause, we salt the passwords. OWASP guidelines says that a value that is generated by a cryptographically secure value is called salt. This salt is then added to the input of hash functions to make the hash for every same or different inputs unique. Therefore for adding more security, we use this method of salting.

After generating the salt on the client side, we will also generate a verifier function. We will use

email, password and the salt value in a PBKDF2 function. The PBKDF2 (password based key derivation function) is a simple key derivation function, the PBKDF2 takes password, salt and the number of iterations to generate a private key. This private key will further be put into the verifier function to get the verifier.

Now we will send the salt, email and verifier to the server. Note that here we are not sending the password to the server. With this our signup process is completed.

In the login process, we take the email and password and send the email to the server. Now if the email exists on server, we will receive a salt value from the server and a temporary public key and the private key will be save by the server. The client further generates a temporary public, private key and this public key generated by client is send to the server. The client then uses the salt received from the server as well as the public key of server and the email, password to generate a proof, and send it to the server. The server will generate its own proof following the above steps and check if both the client side and server sides proof matches, if it match we are logged in.

This is the procedure involved during the login phase.

### **3.2.3 Upload:**

When a user is first created, we also generate a big random string which we use as a key for our AES encryption. We then encrypt this key using some password called the media password. This password is also not stored anywhere. Loss of this password would mean loss of data as without it, we can't recover anything. After encryption this key is saved into the database.

This key is sent to client after successful login and user is asked for media password again to decrypt this key to get the original one back. This decrypted key is then used to decrypt all the files and other metadata so that it becomes user readable. During upload, a file gets selected and we extract its name from its metadata and encrypt it using media password, we also



generate its new name using uuid library. We save this new name and encrypted name as key value pair in the database.

We load the file into a variable and convert it to a WordArray. After that we encrypt this WordArray using AES in CryptoJS library. After encryption we save it inside a folder named after the user.

If anyone gains access of the system then they will not know what file is a particular file as everything is encrypted.

The flowchart for the upload is as follows –

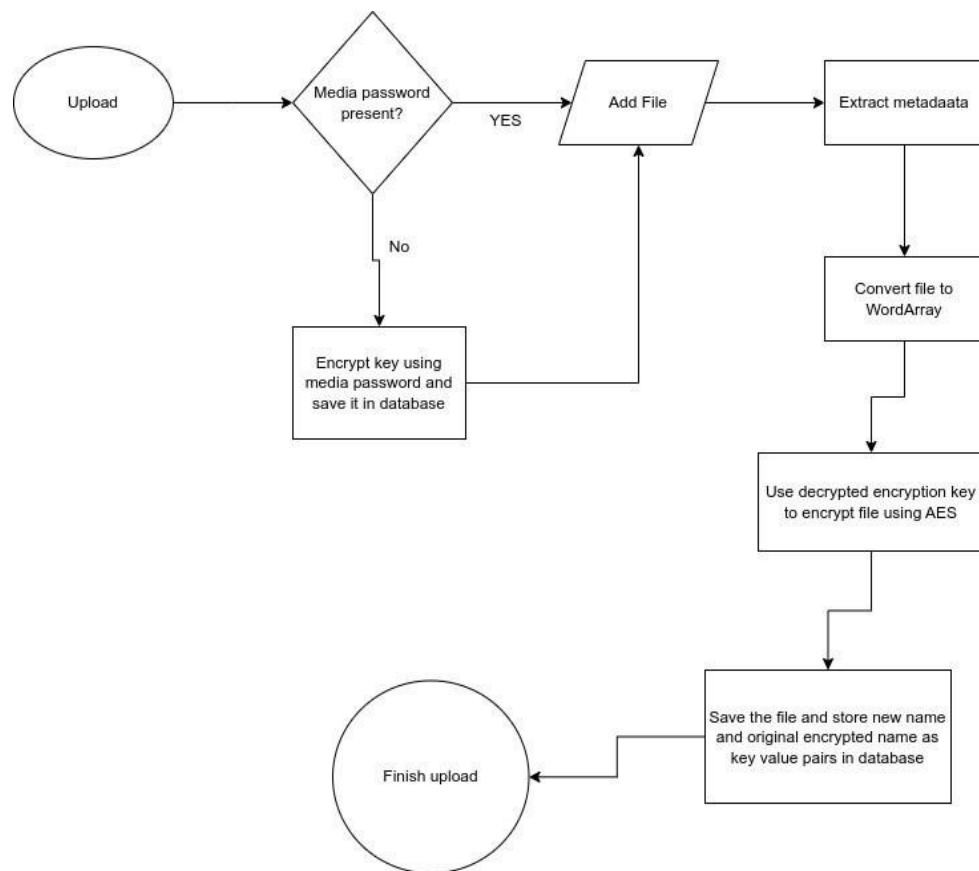


Figure 10: Flowchart for Upload

### 3.2.3 Download:

When user clicks on the file listed on the content page, it gets downloaded in the background in the encrypted format, after that we use decrypted key to decrypt this file. This decrypted data is still in unusable format (WordArray) so we have to convert it from WordArray to Uint8Array and then save it as a blob to a variable and download it finally. We also decrypt the encrypted name from database and use it as a new file name so that user can identify it.

The flowchart for the process if download in our project is as follows –

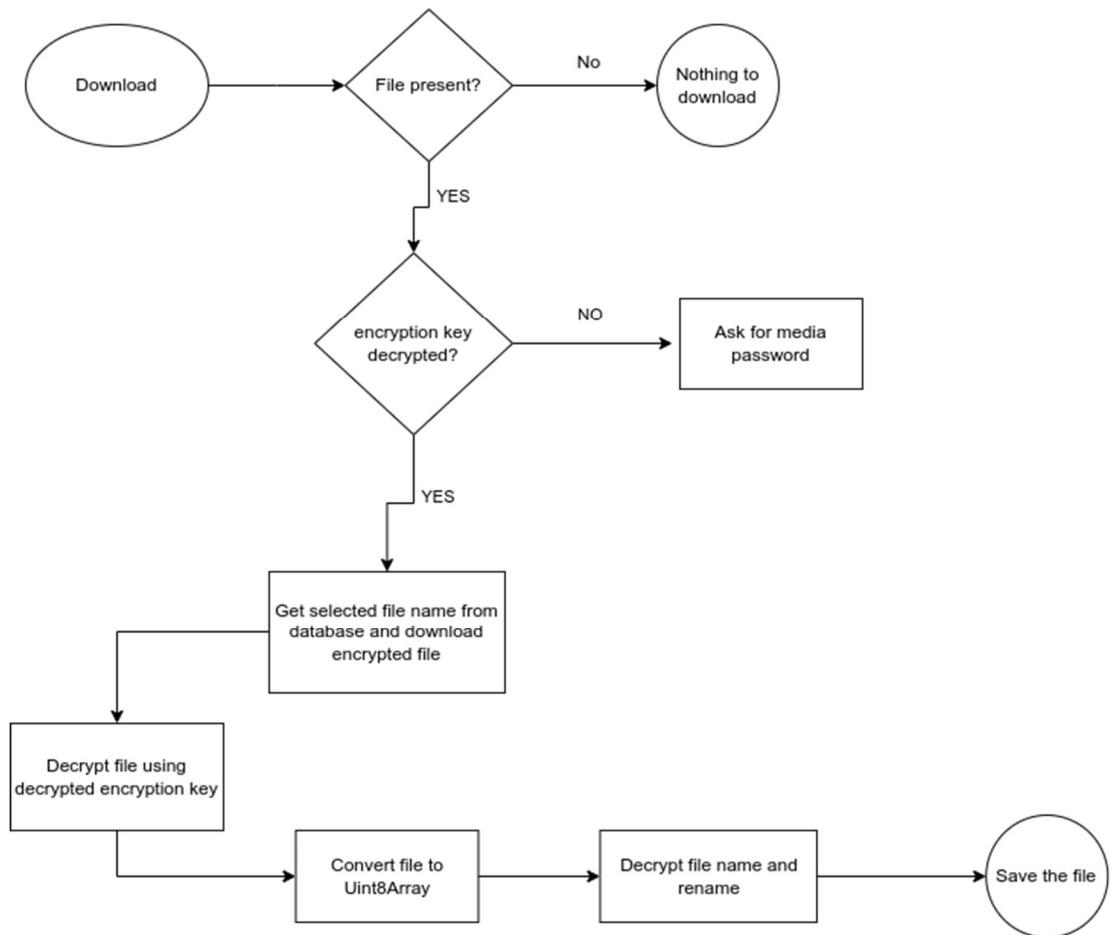


Figure 11: Flowchart for Download

### 3.2.4 SRP in our android app:

When user clicks In our android app the working and implementation of the android app is as follows -

The Secure Remote Password (SRP) protocol enables secure user authentication over public and untrusted networks by ensuring that passwords are not stored anywhere. During the account creation process, users supply a server with a salt and a verifier, calculated using their username, password, and a randomly generated salt, with the verifier being created via the PBKDF2 function. To enhance security and resist brute force attacks, the number of iterations in PBKDF2 can be increased, though this also increases the computational load, especially noticeable on mobile devices compared to PCs.

When logging in, a user generates a secret and a corresponding public ephemeral value, sending the public value and their username to the server to identify their account. The server retrieves the salt and verifier using the username, generates its own ephemeral values, and sends the salt back to the client. The client then uses their credentials and both the server's and their own ephemeral values to derive a shared session key, sending proof of this key to the server. If the server successfully verifies this proof using its verifier and private key, it sends its proof back to the client, who must also verify it to complete authentication.

If a user logs in for the first time, they are prompted to encrypt a large random string with another password, which is not stored by the system, thus securing the data. The encrypted string is then stored in the backend using AES encryption, the strongest available method, significantly hindering brute force attacks. Users are responsible for remembering their encryption password, as losing it would result in inaccessible data.

File uploads are encrypted client-side using AES and then stored with their metadata in the backend. Each file's original name is encrypted and stored in MongoDB. When logging back in, users decrypt their key locally with their second password, maintaining decryption capability without exposing their password. This key is only available during the session and is lost when the tab is closed.

Files are encrypted on the client-side before being uploaded to the server. For downloads, the server sends encrypted files that users decrypt locally. Browsers handle encrypted files as Blobs, facilitating file transfers and operations, but Android lacks Blob support. On Android, files are read in base64 encoding, encrypted, and then sent to the server. Upon downloading, files are decrypted, and the resulting data is handled differently depending on the platform: as base64 on Android and as a Blob in browsers. In Browsers, we can store the file as Blob. This makes it easy for transferring a file and applying operations on it. On Android, we don't have Blob support. Hence we cannot read a file just the way we do on browsers. On android, we first read a file in base64 encoding. We read it using readstream for faster operations. After this we create a wordarray after parsing this base64 string. This wordarray is then encrypted using AES and converted to a string. This string is wrapped in RNFetchBlob and sent to server.

Download part is same for both, except a blob is created in browser whereas base64 encoding is done on Android app. First after getting the encrypted file, we decrypt it using the decrypted key.

After this we get a CipherParams object, we convert this to Uint8Array. This is then converted to base64 in android and blob in browsers. This is the final step, now it gets written to the local disk and is decrypted successfully.

### **3.3 DATA PREPARATION**

Our testing data consists of PDFs, PNGs and other similar files. After encrypting them and downloading and decrypting them, we can check their MD5 hash to check their integrity.

## 3.4 IMPLEMENTATION

### 3.4.1 MongoDB Database Schema

```
const User = new mongoose.Schema({
  email: {
    type: String,
    required: true,
    unique: true,
    index: true,
  },
  salt: {
    type: String,
    required: true,
  },
  verifier: {
    type: String,
    required: true,
  },
  tempEphemeral: {
    type: String,
  },
  encryptedKey: {
    type: String,
  },
  files: {
    type: Array,
    default: [],
  },
  mediaPassword: {
    type: Boolean,
    default: false,
  },
});
```

Code snippet of MongoDB database schema

Here we use,

Email: for identifying the user

Salt: necessary for login process

Verifier: required for SRP protocol

TempEphemeral: ephemeral key for authentication

Encryptedkey: encrypted version of key used in AES decryption

Files: array of files stored by this user

MediaPassword: Boolean which checks whether encrypted key is added by user during account creation

### 3.4.2 Signup

```
router.post('/signup', async (req, res) => {  
  const { email, salt, verifier } = req.body;  
  const user = new User({  
    email,  
    salt,  
    verifier,  
  });  
  
  try {  
    await user.save();  
    res.status(200).json({ message: 'User created successfully' });  
  } catch (err) {  
    console.log('error', err);  
    res.status(409).json({ message: 'Already exists' });  
  }  
});
```

Code snippet of Signup route in the backend

### 3.4.3 Login

```
const router = require('express').Router();
const srp = require('secure-remote-password/server');
const User = require('../models/users');

router.post('/login', async (req, res) => {
  console.log('enter', req.body.email);
  let user = await User.findOne({ email: req.body.email });
  if (!user) {
    return res.status(400).json({ message: 'User not found' });
  }

  const serverEphemeral = srp.generateEphemeral(user.verifier);
  user.tempEphemeral = serverEphemeral.secret;
  await user.save();

  res.status(200).json({
    salt: user.salt,
    serverEphemeral: serverEphemeral.public,
  });
});

router.post('/loginWithProof', async (req, res) => {
  const { email, clientEphemeral, clientProof } = req.body;
  const { tempEphemeral, salt, verifier, encryptedKey, files, mediaPassword }
    = await User.findOne({ email });

  try {
    const serverSession =
      srp.deriveSession(tempEphemeral, clientEphemeral, salt, email, verifier, clientProof);
    res.status(200).json({ serverProof: serverSession.proof, key: encryptedKey, files, mediaPassword });
  } catch (err) {
    res.status(401).json({ message: err.message });
  }
});

module.exports = router;
```

Code snippet of the route in backend which deals with Login using SRP protocol

### 3.4.4 Adding media password (encryption key)

```
const AddEncryptedKey = () => {
  setErrorMessage('');

  if (Password !== confirm_password) {
    setErrorMessage('Passwords do not match');
    return;
  }

  if (Password.length > 0) {
    RNSecureStorage.getItem('user_data')
      .then((data) => {
        user_data = JSON.parse(data)

        const email = user_data.email;
        const dec_key = CryptoJS.PBKDF2(srp.generateSalt(), srp.generateSalt(), {
          keySize: 256 / 32,
          iterations: 100,
        }).toString();
        console.log("dec", dec_key);
        const encryptedKey = CryptoJS.AES.encrypt(dec_key, Password).toString();
        console.log(encryptedKey);
        const decryptedKey
          = CryptoJS.AES.decrypt(encryptedKey, Password).toString(CryptoJS.enc.Utf8);
        console.log(decryptedKey);

        fetch(`http://${SERVER}/api/setEncryptedKey`, {
          method: 'POST',
          body: JSON.stringify({ email, encryptedKey }),
          headers: {
            'Content-Type': 'application/json',
          },
        })
          .then((resp) => resp.json())
          .then((resp) => {
            console.log(resp.message);
          });
      });
  }
}
```



```

console.log(JSON.stringify({
  ...user_data,
  "mediaPassword": true,
}));

RNSecureStorage.setItem('user_data', JSON.stringify({
  ...user_data,
  key: encryptedKey,
  "mediaPassword": true,
}))
.then(() => console.log('Media Password Set'))
.catch((err) => console.log(err.message));

dispatch(initialize(dec_key));
navigation.dispatch(
  CommonActions.reset({
    index: 0,
    routes: [
      {
        name: 'Home',
      }
    ],
  })
);
}
.catch((err) => {
  console.log(err.message);
  setErrorMessage('Error setting media password');
});
})
.catch((err) => console.log(err.message));
}
}

```

Code snippet of the route in backend which saves encrypted key in the database.

### 3.4.5 Upload

```
const fileUploader = async () => {
  setDownloadStatus('');
  setUploadStatus('');
  const file = await filePicker();
  if (file) {
    console.log(file);
    const enc_key = key;

    if (file.size > 50000000) {
      setErrorMessage('File too large, max size is 50MB');
      return;
    }

    setUploadStatus('Reading file...');
    await new Promise(resolve => setTimeout(resolve, 0));

    let fileData = ''
    RNFetchBlob.fs.readStream(file.uri, 'base64', 8190000, 10)
    .then((ifstream) => {
      ifstream.open()
      ifstream.onData((chunk) => {
        console.log('ifstream data', chunk)
        fileData += chunk
      })
      ifstream.onError((err) => {
        console.log('ifstream error', err)
      })
      ifstream.onEnd(async () => {
        console.log('ifstream end')
        setUploadStatus('Creating WordArray...');
        await new Promise(resolve => setTimeout(resolve, 0));
        const wordArray = CryptoJS.enc.Base64.parse(fileData);
```

```

const wordArray = CryptoJS.enc.Base64.parse(fileData);
setUploadStatus('Encrypting...');
await new Promise(resolve => setTimeout(resolve, 0));
const encryptedFile = CryptoJS.AES.encrypt(wordArray, enc_key).toString();
setUploadStatus('Generating a temporary file...');
await new Promise(resolve => setTimeout(resolve, 0));
RNFS.writeFile(`${RNFS.TemporaryDirectoryPath}/encryptedFile.txt`,
encryptedFile, 'utf8')
  .then(async () => {
    await new Promise(resolve => setTimeout(resolve, 0));
    console.log('TEMP FILE WRITTEN!');
  })
  .catch((err) => {
    console.log(err.message);
  });
const fileName = uuidv4();
const enc_original_name = CryptoJS.AES.encrypt(file.name, enc_key).toString();

setUploadStatus('Uploading...');
await new Promise(resolve => setTimeout(resolve, 0));

RNFetchBlob.fetch('POST', `http://${SERVER}/api/upload`, {
  'Content-Type': 'multipart/form-data',
}, [
  { name: 'email', data: email },
  { name: 'fileName', data: fileName },
  { name: 'enc_original_name', data: enc_original_name },
  { name: 'userFile', filename: fileName,
    data: RNFetchBlob.wrap(`${RNFS.TemporaryDirectoryPath}/encryptedFile.txt`) },
])
  .then((res) => res.json())
  .then((data) => {
    setFiles((files) => [...files, { fileName, enc_original_name }]);
    RNSecureStorage.getItem('user data')
  })

```

```

.then((data) => {
  user_data = JSON.parse(data);
  RNSecureStorage.setItem('user_data', JSON.stringify({
    ...user_data,
    files: [...user_data.files, { fileName, enc_original_name }],
  })))
  .then(() => {
    RNFS.unlink(`${RNFS.TemporaryDirectoryPath}/encryptedFile.txt`)
    .then(() => {
      console.log('TEMP FILE REMOVED!');
    })
    .catch((err) => {
      RNFS.unlink(`${RNFS.TemporaryDirectoryPath}/encryptedFile.txt`)
      .then(() => {
        console.log('TEMP FILE REMOVED!');
      })
      console.log(err.message);
    });
    setUploadStatus('Uploaded!');
    setTimeout(() => {
      setUploadStatus('');
    }, 5000);
    console.log('File added to user data')
  })
  .catch((err) => {
    RNFS.unlink(`${RNFS.TemporaryDirectoryPath}/encryptedFile.txt`)
    .then(() => {
      console.log('TEMP FILE REMOVED!');
    })
    console.log(err.message)
  });
});

```

```

    })
    .catch((err) => {
      console.log(err.message);
      setErrorMessage('Error reading file, it may be too big to store in memory');
      return;
    });
  }
};

function convertWordArrayToUint8Array(wordArray) {
  var arrayOfWords = wordArray.hasOwnProperty('words') ? wordArray.words : [];
  var length = wordArray.hasOwnProperty('sigBytes') ? wordArray.sigBytes : arrayOfWords.length * 4;
  var uInt8Array = new Uint8Array(length),
      index = 0,
      word,
      i;
  for (i = 0; i < length; i++) {
    word = arrayOfWords[i];
    uInt8Array[index++] = word >> 24;
    uInt8Array[index++] = (word >> 16) & 0xff;
    uInt8Array[index++] = (word >> 8) & 0xff;
    uInt8Array[index++] = word & 0xff;
  }
  return uInt8Array;
}

```

Code Snippet from Frontend which handles file upload

```

const user = await User.findOne({ email: req.body.email });
const file = {
  fileName: req.body.fileName,
  enc_original_name: req.body.enc_original_name,
};
console.log(file);
user.files.push(file);
await user.save();
res.status(200).json({ message: 'File uploaded successfully' });
});

```

Code snippet from backend which captures file received from client and stores it



### 3.4.6 Download

```
const Downloads = ({navigation}) => {
  const [files, setFiles] = useState([]);

  React.useEffect(() => {
    RNFS.readDir('/storage/emulated/0/Android/media/com.lockstash_app/')
      .then(result => {
        setFiles(result.map(file => ({ name: file.name, uri: file.path })));
      })
      .catch(err => {
        setErrorMessage(err.message);
      });
  });
}, []);
```

```
router.get('/download', (req, res) => {
  const path = `../uploads/${req.query.email}/${req.query.fileName}`;
  res.sendFile(p.join(__dirname, path));
});
```

Code snippet from frontend which handles file downloading and saving

## 3.5 KEY CHALLENGES

Our main challenge was to prevent the data loss even if the storage gets into wrong hands. Storing password in the database even after hashing it is not secure. One can use rainbow table attack to get all the easy passwords. Hence, we decided to use industry standard SRP protocol which is used by companies like Proton AG. Since password is not saved anywhere, it makes it really hard to compromise accounts.

To prevent storage of encryption key, we never save it on any device and it is only used temporarily until user is logged in. Refreshing the page would lead to loss of this key and one would need to relogin to reaccess their data. Also, even after encrypting a file, we need to remove their names otherwise one can guess the contents based on the file name. We used a hashmap to link old names to new ones and encrypted old names so only original owners can find them

# CHAPTER 4: TESTING


## 4.1 TESTING STRATEGY

We checked account creation and validated passwords, checked if media password is incorrect then do we get access without permission, checked uploading of different kinds of media to the server.

We also compared the integrity of the files after encrypting and uploading and then downloading and decrypting from the server with the original files.

## 4.2 TEST CASES AND OUTCOMES

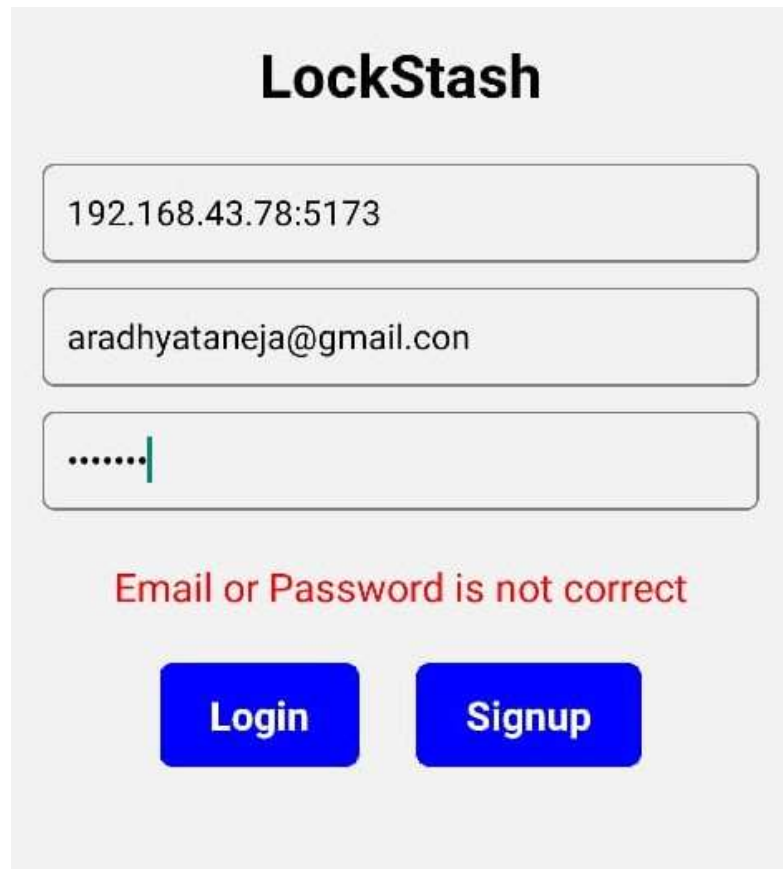
### 4.2.1 ACCOUNT CREATION



The screenshot shows a web interface for 'LockStash'. At the top, the title 'LockStash' is displayed in bold black text. Below the title are three input fields: the first contains the IP address '192.168.43.78:5173', the second contains the email address 'aradhyataneja@gmail.com', and the third contains a masked password '.....'. Below the input fields, a green message reads 'User created successfully'. At the bottom, there are two blue buttons: 'Login' and 'Signup'.

Figure 12: User was created successful

## 4.2.2 VALIDATING PASSWORD AND EMAIL



The image shows a login form for 'LockStash'. It has three input fields: the first contains the IP address '192.168.43.78:5173', the second contains the email 'aradhyataneja@gmail.com', and the third contains a password represented by seven dots. Below the fields is a red error message: 'Email or Password is not correct'. At the bottom are two blue buttons labeled 'Login' and 'Signup'.

Figure 13: Providing incorrect login information will generate an error



4.2.3 ADDING MEDIA PASSWORD

The screenshot shows a mobile application interface for adding a media password. At the top left is a back arrow. To its right is the title 'Add Media Password' and a black 'LOGOUT' button. Below this is a light gray container with two password input fields. The first field contains seven dots. The second field contains seven dots and a vertical cursor. At the bottom center of the container is a blue 'SUBMIT' button.

Figure 14: Creation of media password

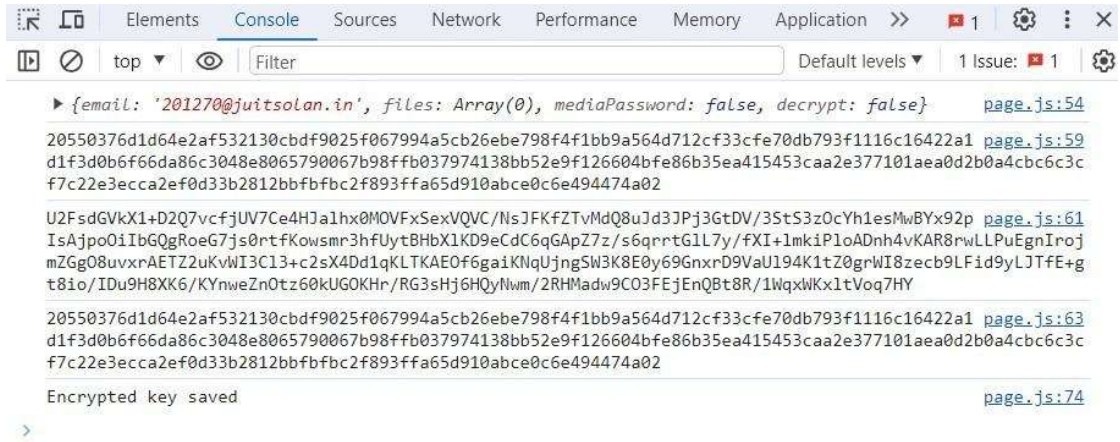


Figure 15 : Debug information showing us original AES key and its encrypted version using media password

#### 4.2.4 CHECKING USER DETAILS IN DATABASE



Figure 16: User details in database are encrypted except email which is used for identification

## 4.2.5 ENCRYPTING AND UPLOADING FILES OF DIFFERENT TYPES

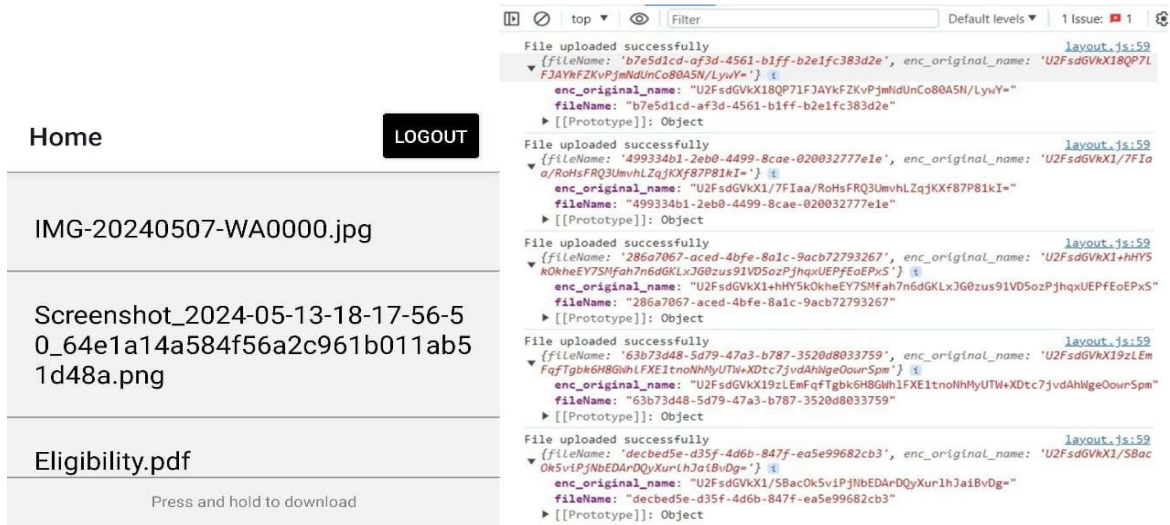


Figure 17: Files which we uploaded and their encrypted details on the right hand side

## 4.2.6 FILE DETAILS STORED IN OUR DATABASE IN ENCRYPTED FORMAT

```

_id: ObjectId('66420888dfc133795e6d6b0c')
email: "aradhyataneja@gmail.com"
salt: "fad703dbe113f38d014f75e7355e824ae2fd1166781b80b328f33b200c1466f0"
verifier: "11cb1a4d2e0fabfc769c6d3fcd58e669687edf96b7e665a74a1a576c3705f6cb4e7d1..."
files: Array (2)
  0: Object
    fileName: "7f7ec263-9997-4600-950c-51540f5ffa2a"
    enc_original_name: "U2FsdGVkX1/300vWSS0IwPTdDUBuzwE5S29kT/MK6dXh0IJZrNcVdQ6YoCGKgP/w"
  1: Object
    fileName: "6d10ba8a-5a38-4605-b121-1505eed88c54"
    enc_original_name: "U2FsdGVkX1+dUogp+0vIQMoxMVihN0Iu9p0AcfPwjR8/oG3s22+cA11UWF3YZaHaUxD+1B..."
mediaPassword: true
__v: 2
tempEphemeral: "5737d166fcc6860dc1fcb6ec0c639b13ce6db15877664ca0fe73e7e3c7cbd935"
encryptedKey: "U2FsdGVkX19n4HNvgq2QGikuEko2ufQaGhm06CXL6yaAXtcqk+3m/wvgUoUT65E3D6fDU..."

```

Figure 19: File details are stored in an array in our database in encrypted format

## 4.2.7 MULTIPLE USERS

```

▶ _id: ObjectId('6568f3e85e8deaab5505be79')
  email: "201270@juitsolan.in"
  salt: "b2bc9494f55d79c778e34e434c3cf5d082e08f10ec92336ac6fa62f7257d2ffff"
  verifier: "7ecc38ee26da1870c0cab16c593bf044863f467ae8e3d3f9afde79b30becb3a712a921..."
  ▶ files: Array (5)
  mediaPassword: true
  __v: 5
  tempEphemeral: "d811e365bc25028546d0539497ca2b126a1062c3966394b99274c796bc088ba7"
  encryptedKey: "U2FsdGVkX1+D2Q7vcfjUV7Ce4HJaLhx0MOVfXSexVQVC/NsJFKfZTvMdQ8uJd3JPj3GtDV..."

_id: ObjectId('6568f5095e8deaab5505be90')
  email: "202020@gmail.com"
  salt: "5894252c4d0bfff6b52b59b820c391937273f3d3e77115f758bda04fe920f95e7"
  verifier: "9077396cfbb3c512cb39165597331484375cd284ae5f3583c800a6ced5f1c545be8d07..."
  ▶ files: Array (empty)
  mediaPassword: true
  __v: 0
  tempEphemeral: "64a8c7fb5a579be43b8a2913bd674c42815ba632e33f4694ef565535db92e92b"
  encryptedKey: "U2FsdGVkX1/cwkVxeFMPxNEzxMMs106Bq4psOhv8Lx4bqOwQYknh/Zn42r5ahhGuE/LR2y..."

```

Figure 20: Data of each user is stored separately and encrypted

## 4.2.8 FILE INTEGRITY CHECK

File Name	File Integrity Comparison
biomedical.docx - 56528 bytes	<b>ORIGINAL FILE</b>
MD5: e40fda40cf6c1561eb0d9bcb917edb40 SHA1: 4c47185ccb19da99358426ff455a4f88772693a8 SHA256: defba911aa0c94378f41b4775a68fd76ae79c4c1a983ac7e6cc19eb62900c861 SHA512: 82315c3c6951dd658531cd766b891d6488b3d23e06d97ee0b8977cbb72de404ae59128c64be7f8ccdf6f6e464c9989abfdaa68b6036702103c601737eb2f04c1	
201270 Aradhya-Taneja.pdf - 592884 bytes	<b>ORIGINAL FILE</b>
MD5: e7ecc9bc75e938395bae84cdc1e4a15c SHA1: 4298e1ba80c44dbbcb239b794030bea88dd00b46 SHA256: 3d8bfb609e048f73814eba6639a41e5c341c8ab453f23ef9add652b87770dacb SHA512: b6e9c3375e99cd99abd401a6aa8d80de45c26e860211ac570aaf75b1f2e457d4f8e77f471aa6512fd11b77da59c45f90d3df1fedf68ba4698cc811cae9b95d19	
biomedical.docx - 56528 bytes	<b>Decrypted File from LockStash</b>
MD5: e40fda40cf6c1561eb0d9bcb917edb40 SHA1: 4c47185ccb19da99358426ff455a4f88772693a8 SHA256: defba911aa0c94378f41b4775a68fd76ae79c4c1a983ac7e6cc19eb62900c861 SHA512: 82315c3c6951dd658531cd766b891d6488b3d23e06d97ee0b8977cbb72de404ae59128c64be7f8ccdf6f6e464c9989abfdaa68b6036702103c601737eb2f04c1	
201270 Aradhya-Taneja.pdf - 592884 bytes	<b>Decrypted File from LockStash</b>
MD5: e7ecc9bc75e938395bae84cdc1e4a15c SHA1: 4298e1ba80c44dbbcb239b794030bea88dd00b46 SHA256: 3d8bfb609e048f73814eba6639a41e5c341c8ab453f23ef9add652b87770dacb SHA512: b6e9c3375e99cd99abd401a6aa8d80de45c26e860211ac570aaf75b1f2e457d4f8e77f471aa6512fd11b77da59c45f90d3df1fedf68ba4698cc811cae9b95d19	

Figure 21: File integrity check showing our files are identical to originals

# CHAPTER 5: RESULTS AND EVALUATION

## 5.1 RESULTS

We successfully implemented self-hosted server which comprises of a secure system which supports passwordless login mechanism using Secure Remote Password Protocol using PBKDF2 private key generation, also we were successful in encrypting 5 different types of files using AES encryption and then we were able to download and decrypt them.

We were able to separate the media of different users and keep their details encrypted.

The files we downloaded from our server were identical to the original ones. We also established this by doing a file integrity check which comprised of several kinds of hashes and all were found to be equal for same files.

We also successfully created an android app and tested all the above things successfully

# **CHAPTER 6: CONCLUSIONS AND FUTURE SCOPE**

## **6.1 CONCLUSION**

Safe and secure storage system is the main issue which our project sought to overcome. The strategy was simple, that is creating a secure storage system where everything from signup, login to upload and download is secured. Also, if the database gets compromised then there must be a system which ensure that the attacker never gets the contents of the file present. We decided to encrypt even file names so they won't be even able to identify which file is which. To achieve this we have used symmetric and asymmetric cryptography. We have used SRP protocol (secure remote password protocol), in addition to this we have used PBKDF2 (password based key derivation function) which is a key derivation mechanism to help in generating secure keys. All of these methods helped us in creating our secure storage system.

We even created an android app which makes it easier for users to upload files to their own servers.

We have also provided our backend which anyone can host and use personally.

## **6.2 FUTURE SCOPE**

Future scope can be to optimise the android application and to integrate the cloud services like Azure, GCP, AWS, etc for ease of hosting the server, saving encrypted files and getting easier access.

## REFERENCES

- [1] Pronika and S. S. Tyagi, "Secure Data Storage in Cloud using Encryption Algorithm," 2021 Third International Conference on Intelligent Communication Technologies and Virtual Mobile Networks (ICICV), Tirunelveli, India, 2021, pp. 136-141, doi: 10.1109/ICICV50876.2021.9388388.
- [2] S. Kiran, R. P. K. Reddy and N. Subramanyan, "A novel crypto system with key entrenched cipher," 2017 4th International Conference on Advanced Computing and Communication Systems (ICACCS), Coimbatore, India, 2017, pp. 1-4, doi: 10.1109/ICACCS.2017.8014597.
- [3] M. B. Yassein, S. Aljawarneh, E. Qawasmeh, W. Mardini and Y. Khamayseh, "Comprehensive study of symmetric key and asymmetric key encryption algorithms," 2017 International Conference on Engineering and Technology (ICET), Antalya, Turkey, 2017, pp. 1-7, doi: 10.1109/ICEngTechnol.2017.8308215.
- [4] Singh, Prabhsimran & Kaur, Kuljit. (2015). Database security using encryption. 2015 1st International Conference on Futuristic Trends in Computational Analysis and Knowledge Management, ABLAZE 2015. 353-358. 10.1109/ABLAZE.2015.7155019.
- [5] T. Saxena and V. Chourey, "A survey paper on cloud security issues and challenges," 2014 Conference on IT in Business, Industry and Government (CSIBIG), Indore, India, 2014, pp. 1-5, doi: 10.1109/CSIBIG.2014.7056957.
- [6] Kioon, Mary & Wang, ZhaoShun & Das, Shubra. (2013). Security Analysis of MD5 Algorithm in Password Storage. Applied Mechanics and Materials. 347-350. 10.2991/isccca.2013.177.

- [7] Shen, Zhirong & Xue, Wei & Fu, Yingxun. (2013). Secure storage system and key technologies. 376-383. 10.1109/ASPDAC.2013.6509625.
- [8] S. Nepal, C. Friedrich, L. Henry and S. Chen, "A Secure Storage Service in the Hybrid Cloud," 2011 Fourth IEEE International Conference on Utility and Cloud Computing, Melbourne, VIC, Australia, 2011, pp. 334-335, doi: 10.1109/UCC.2011.55.
- [9] Z. Yong-Xia and Z. Ge, "MD5 Research," 2010 Second International Conference on Multimedia and Information Technology, Kaifeng, China, 2010, pp. 271-273, doi: 10.1109/MMIT.2010.186.
- [10] X. -h. Wu and X. -j. Ming, "Research of the Database Encryption Technique Based on Hybrid Cryptography," 2010 International Symposium on Computational Intelligence and Design, Hangzhou, China, 2010, pp. 68-71, doi: 10.1109/ISCID.2010.105.
- [11] P. Zeng, Y. Shen, Z. Qiu, Z. Qiu and M. Guo, "SRP: A routing protocol for data center networks," The 16th Asia-Pacific Network Operations and Management Symposium, Hsinchu, Taiwan, 2014, pp. 1-6, doi: 10.1109/APNOMS.2014.6996564.
- [12] Moradi, Fereidoun & Mala, Hamid & Ladani, Behrouz. (2015). Cryptanalysis and strengthening of SRP+ protocol. 91-97. 10.1109/ISCISC.2015.7387904.
- [13] Xiurong Chen, Xiaochao Li, Yihui Chen, Pengtao Li, Jianli Xing and Lin Li, "A modified PBKDF2-based MAC scheme XKDF," TENCON 2015 - 2015 IEEE Region 10 Conference, Macao, 2015, pp. 1-6, doi: 10.1109/TENCON.2015.7373109
- [14] J. Herrera and M. L. Ali, "Concerns and Security for Hashing Passwords," 2018 9th IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON), New York, NY, USA, 2018, pp. 861-865, doi: 10.1109/UEMCON.2018.8796720.



- [15] P. Bharathi, G. Annam, J. B. Kandi, V. K. Duggana and A. T., "Secure File Storage using Hybrid Cryptography," 2021 6th International Conference on Communication and Electronics Systems (ICCES), Coimbatre, India, 2021, pp. 1-6, doi: 10.1109/ICCES51350.2021.9489026
- [16] T. M. Zaw, M. Thant and S. V. Bezzateev, "Database Security with AES Encryption, Elliptic Curve Encryption and Signature," 2019 Wave Electronics and its Application in Information and Telecommunication Systems (WECONF), St. Petersburg, Russia, 2019, pp. 1-6, doi: 10.1109/WECONF.2019.8840125.
- [17] G. Raj, R. C. Kesireddi and S. Gupta, "Enhancement of security mechanism for confidential data using AES-128, 192 and 256bit encryption in cloud," 2015 1st International Conference on Next Generation Computing Technologies (NGCT), Dehradun, India, 2015, pp. 374-378, doi: 10.1109/NGCT.2015.7375144.
- [18] A. Mousa, M. Karabatak and T. Mustafa, "Database Security Threats and Challenges," 2020 8th International Symposium on Digital Forensics and Security (ISDFS), Beirut, Lebanon, 2020, pp. 1-5, doi: 10.1109/ISDFS49300.2020.9116436.
- [19] Yuan, X., Chen, Y. Secure routing protocol based on dynamic reputation and load balancing in wireless mesh networks. *J Cloud Comp* 11, 77 (2022). <https://doi.org/10.1186/s13677-022-00346-x>
- [20] K. S. Charan, H. V. Nakkina and B. R. Chandavarkar, "Generation of Symmetric Key Using Randomness of Hash Function," 2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT), Kharagpur, India, 2020, pp. 1-7, doi: 10.1109/ICCCNT49239.2020.9225280.
- [21] A. Vichare, T. Jose, J. Tiwari and U. Yadav, "Data security using authenticated encryption and decryption algorithm for Android phones," 2017 International Conference on Computing, Communication and Automation (ICCCA), Greater Noida, India, 2017, pp. 789-794, doi: 10.1109/CCAA.2017.8229903.

- [22] N. A. Fauziah, E. H. Rachmawanto, D. R. I. Moses Setiadi and C. A. Sari, "Design and Implementation of AES and SHA-256 Cryptography for Securing Multimedia File over Android Chat Application," 2018 International Seminar on Research of Information Technology and Intelligent Systems (ISRITI), Yogyakarta, Indonesia, 2018, pp. 146-151, doi: 10.1109/ISRITI.2018.8864485.
- [23] M. Alkhyeli, S. Alkhyeli, K. Aldhaheeri and H. Lamaazi, "Secure Chat Room Application Using AES-GCM Encryption and SHA-256," 2023 15th International Conference on Innovations in Information Technology (IIT), Al Ain, United Arab Emirates, 2023, pp. 180-185, doi: 10.1109/IIT59782.2023.10366418.
- [24] I. Tariq, M. Nazish, S. Ashaq, I. Sultan and M. T. Bandy, "A Performance Comparison of Hashed and Authenticated Advanced Encryption Standard," 2022 Smart Technologies, Communication and Robotics (STCR), Sathyamangalam, India, 2022, pp. 1-5, doi: 10.1109/STCR55312.2022.10009112.
- [25] B. B. Ahamed and M. Krishnamoorthy, "Fraction of Data in a Secure Client-Server Communication Method Using Key Exchange," 2022 International Conference on Computing, Communication, Security and Intelligent Systems (IC3SIS), Kochi, India, 2022, pp. 1-4, doi: 10.1109/IC3SIS54991.2022.9885557