Article

Not peer-reviewed version

Authenticate and Verification Source Files using SHA256 and HMAC Algorithms

Wisnu Uriawan [*] , Ray Ramadita [*] , Rizky Dwi Putra [*] , Rizqi Ilham Siregar [*] , Risyad Addiva [*]

Posted Date: 1 July 2024

doi: 10.20944/preprints202407.0075.v1

Keywords: AES, Cryptography, HMAC, SHA256

Article

Authenticate and Verification Source Files using SHA256 and HMAC Algorithms

Wisnu Uriawan [1], Ray Ramadita [2], Rizky Dwi Putra [3], Rizqi Ilham Siregar [4] and Risyad Addiva [5]

[1] Informatics Department, UIN Sunan Gunung Djati Bandung, Jawa Barat, Indonesia; wisnu.uriawan@uinsgd.ac.id
[2] Informatics Department. UIN Sunan Gunung Djati Bandung, Jawa Barat, Indonesia; rayramadita12@gmail.com
[3] Informatics Department, UIN Sunan Gunug Djati Bandung, Jawa Barat, Indonesia; rizkydwiputra6666@gmail.com
[4] Informatics Department, UIN Sunan Gunug Djati Bandung, Jawa Barat, Indonesia; rizqiilhamsiregar@gmail.com
[5] Informatics Department, UIN Sunan Gunung Djati Bandung, Jawa Barat, Indonesia; risyadaddiva@gmail.com

Abstract: Because data authenticity and validity are becoming more and more of a concern in the digital age file security and integrity must be guaranteed. This paper investigates the application of HMAC-SHA256 which combines the Secure Hash Algorithm 256 (SHA256) with a private key for transmission-side source file authentication and verification. The study creates a prototype application to evaluate the efficacy of HMAC-SHA256 in securing file transfers by examining previous investigations on SHA and HMAC. The findings demonstrate that SHA256 produces unique file signatures which successfully guards against unwanted changes. Through the use of cryptographic hashes and secret keys to create distinct digital signatures that guarantee file authenticity HMAC improves security. The study shows that HMAC-SHA256 protects data integrity and detects tampering effectively. It does however stress how crucial it is to comprehend cryptographic concepts and apply these strategies with caution in order to prevent vulnerabilities. Overall the study finds that when properly implemented and updated HMAC-SHA256 greatly enhances file security in the digital age.

Keywords: AES; Cryptography; HMAC; SHA256

1. Introduction

In the ever-evolving digital age, the growth of information technology has presented new challenges related to file security and integrity, raising concerns about the validity and authenticity of data sources. The circulation of files between devices and platforms also poses the risk of manipulation or unauthorized insertion of content, which can have a serious impact on user security and privacy.

The advancement of technology and the thoughts of researchers create sophisticated technologies and help many people do things, such as information that is now quickly widespread with the internet, even those of us who used to long and complicated in sending files now only use the internet we can exchange files.

In the era of distributed systems, where file and data exchanges occur across vast and diverse networks, the implementation of cryptographic technology is becoming increasingly vital. Distributed systems can ensure the security, integrity, and authenticity of files transferred between system nodes. Building dependable systems in intricate and dynamic distributed environments is facilitated by this strong foundation [1].

When it comes to technological advancement, cryptography has been the foundation for the creation of more modern sophisticated systems like blockchain. By using this new technology data security and stability in a distributed network can be guaranteed without requiring a central administration to oversee or validate transactions because it makes use of the cryptography concept. Thus cryptography is a tool for data protection as well as a contributing factor to recent technological advancements in a number of other areas [2].

Furthermore digital transactions including online payments and information transfers will be very safe when technology and cryptography are applied. Virtual signatures and encryption protocols are used by cryptography technology to ensure that no transaction is altered by unauthorized parties. Users' trust in the platform or system they are using is bolstered by this [3].

 doi:10.20944/preprints202407.0075.v1

As a way of handling these challenges cryptography has become increasingly important. SHA256 and other cryptographic hashes are some of the most popular cryptography techniques. One can generate a unique signature mark from a file using this hash algorithm. Because even small changes to the file will result in a different hash this process allows the integrity of the file to be verified.

Additionally a key factor in guaranteeing the integrity and authenticity of files is the HMAC (Hash-based Message Authentication Code) algorithm. A file or message can be signed with a secret key that is only known to authorized parties by using HMAC. As a result the recipient of the message or file can confirm that it originated from a reliable source and was not altered in transit [4].

In the financial and healthcare sectors the importance of using cryptography technology to protect sensitive data has grown overtime as technologies advanced. Companies will be able to use hash and HMAC algorithm in safeguarding their sensitive data against unauthorized changes or the introduction of malicious content.

Additionally the use of cryptography technology contributes to building user and system trust because of its ability in safeguarding data therefore guaranteeing data security. Users can be assured that the information they receive or send is authentic and hasn't been tampered with by third parties by using the concept of file authentication [5].

Because cybercrime is still on the rise, researchers are not content to just stick with the many efficient methods that cryptography technology offers to safeguard the security and authenticity of their files. Instead they are constantly coming up with new more advanced techniques to tackle these problems and their efforts are aided by the widespread cryptography community [6].

Therefore cryptography technology serves as both a foundation for future technological advancements and innovations and a solution to the problems with file security and integrity that exist today. Through continuous research and development efforts, we can continue to strengthen our information security systems and ensure that our data remains safe and secure in the ever-evolving digital age.

This paper's remainder is structured as follows: Section 1 introduces the credit scoring and background. Section 2 related work. Section 3 are methods. Section 4 result and discussion. Section 5 concludes this paper.

2. Related Work

Previous research indicates that hashing, specifically SHA256 combined with HMAC, a powerful and secure hash function is important for maintaining the integrity of digital data. A lot of research has been done on the efficacy and security of this file integrity authentication technique as well as how it can detect any unauthorized changes or modifications in digital data.

Rezky M. Nasution (2022) discusses about how audio files are easily accessed by careless people which makes them easy targets for fraud eavesdropping and data theft. They can be protected with the use of cryptographic techniques. Secure Hash Algorithm (SHA) , a standard hash function released by NIST, is one frequently used cryptographic technique. A message digest or 160 bits of output is generated by the SHA-1 variation of SHA. This study addresses the use of the SHA-1 algorithm for audio file authenticity detection particularly when sharing private audio files over public networks. Preventing unwanted alterations or alterations to audio files is the aim in order to reduce fraud hoaxes and mishandling of audio files [7].

Quist-Aphetsi K. and Baffour Senkyire I. (2019) asserts that technological advancements have made it easier for careless individuals to alter and utilize private digital photos for illicit purposes. Consequently it is suggested that forensic digital images be validated using the SHA-256 algorithm. By making sure that they are not changed or manipulated without permission this attempts to increase the security of private digital photos. High levels of authentication are anticipated with SHA-256 reducing the possibility of sensitive digital images being misused [8].

Word documents with a .docx extension are one type of digital file that Sopiana Nainggolan (2022) emphasizes the significance of the SHA-256 hash algorithm in detecting changes. Computers make it easy to duplicate files creating a perfect replica of the original. Still, even minute modifications like a

 doi:10.20944/preprints202407.0075.v1

single pixel shift in a digital picture can be picked up by SHA-256. Because this algorithm is intended to detect changes with nearly total precision even with its short 256-bit length it is regarded as secure. Based on earlier research SHA-256 has shown its effectiveness and dependability, as such the SHA-256 was chosen to be used in this study [9].

In the field of cryptography there has also been recent research on the use of elliptic curve cryptography (ECC) in conjunction with SHA-256 for increased security. ECC is suitable for settings with limited resources because it provides a higher level of security with smaller key sizes than traditional RSA cryptography. Combining SHA-256 and ECC ensures data integrity and authenticity even in low-power devices providing a robust framework for protecting digital communications. The convergence of security and computational efficiency has shown to be particularly beneficial in Internet of Things (IoT) applications [10,11].

Another trend is the use of blockchain technology to verify file integrity. Hashing digital files that later can be stored in a decentralized tamper-proof ledger that is made possible by blockchain technology. Blockchain technology guarantees that any modification in the file can be quickly identified when paired with SHA-256 offering a transparent and safe way to track file changes over time. In supply chain management and digital rights management where preserving the integrity of papers and media files is essential this strategy has been successfully applied [12,13].

Additionally advances in machine learning have been applied to improve the identification of irregularities and unapproved modifications in electronic documents. Strengthening machine learning algorithms to detect patterns and anomalies in normal file behavior can provide an extra security measure. Because machine learning-based anomaly detection can provide real-time insights into possible security breaches when combined with SHA-256 and HMAC it is a useful tool for large-scale dynamic systems [14,15].

Serverless computing platforms such as AWS Lambda and Azure Functions have completely changed the way secure applications are deployed and managed. Because these platforms take care of the underlying infrastructure automatically, developers can focus on writing code. Without having to cope with the headaches of server management developers can ensure the security of their applications by integrating SHA-256 and HMAC for data integrity verification in serverless functions. This approach is scalable and economical which is very advantageous for applications with fluctuating workloads [16,17].

Since the use of end-to-end encryption or E2EE in communication platforms ensuring the confidentiality and integrity of data has gained a lot of traction. Intermediaries are unable to decrypt data in E2EE so only the users who are communicating can read the messages. SHA-256 for message integrity and HMAC for authentication work together to provide an extremely secure communication channel when combined with E2EE. This combination protects against listening in and tampering. This has worked particularly well with secure messaging apps and online collaboration tools [18,19].

As specified by Kumar Rakesh et al., based on MongoDBs adaptability in managing both semi-structured and unstructured data, it has shown to be a useful database for hashed data storage and metadata. MongoDB offers plenty of flexibility when it comes to changing the format of data because it stores data in documents that resemble JSON. This adaptability is especially helpful for applications that need to store various types of data like book metadata. Due to its scalability which enables it to efficiently handle enormous volumes of data MongoDB is an excellent option for large-scale applications. Furthermore sensitive data is shielded from unwanted access by its robust security features which include authorization authentication and encryption at rest [20,21].

In JS environments it is common to highlight the use of Multer, a middleware designed to handle multipart/form-data, the standard encoding for file uploads on the web. After processing the files it receives Multer makes the data available to the server either in memory or on disk. Applications that request file uploads from users must make sure that files are efficiently parsed and stored. Research indicates that Multer combination with Express improves its usefulness by enabling developers to

 doi:10.20944/preprints202407.0075.v1

design endpoints that manage file uploads safely and effectively thus bolstering the application's overall functionality [23**?**].

A popular foundational framework for developing server-side applications is Express. It's a simple adaptable Node.js web application framework offering a large feature set for mobile and web applications. Express user-friendly routing system and middleware integration make managing HTTP requests and responses easier. Express is the main component of server-side architecture in a number of studies controlling data flow and business logic. Building scalable and maintainable microservices is made easier by its robust community support and modular design. To improve application capabilities developers can quickly define routes to handle various request types and integrate middleware such as Multer by using Express [22,23].

Modern web development requires two things above all else: client-side navigation that works seamlessly and efficient server-side rendering. Thanks to its performance optimizations and extensive feature set Next.js is the framework of choice for developing contemporary web applications. Creating static and dynamic websites is made easier with this React-based framework which offers developers and users a cohesive experience. Next offers integrated server-side rendering support, thanks to it, web apps are more accessible and user-friendly with faster loading times. Also, Next.js Code management is made simpler by the file-based routing system which enables scalable and user-friendly routing. Additionally, the framework allows for incremental static regeneration which combines the advantages of dynamic and static rendering to guarantee real-time updates and optimal performance. Following. With the help of the robust ecosystem of plugins and extensions that improve its functionality, developers can easily create web applications that are scalable, maintainable, and high-performing [24,25].

On the other hand, Postman is definitely an integral tool for API testing and validation [26]. This is one such versatile platform with which any developer can have a user-friendly interface to send HTTP requests and inspect server responses, or even automate testing workflows. Additionally, it has features beyond interaction with very basic APIs; it supports the generation of full-featured test suites using collections and scripts to easily and thoroughly test endpoints dynamically under various conditions. Its environment management features allow straightforward testing of different deployment stages, ranging from development to staging and finally to production, by dynamically controlling environment variables and their configurations [27]. In addition, Postman collaborates with team members using shared collections and automated testing for consistency in the API behavior across development cycles [28].

The advanced encryption standard or AES is well known for its strong security and effectiveness when encrypting digital data. The National Institute of Standards and Technology (NIST) created AES which is now the industry standard for encryption because of its capacity to manage massive volumes of data in a secure manner. Key lengths of 128, 192, and 256 bits are supported and it functions with fixed block sizes of 128 bits. AES is a dependable method for protecting sensitive data such as private communications and financial and medical records according to several studies. Studies have indicated that AES' strong defense against brute force attacks makes it the best technique for securing data while it's in transit and at rest. Because AES is effective at both data encryption and decryption it is a good choice for high-speed processing applications like streaming services and real-time communications. The fact that it is widely used in many security protocols including SSL/TLS for safe online communications emphasizes how crucial it is to contemporary cryptography methods [29,30].

The recent HMAC-SHA256 research expands upon the groundwork laid by earlier research on the topic. In this study, HMAC will be combined with SHA256 to improve and strengthen the defense against unwanted access a secret key is added to the authentication process. source file authentication and verification are accomplished through the use of HMAC-SHA256. By creating a unique hash code for every file and comparing it to the original hash code to identify any tampering or unauthorized changes made during transmission HMAC-SHA256 ensures message integrity and authenticity. In

 doi:10.20944/preprints202407.0075.v1

contrast, AES is used to encrypt sensitive data including file hashes and encryption keys guaranteeing that these details are kept private and secure during storage and transmission procedures. Furthermore, Multer is used for efficient file handling, Express for reliable server-side application management, MongoDB for safe and scalable data storage and, Next.js is used in front-end programming.

3. Methodology

This research uses HMAC-SHA256 (Hash-Based Message Authentication Code) to increase the hash security of a file by encrypting the original hash using a predefined secret key. This algorithm was chosen because it utilizes a secret key to ensure that the original hash is not easy to crack in order to ensure that only those who have the key can decrypt the original hash.

The research begins by reviewing related research, especially research on the implementation or study of the Secure Hash Algorithm (SHA) and Hash-Based Message Authentication Code (HMAC). After that, it continues with the development of an application prototype with the following user story:

- Users are faced with several input forms such as images and text. Image input is used as an image container that will be sent to the server for hash analysis, while text input is used as a secret key container determined by the user himself where later the secret key is used to encrypt the image hash with the help of HMAC-SHA256.

  Acceptance Criteria:

  - Image Input Form: A form where users can upload an image.
  - Text Input Form: A form where users can input a secret key.

- The backend retrieves the metadata and hash of the uploaded image, then performs HMAC-SHA256 encryption with the secret key combination that was previously inputted.

  Acceptance Criteria:

  - Retrieve Metadata and Hash: The backend retrieves the necessary metadata and calculates the hash of the uploaded image.
  - HMAC-SHA256 Encryption: The backend encrypts the image hash using the provided secret key.

- The image hash, image hash encryption result, metadata, and other details are stored in the database.

  Acceptance Criteria:

  - Store Data: The system stores the image hash, encrypted hash, metadata, and other relevant details in the database.
  - Data Integrity: Ensure that the data stored is accurate and retrievable for future processes.

 doi:10.20944/preprints202407.0075.v1

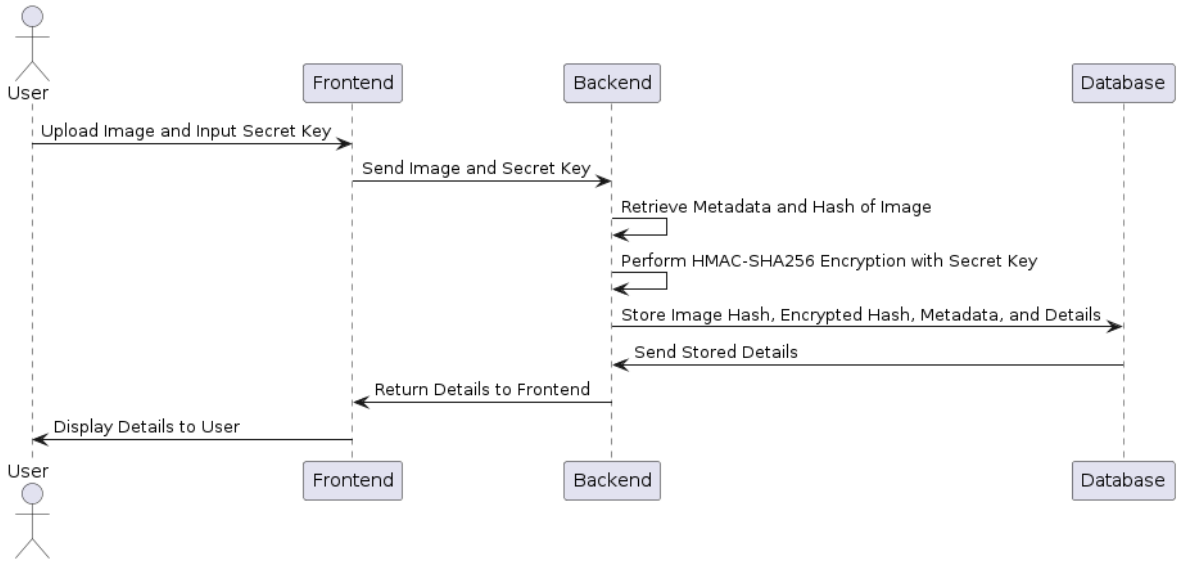

Figure 1. UML diagram illustrating the user interaction with the system for image and secret key input, backend processing, and database storage and retrieval.

Then, the next user story is a checking scenario with the following details:

- Users get images that are the same or similar to those in the system from various sources such as social media or other communication applications (Telegram, Whatsapp, Discord, etc.).

  Acceptance Criteria:

  - Users can obtain images from various sources such as social media, and communication applications (Telegram, Whatsapp Discord, etc.).
  - The system should support common image formats (e.g., JPEG, PNG).
- The user wants to check the authenticity of the image then opens the system that has been built in this research and is faced with an image and text input form. Image input is used as an image container which will later be sent to the server for hash analysis, while text input is used as a secret key container (users must have the secret key of the original owner of the image they want to check) where the secret key will be used to decrypt the image hash which has previously been generated using HMAC-SHA256.
- The backend matches the decrypted data with the data already in the database.

  Acceptance Criteria:

  - The backend receives the uploaded image and secret key.
  - The backend retrieves the metadata and hash of the uploaded image.
  - The backend decrypts the image hash using the provided secret key.
  - The backend compares the decrypted hash with the stored hashes in the database.
  - If a match is found, the system identifies the matching data.
- The backend sends a response back to the frontend in the form of which data matches what is in the database.

  Acceptance Criteria:

  - The backend sends a response to the frontend indicating whether a match was found.
  - The response should include details of the matching data if a match is found.
  - The frontend displays the result to the user, indicating whether the image is authentic.
  - If the image matches, the system should show relevant metadata and details of the match.
  - If no match is found, the system should notify the user that the image does not match any records in the database.

 doi:10.20944/preprints202407.0075.v1

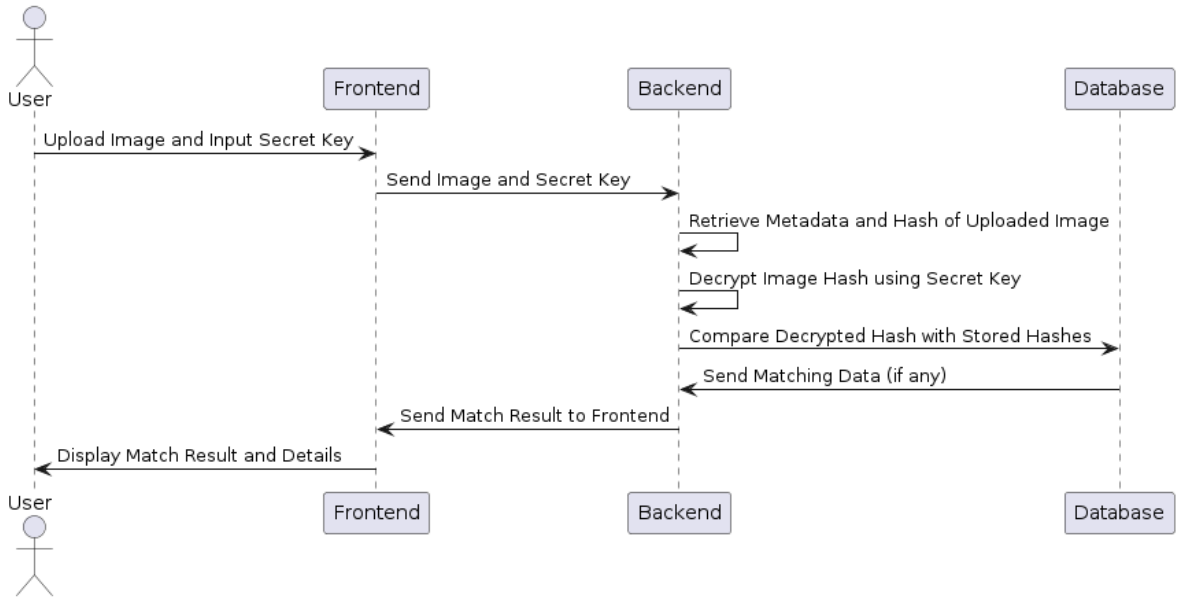

Figure 2. UML diagram illustrating the user interaction for checking the authenticity of an image by uploading it and providing a secret key, followed by backend processing and database verification.

3.1. Research Review

This research proposes a methodology to improve image file authentication using HMAC-SHA256. The research process begins with a review of related research, specifically focusing on the implementation and study of the Secure Hash Algorithm (SHA) and Hash-based Message Authentication Code (HMAC). From these related journals, the findings of previous research and the methodologies used were analyzed and will influence our research.

3.2. Technologies and Tools

The technologies and tools used in this research are as follows MongoDB for database, Multer for file handling, express for Rest-API, NextJS for frontend, Postman for API testing, HMAC and SHA256 for securing hash file, to provide strong defense against tampering and unauthorized access, sensitive information including file hashes and encryption keys will also be encrypted using AES.

3.2.1. MongoDB

In this research, the database used is MongoDB, where MongoDB is a NoSQL database. MongoDB is a well-known NoSQL (Not Only SQL) database system used to store and manage data in JSON (JavaScript Object Notation) document format. Unlike traditional relational databases such as SQL that use rigid table structures and schemas, MongoDB provides greater flexibility for storing unstructured or semi-structured data [31].

 doi:10.20944/preprints202407.0075.v1

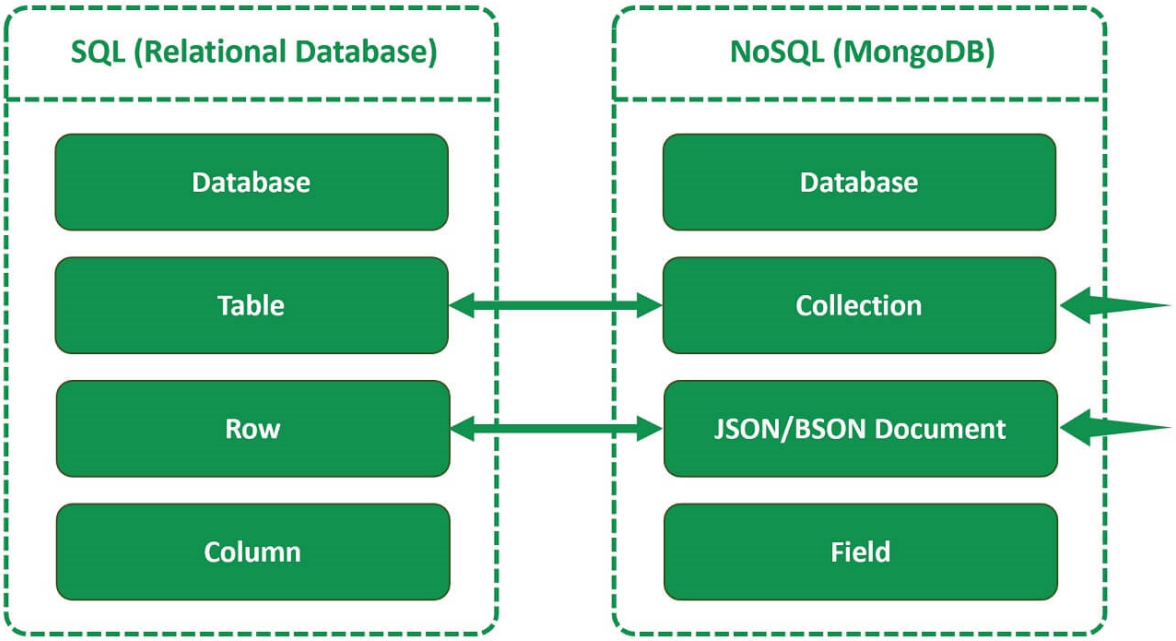

Figure 3. Format Difference between Relational Database Management System (RDBMS) and MongoDB. Source: pragimtech

MongoDB, as a leader in NoSQL database systems, offers many advantages that appeal to developers. The flexibility of its schema can be analogized to a blank canvas ready to accept data in any form, which is ideal for data that is constantly evolving or has an unpredictable structure[32].

Its high scalability capabilities allow for the handling of very large data volumes and distribution across multiple servers, making it a great choice for large-scale web applications. It is extremely fast with a high-performance storage engine and efficient queries, enabling data access in record time [33].

Common MongoDB Use Cases: Solving a Wide Range of Data Challenges MongoDB not only excels in flexibility and scalability but also in handling a wide range of use cases. For web applications, MongoDB serves as a reliable data warehouse, neatly storing user, product, order, and other dynamic data [34].

MongoDB's document-oriented structure is ideal for this application, as it allows for the flexible and efficient storage of complex and evolving metadata [32]. Thanks to the database's scalability, data can be retrieved and verified quickly even with an increasing number of books and related metadata. Performance is thus maintained [31,33].

In addition MongoDBs strong security features, such as authentication authorization and encryption at rest provide an extra degree of defense to guarantee that hashed data and metadata are safely stored and accessible to authorized users only. This research shows how NoSQL databases can be used to effectively enhance data integrity and security in digital applications ensuring dependable and efficient data management and retrieval [34]. Therefore MongoDB is chosen to be used for secure file storage in this study.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 1 July 2024 doi:10.20944/preprints202407.0075.v1

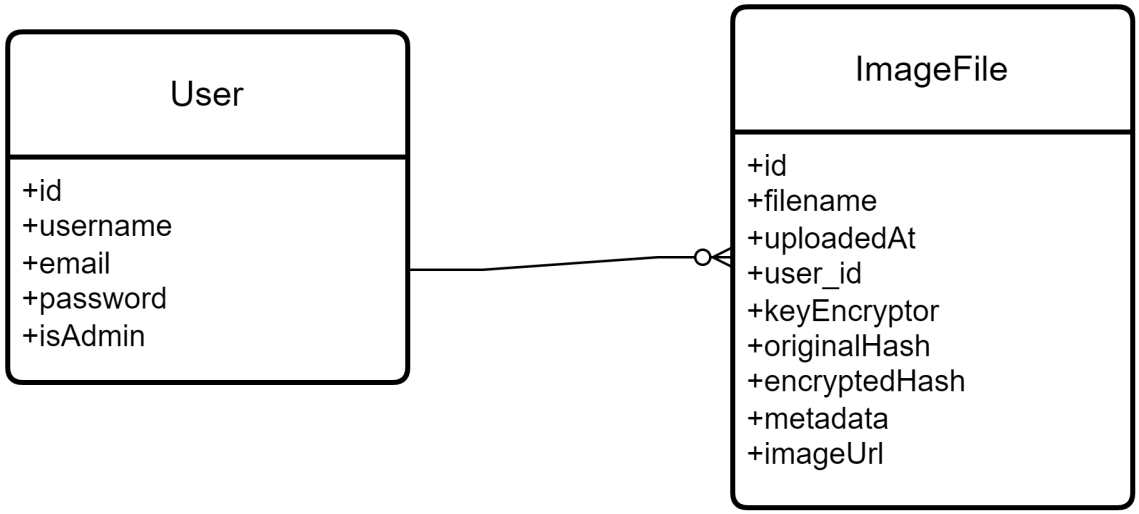

Figure 4. Entity Relationship Diagram of This Project

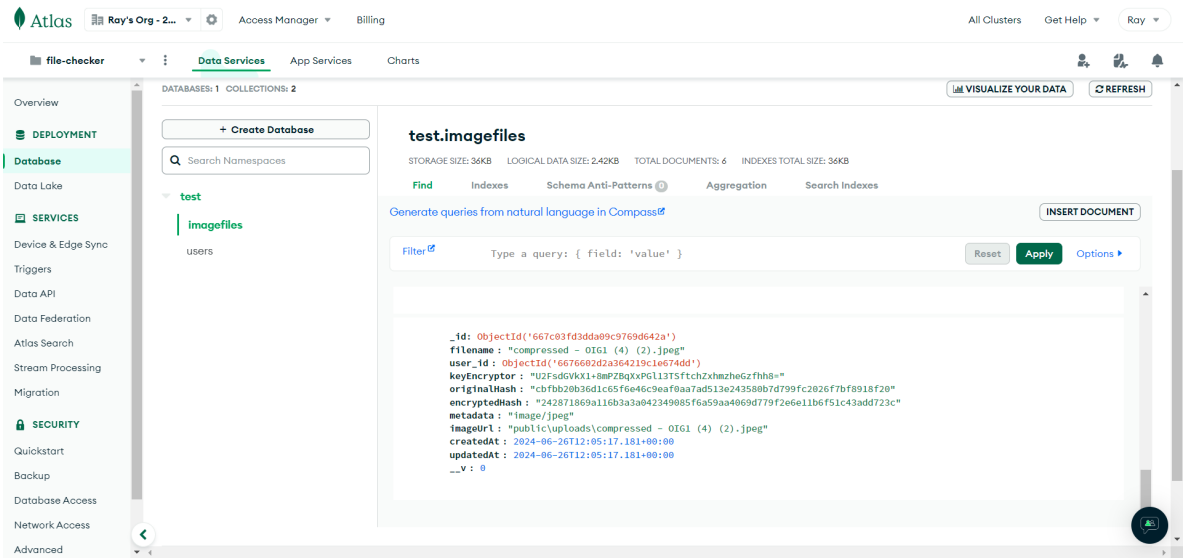

Figure 5. Example of MongoDB ImageFile Document

 doi:10.20944/preprints202407.0075.v1

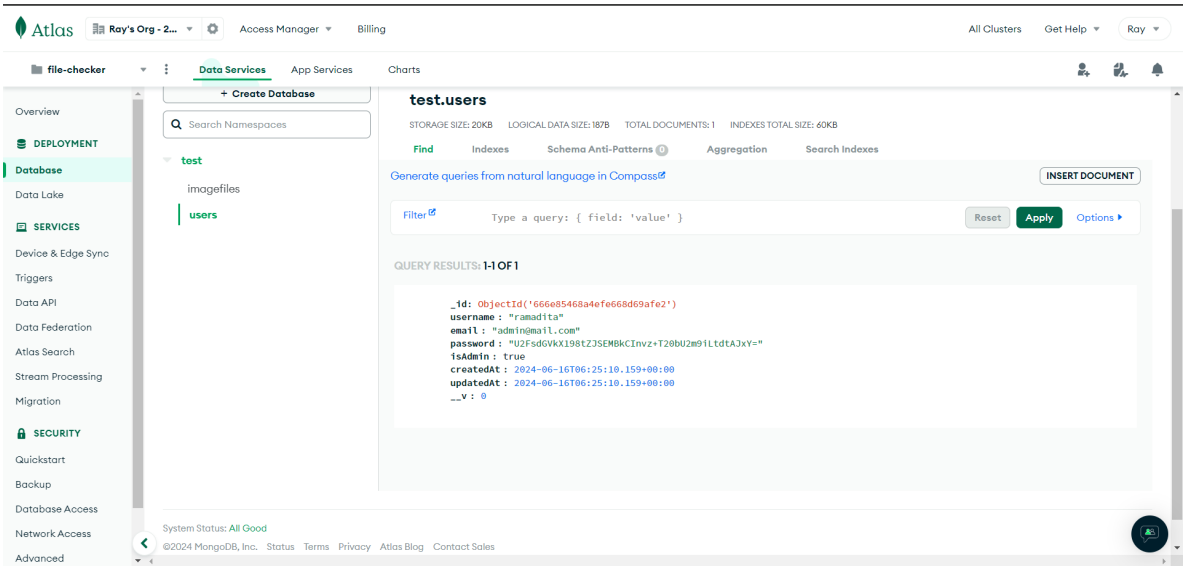

Figure 6. Example of MongoDB User Document

3.2.2. Multer

A vital Node middleware in node.js apps specializing managing multipart/form-data the common file upload format is called Multer. Through request inspection and file extraction from the payload it streamlines the file upload process. This feature will be necessary for applications that ask users to upload different kinds of files such as documents multimedia files and images [23].

One of Multers key advantages is its flexibility with regard to file storage following upload. Developers can set Multer up to store files in memory (as buffers) or directly to disk. This adaptability allows programs to be optimized based on factors such as file size, upload frequency, and server resources enabling efficient file upload management [23,35].

 doi:10.20944/preprints202407.0075.v1

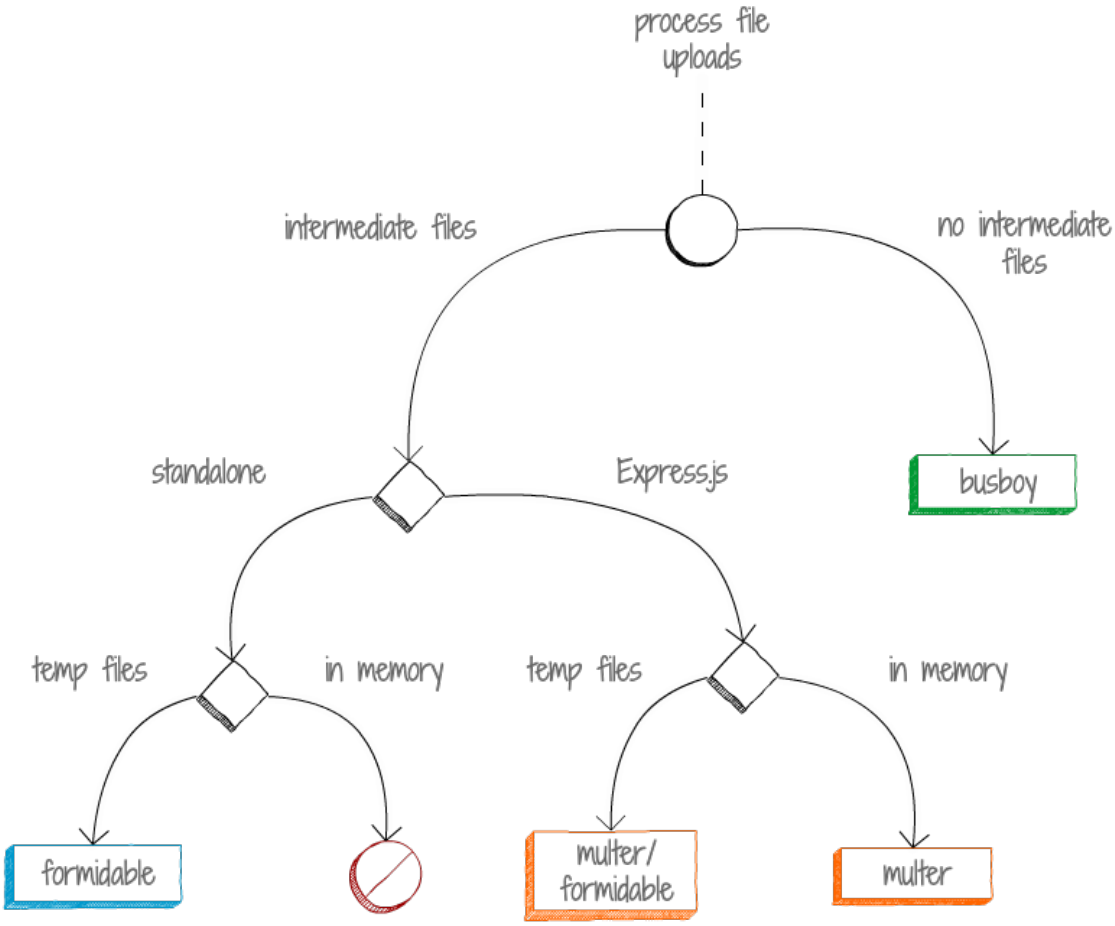

Figure 7. Handling File Uploads. Source: byte archer

Multer's seamless integration with Express, a popular Node, facilitates its integration into application routes. A web framework in JavaScript. Developers can designate specific endpoints in Express where the Multer middleware handles file uploads ensuring that uploaded files are safely managed and available for additional application logic. Strong file management functionalities are made possible by Multer in practical applications such as e-commerce platforms and content management systems. To ensure that these resources are secure and only accessible by authorized users Multer might for instance supervise the uploading of high-resolution pictures or video files into a digital asset management system [23].

All things considered Multer is critical for enhancing application performance because it simplifies and secures file upload management. Because of its seamless integration with Express and customizable storage options it is the preferred option for Node.js apps that require efficient file management features.

3.2.3. Express Framework

An adaptable framework that is Express, owing to its extensive middleware ecosystem and versatility the JavaScript web application framework is widely acknowledged as a dependable tool for creating server-side applications. Because of how robust its middleware integration and straightforward routing mechanism handling HTTP requests and responses is made easier [36].

 doi:10.20944/preprints202407.0075.v1

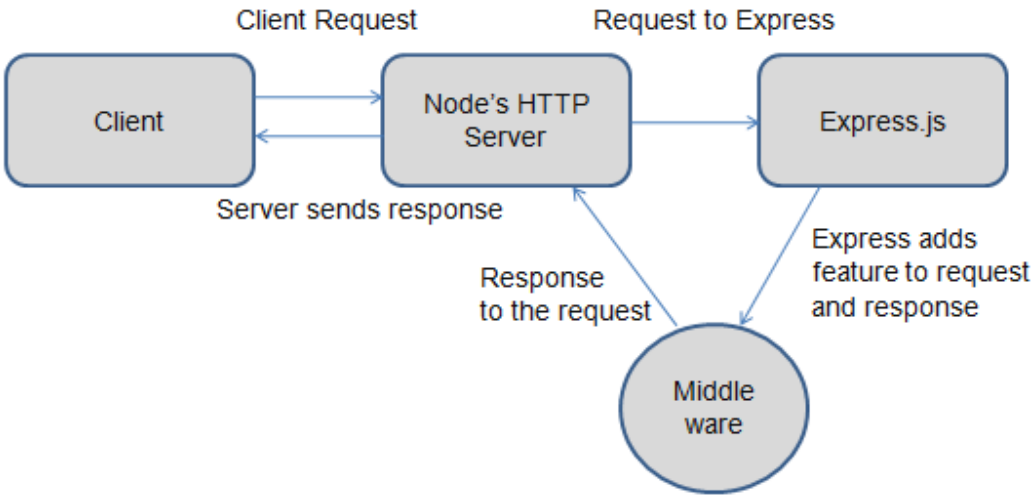

Figure 8. How Express.js link to Node.js. Source: eTutorialsPoint

Express, which is built on features necessary for effectively managing data flow and implementing business logic, is the foundation upon which server-side applications are constructed. Applications can be more functional thanks to middleware like Multer and Express which can handle various request types, define routes and handle other request types, thus its potential is hugely increased by those functions [23].

Smooth file upload processing is made possible by Multer's interface with Express. Applications that need to understand and safely store uploaded files might use Multer, a specialized middleware to handle multipart/form-data. For websites that request file submissions this feature is necessary to ensure safe and efficient file processing [23,36].

Express is a well-liked tool for developers creating scalable and robust online applications because of its modular architecture. It is a recommended framework for projects requiring flexibility and excellent performance due to its seamless integration with other Node.js modules and frameworks.

3.2.4. NextJS

Is a web development framework built on React, a popular JavaScript library for building user interfaces (UI). Next.js adds important features that simplify and optimize the modern web development process.

Next.js has a number of advantages over pure React. One such advantage is the server-side rendering (SSR) capability, which allows React components to be rendered on the server instead of just in the browser. This improves website performance and search engine optimization (SEO) as web page content can be instantly understood by search engines. In addition, Next.js supports static site generation (SSG), which allows the creation of pre-rendered static websites, allowing them to load very quickly, especially for static content such as blogs or landing pages.

The built-in routing features in Next.js make it easy for developers to determine how URLs are mapped to the appropriate React components. Next.js also provides various ways to retrieve data from external sources, such as APIs, for display on web pages. Another advantage of Next.js is its ease of use, especially for developers who are already familiar with React. Next.js has a large and active developer community and would be able to offers plenty of help and resources thats available when needed.

 doi:10.20944/preprints202407.0075.v1

Next.js presents a number of fundamental ideas that form the basis of the framework. In order to make the website easier to manage and develop the pages system is first used to define its structure. Each page is represented by a separate React file. Smooth integration of serverless backends with React.js frontends is made possible for ease of use in creating API routes. Furthermore because static web pages are pre-generated Next.js allows for Static Site Generation (SSG) which offers exceptional scalability and speed. Last but not least Next.js allows for Server-Side Rendering (SSR) which enhances SEO and website performance for dynamic content by type-checking web pages on the server before they are sent to the browser.

A number of extra features that Next.js would provide improvement to the web development process. Optimizing images automatically for better SEO and performance is the goal of the Image Optimization feature. Furthermore IntelliSense and type-checking are two features that TypeScript, a JavaScript superset offers, is supported by Next.js. Because of its built-in internationalization features Next.js makes it simple for multilingual web development. Multiple ways to retrieve data from external sources including databases headless CMS and APIs are also provided by Next.js.

Security and scalability are core design principles of Next.js. Assuring your website is safe and scalable to meet demands for high traffic it adheres to web security best practices and integrates with top cloud hosting platforms.

The best framework for creating contemporary easy-to-develop and SEO-friendly websites is Next.js. Next.js boasts a vast community and sophisticated features. For developers who want to create incredible online experiences, Next.js is one of the best options.

3.2.5. Postman

APIs or applications programming interfaces are connecting various services and applications and are becoming an essential part of today's digital world. Testing is essential to ensuring that APIs are safe and functional. For easy and trustworthy API testing in this case Postman is the ideal tool. With the vast feature set of Postman developers can create, send and analyze API requests with ease as well as debug and validate API responses.

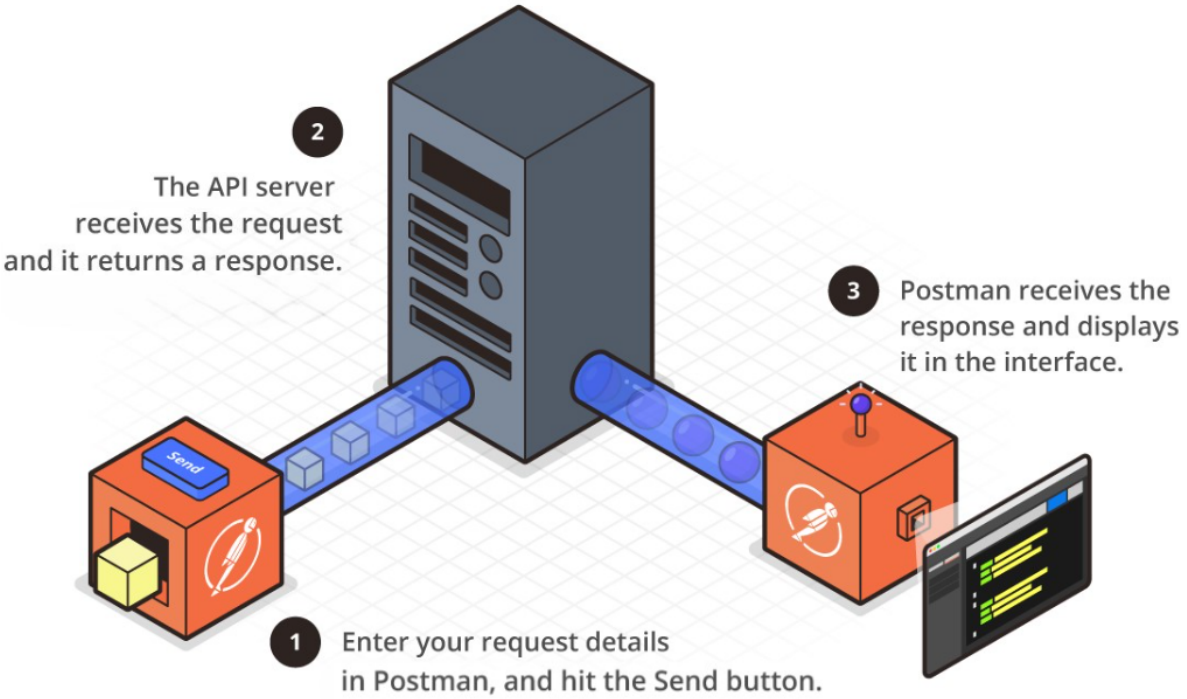

Figure 9. How Postman Work. Source: Adapted from Postman

 doi:10.20944/preprints202407.0075.v1

With its many important features that simplify things for developers Postman is a very helpful tool for API testing. It makes it simple to create an array of API request types including GET, POST, PUT, and DELETE requests. The headers bodies and parameters required for every API request can also be added using Postman. Postman not only communicates with the server via API requests but it also clearly shows the response complete with headers bodies and status codes. Postman also offers tools like JSON Viewer and Console for debugging and response verification. Users can organize their API requests into collections and folders for easier organization and management. Moreover, Postman allows script execution before and after API requests to perform automation tasks. The testing and mocking features in Postman are useful for verifying API functions and simulating API responses. Postman also facilitates collaboration between teams in API testing by allowing the sharing of collections and environments [26].

To implement Postman into verifying the source and authenticity of files using SHA256 and HMAC algorithms, several requests to Postman will be made which can be used as follows:

Table 1. Postman Request Method with Headers and Body

Method	Endpoint	Headers	Body
POST	/register	None	email, username, password, isAdmin (false by default)
POST	/login	None	username, password
POST	/upload	token (Bearer + generated token)	file (with file type), keyExtractor
POST	/check/:id	token (Bearer + generated token)	file (with file type), keyExtractor

3.2.6. HMAC and SHA256

HMAC (Hash-based Message Authentication Code) acts as a guardian that ensures the integrity of the message. By working with cryptographic hash functions such as SHA256, HMAC generates a unique authentication code for each message. This code acts like a digital fingerprint that cannot be copied, allowing you to detect if the message has been altered or tampered with during transmission.

SHA256 (Secure Hash Algorithm 256) is a reliable algorithm in the world of cryptography. It accepts a message as input and through a complex hashing process, produces a unique and immutable 256-bit long output string. This output string would then serves as a digital fingerprint or signature of the message, marking the message's identity and ensuring its integrity [37].

Data security is double-layered thanks to the combined strength of SHA256 and HMAC. After generating an authentication code using SHA256 HMAC appends the code to the initial message. SHA256 is applied once more to the message and authentication code upon receipt. It is verified that the message is real and hasnt been tampered with if the generated authentication code matches the one thats attached. Nevertheless a security alert is triggered in the event that the authentication code differs indicating that the message has been changed or manipulated [38].

 doi:10.20944/preprints202407.0075.v1

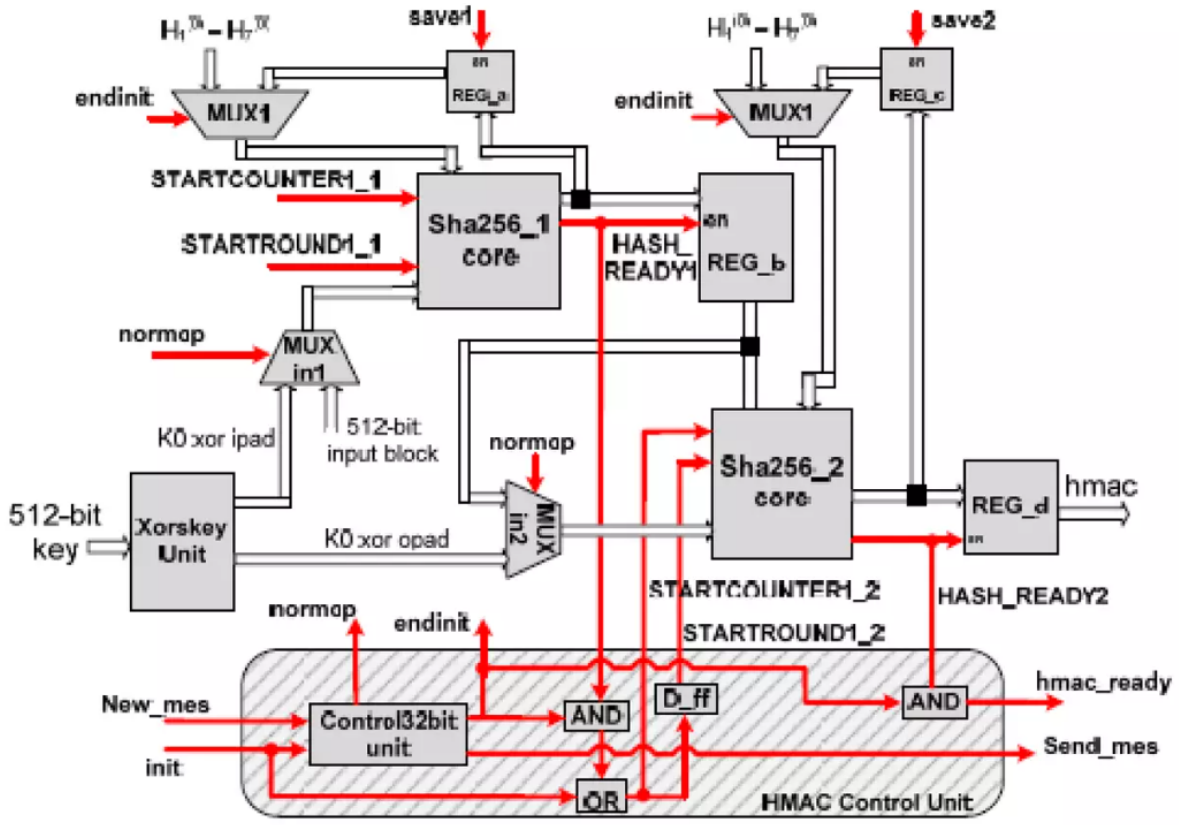

Figure 10. HMAC-SHA256 Architecture. Source: Mensah 2023

The fact that HMAC-SHA256 is resistant to length extension attacks is one of its most important features. Attackers can alter the length of the input message in conventional hashing techniques and still generate a valid hash. By adding a secret key to the hashing process, HMAC reduces this risk and makes it computationally impossible for an attacker to create a legitimate HMAC without the secret key. In secure communications and data integrity applications where message authenticity must be guaranteed this feature is especially helpful [39].

The combination of HMAC and SHA256 is used for message integrity and authentication in addition to being a part of many secure protocols and systems. For instance the SSL/TLS protocols which form the basis for safe online communications depend heavily on HMAC-SHA256. In addition it is employed to secure API requests ensuring that data is accurate and unaltered throughout transmission between clients and servers. The cybersecurity community's confidence in the dependability and reliability of HMAC-SHA256 is demonstrated by its widespread use [40].

HMAC-SHA256 is also very efficient and suitable for both software and hardware implementations. Because of its computing efficiency it can be used in environments with limited resources like embedded systems and Internet of Things devices without compromising security. Rapid authentication of even massive volumes of data is possible thanks to the SHA256 algorithm's efficient application and the methodical HMAC construction process [41].

Because of its adaptability, HMAC-SHA256 is able to be used in multi-factor authentication systems. HMAC-SHA256 would improves the security of the user authentication procedure by creating safe time-based one-time passwords (TOTPs). Each of these TOTPs will be generated uniquely and has a limited validity period because they rely on a shared secret and the current time. By the need to have the correct secret key for systems to validate the code this approach greatly lowers the possibility of unauthorized access [42].

 doi:10.20944/preprints202407.0075.v1

3.2.7. Advanced Encryption Standard

The symmetric encryption algorithm known as Advanced Encryption Standard (AES) is extensively employed in data security because of its robust security features and high level of efficiency. It is adaptable to different security requirements because it runs on fixed block sizes of 128 bits with key sizes of 128, 192, or 256 bits [43]. During file uploads and verification procedures, AES is used to encrypt file hashes and related keys in order to protect data integrity and confidentiality.

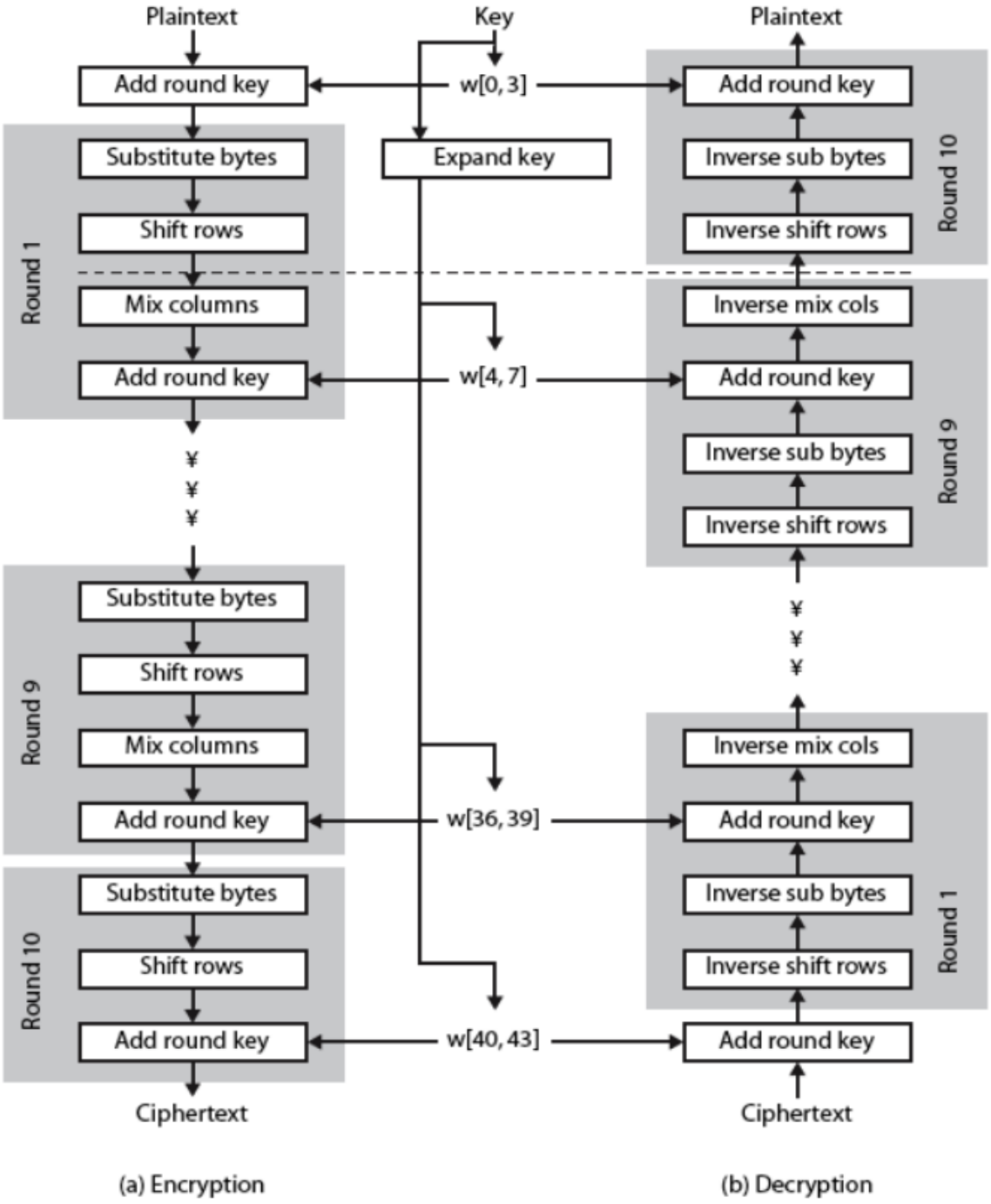

Figure 11. Basic Structure of AES. Source: Abdullah 2017

The first step in the encryption process is adding a round key. Next rows and columns are shifted round keys for 128, 192, and 256-bit keys are added and nine eleven or thirteen rounds of substitution

 doi:10.20944/preprints202407.0075.v1

are performed. In order to generate the ciphertext the mixing columns step is skipped in the final round. The original file hash and the keyEncryptor are then safely encrypted and out of the reach of unauthorized users thanks to the storage of this ciphertext in the database [30].

The file hash and the keyEncryptor are encrypted by the backend using AES during the file upload process and are then stored in the database. This encryption makes sure that the encrypted values are safe even in the event that the database is hacked. An AES encryption is applied to the keyEncryptor that the user provides and its hash is compared with the encrypted keyEncryptor that has been stored in case the file needs to be verified. The authenticity and integrity of the file are verified if they match [29].

To ensure reliable and secure encryption the AES encryption process is implemented using well-known cryptographic libraries. These libraries pre-built encryption and decryption functions for key generation make the implementation process dependable and effective. The entire security of the file upload and verification process is improved by the system's use of AES encryption which guards sensitive data like file hashes and encryption keys against manipulation and unwanted access [43].

4. Result and Discussion

4.1. Result

This study concentrates on SHA256 and HMAC-based secure file transfers between platforms and devices, and authenticates and verifies source files. The findings demonstrate that both HMAC algorithms and cryptographic hashes can be used to maintain file security. As an efficient integrity check to ensure that files have not been altered or tampered with while being transferred or stored SHA256 is used to give each file a unique signature, this will then stop unintentional changes. These results demonstrate the effectiveness of SHA256 to protect data integrity and provide a reliable means of identifying attempted manipulation. Because HMAC algorithms provide a unique digital signature to every file, they will enhance file security. By combining the cryptographic hashes with secret keys this digital signature verifies the legitimacy of the file.

The study has proven that by verifying that files are genuine and have not been changed by unauthorized parties, HMAC is able to provide an additional layer of security. Notwithstanding the innate advantages, the investigation also recognized a number of obstacles. In order to implement these cryptographic techniques one must have a solid understanding of their underlying principles and exercise caution to prevent any potential vulnerabilities. However cryptographic technologies such as SHA256 and HMAC can greatly improve file security in the digital age if implemented correctly and continued to be improved.

4.2. Postman

To test whether the system works or not, it is done using Postman. Postman was chosen because it can fulfill requests that you want to use such as POST, DELETE, GET, EDIT. And postman can real-time do all these requests by directly entering data into the database. Using a postman is quite easy, just enter the fire that has been set then enter the request according to the fire, and enter the body according to the provisions. Here are some test results using Postman.

4.2.1. Register

In the system created, it is necessary to register first so that the system can find out and identify who sent the file in order to maintain the originality of the file. The importance of user information in sending files to avoid piracy and manipulation of file data. When registering, it will perform a POST to send information to the database. Example of implementation as follows:

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 1 July 2024 doi:10.20944/preprints202407.0075.v1

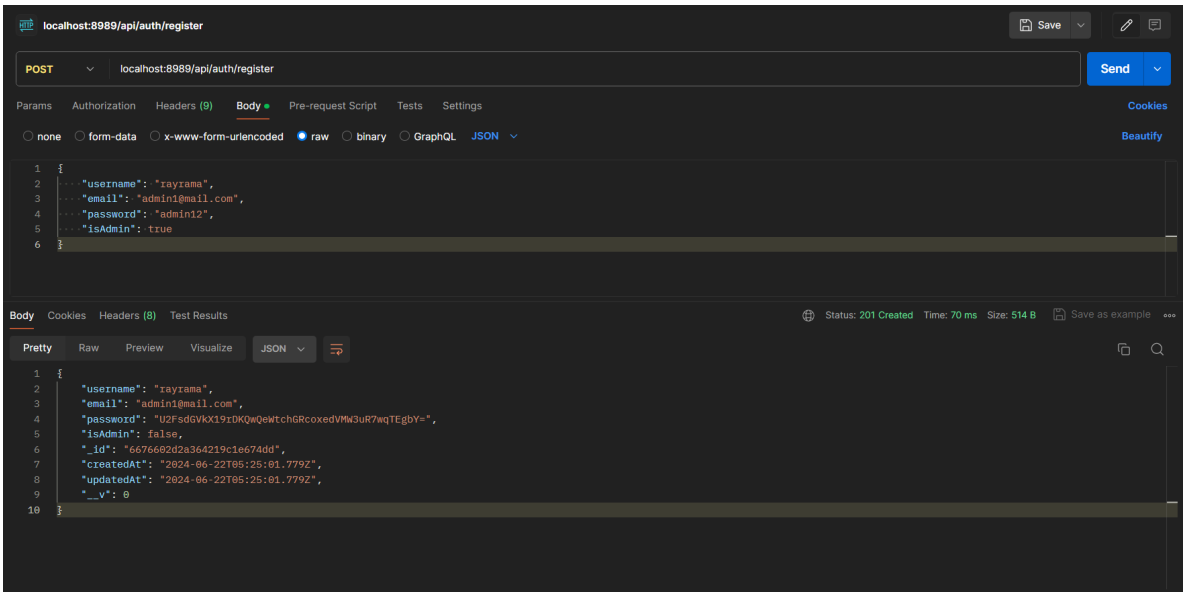

Figure 12. Example of Postman Register User

After registering, the data of the registered user will automatically be stored in the database. The data stored includes ID, username, email, password, role (admin/not admin), and creation date. All of this information is stored to make it clearer about the source of the file being sent. At the time of successful registration, the system will automatically provide a different access token for each account to distinguish each user and characterize the user. Access tokens are also useful for bearer tokens when sending files and also as authorization when sending files.

4.2.2. Login

To send a file, the user must login with the account that has been registered to the system. The login is required to identify the user in sending files and the access token is required to send files. In logging in, a POST request is required with a username and password body. The user must enter the same username and password as the registered account in the database. Example of login usage:

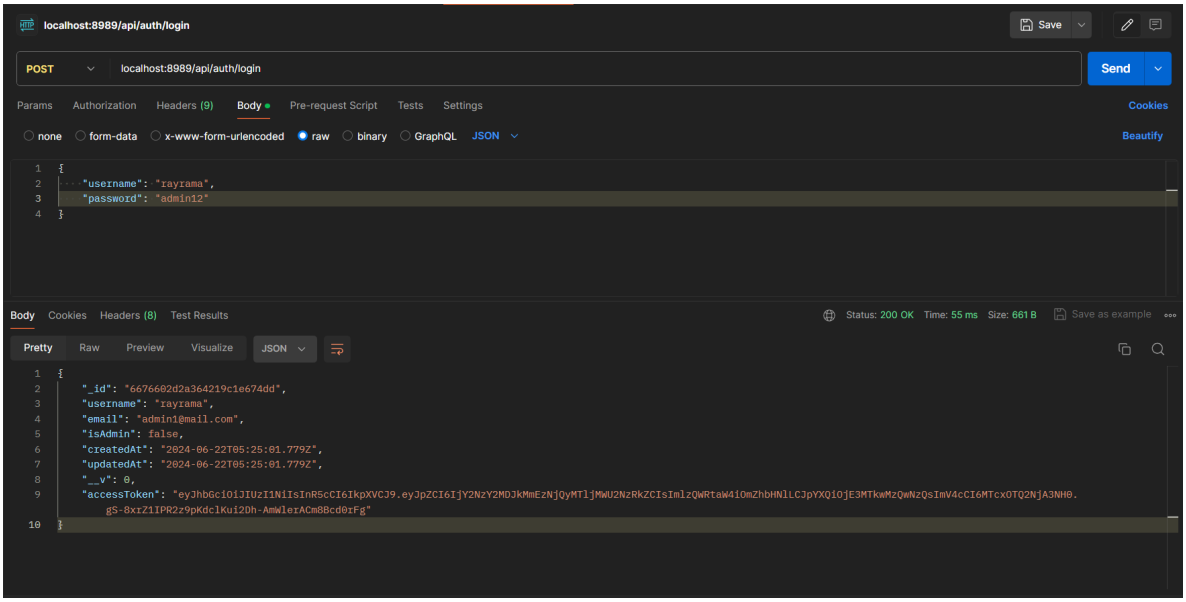

Figure 13. Example of Postman Login User

 doi:10.20944/preprints202407.0075.v1

If the login is successful, it will read the ID, username, email, role, creation date & change date, and access token. Very important data in the account is the access token where the access token is unique and will be different for each registered account and will be important data in sending files to source file authenticity and file delivery. Here's an example of an account that has successfully logged in:

4.2.3. Upload

To upload, users are required to log into a registered account because this process requires an access token to be used as an authorization header. The access token is unique data and must be different for every login session. Access tokens are needed so that only registered users can upload or check the authenticity of image files.

In this scenario, the backend performs encryption not only on the file hash, but also encrypts the "keyEncryptor" into the same AES hash as the "password" field. This is done to secure the "keyEncryptor" so that it is not easy to read and the uploader must really remember it.

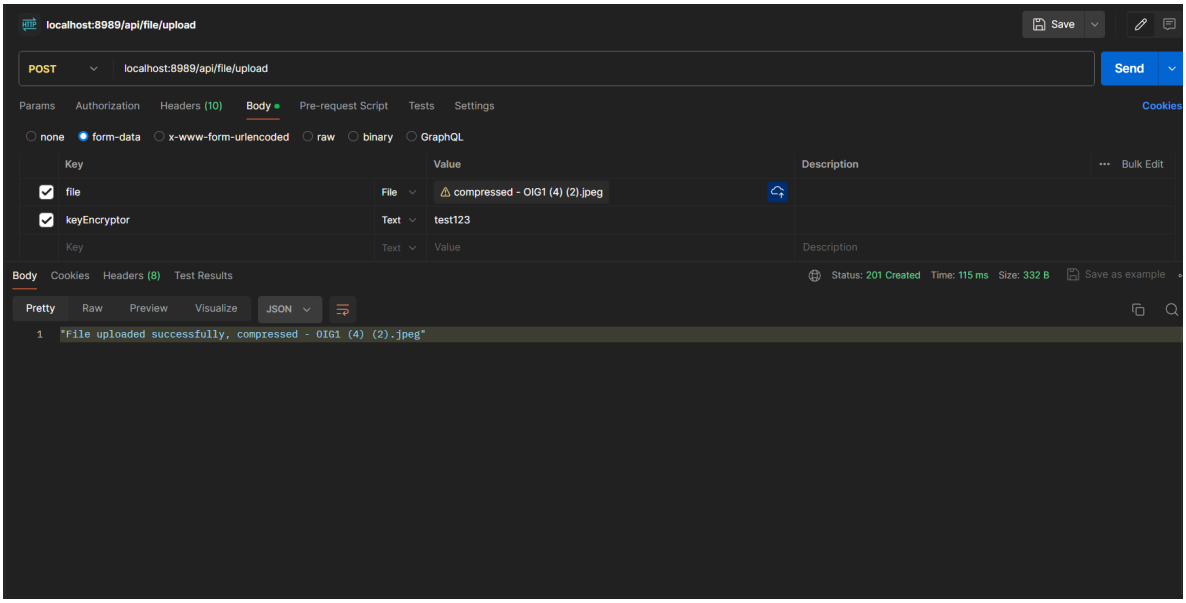

Figure 14. Success uploaded file to server

When successfully uploading the image file, the system will respond with a short answer without displaying details such as the image ID or encrypted hash. This is done for security reasons. But in the database, the image file that is uploaded should be like this:

 doi:10.20944/preprints202407.0075.v1

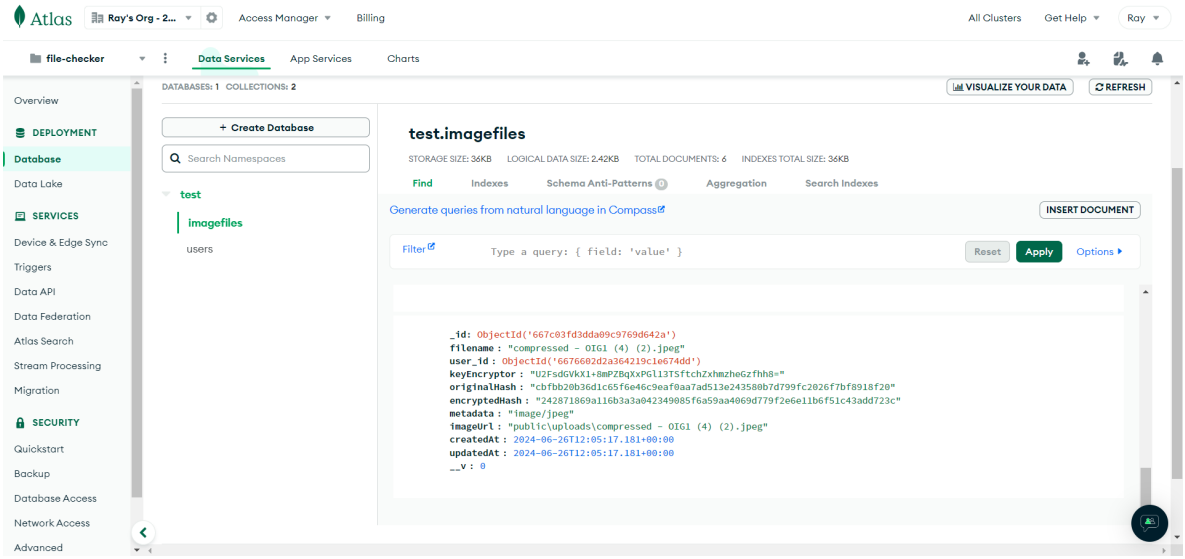

Figure 15. Uploaded File Image in Database

All encryption processes have been done in the backend automatically because if all processes run in the frontend, other people can manipulate all data.

4.2.4. File Authority and Source Check

To perform file checking, the user needs a key that was previously determined by the person who uploaded the image file earlier. This is done to ensure that only people who have the key can check whether the files are the same or not.

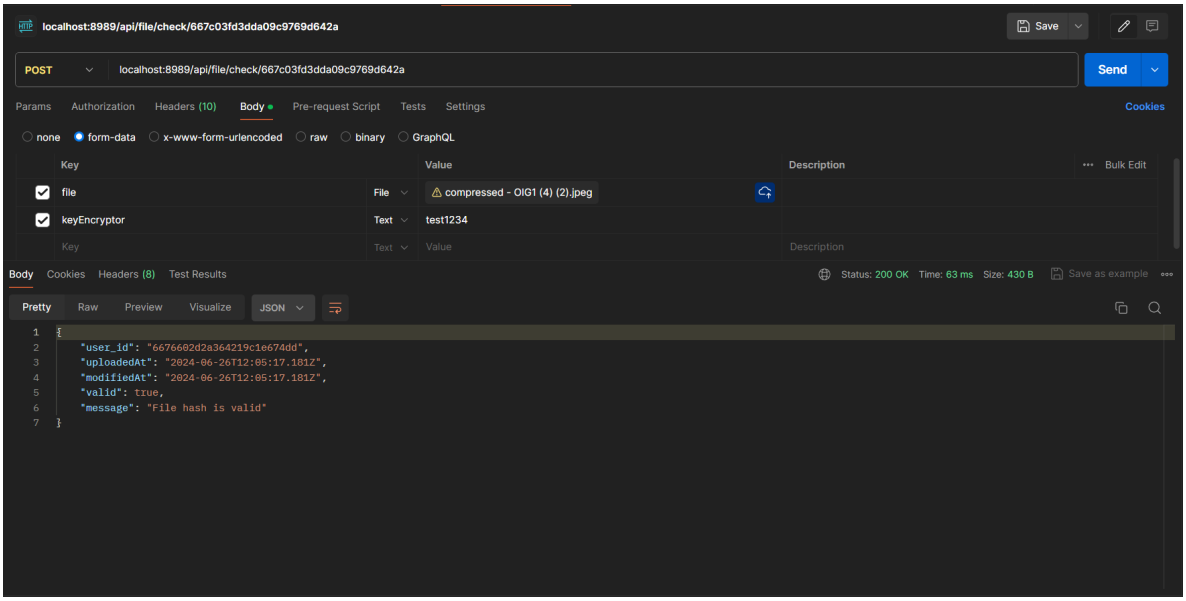

Figure 16. When file hash and keyEncryptor are matched

In this scenario, if the file hash that is checked is the same as the one in the database but does not enter a valid "keyEncryptor" or the same as the data in the database, the system will still reject it because the "keyEncryptor" is used to encrypt the file hash if the value is different, the encryption result will be different. The backend way to verify the sent "keyEncryptor" with the one in the database is to perform AES encryption on the inputted "keyEncryptor" and perform hash matching on the "keyEncryptor" data that has been hashed in the upload scenario.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 1 July 2024 doi:10.20944/preprints202407.0075.v1

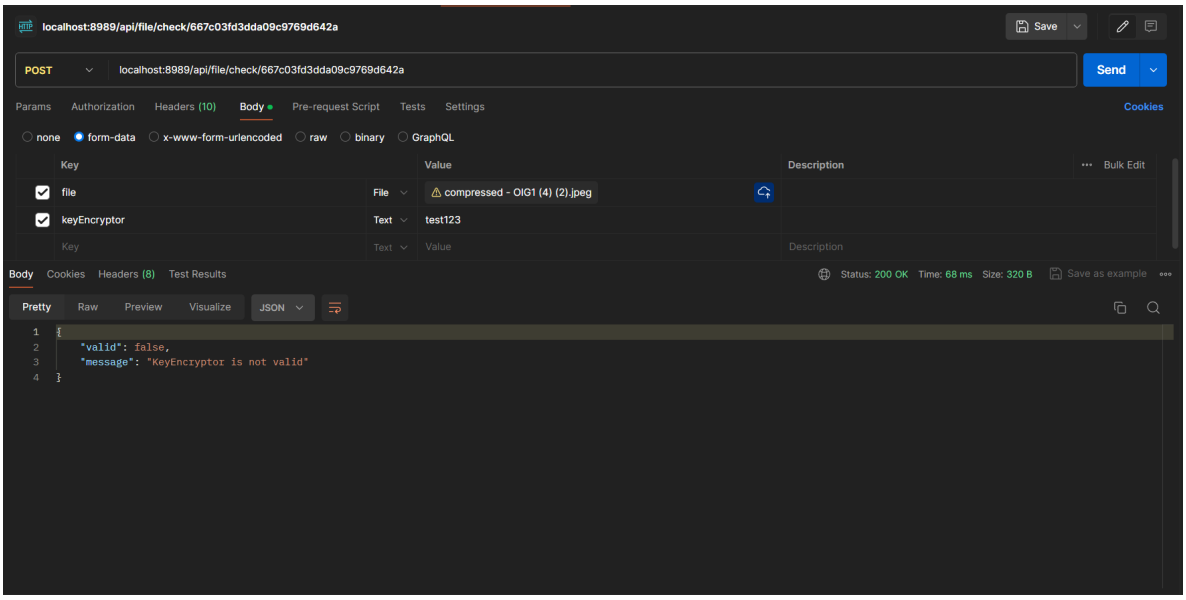

Figure 17. When file hash and keyEncryptor are not matched

4.3. Discussion

The study's emphasized the significance of securing file integrity by utilizing the SHA256 and HMAC algorithms. The outcomes confirm how well these cryptographic methods work to protect the security and integrity of files. As a strong integrity check to make sure that files won't change while being transferred or stored SHA256 is used to uniquely sign each file. This provides a robust defense against unwanted modifications and a reliable way to identify any tampering.

The study illustrates how well SHA256 maintains data integrity. File security is further improved by using HMAC algorithms to produce a distinct digital signature for every file. The validity of the file is attested to by this digital signature which was produced by fusing cryptographic hashes and secret keys. An extra degree of security is added to the overall security framework by HMAC which makes sure that files haven't been changed by unauthorized parties.

HMAC also adds another degree of security by guaranteeing the legitimacy of files. However an understanding of the underlying theories of SHA256 and HMAC is necessary in order to use them effectively. To avoid any possible weak spots that could compromise file security implementers must exercise caution. This study shows that SHA256 and HMAC are two examples of regularly developed and appropriately implemented cryptographic techniques that can greatly enhance file security in the digital age.

By integrating HMAC algorithms and cryptographic hashes this method provides a complete security solution that prevents unauthorized changes and ensures file authenticity. As digital communication and data sharing increase robust cryptography solutions become more and more important. Potential for future research include SHA256 and HMAC algorithm applications to further augment and enhance file transfer security. Since improved implementations of these techniques could reduce the computational burden and also improve performance in resource-constrained places more research is needed in this area.

To build multi-layer security frameworks that are more resilient, future research could expand on the study of quantum-resistant cryptographic algorithms which is a promising field for further investigation. The development of quantum computing technology could make conventional cryptographic techniques more susceptible to attacks from these machines. The long-term security of digital file transfers could be guaranteed by creating and utilizing quantum-resistant algorithms.

Further research endeavors could examine the viability of utilizing machine learning techniques to expeditiously detect and address potential vulnerabilities. Improving security and taking preventive action will possibly be made simpler by training artificial intelligence models to recognize patterns

 doi:10.20944/preprints202407.0075.v1

of illegal access or manipulation. Examining the potential of using blockchain applications may also be important for secure file transfers. Immutable decentralized file transfers made possible by blockchain technology allow for the tracking and recording of modifications. As a result of it file transfers will become more reliable and secure. Further investigation into the security and effectiveness of cryptographic techniques is necessary to ensure secure data transfers because these fields address current problems and guard against potential threats in the digital sphere.

5. Conclusions

In the age of technology, the growth of information technology poses new challenges related to the security and integrity of digital files. File exchanges that occur between devices have the risk of data manipulation and the insertion of malicious and unauthorized content, which has a serious impact on user security and privacy. To meet these challenges, cryptography with hash methods such as SHA256, AES encryption, HMAC (Hash-Based Message Authentication Code) algorithm, and other hash algorithms and functions play a key role in the technology field. Hash functions such as SHA256 generate a unique fingerprint for each file to verify its integrity, while AES encryption and HMAC ensure the authenticity and integrity of data through digital signatures using secret keys. Combining the two methods together ensures that the data or file remains authentic and unchanged during transmission.

Not only does it play a role in maintaining data security, but cryptography also plays a role in building trust between users and systems. Through verification mechanisms of file integrity and authenticity, users can be assured that the information received or transmitted has not been manipulated by unauthorized third parties. While cryptography offers an effective solution for maintaining file security and integrity, it is also important to continuously develop new methods and improve system security. Cyber threats are evolving every day and the cryptography community's involvement in research and development is crucial to addressing these challenges.

Therefore, cryptography is not only a solution to today's data and file security and integrity challenges, but also an important foundation to drive future technological innovation and development. Through continuous research and development, we can continue to strengthen information security systems and ensure that our data remains safe and secure in the ever-evolving digital age.

Acknowledgment

The author's wishes to acknowledge the Informatics Department UIN Sunan Gunung Djati Bandung, which partially supports this research work.

References

1. Andrew, J.; Isravel, D.P.; Sagayam, K.M.; Bhushan, B.; Sei, Y.; Eunice, J. Blockchain for healthcare systems: Architecture, security challenges, trends and future directions. *Journal of Network and Computer Applications* **2023**, p. 103633.
2. Zheng, Z.; Xie, S.; Dai, H.; Chen, X.; Wang, H. An overview of blockchain technology: Architecture, consensus, and future trends. 2017 IEEE international congress on big data (BigData congress). Ieee, 2017, pp. 557–564.
3. Dong, S.; Abbas, K.; Li, M.; Kamruzzaman, J. Blockchain technology and application: an overview. *PeerJ Computer Science* **2023**, *9*, e1705.
4. Shaker, S.H. HMAC modification using new random key generator. *IRAQI Journal of Computers, Communication and Control and Systems Engineering* **2014**, *14*, 72–82.
5. David, D.S.; Anam, M.; Kaliappan, C.; Selvi, S.; Sharma, D.K.; Dadheech, P.; Sengan, S. Cloud Security Service for Identifying Unauthorized User Behaviour. *Computers, Materials & Continua* **2022**, *70*.
6. Sood, N. Cryptography in Post Quantum Computing Era. *Available at SSRN 4705470* **2024**.
7. Nasution, R.M. Implementasi Metode Secure Hash Algorithm (SHA-1) Untuk Mendeteksi Orisinalitas File Audio. *Bulletin of Computer Science Research* **2022**, *2*, 73–84.

8. Quist-Aphetsi, K.; Senkyire, I.B. Validating of Digital Forensic Images Using SHA-256. *2019 International Conference on Cyber Security and Internet of Things (ICSIoT)* **2019**, pp. 118–121. doi:10.1109/ICSIoT47925.2019.00028.
9. Nainggolan, S. Implementasi Algoritma SHA-256 Pada Aplikasi Duplicate Document Scanner. *Resolusi : Rekayasa Teknik Informatika dan Informasi* **2022**, *2*, 201–213. doi:10.30865/resolusi.v2i5.368.
10. Harinath, D. Enhancing Data Security Using Elliptic Curve Cryptography in Cloud Computing. *International Journal of Science and Research (IJSR)* **2016**, *5*, 1884–1890. doi:10.21275/v5i7.ART2016624.
11. Gilchrist, J. Enhanced Elliptic Curve Digital Signature Algorithm Authentication System Utilising Cryptographic Key Tweaking **2024**.
12. Jahan, F.; Mostafa, M.; Chowdhury, S. SHA-256 in Parallel Blockchain Technology: Storing Land Related Documents. *International Journal of Computer Applications* **2020**, *175*, 33–38. doi:10.5120/ijca2020920911.
13. Agarwal, U.; Rishiwal, V.; Tanwar, S.; Chaudhary, R.; Sharma, G.; Bokoro, P.N.; Sharma, R. Blockchain Technology for Secure Supply Chain Management: A Comprehensive Review. *IEEE Access* **2022**, *10*, 85493–85517. doi:10.1109/ACCESS.2022.3194319.
14. Tatineni, S. MACHINE LEARNING APPROACHES FOR ANOMALY DETECTION IN CYBERSECURITY: A COMPARATIVE ANALYSIS. *INTERNATIONAL JOURNAL OF COMPUTER ENGINEERING & TECHNOLOGY* **2021**, *12*, 42–50.
15. Devineni, S.K.; Kathiriya, S.; Shende, A. Machine Learning-Powered Anomaly Detection: Enhancing Data Security and Integrity. *Journal of Artificial Intelligence & Cloud Computing* **2023**, pp. 1–9. https://doi.org/10.47363/JAICC/2023(2)184.
16. Building Applications with Serverless Architectures, 2024.
17. Cinar, B. The Rise of Serverless Architectures: Security Challenges and Best Practices. *Asian Journal of Research in Computer Science* **2023**, *16*, 194–210. doi:10.9734/ajrcos/2023/v16i4382.
18. Sojitra, L. Cryptography Behind WhatsApp's End-to-End Encryption and Its Security Challenges, 2024.
19. Salsabiila. Analysis of End-to-End Encryption Implementation Across Different Meta's Applications, 2023.
20. Kumar, R.; Charu, S.; Bansal, S. Effective way to handling big data problems using NoSQL Database (MongoDB). *J. Adv. Database Manag. Syst* **2015**, *2*, 42–48.
21. MongoDB Documentation. https://www.mongodb.com/docs/. Accessed: 2024-06-17.
22. Docs, M.W. Express - Node.js web application framework. https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs. Accessed: 2024-06-17.
23. Megida, D. Multer: Easily upload files with Node.js and Express. https://blog.logrocket.com/multer-nodejs-express-upload-file/, 2022.
24. Next.js documentation.
25. Saraswathi, D.M.; Satya, D.; Sowmya, T.N.S. Student Portfolio Designing using Nextjs. *International Journal of Soft Computing and Engineering* **2023**, *13*, 21–24. doi:10.35940/ijsce.A3598.0313123.
26. Postman. https://www.postman.com/product/tools/.
27. What is Postman? https://www.postman.com/product/what-is-postman/.
28. Postman Collaboration, 2024. Retrieved from https://www.postman.com/product/integrations/.
29. Dworkin, M.J. Advanced Encryption Standard (AES), 2023. doi:10.6028/NIST.FIPS.197-upd1.
30. Abdullah, A. Advanced Encryption Standard (AES) Algorithm to Encrypt and Decrypt Data **2017**.
31. Nayak, A.; Poriya, A.; Poojary, D. Type of NOSQL databases and its comparison with relational databases. *International Journal of Applied Information Systems* **2013**, *5*, 16–19.
32. pragimtech. Relational and non relational databases.
33. Wu, L.; Yuan, L.; You, J. Survey of large-scale data management systems for big data applications. *Journal of computer science and technology* **2015**, *30*, 163–183.
34. Kanade, A.; Gopal, A.; Kanade, S. A study of normalization and embedding in MongoDB. 2014 IEEE International Advance Computing Conference (IACC). IEEE, 2014, pp. 416–421.
35. Chat, D.S. File Upload using Nodejs, Express Multer. https://deadsimplechat.com/blog/file-upload-using-nodejs-multer-express/, 2023.
36. Express - Node.js web application framework. https://expressjs.com/. Accessed: 2024-06-17.
37. Sarwar, M.I.; Maghrabi, L.A.; Khan, I.; Naith, Q.H.; Nisar, K. Blockchain: A Crypto-Intensive Technology-A Comprehensive Review. *IEEE Access* **2023**.
38. Azeez, N.A.; Chinazo, O.J. ACHIEVING DATA AUTHENTICATION WITH HMAC-SHA256 ALGORITHM. *Computer Science & Telecommunications* **2018**, *54*.

 doi:10.20944/preprints202407.0075.v1

39. Krawczyk, D.H.; Bellare, M.; Canetti, R. HMAC: Keyed-Hashing for Message Authentication. RFC 2104, 1997. doi:10.17487/RFC2104.
40. Rescorla, E. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, 2018. doi:10.17487/RFC8446.
41. 3rd, D.E.E.; Jones, P. US Secure Hash Algorithm 1 (SHA1). RFC 3174, 2001. doi:10.17487/RFC3174.
42. M'Raihi, D.; M'Raihi, D.; Hoornaert, F.; Naccache, D.; Bellare, M.; Ranen, O. HOTP: An HMAC-Based One-Time Password Algorithm. RFC 4226, 2005. doi:10.17487/RFC4226.
43. Buchanan, W.J.; Li, S.; Asif, R. Lightweight cryptography methods. *Journal of Cyber Security Technology* **2017**, *1*, 187–201. doi:10.1080/23742917.2017.1384917.