International Journal of Scientific Research & Engineering Trends
Volume 12, Issue 2, Mar-Apr-2026, ISSN (Online): 2395-566X

QuantumDrive: A Secure Architecture for Ephemeral QR-Based File Sharing

Aryan D. Vekariya
Dept. Of Information Technology Parul Institute Of Engineering And Technology Parul University, Vadodara, Gujarat, India

Abstract—When working across different devices, moving a file securely from a phone to a laptop is surprisingly annoying. People usually end up emailing files to themselves or logging into a cloud drive just for one small transfer. This not only wastes time but also leaves junk data sitting on third-party servers forever. To fix this, we built QuantumDrive, a simple web app that lets you transfer files directly without forcing you to make an account. We used React for the frontend and Node.js for the server. Instead of saving your files permanently, the app encrypts them right inside your browser. The receiver then downloads the file using a temporary QR code, and the file gets deleted from the server shortly after. This paper explains how we designed the app, how the encryption works, and what we learned while building it during our internship.

Keywords—Temporary storage, File sharing, AES-256-GCM, QR codes, React, Fastify, Web Crypto.

I. INTRODUCTION

Software developers and students constantly need to move logs, design assets, or quick code scripts between different computers. Right now, doing this securely is a bit of a headache. It is really frustrating to dig around for a USB drive or go through the whole process of logging into Google Drive just to transfer a single 5MB configuration file.

This habit of using cloud storage for temporary files kills productivity and creates multiple leftover copies of sensitive data online. We built QuantumDrive to solve this exact everyday problem. Instead of building another huge cloud storage platform where files live forever, we designed it to be a secure, temporary tunnel.

With our app, a user can pick a file, and their browser scrambles it using encryption before it even touches the internet. The file gets parked on our server as an unreadable block of data. Then, the person receiving the file just scans a QR code. They get the file, it decrypts on their end, and everyone moves on. This report breaks down the code architecture and how we handled the security pipeline.

II. LITERATURE REVIEW

Most of the popular tools out there, like Dropbox and Google Drive, are built for long-term storage. While they are great for keeping backups, they force you to create a user profile and keep your files on their servers until you remember to manually delete them.

A few projects have tried to solve this by focusing on temporary, self-destructing files. A good example was Firefox Send (which is no longer around) and newer websites like Wormhole. The main idea behind these tools is time-limited, encrypted sharing. Research shows that modern web browsers are now powerful enough to handle heavy encryption tasks locally using the Web Cryptography API, which means we do not have to rely on the backend server to secure the data.

However, most of these existing platforms still rely on copying and pasting links in chat apps or emails. We decided to use QR codes instead. A QR code bridges the gap between two physical devices perfectly—like moving a file from a PC monitor straight to a smartphone camera without typing anything. That is the core idea we built QuantumDrive around.

III. SYSTEM DESIGN AND TOOLS

We wanted to keep the project clean and easy to maintain, so we picked a modern JavaScript stack.

The Tech Stack

- **Frontend:** We used React (version 19.2) and Vite to build the user interface and generate the QR codes quickly.
- **Backend:** We set up a Node.js server using Fastify. Fastify is really lightweight and handles HTTP requests incredibly fast.
- **Database:** We chose MongoDB to store the locked files. The best part about MongoDB for this project is its automated TTL (Time-To-Live) indexing, which automatically wipes the data after 24 hours without us having to write extra cleanup scripts.
- **Security:** We used the native Web Crypto API built into modern browsers. It handles the AES-256-GCM encryption and SHA-256 hashing right on the user's machine.

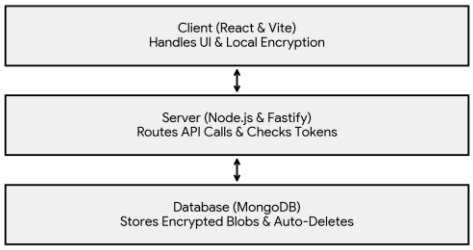

Fig. 1. QuantumDrive System Architecture

IV. HOW IT ACTUALLY WORKS

To make sure the project made sense, we mapped out exactly how the data flows from the sender to the receiver.

The Upload Process

Everything starts with the Sender. When they upload a file, the browser reads it as a raw byte array. We wrote a script that generates a random 64-byte salt and a secret key. The browser then encrypts the file locally. Our server only receives this locked file and has absolutely no way to read it. The server then hands back a special JSON Web Token (JWT), which our React app turns into a QR code on the screen.

Fig. 2. How Files Move Between Users

The Download Process

On the other side, the Receiver opens their phone camera or our web app and scans that QR code. Scanning the code sends a quick check to our server to make sure the file is still there and hasn't been downloaded yet. If it's safe, they click download. The server sends over the locked blob, and the receiver's browser uses the secret key hidden inside the QR code to decrypt the file back to normal.

Because our server doesn't know what kind of file it was holding, the receiver's browser actually reads the first few "magic bytes" of the decrypted data. For example, if it sees the hex code "FF D8 FF", it knows it's a JPEG image and saves it properly.

V. TESTING AND SECURITY

We tested the prototype mostly by sending different types of files and checking what happens on slow networks.

Speed and Performance

Because we kept the architecture lightweight, it runs really fast. When we tested with a standard 1MB picture, the browser took about 200 milliseconds to scramble the file. Getting a response from our Fastify server to start the transfer only takes about 20 to 50 milliseconds. Overall,

staging a file on the screen takes about half a second, plus normal internet upload time.

Security Rules

To stop people from abusing the server or filling up our database, we put some strict rules in place. If a user isn't logged in, they can only send files up to 30MB, and we use basic device fingerprinting to limit them to 3 free transfers. Also, our database is strict about the one-time download rule. Once a file is downloaded successfully, the server flips a "consumed" switch, and anyone else who tries to scan that same QR code will just get an error.

VI. CONCLUSION AND NEXT STEPS

Building QuantumDrive proved that you don't need a massive, complicated system to share files securely. By pushing the heavy lifting of encryption to the user's browser and using the server just as a dumb relay, we got rid of all the privacy risks you normally get with file-hosting websites. Using MongoDB's auto-delete feature saved us a lot of headache with server maintenance.

If we wanted to take this project further in the future, the next logical step would be horizontal scaling. We could add a load balancer and swap out MongoDB for a cloud storage bucket like Amazon S3, which would handle huge video files much better. We also thought about adding WebSockets so the sender's screen could show a cool animation the exact second the receiver downloads the file.

Acknowledgment

A big thanks to the developers behind React, Fastify, and MongoDB for making their tools open and easy to learn. We also really appreciate the guidance from our mentors who helped us shape this internship project into a real, working application.

REFERENCES

1. W3C, "Web Cryptography API," W3C Recommendation, Jan. 2017. [Online]. Available: https://www.w3.org/TR/WebCryptoAPI/
2. D. McGrew and J. Viega, "The Galois/Counter Mode of Operation (GCM)," Submission to NIST Modes of Operation Process, 2004.
3. M. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)," RFC 7519, May 2015.
4. Node.js Foundation, "Node.js Documentation," [Online]. Available: https://nodejs.org/docs/