TITLE OF PROJECT REPORT

A PROJECT REPORT

Submitted by

Shreyansh Saha

(23BCY10217)

in partial fulfillment for the award of the degree
of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING
(Cyber Security and Digital Forensics)

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING

VIT BHOPAL UNIVERSITY

KOTHRIKALAN, SEHORE
MADHYA PRADESH – 466114

FEB – APRIL 2025

A PROPOSED DESIGN AND IMPLEMENTATION OF IOT BASED SMART WATCH FOR MONITORING APP

A PROJECT REPORT

Submitted by

Shreyansh Saha (23BCY10217)

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

(Cyber Security and Digital Forensics)

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING

VIT BHOPAL UNIVERSITY

KOTHRIKALAN, SEHORE
MADHYA PRADESH - 466114

APRIL 2025

VIT BHOPAL UNIVERSITY, KOTHRIKALAN, SEHORE
MADHYA PRADESH – 466114

BONAFIDE CERTIFICATE

Certified that this project report titled **"SECURE FILE TRANSFER"** is the bonafide work of "Shreyansh Saha **(23BCY10217)"** who carried out the project work under my supervision. Certified further that to the best of my knowledge the work reported at this time does not form part of any other project/research work based on which a degree or award was conferred on an earlier occasion on this or any other candidate.

PROGRAM CHAIR	**PROJECT SUPERVISOR**
Dr. D. Saravanan	Dr. Irfan Alam
Assistant Professor Sr.	Assistant Professor
School of Computing Science and Engineering,	School of Computing Science and Engineering,
VIT Bhopal University.	VIT Bhopal University.

The Capstone Project Examination is held on _______________

ACKNOWLEDGEMENT

First and foremost I would like to thank the Lord Almighty for His presence and immense blessings throughout the project work.

I wish to express my heartfelt gratitude to **Dr D. Saravanan**, Program Chair, Cyber Security and Digital Forensics for much of his valuable support encouragement in carrying out this work.

I would like to thank my internal guide Dr. Irfan Alam for continually guiding and actively participating in my project, giving valuable suggestions to complete the project work.

I would like to thank all the technical and teaching staff of the School Computing Science and Engineering, who extended directly or indirectly all support.

Last, but not least, I am deeply indebted to my parents who have been the greatest support while I worked day and night for the project to make it a success.

ABSTRACT

The goal of SpassShare is to securely share files between two clients using password as the method of protection. In today's world, various secure file sharing solutions exist, but they are usually very complex and are suited for transfer of highly secure and confidential files. Despite many of them being open source, their working mechanism is often very complicated to be understood or modified. This can be overwhelming for users who just want to transfer simple files quickly and safely.

Thus, for sharing simple files, SpassShare can be used which aims to be very modular, customisable and completely open source. The working mechanism of SpassShare is very simple yet effective for simple and small file transfer, and thus the user can have confident that the software doesn't do anything shady or hidden. Everything in the software is transparent, and users can freely inspect the code and see exactly how it works.

SpassShare doesn't try to do too much. It only focuses on what is required—secure, password-based file sharing without unnecessary complexity. It is ideal for people who want a clean and straightforward way to send files without diving deep into complex setups or configurations. It is meant to be easy to use, easy to understand, and reliable for the use cases it is made for.

TABLE OF CONTENTS

CHAPTER 1
PROJECT INTRODUCTION

1.1 Introduction

We already have many secure file sharing solutions in today's world, but most of them are closed source and some, despite being open source, are very complex to understand and modify as per the use case. These solutions might be beneficial for sharing extremely important and confidential files, where the protection of the files is much more important and it outweighs other tradeoffs like ease of use or simplicity. These tools are often built with heavy encryption, strict protocols and complicated mechanisms that make them suitable for professional or enterprise-level scenarios.

But for sharing simple files over a network where so much protection isn't desired, these tradeoffs of today's software often become too much sometimes. The user ends up spending more time figuring out how to use the software rather than actually sharing the file. In these cases, the complexity becomes a hurdle rather than a feature.

Hence, SpassShare aims to be a very simple implementation of password-based file sharing software. It is not trying to compete with high-end secure solutions, rather it is made for the user who just wants to send a file safely and quickly with minimal setup. It has a very simple and modular code base which can be understood by any user with enough coding experience and can be modified as per the needs. The whole idea is to make things as clear and straightforward as possible, so users know exactly what the software is doing at every step. It is both open source and very accessible, making it a perfect choice for developers or users who value transparency, simplicity and control.

1.2 Motivation for the work

The sole motivation for SpassShare was to create a software which is simple and can be understood and modified by the user in a world where only more and more complex and closed source tech is being developed. Today, most new tools and software are becoming harder to understand, harder to trust, and harder to modify. Even the open source ones are often so large and filled with unnecessary features that going through their code becomes a task in itself. This makes it very difficult for an average user or even a developer to fully know what the software is doing in the background.

SpassShare was born out of the idea to go the opposite way. Instead of adding more complexity, it removes everything unnecessary and keeps only what is needed. It is built to give the user full clarity and control over the file sharing process, with every part of the software being visible, editable, and understandable. It’s for people who don’t want to rely on black-box tools for doing something as basic as sending a file. It is meant to bring back the kind of simplicity and openness that’s slowly disappearing from modern software.

1.3 Problem Statement

Create a simple file sharing app which uses a simple protection mechanism and the code base is open source and simple enough to be understood and modified by the user. In today’s world, most file sharing apps are either too complex or come with closed source code, making it hard for users to know what is happening behind the scenes. Even the ones that are open source often have bloated code bases which are not easy to go through or change according to one’s needs.

The goal here is not to build a highly advanced or enterprise-grade tool, but to build something that just works for basic file sharing with minimal setup and effort. The protection mechanism should be simple, like password-based access, which is enough for regular use cases where military-grade security is not needed. At the same time, the code should be clean, well-structured and fully open so that anyone with a little bit of coding experience can read, understand, and modify it if required.

This app should focus more on simplicity, transparency and ease of use rather than complex features. The end goal is to provide users with a lightweight tool they can trust, tweak and use without going through a steep learning curve.

1.4 Summary

SpassShare is a simple password-protected file sharing web app that is lightweight, easily deployable, and built with a clean and modular codebase. It is designed to offer basic file sharing with just enough protection for everyday use, without any of the heavy setup or complexity seen in other tools. The code is open source, easy to understand, and can be modified as needed without any performance tradeoffs. SpassShare focuses on keeping things minimal, transparent, and fully in the user’s control.

CHAPTER 2

RELATED WORK INVESTIGATION

2.1 Introduction

Unlike SpassShare various others file sharing softwares exists, but they arent very simple, some are proprietery and some despite being open source are very complex to understand and customise. However they do have very important use case, which is sharing highly confedential files where the users might want more robust protection techniques than just using passwords to protect their data.

2.2 Existing password protected file sharing tools

2.2.1 Cryptomator

The Android-based Cryptomator library is an open source application with client-side encryption feature that provides encryption services from the client side to secure files stored on public cloud services such as Dropbox, Google Drive, One Drive, iCloud Drive, and WebDAV [1]. The Cryptomator application was initiated by Skymatic in February 2014 and its Android version has progressed to the Beta version until Cryptomator 1.0 was released on October 17, 2017 [2]. The Cryptomator application receives the CeBIT Innovation Award 2016 for Usable Security and Privacy [2]. The Cryptomator application provides an open source library for cryptomator's developer so that it can be developed continuously to detect backdoor or vulnerability [1]. The lopen source library using an AES-256 algorithm as an encryption algorithm to provide confidentiality services [3]. Smartphone technology has enabled the growth of a variety of sophisticated Android-based applications [4]. Based on a survey from statista.com in 2018, that Android- based applications in the period December 2009 to December 2017 experienced an increase of up to 3.5 billion applications in the Google Play Store. This proves that the application made adjust to the existing smartphone hardware. However, smartphone applications are still constrained in terms of power consumption, calculation speed, code size, and wireless network bandwidth [4] [5]. The implementation of algorithms carried out on the Android operating system contains a more efficient cryptographic algorithm suchas TEA and XTEA [6]. For 64- bit plaintext size, TEA and XTEA reach speeds of 40µs for encryption and decryption processes [6]. Based on the code size and cycle count, the TEA algorithm in the first ranks for the best performance of other algorithms such as XTEA, DESXL, Noekeon, Klein, then AES [7]. While based on the cycle/byte metric, the best-performing algorithm is TEA, XTEA, Noekeon, AES, DESXL, then Klein [7]. In terms of algorithms that require the lowest code size, the best- performing algorithms are, TEA, XTEA, Twine, and LED [7]. As for algorithms that require less RAM, namely, XTEA, Lblock, TEA, and MIBS [7]. Overall TEA and XTEA occupy the best performance in terms of memory metrics [7]. Another reference states that for throughput and speed, the TEA and XTEA algorithms rank first with good performance on a variety of platforms, followed by AES and Klein [8]. The TEA algorithm appropriate if we need good speed performance and light memory[9]. Based on the above background, research will be conducted on comparing of the performance of the implementation of the TEA and AES algorithms on the Android-based cryptomator open source library. To find out the comparison of the performance of the implementation of the TEA and AES algorithms in the open source library, a cryptomator application was developed to implement the Android-based cryptomator open source library.

2.2.2 Tresorit

Tresorit was founded in 2011 as a provider mainly geared towards businesses. Currently, the Swiss Post holds a major- ity stake in the company. Tresorit has released a whitepaper containing the technical details of their protocol [11]

Tresorit uses zero-knowledge end-to-end encrypted platform and one of their product is file sharing. End-to-end-encrypted file transfer is the most secure way to exchange files and documents.

Using client-side end-to-end encryption coupled with zero-knowledge authentication, Tresorit encrypts every file and relevant metadata on your devices with unique, randomly generated encryption keys. These keys are never sent to the servers unencrypted. This means only data owners and any intended recipients can access the data stored in Tresorit, allowing you to share files with trusted recipients in full confidence. [12]

2.3 How SpassShare compares to Cryptomator and Tresorit

SpassShare, Cryptomator, and Tresorit all provide ways to share and protect files, but they differ significantly in their approach to security. SpassShare uses password-based protection, where files are stored in their original form on the server and only the password is hashed and stored in the database. This makes it easier for users to access and share files but leaves the files vulnerable if the server or file system is compromised. In contrast, Cryptomator employs client-side encryption, where files are encrypted on the user's device before being uploaded, using AES-256 encryption. Both the content and filenames are encrypted, ensuring the file remains secure even if third parties or cloud providers access the storage. Similarly, Tresorit provides end-to-end encryption using AES-256, ensuring files are encrypted on the user's device with unique keys and random IVs. Unlike SpassShare, Tresorit employs zero-knowledge encryption, meaning even their own servers cannot access your files. The key difference between Cryptomator and Tresorit is that Cryptomator uses a vault system—an encrypted folder where files are stored, while Tresorit does not use this concept but provides similar security through its encryption system. Overall, while SpassShare offers basic password protection, both Cryptomator and Tresorit provide far stronger security through client-side encryption, with Cryptomator offering a vault system and Tresorit ensuring zero-knowledge encryption.

2.4 Related works

- Library Open Source Develop Cryptomator Android-based cryptomator open source library is an open core project library that is applied on the Android cryptomator application. The library build so it can be developed to add performance from the application [2].
- Chaudhry et al. Authenticated Encryption Scheme Tiny Encryption Algorithm (TEA) is an algorithm created by David J. Wheeler and Roger M. Needham from Cambridge University in 1994. TEA is a block cipher algorithm designed for minimal memory usage and a maximum speed of encryption. TEA uses the Feistel network structure with 64-bit blocks and 128-bit key inputs. TEA stores 64-bit inputs into L0 and R0 respectively 32-bit, while 128-bit keys are stored into k[0], k[1], k[2], and k[3] which each contain 32-bit
- Comparative research is a research method used to compare two or three events by looking at the causes [10]. Further information regarding comparative research according to Sugiyono (2014) is a study to compare the state of one or more variables in two or more different samples, or two different times.

2.5 Other secure file transferring techniques

1. Cloud-based Secure File Sharing Solution
2. Secure File Transfer Protocol (SFTP)
3. Using email services like Gmail

2.6 Pros and cons of SpassShare

2.6.1 Pros

- **Extremely lightweight and readily deployable:** Quick to set up and easy to run with minimal system requirements.
- **Completely open source:** Fully transparent code that anyone can inspect, modify, and contribute to.
- **Codebase is simple and modular enough to be understood and modified:** Easy for developers to customize and extend the software without difficulty.
- **Can act as a base software and can be extended with more security features easily:** Provides a foundation that can be built upon for added functionality.
- **Password is hashed using bcrypt, a mature hashing algorithm that can't be cracked easily:** Utilizes a secure, widely trusted hashing algorithm to protect user passwords.
- **No unnecessary features:** Focuses purely on basic file sharing, making it clean and free of bloat.
- **Local deployment option:** Can be deployed on any server or local machine, giving users full control.
- **Low resource consumption:** The app is designed to use minimal server and client resources, making it efficient.
- **Customizable UI:** The user interface is simple and can be easily adjusted to fit the user's preferences.
- **Minimal learning curve:** Even users with basic technical knowledge can understand and use the app effectively.

2.6.2 Cons

- **It is not recommended to use SpassShare for sharing extremely important or highly confidential files:** Best suited for low-risk file sharing, not for high-stakes data.
- **The only security mechanism is password, which can be guessed by an attacker:** Passwords are the only line of defense, making it vulnerable to weak or common passwords.
- **No built-in encryption for files:** Files are not encrypted, so they can be accessed by anyone who has the correct password.
- **Limited security features:** Lacks advanced security mechanisms like multi-factor authentication or encryption protocols.

- **No file expiration or auto-delete mechanism:** Once files are uploaded, they remain available unless manually deleted.
- **No audit trail or logging:** There's no built-in way to track who accessed or downloaded the files.
- **Vulnerable to brute-force attacks if weak passwords are used:** Without additional protection, weak passwords can be cracked over time.
- **Requires manual updates for improvements:** Users need to manually update the codebase if they want new features or fixes.
- **No centralized file management:** Each file is managed individually, and there’s no system for managing multiple uploads or user accounts

CHAPTER 3
METHADOLOGY

SpassShare is developed using the MERN stack (MongoDB, Express.js, React, Node.js) along with bcrypt for hashing the passwords. The files that are uploaded by users are stored on the server in a designated folder, and the metadata such as the file name, file path, and optional password hash are stored in MongoDB.

Users can upload files through the application and have the option to set a password for added security. If a password is provided, it is hashed using bcrypt before it is stored in the database. When someone tries to access the file, the application checks if password protection is enabled. If a password is set, the user must enter the correct password, which is verified using bcrypt's hashing algorithm.

The app also tracks the download count for each file. This helps in monitoring file usage and adds an additional layer of tracking. To ensure unauthorized access is prevented, the application uses server-side validation at every step.

Express.js middleware (specifically multer) is used to handle file uploads and request parsing. When the user downloads a file, the app serves the file via res.download(), a built-in function in Express.js that makes the file available for download after validation and checks have been completed.

This setup ensures a straightforward yet effective mechanism for securely sharing files while keeping the codebase clean, modular, and easy to manage.

CHAPTER 4
REQUIREMENTS AND IMPLEMENTATION

4.1. Requirements

1. **Node.js** and **npm** (runtime environment)
2. **MongoDB** (for storing file info and password hashes)
3. **Express.js** (backend web framework)
4. **Multer** (for handling file uploads)
5. **bcrypt** (for password hashing)
6. **EJS** (for rendering frontend pages)
7. Basic knowledge of HTML and CSS (for frontend)
8. A .env file with:
    - 8.1 DATABASE_URL (MongoDB connection string)
    - 8.2 PORT (port number, e.g. 3000)

4.2. Implementation

1. Set up a basic **Express.js** server in app.js
2. Connected to **MongoDB** using mongoose and a connection string from .env
3. Created a **Mongoose model** (File.js) to store:
    - 3.1 File path
    - 3.2 Original file name
    - 3.3 Optional password (hashed)
    - 3.4 Download count
4. Used **Multer** to handle file uploads:
    - 4.1 Files are stored in a local uploads/ folder
    - 4.2 Each file is given a randomly generated unique name automatically
5. Created an **upload route** (/upload):
    - 5.1 Handled single file upload via form input named "file"
    - 5.2 If user enters a password, hashed it with **bcrypt**
    - 5.3 Saved file data and password hash (if any) to the database
    - 5.4 Rendered a unique file link on the homepage
6. Created a **download route** (/file/:id) using .route().get().post():

6.1 Looked up the file in the database using its ID

6.2 If the file has a password:

6.3 Rendered a password input form (GET)

6.4 Verified the entered password with **bcrypt.compare()** (POST)

6.5 f the password is correct (or not required):

6.6 Increased the file's download count

6.7 Sent the file using res.download()

7. Used **EJS** templates for frontend rendering:

    7.1 index.ejs – upload form and file link

    7.2 password.ejs – password prompt if the file is protected

8. Handled form data using express.urlencoded() middleware
9. Served static CSS files using express.static('public')

CHAPTER 5
CONCLUSION

5.1 Conclusion

This project successfully implements a basic yet secure file-sharing web app that uses password protection as the primary security mechanism. Users can easily upload files and have the option to set a password to restrict access, ensuring that only authorized individuals can download the files. The application is built with simplicity in mind, ensuring it remains lightweight and efficient for everyday use.

Additionally, SpassShare is fully open-source, making it transparent and accessible to developers. The clean and modular codebase allows users to understand how the system works and modify or extend it according to their specific needs. Whether for personal use or as a foundation for more advanced projects, SpassShare provides a flexible and secure file-sharing solution that can be tailored to a variety of use cases.

5.2 Future Enhancements

In the future, the password protection system in SpassShare can be replaced with **JWT (JSON Web Tokens)** for enhanced security and flexibility. JWTs provide a more scalable and secure way of handling user authentication and access control. Instead of requiring users to enter a password every time they access a file, users will authenticate once and receive a token.

This token can then be used to access the file securely, without the need for repeated password entries. By using JWTs, we can create a smoother user experience while reducing the risk of weak password usage or password fatigue. Furthermore, JWT-based authentication can be more efficient in large-scale systems where multiple users need access to various files. The system can be extended to include features such as token expiration, refresh tokens, and more robust access management, allowing for a more seamless and secure file-sharing experience.

REFERENCES

[1] Akerlund, Geoff. 2016. Cryptomator Is an Open-Source App to Encrypt Your Cloud Files. Available at: http://www.backupreview.com/cryptomator-open-source-app-to- encrypt-your-cloud-files/ [Accessed October 29, 2017]

[2] Skymatic. 2017. Security Architecture Cryptomator. Available at : https://cryptomator.org/security/architecture/ [Accessed Januari 19, 2018]

[3] Kreusch, Markus, dan Christian Schmickler. 2017. Privacy with Cryptomator End to End Cloud Encryption – User Friendly and Open Source. Skymatic. Android Security Symposium 2017. Vienna. Austria.

[4] Hung, Shih-Hao, Chi-Seng Shih, Jeng-Peng Shieh, Cheen-Pang Lee, Yi-Hsiang Huang. 2012. Executing Mobile Applications on The Cloud: Framework and Issues. Department of Computer Science and Information Engineering, National Taiwan University, Taipei 106, Taiwan.

[5] B.-G. Chun, P. Maniatis, Augmented smartphone applications through clone cloud execution, in: Proceedings of the 12th Workshop on Hot Topics in Operating Systems, USENIX Association, 2009, pp. 8–8.

[6] Malina,Lukas,ClupekVlastimil,MartinasekZdenek,HajnyJan,Oguchi Kimio, Zeman Vaclav.Evaluation of software oriented block ciphers on smartphones. Foundations and practice of security. Berlin: Springer International Publishing; 353–68.

[7] J. Mohd, Bassamet. et.al.2015. A Survey For Lightweight Block CipherFor Low- Resource Devices: Comparatif Study And Open Issues. Journal Network And Computer Applications.

[8] Andem, Vikram Reddy. 2003. A Cryptanalysis Of The Tiny Encryption Algorithm [Thesis]. University Of Albama : Departement Of Computer Science. Cazorla M, Marquet K, and Minier M. Survey and benchmark of lightweight block ciphers for wireless sensor networks. In: Proceedings of the SECRYPT; 2013.

[9] Dennis, A., Wixom, B. H. & Roth, R. M., 2012. System Analysis and Design, Fifth Edition. USA: John Wiley & Sons, Inc.

[10] Dennis, A., Wixom, B.H. & Tegarden, D., 2015. Systems analysis and design: An object-oriented approach with UML, Wiley

[11] Tresorit AG. 2024. Tresorit - Encryption Whitepaper. https://tresorit.com/ resources.

[12] Web access - tresorit. (n.d.). Tresorit. https://web.tresorit.com/