APPLIED INFORMATION TECHNOLOGY AND COMPUTER SCIENCE
e-ISSN: 2773-5141

AITCS

Vol. 5 No. 2 (2024) 232-253
https://publisher.uthm.edu.my/periodicals/index.php/aitcs

Secure Messaging using HIGHT for Android Smartphones

Khok Xuan Han[1], Sapiee Jamel[1*],

[1] *Faculty of Computer Science and Information Technology, Universiti Tun Hussein Onn Malaysia, Parit Raja, Batu Pahat, 86400, MALAYSIA*

*Corresponding Author: sapiee@uthm.edu.my
DOI: https://doi.org/10.30880/aitcs.2024.05.02.013

Article Info

Received: 17 July 2024
Accepted: 16 October 2024
Available online: 15 December 2024

Keywords

Secure Messaging, Android, Encryption, HIGHT

Abstract

In today's digital world, secure mobile communication is essential. This project explores the integration of the HIGHT (High Security and Lightweight) algorithm and End-to-End Encryption (E2EE) on Android smartphones. The goal is to create a secure messaging application using HIGHT, addressing the need for efficient, secure communication. Traditional methods like Advanced Encryption Standard (AES) are too resource-intensive for Android devices. The proposed chat application uses the HIGHT algorithm with HMAC-SHA256 for confidentiality and authenticity. Key features include user authentication via phone number and OTP verification through Firebase, and secure key management using Elliptic Curve Diffie–Hellman (ECDH). The application provides end-to-end encrypted messaging, secure file sharing, and a user-friendly interface, ensuring privacy and data security while maintaining performance. The application was developed using Java, Android Studio, and Firebase, following Object-Oriented Software Development (OOSD) principles. Security was thoroughly tested using Mobile Security Framework (MobSF) and AppSweep, identifying areas for improvement and ensuring industry standards adherence. User acceptance testing with 30 beta users from UTHM confirmed the application's functionality and user-friendliness. Additionally, encryption time performance was evaluated, with Logcat outputs showing the HIGHT algorithm's encryption process consistently ranges between 0 ms and 1 ms. This low latency is crucial for real-time messaging, demonstrating the high resource efficiency in terms of algorithm's efficiency and scalability. Overall, the measured encryption times are excellent, reflecting the algorithm's suitability for real-time messaging while ensuring robust security.

1. Introduction

In today's digital world, messaging applications on smartphones are used to exchange sensitive information daily, including private conversations, financial transactions, medical records, and business communications. Protecting the confidentiality and integrity of these messages is essential to safeguard users' privacy and prevent unauthorized access [1]. Encryption is crucial for secure communication, with the choice of algorithm being key to security levels. The Advanced Encryption Standard (AES) is a widely used and important block cipher for both hardware and software applications [2]. However, for devices with limited resources like Android smartphones, AES can be too resource intensive. A study by [3] tested the impact of different encryption methods on smartphone battery life, finding that AES used the most power. Other algorithms like RC6, Bluefish, and DES also consumed significant power, making them impractical for mobile use. The

researchers suggest developing new, more efficient encryption algorithms because traditional methods are too slow and power-hungry for devices with limited storage, power, and processing capabilities. Devices like RFID tags, medical equipment, smartphones, and embedded smart objects typically have limited storage, power, and processing capabilities [4].

To address this challenge, this project focuses on developing a secure messaging application for Android smartphones using the HIGHT encryption algorithm. HIGHT (HIGh security and lightweigHT) is a block cipher with a 64-bit block size and a 128-bit key length, making it suitable for lightweight hardware implementations such as embedded CPUs in nano-sensor network systems, smartphones, and RFID tags. The HIGHT block cipher was developed by the National Security Research Institute (NSRI) and the Korea Information Security Agency (KISA) [5]. It is an ISO/IEC international standard block cipher, included in ISO/IEC 18033-3:2010 [6]. The objectives of this project include:

1. Selecting and implementing HIGHT for message encryption.
2. Developing the Android application.
3. Evaluating the performance and security of the implemented algorithm.

The scope is limited to secure personal messaging and excludes large-scale industrial or commercial communication solutions. Key features of the proposed messaging application include end-to-end encryption using the HIGHT algorithm, real-time message transmission, secure messaging, locking chat for privacy, and a user-friendly interface. The project's significance lies in addressing privacy and data security concerns, contributing to advancements in secure communication solutions, empowering users to protect their privacy, serving as an educational resource, and raising awareness about cybersecurity issues.

In this paper, Section 2 discusses the related works and system that are currently available in the market and provides a comparison between the available system and the proposed system. Section 3 outlines the technical implementation, detailing the chosen object- oriented software development approach chosen for its emphasis on modularity, maintainability, and code reusability. Each phase is outlined with its specific activities and goals. Section 4 explains the system analysis and design, outlining the architecture and components of the HIGHT-powered messaging application. Finally, the concluding section summarizes the achieved work and outlines exciting possibilities for future contributions in secure mobile communication.

2. Related Work

This section covers secure messaging, Android Studio security features, encryption algorithms, and a comparison of different secure messaging systems. It provides an overview of the current landscape and challenges in developing secure communication platforms.

2.1 Secure Messaging

The prevalence of instant messaging and social networks like WhatsApp and Telegram underscores their vital role in personal and professional communication [7]. With these platforms accessible on mobile phones and computers, securing these channels is crucial. Secure messaging involves protecting communication infrastructures, such as emails and messaging apps, through mechanisms like End-to-End Encryption (E2EE) [1]. It ensures the protection of information exchanged between individuals or groups [8], safeguarding sensitive data, such as health information, transmitted beyond an organization's boundaries [9]. Secure messaging not only protects privacy but also personal information, financial data, and confidential conversations from unauthorized access. It builds trust, ensures regulatory compliance, mitigates financial and legal risks, and enhances customer confidence and loyalty. It also defends against cyber threats like malware and ransomware [10]. However, implementing secure messaging effectively faces challenges, particularly in establishing trust, ensuring conversation security, and maintaining transport privacy [11]. Trust establishment involves verifying communication parties, distributing cryptographic keys, and validating associations with the owning entity. Conversation security focuses on protecting messages, while transport privacy aims to conceal communication metadata. Addressing these challenges is essential for maximizing the benefits of secure messaging in today's digital world.

2.2 Android Studio Security

Android, an open-source operating system led by Google, holds a significant 70.5% market share globally as of Q3 2023, with Apple's iOS at 28.8% [12]. This widespread adoption makes Android a crucial platform for secure messaging solutions. This section reviews Android's security features and their potential to enhance secure messaging applications.

- **Application Sandbox**: Android employs Linux user-based protection to isolate application resources, assigning a unique user ID (UID) to each app, ensuring security by restricting interactions and system access [13]. This sandbox model, based on UNIX-style user separation, effectively protects against malicious actions from other apps [14].

- **Application Signing**: All Android apps must be signed by their developers. This process authenticates the developer and ensures that apps are genuine and unaltered when delivered via Google Play or the device's package installer [14].
- **Authentication**: Android uses user-authentication-gated cryptographic keys and supports multiple authentication methods, including fingerprint sensors and the Gatekeeper subsystem for pattern/password authentication within a Trusted Execution Environment (TEE). Enhanced security features in Android 9 include Protected Confirmation for crucial transactions [14].
- **Biometrics**: Android 9 and higher versions feature the BiometricPrompt API, allowing developers to integrate biometric authentication. Strong biometrics are required for this API, providing a balance between convenience and security [14].
- **Data Encryption and Keystore**: Android offers symmetric encryption keys, and a hardware backed Keystore for secure key management and cryptographic operations [14].
- **Security-Enhanced Linux (SELinux)**: SELinux enforces mandatory access controls over all processes, including those with root privileges, enhancing protection against malicious software and code vulnerabilities [14].
- **Trusty TEE and Verified Boot**: Trusty provides a secure OS isolated from the main system, while Verified Boot ensures that all executed code is from a trusted source, protecting against unauthorized code execution and corruption [14].

In conclusion, Android's comprehensive security features make it an ideal platform for developing secure messaging applications. Effective utilization of these features can significantly enhance the security and privacy of messaging communications on Android devices.

2.3 Encryption

At its core, encryption protects information by using mathematical algorithms to scramble data, making it accessible only to those with the correct key to unscramble it [15]. This process turns readable "plaintext" into unreadable "ciphertext" and requires a decryption key—a sequence of numbers or a password generated by an algorithm—to revert it back to plaintext. Secure encryption methods use numerous cryptographic keys, making it difficult for unauthorized individuals to guess or compute the correct key through brute force attacks. The two main types of encryption algorithms are symmetric and asymmetric.

- **Symmetric Encryption**: Uses a single secret key for both encryption and decryption. Common methods include AES (Advanced Encryption Standard), DES (Data Encryption Standard), IDEA (International Data Encryption Algorithm), and Blowfish [16].
- **Asymmetric Encryption**: Also known as Public-Key Cryptography, uses two different keys: a public key for encryption and a private key for decryption. Common methods include RSA (Rivest Shamir Adleman), DSS (Digital Signature Standard), ECC (Elliptical Curve Cryptography), and Diffie-Hellman [16].

Due to the small size, limited computational power, and restricted memory and energy resources of devices like smartphones, traditional encryption algorithms can be too resource-intensive for these environments [17]. Therefore, there's a growing need for lightweight encryption algorithms that can efficiently secure information on such devices. In summary, encryption is essential for protecting information by converting it into a secure format accessible only to those with the correct key. However, there's a need for more efficient, lightweight encryption methods for devices with limited resources.

2.4 HIGHT Algorithm

HIGHT, is a lightweight encryption algorithm with a 64-bit block length and 128-bit key length, making it suitable for low-cost, low-power, and resource-constrained implementations. Its 32-round iterative structure, based on a generalized Feistel network variant, employs simple operations like XOR, addition mod 2^8, and left bitwise rotation, making it more hardware oriented. This simplicity results in remarkably low resource consumption, rendering HIGHT ideal for platforms with limited resources [5].

The encryption process of HIGHT involves key scheduling, initial transformation, round function, and final transformation. This process leverages whitening keys and subkeys, denoted as WK and SK, and employs mathematical operations such as addition modulo 2^8, subtraction modulo 2^8, and XOR. The round function, executed for 32 rounds, involves dividing the data block, applying a round function with a subkey and round constant, and swapping the halves for enhanced data mixing. The final transformation acts as a finishing touch, ensuring thorough data mixing and protection through the application of secret keys [5].

The design principles of HIGHT are tailored to offer flexibility in designing inner auxiliary functions while ensuring a lightweight round function. Its structure is generalized Feistel-like, providing similar costs for both encryption and decryption processes, simplifying circuit implementation. HIGHT's strategic combination of operations enhances resistance against attacks, and its key schedule algorithm utilizes a single 128-bit register

for both encryption and decryption. The incorporation of whitening keys in the first and last rounds prevents direct exposure of inputs to certain functions from plaintexts and ciphertexts. Additionally, the sequence generated by a linear feedback shift register enhances subkey byte randomness, increasing resistance against slide attacks. HIGHT's design principles prioritize efficiency, security, and adaptability to specific processor environments, making it a compelling option for resource-constrained platforms [5]. Figure 1 shows the overview of HIGHT encryption process.

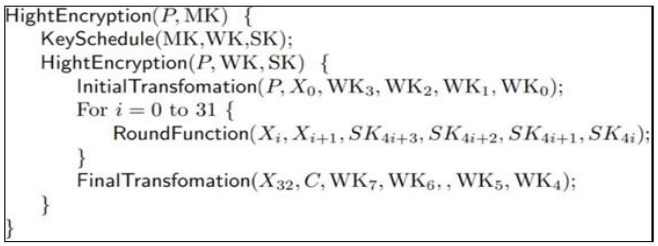

Fig.1 *Overview of HIGHT encryption process* [5]

2.5 System Comparison

Facebook Messenger is a popular messaging application with various features for engaging with customers. It has an intuitive user interface that emphasizes the chat window and offers easy access to conversations. The platform includes multimedia functions, personalization options, and security features such as encryption and two-factor authentication [18]. However, there are concerns about its closed-source E2EE protocol and data privacy issues following a significant data breach in 2021 [19].

WeChat, launched by Tencent in 2011, has evolved into a comprehensive social platform with over 1 billion monthly active users. Initially a text messaging app, it now includes gaming, payment services, and social networking functions. The main interface features chat tabs for easy navigation [20]. Concerns arise from the lack of security features like end-to-end encryption and extensive metadata collection [21].

Signal was launched in 2014 and created by Moxie Marlinspike. Signal's user interface is intuitive and familiar, allowing easy navigation for users. The app prioritizes security through end-to-end encryption using the open-source Signal Protocol, ensuring secure communication across text messages, calls, and media transfers. It also includes privacy features like disappearing messages and screen security against screenshots. Additionally, Signal does not collect metadata to strengthen user privacy. This combination of usability with strong encryption makes Signal a standout option in the secure messaging realm—a potential blueprint for applications seeking a balance between user experience and robust security and privacy [22].

Table 1 presents a thorough comparison of its capabilities with those of well-known platforms such as Facebook Messenger, WeChat, and Signal. This analysis showcases the strengths and weaknesses of each platform in terms of interface, features, security protocols, and data privacy practices. By carefully assessing these aspects, both developers and users can gain valuable insights into the distinctive value proposition provided by the HIGHT-powered solution and how it addresses the crucial challenges of balancing user experience with strong security and privacy in today's digital environment.

Table 1 *Comparison between the existing application and the proposed application*

System/Feature	Facebook Messenger [18]	WeChat [20]	Signal [22]	Proposed System
Encryption Algorithm	AES	AES	Signal Protocol (uses AES)	HIGHT
End-to-End Encryption (E2EE)	Optional (Secret Conversations)	No	open-source Signal Protocol	ECDH with ecp256r1
Key Management	Closed source	Closed source	Open-source, secure key management	Secure key management via Android Keystore
Privacy Features	2FA, disappearing messages, blocking	Limited	Disappearing messages, screen security	Locking chat, secure messaging

Table 1 *Comparison between the existing application and the proposed application (cont.)*

System/Feature	Facebook Messenger [18]	WeChat [20]	Signal [22]	Proposed System
Multimedia Functions	Image/video sharing, voice/video calls	Extensive multimedia and social features	Image/video sharing, voice/video calls	Image sharing, voice calls
User Authentication	2FA, login alerts	Phone number, password, SMS verification	Phone number, one time password, and PIN	Phone number and one time password
Message Authenticity	Not specified	Not specified	Verified with Signal Protocol	Ensured with HMAC-SHA256
Resource Efficiency	Moderate	Moderate	High	High, optimized for low resources

3. Methodology

This project will design and develop by using the object-oriented software development methodology. There are Object-oriented software development methodology has four main phases, which are requirement analysis, object-oriented analysis, object-oriented design and implementation and testing.

3.1 Object-Oriented Requirement Analysis

Object-oriented requirement analysis is vital in object-oriented design and development, focusing on analyzing and specifying a system's needs in terms of objects, their interactions, and behaviors. Before detailed analysis or modeling begins, it is essential to define the proposed application, including its objectives and project scope. The functionality and features of the proposed application are detailed through problem statements, expected outcomes, and project significance. Requirement analysis involved reviewing research papers, online blogs, and websites, and analyzing weaknesses in existing systems to gather diverse perspectives and insights from a broad user base prioritizing effectiveness and privacy. Both software and hardware tools are crucial for development. The hardware requirements for the proposed messaging encryption application are outlined as follows: The development will be conducted on a TUF Gaming Asus Laptop equipped with an Intel(R) Core (TM) i5-10300H CPU @ 2.50GHz, 16 GB of RAM, and Windows 11 64-bit operating system. Additionally, the application will be tested on a mobile device running Android Version 11.0 to ensure compatibility and performance. The software requirements include various tools for different aspects of development and documentation. Microsoft Word will be used for documentation, and Canva for creating slide presentations. Microsoft Project will help build the Gantt chart for project management. Android Studio will be the primary development environment for creating the mobile application in Java. SequenceDiagram.org will be used for drawing sequence diagrams, Uizard for wireframes, and Visual Paradigm for class and activity diagrams. Finally, Firebase will serve as the system's database to manage and store data securely. Additionally, a Gantt chart in Appendix A outlines the time estimation for each development stage.

3.2 Object-Oriented Analysis

The Object-Oriented Analysis (OOA) phase focuses on understanding the system and application requirements, which include user requirements, functional requirements, and non-functional requirements. The insights gained from the requirement analysis phase are used to plan and create the proposed application. A study of current applications—specifically WeChat, Signal, and Facebook Messenger—is conducted to understand their design and use these as guidelines for the proposed messaging encryption application using the HIGHT algorithm. During this phase, the new application is designed using Unified Modelling Language (UML). UML is used to illustrate:

- Use Case Diagrams: To show how users interact with the system.
- Sequence Diagrams: To illustrate the flow of messages between objects in the system.
- Activity Diagrams: To depict the flow of activities within the system.
- Class Diagrams: To demonstrate the relationships between classes in the system.

3.3 Object-Oriented Design

The object-oriented design phase involves creating the complete architecture of the proposed application by utilizing the results from previous stages to develop user interfaces and a database. User interface design is crucial for enabling interaction between the user and the application; it should be straightforward and easy to

use, guiding users effectively. Equally important is the database design, which provides a comprehensive data model for storing all information. The database must be constructed carefully to avoid redundant data. During this process, entities and attributes for the application are identified, along with their metadata—such as data type, size, and primary key—in the data dictionary. This ensures that all associated data is correctly stored and retrieved. By focusing on these aspects, the object-oriented design phase ensures a robust, user-friendly, and efficient application architecture.

3.4 Object-Oriented Software Development (OOSD)

In the final stage of OOSD, programming and testing are carried out using Java, with Android Studio for coding and Firebase for storing user data. Class objects and their relationships are implemented, and the interaction between user interfaces and the database is established. Thorough testing ensures the prototype aligns with project specifications, focusing on functionality, usability, bug identification, and error detection. User Acceptance Testing (UAT) confirms that the application meets expectations. After meeting all project criteria, maintenance addresses any remaining issues. Security testing is crucial, involving static analysis tools like Mobile Security Framework (MobSF) and AppSweep. MobSF is an open-source mobile application security testing tool that performs comprehensive static analysis on Android applications to identify security vulnerabilities, privacy issues, and compliance gaps by examining the application's code and configuration files. AppSweep, developed by Guardsquare, provides automated security analysis, focusing on identifying vulnerabilities and potential security issues. Both tools align with the OWASP Mobile Application Security Verification Standard (MASVS), ensuring a thorough security assessment.

4. System Analysis and Design

This section 4.1 will discuss the general system architecture. Section 4.2 will delve into the system requirement analysis. Section 4.3 will explain the Unified Modelling Language (UML), followed by Section 4.4, which will discuss the database design. Section 4.5 will explain the user interface design. Lastly, Section 4.6 will provide the conclusion.

4.1 System Analysis and Design

A general system architecture diagram illustrates the mapping of functionality onto hardware and software components, as well as how users interact with these elements. Figure 2 shows the application architecture for a message encryption application using HIGHT for Android smartphones. The system architecture diagram illustrates a secure messaging process between two users (User A and User B) on Android smartphones. Each smartphone employs the Elliptic Curve Diffie-Hellman (ECDH) protocol for key exchange, generating cryptographic keys securely managed by the Android Keystore. Messages are encrypted using the HIGHT algorithm, ensuring lightweight and efficient encryption. To verify message authenticity and integrity, HMAC-256 is used to generate a message authentication code (MAC). The process begins with User A composing a message, which is then encrypted with HIGHT and authenticated with HMAC-256. The encrypted message, MAC, and associated data (such as a nonce for added security) are transmitted to User B. Upon receipt, User B's device uses the shared secret key from ECDH, stored in the Android Keystore, to decrypt the message with HIGHT and verify the MAC with HMAC-256. This architecture ensures end-to-end encryption and authentication, maintaining the security, privacy, and integrity of the communication. Admin is allowed only to monitor users' activity via Firebase security features by leveraging the users and permission settings.

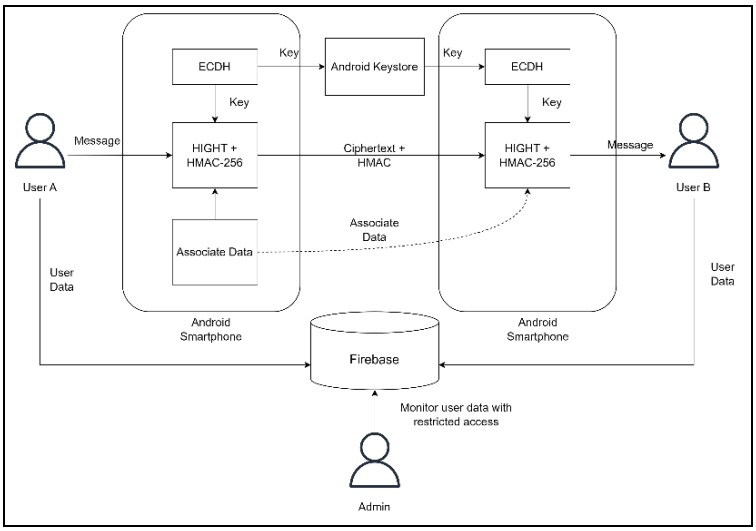

Fig.2 *General System Architecture Diagram*

4.2 System Requirement Analysis

The requirement analysis consists of system requirements and user requirements. Its main goal is to comprehend the application's usage within the specified flow and to ensure that the system is developed in accordance with user needs. Therefore, thorough analysis is required for essential aspects to ensure the efficient operation of the message encryption application. If these requirements are not met, issues related to installation or performance may occur.

4.2.1 Functional Requirement Analysis

All the functional requirement features that need to be integrated into the application should be included. These include defining input for the application, performing operations, and achieving specific outcomes as per user requirements. These functional requirements are crucial in ensuring that the finished application aligns with user needs. Below is Table 2 displaying all the functional requirements to be implemented in the message encryption application using HIGHT.

Table 2 *Functional Requirement of Proposed System*

No	Module	Functionality
1.	User Authentication	Register and log in users using phone number and OTP verification via Firebase.
2.	Key Management	Manage ECDH key pairs using Android Keystore for secure key exchange.
3.	Encryption	Encrypt and decrypt messages using HIGHT algorithm.
4.	Message Authentication	Generate and verify MACs using HMAC-256 for message integrity and authenticity.
5.	Key Exchange	Implement ECDH for secure key exchange between users.
6.	Real-Time Messaging	Ensure real-time transmission and reception of encrypted messages.
7.	View Message	Allow users to view received and sent messages.
8.	Delete Message	Enable users to delete messages and clear conversations.
9.	Group Chat	Allow users to create and participate in group chats.
10.	Edit Profile	Enable users to edit their profile information.
11.	Voice Call	Support real-time voice calls between users.
12.	Video Call	Support real-time video calls between users.
13.	Send Image	Allow users to send images.
14.	View User's Profile	Enable users to view other users' profiles.
15.	Clear Chat	Allow users to clear chat history.
16.	Search Conversation	Provide a search feature to find specific conversations.
17.	User Lists	Display a list of users.

4.2.2 Functional Requirement Analysis

Table 3 shows non-functional requirement analysis of the message encryption application. A non-functional requirement defines a software system's quality by presenting a set of criteria for evaluating the system's processes. These requirements are critical to ensuring the efficiency and productivity of the entire system. Similarly, attributes such as security, privacy, availability, reliability, scalability, and performance all indicate criteria used to evaluate proposed systems. Imposing constraints based on these requirements is crucial in designing systems that meet user needs effectively.

Table 3 *Non-Functional Requirement of Proposed System*

No	Requirement	Functionality
1.	Operational	• The application is only available for mobile devices with Android operating system. • The application should be user-friendly. • The application should be easily maintained andupdated at any time. • The application should be reliable, minimizing the occurrence of errors, crashes, or disruptions.
2.	Performance	• The application should have low latency, providing quick response times for user interactions. • The message transmit time between user should not exceed 2 seconds.
3.	Security	• All user communications should be secured withend-to-end encryption. • The application provides appropriate error notifications. • User data, including personal information, metadata, and communication content, should be encrypted when stored in database. • All users can only login after successfully authenticated.

4.3 System Requirement Analysis

The Unified Modelling Language was created to provide a consistent, visually expressive modelling language for the structural and functional design of complex software systems. The UML diagram uses a variety of representations to illustrate the results of the analysis, including use case diagrams, class diagrams, sequence diagrams, and activity diagrams. Each one has a unique purpose.

4.3.1 Use Case Diagram

A UML use case diagram is the basic form of system/software requirements for an undeveloped software programme. Use cases define the intended behaviour (what) rather than the actual technique of achieving it (how). Once defined, use cases can have both written and visual representations.

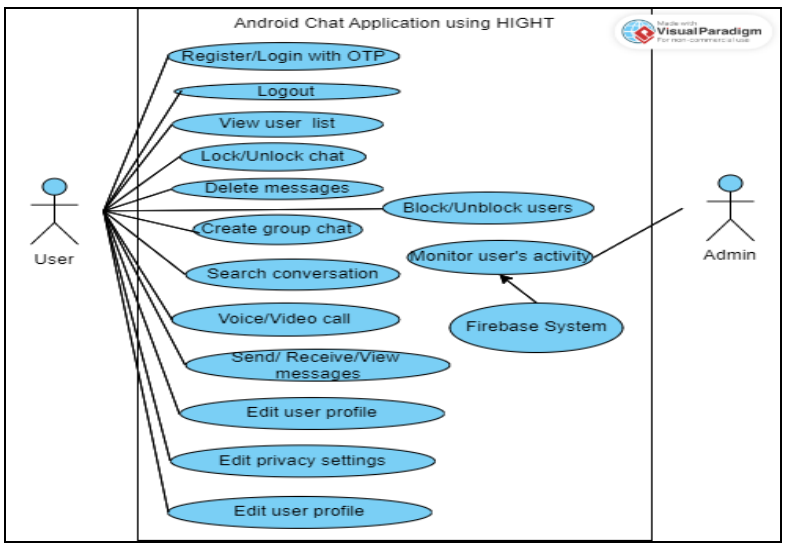

Fig.3 *Use Case Diagram*

The use case diagram of the proposed system is shown in Figure 3, illustrating the functionalities of an Android chat application using HIGHT encryption, highlighting interactions between users and the system. The

primary actors are user and the admin. User can perform various actions including registering or logging in with OTP, logging out, viewing the user list, locking and unlocking chats, deleting messages, creating group chats, searching conversations, initiating voice and video calls, sending, receiving, and viewing messages, editing their user profile, adjusting privacy settings, and blocking or unblocking other users. The admin's role is to monitor user activity within the Firebase system, such as tracking login times, message transactions, and profile updates, ensuring the application's security and proper usage. The Firebase system acts as the backend service where user activity data is stored and monitored.

4.3.2 Sequence Diagram

A sequence diagram illustrates the control structure between objects and how functions are carried out. It is particularly useful for visualizing the dynamic behaviour of a system during a specific scenario or process. Figures 4 to 11 show the sequence diagrams of the proposed Android chat application using HIGHT.

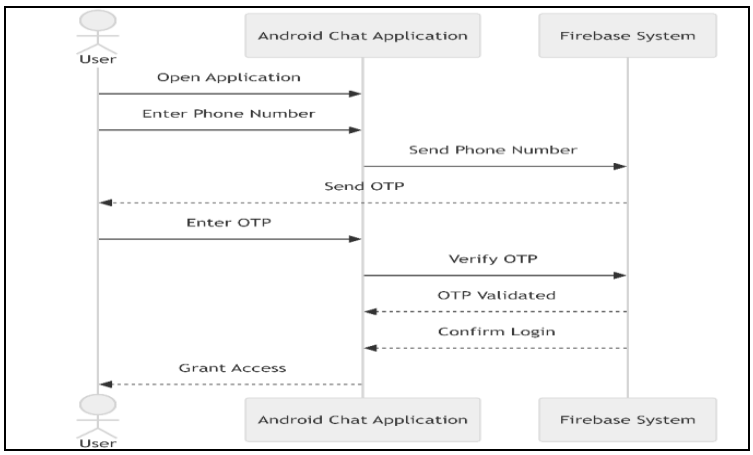

Fig.4 *Sequence Diagram for User Registration/Login with OTP*

Figure 4 shows the process of user registration and login using OTP. The user enters their phone number, receives an OTP from Firebase, and inputs the OTP into the application. The OTP is verified by Firebase, and upon successful authentication, the user gains access to the application

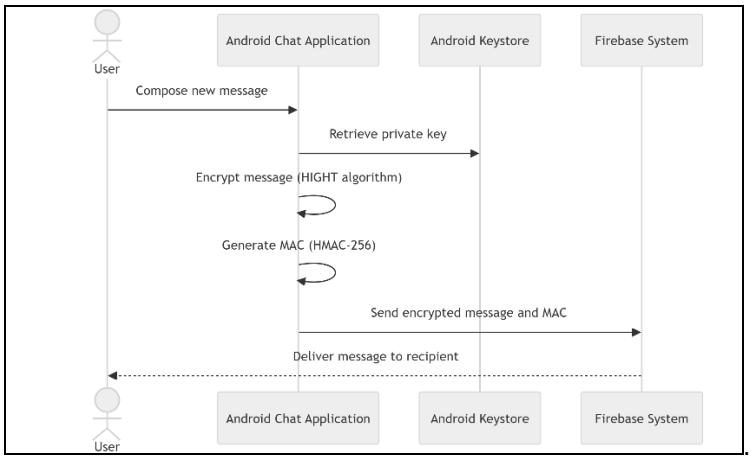

Fig.5 *Sequence Diagram for User Sending an Encrypted Message*

Figure 5 illustrates the process of sending an encrypted message. The user composes a message, which the application encrypts using the HIGHT algorithm and a private key from the Android Keystore. The encrypted message and a MAC using HAMC-256 are sent to Firebase for delivery to the recipient.

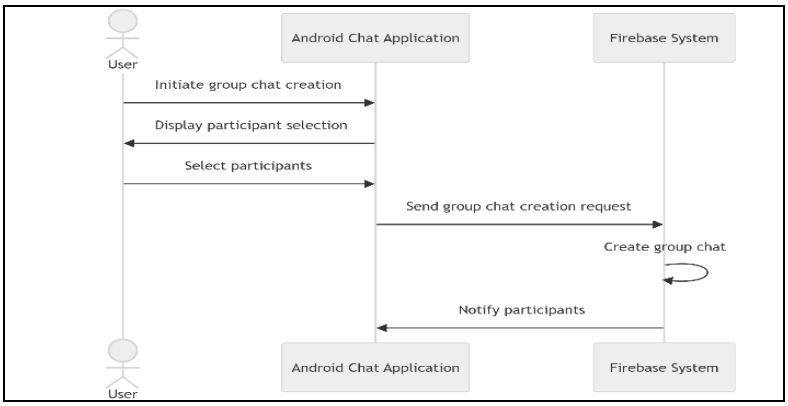

Fig.6 *Sequence Diagram for User Creating a Group Chat*

Figure 6 details the creation of a group chat. The user selects participants, and the application sends a group creation request to Firebase. Firebase processes the request and notifies the participants about their inclusion in the new group chat.

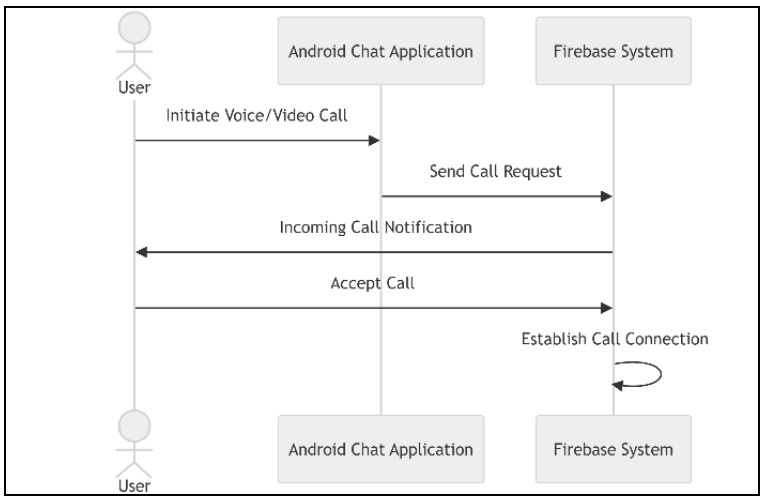

Fig.7 *Sequence Diagram for User Initiating and Accepting Voice/Video Call*

Figure 7 showcases the process of initiating and accepting a voice or video call. The user initiates a call, which sends a request to Firebase. Firebase notifies the recipient, who accepts the call, establishing the connection.

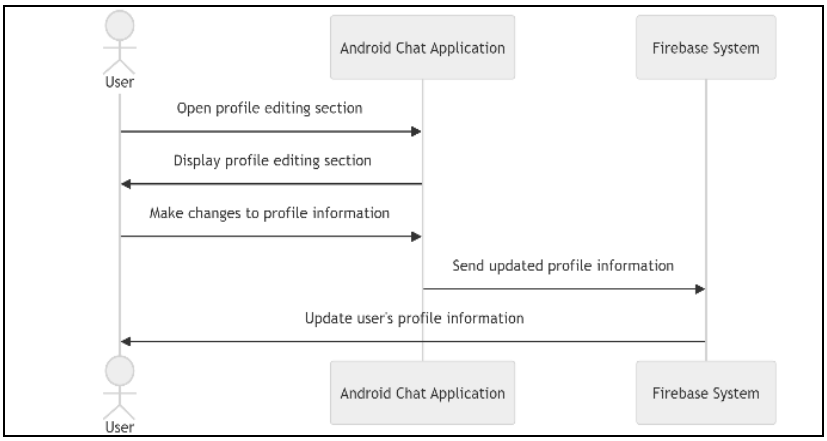

Fig.8 *Sequence Diagram for User Editing User Profile*

Figure 8 illustrates the process of editing a user profile. The user updates their profile information, which the application sends to Firebase. Firebase updates the database and confirms the changes to the application.

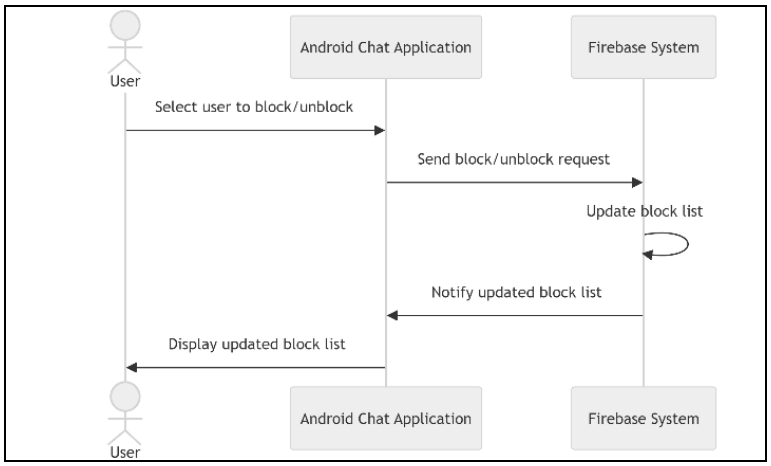

Fig.9 *Sequence Diagram for Blocking/Unblocking a User*

Figure 9 shows the process of blocking or unblocking a user. The user selects another user to block or unblock, and the application sends a request to Firebase. Firebase updates the block list and notifies the application of the changes.

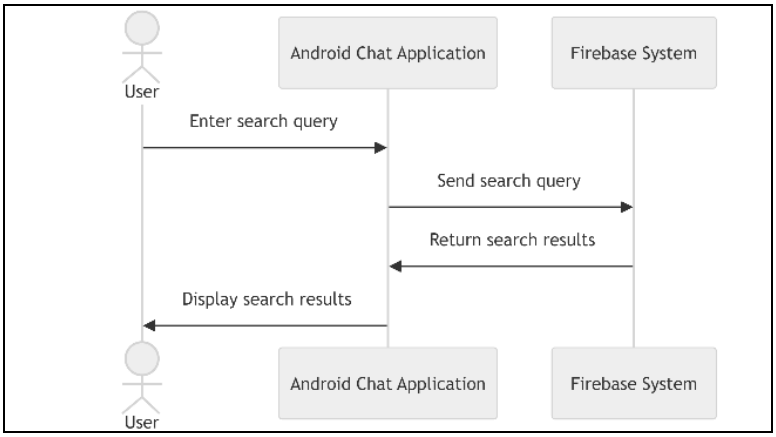

Fig.10 *Sequence Diagram for User Searching Conversations*

Figure 10 details the process of searching conversations. The user enters a search query, which the application sends to Firebase. Firebase processes the query and returns the relevant results to the application, which displays them to the user.

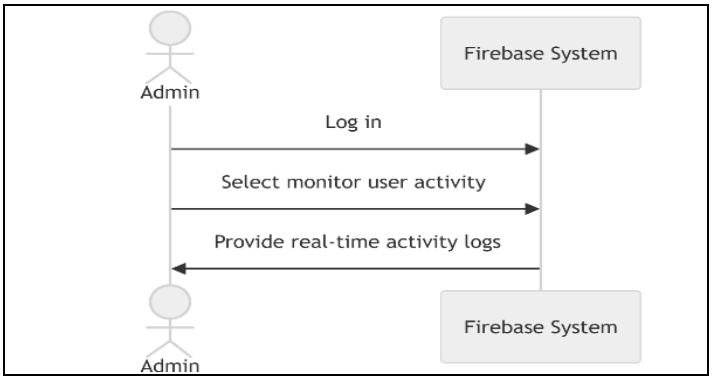

Fig.11 *Sequence Diagram for Admin Monitoring User Activity*

Figure 11 illustrates the process of an admin monitoring user activity. The admin logs into Firebase, views user log activity, manages permissions, and oversees storage management. Firebase provides real-time monitoring and management capabilities.

Applied Information Technology and Computer Science Vol. 5 No. 2 (2024) p. 232-253

4.3.3 Activity Diagram

An activity diagram, like a flowchart or a data flow diagram, visually depicts a series of actions or the flow of control in a system. Figure 12 (a) shows the activity diagram for user which outlines the workflow of a secure messaging application, starting with user registration and profile setup. Users begin by registering through phone authentication, where they enter a verification code. If the code is correct, they proceed to upload and save their profile picture and details. After setting up their profile, users can access various main menu options: viewing individual chats, participating in group chats, managing the user list, updating their profile, and adjusting user settings. In the chats section, users can read, send, delete, or search messages. In the group chats section, users can view existing groups or create new ones by uploading a group image and name and adding users. They can also manage their user profile, update information, and change privacy settings. The diagram also includes options for logging out and closing the application by pressing on back button to close the application. Figure 12 (b) activity diagram illustrates the process of an admin accessing and using Firebase, specifically focusing on the "Develop Viewer" role. The diagram begins with the admin logging into the Firebase system. Once logged in, a decision point determines the admin's role. In this case, the role identified is "Develop Viewer." As a Develop Viewer, the admin has read access to various Firebase services, including Google Analytics, Realtime Database and Cloud Firestore, Authentication, Hosting, Storage, Functions, and ML Kit. The diagram concludes with the admin logging out of Firebase.

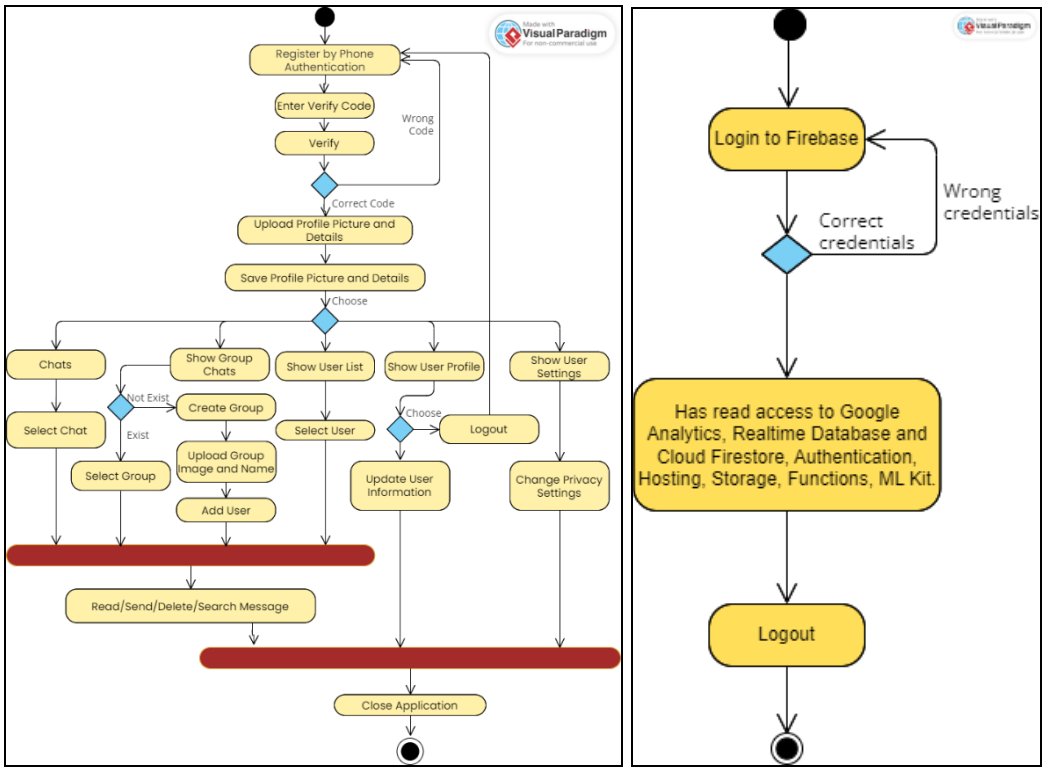

Fig.12 *Activity Diagram (a)User; (b) Admin*

4.3.4 Class Diagram

A class diagram in the Unified Modelling Language (UML) is a form of static structural diagram that shows the system's classes, the attributes of those classes, the operations, and all the relationships between objects. The class diagram of the proposed system is shown in Figure 13. This class diagram outlines the main components and their interactions in a secure messaging application. The User class, with attributes like uid and name, includes methods for logging in, editing profiles, and managing user interactions (e.g., blocking/unblocking users). Users can send and receive Message objects, which have details such as messageId, senderId, and content, with methods for sending, deleting, and receiving messages. The Group class manages chat groups, with attributes like groupId and adminId, and methods for creating groups and adding/removing users. Group messages are handled by the GroupChat class, which includes methods for sending, deleting, and viewing group messages. The ChatList and GroupList classes manage collections of chats and groups, respectively, allowing users to view and update these lists. Notifications are managed by the Notification class, which includes methods for sending and viewing notifications. The EncryptionUtil class, featuring methods for encryption and decryption,

incorporates the HIGHT class, which implements the HIGHT encryption algorithm. This diagram effectively illustrates the relationships and data flow between these classes, ensuring secure message transmission and management within the application.

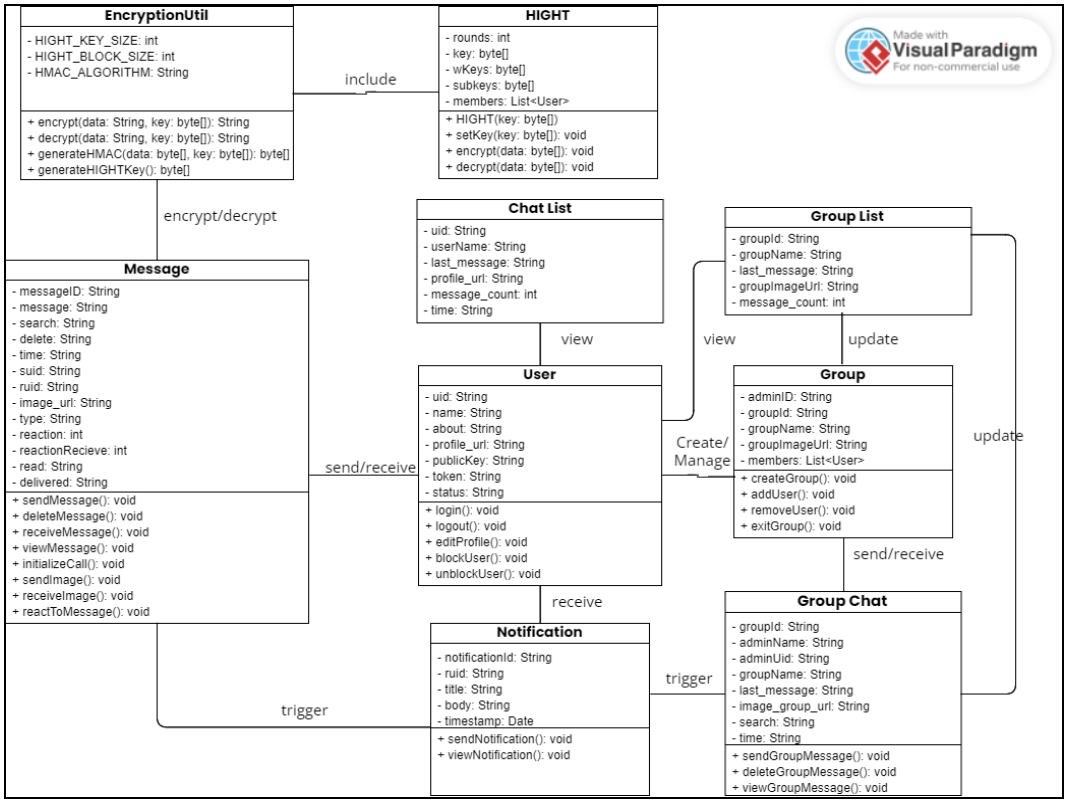

Fig.13 *Class Diagram*

4.4 System Implementation and Testing

This section provides a detailed overview of the implementation and testing phases of the proposed chat application. It describes the development environment, the implementation of key features, and the comprehensive testing strategies used to ensure the application's functionality, security, and performance. This includes the alpha unit testing using test cases and beta testing from the User Acceptance Test.

4.4.1 ECDH Implementation

The ECDH (Elliptic Curve Diffie-Hellman) key exchange mechanism is implemented to enhance security during the user authentication process. When the user successfully logs in using an OTP, the generateAndStoreECDHKeyPair method is invoked. This method generates a new ECDH key pair using the ECDHUtil.generateKeyPair() function. The generated key pair consists of a public key and a private key. The public key is then encoded and stored in SharedPreferences for future use in key exchanges, ensuring secure communication between users. The private key is automatically managed and securely stored by the Android Keystore system, eliminating the need for explicit storage in the app. This ensures that the private key is protected against unauthorized access. Figure 14 show the ECDH key pair generate after user register successfully.

```
84    private void generateAndStoreECDHKeyPair() {
85        try {
86            // Generate ECDH key pair
87            KeyPair keyPair = ECDHUtil.generateKeyPair();
88            PublicKey publicKey = keyPair.getPublic();
89            PrivateKey privateKey = keyPair.getPrivate();
90
91            // Save the public key to SharedPreferences
92            SharedPreferences sharedPreferences = getSharedPreferences( name: "ECDHKeys", MODE_PRIVATE);
93            ECDHUtil.storePublicKey(sharedPreferences, publicKey);
94
95            // Note: Private key is securely stored by Android Keystore.
96
97        } catch (Exception e) {
98            Toast.makeText( context: OTPAct.this,  text: "Error generating ECDH key pair: " + e.getMessage(), Toast.LENGTH_SHORT).show();
99        }
100   }
```

Fig.14 *Generate ECDH key pair after registration*

Figure 15 shows the ECDH key pair generation function which is configured with a KeyGenParameterSpec specifying the key alias (KEY_ALIAS), purpose (KeyProperties.PURPOSE_AGREE_KEY), elliptic curve (ECGenParameterSpec with "secp256r1"), and digests (SHA-256 and SHA-512). The public key generated is then stored in the Firebase NoSQL database shown in Figure 16.

```
// Generate an ECDH key pair
public static void generateKeyPair() throws Exception {
    KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance(KeyProperties.KEY_ALGORITHM_EC, ANDROID_KEYSTORE);
    keyPairGenerator.initialize(new KeyGenParameterSpec.Builder(
            KEY_ALIAS,
            KeyProperties.PURPOSE_AGREE_KEY)
            .setAlgorithmParameterSpec(new ECGenParameterSpec( stdName: "secp256r1"))
            .setDigests(KeyProperties.DIGEST_SHA256, KeyProperties.DIGEST_SHA512)
            .build());
    keyPairGenerator.generateKeyPair();
}
```

Fig.15 *ECDH key generate function*

Fig.16 *User's public key store in database*

4.4.2 HIGHT Implementation

Figure 17 shows the implementation of HIGHT in the application. The EncryptionUtil class is designed for encrypting and decrypting messages using the HIGHT algorithm. It includes constants for key sizes, block sizes, and the HMAC algorithm. The encrypt() method verifies the key size, pads the plaintext to fit 64-bit blocks, and encrypts it using the HIGHT algorithm. After encryption, it generates an HMAC for authenticity and combines the ciphertext and HMAC, encoding the result in Base64. The decryptWithHMAC() method reverses this process: it separates the ciphertext and HMAC, validates the HMAC, decrypts the ciphertext, and trims any padding. Additional methods include generateHMAC() to create an HMAC and generateHIGHTKey() to generate a random 128-bit HIGHT key.

```
public class EncryptionUtil {
    private static final int HIGHT_KEY_SIZE = 16; // 128 bits
    private static final int HIGHT_BLOCK_SIZE = 8; // 64 bits
    private static final String TAG = "EncryptionUtil";
    private static final String HMAC_ALGORITHM = "HmacSHA256";

    public static String encrypt(String valueToEnc, byte[] hightKey) throws Exception {...}

    public static String decryptWithHMAC(String encryptedValue, byte[] hightKey) throws Exception {...}

    private static byte[] generateHMAC(byte[] data, byte[] key) throws Exception {...}

    public static byte[] generateHIGHTKey() throws Exception {...}
}
```

Fig.17 *EncryptionUtil Class with HIGHT Implementation*

Figure 18 shows the HIGHT algorithm. The HIGHT class handles core encryption and decryption. It defines constants for block and key sizes and the number of encryption rounds. The constructor initializes the HIGHT instance with a key and sets up whitening keys and subkeys. The encrypt() and decrypt() methods handle 8-byte blocks of data. The initialTransform() and inverseInitialTransform() methods perform initial and final transformations, while performRounds() executes the main encryption and decryption rounds. These classes (EncryptionUtil and HIGHT) work together to securely encrypt and protect messages from tampering.

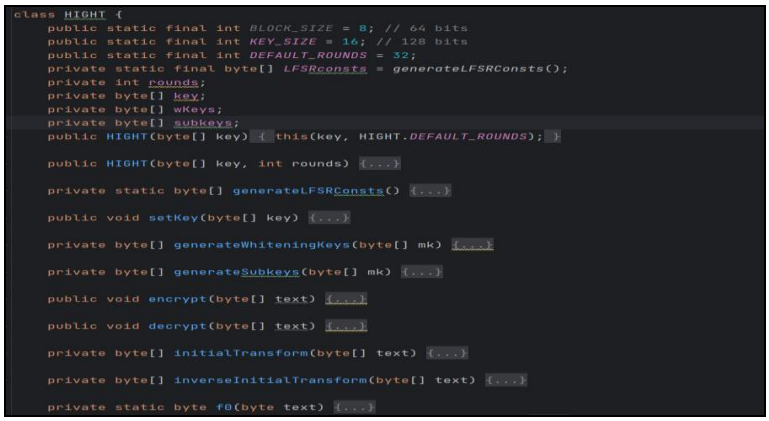

Fig.18 *HIGHT Class*

Figure 19 shows the code snippet which demonstrates the application of encryption before sending a message. The EncryptionUtil.encrypt() method encrypts the plaintext message using a specified key (hightKeyHolder[0]). This process ensures the message is securely encrypted before transmission. The encrypted message is stored in database shown in Figure 20.

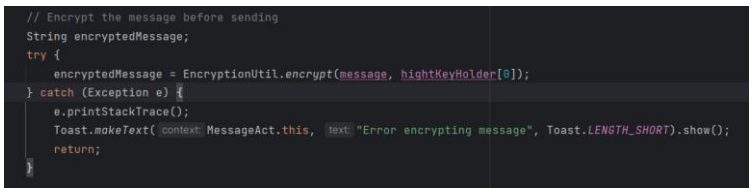

Fig.19 *Encrypt message before transmission*

Fig.20 *Encrypted message stored in database*

Figure 21 shows the code snippet which applies the decryption of a message using the EncryptionUtil.decryptWithHMAC() method. It tries to decrypt the message with the HIGHT key from hightKeyHolder[0]. If a SecurityException occurs, it indicates tampering, logs the error, and updates the UI with "Error: Message has been tampered with" shown in Figure 22.

Fig.20 *Decrypt message when receive*

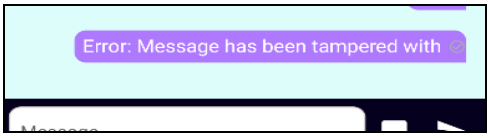

Fig.21 *Error message when encrypted message is tampered*

Applied Information Technology and Computer Science Vol. 5 No. 2 (2024) p. 232-253

4.4.3 Application Interface

Figure 22 shows the authentication interface. The authentication interface for the proposed chat application features a two-step process: obtaining an OTP and verifying it for login. The "Get OTP" screen allows users to enter their country code and mobile number, then request an OTP by clicking the "GET OTP" button. The "Enter OTP" screen lets users input the received OTP and submit it for verification using the "SUBMIT OTP" button. These screens ensure that only users with access to the provided mobile number can log in, enhancing security through phone-based authentication.

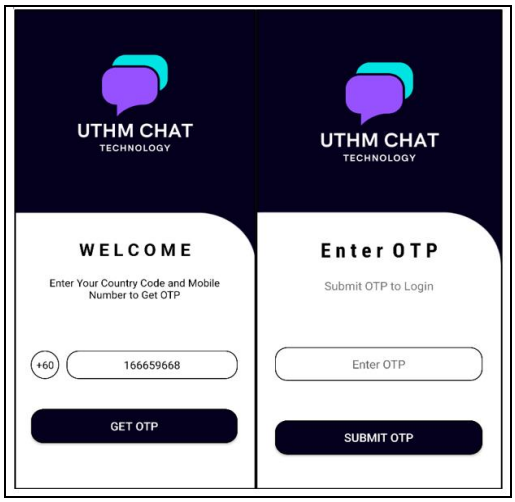

Fig.22 *User Authentication Interface*

Figure 23 shows the user profile interface. The user profile interface guides users through setting up and managing their profiles. Upon first login, users are presented with the setup screen (left image), where they enter their username, about section, and phone number, then click "ADD" to save their profile. Once the profile is set up, users see the detailed profile screen (right image), which displays their entered information and allows them to edit details by clicking "EDIT" or log out using the "LOGOUT" button. Additionally, both screens provide access to the "View Blocked Users" option. This module ensures a personalized user experience by allowing users to manage their profiles and preferences effectively.

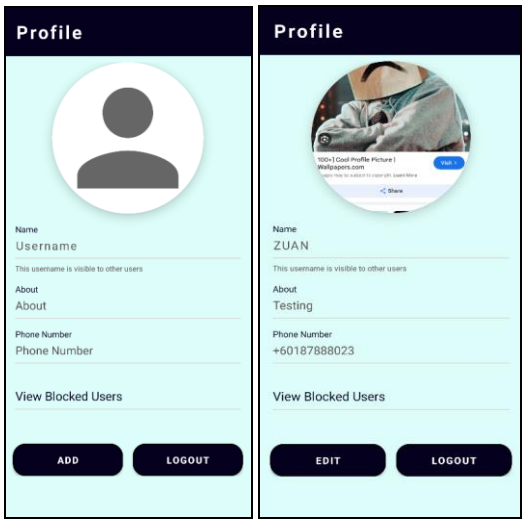

Fig.23 *User Profile Interface*

Figure 24 shows the tabbed interface. The application features a tabbed interface that allows users to navigate between different sections: Chats, Groups, and Status. The Chats tab displays a list of individual conversations, enabling users to view and manage their personal messages. The Groups tab shows the group chat list, allowing users to create new groups and view existing group conversations. The Status tab, which is yet to be implemented, is intended for users to post and view status updates. Tapping on the floating message button will show the list of users to be selected to start a chat activity as shown in Figure 25.

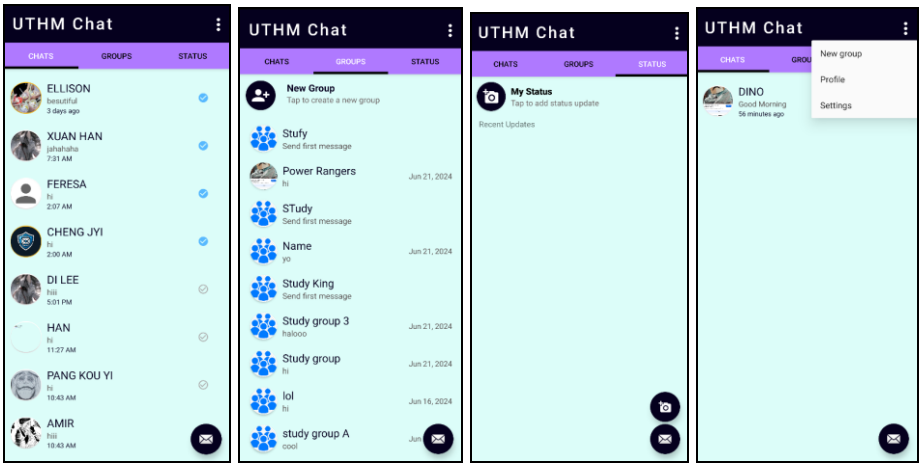

Fig.24 *Tabbed Interface*

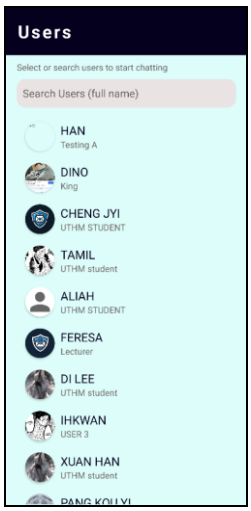

Fig.25 *User List Interface*

Figure 26 shows the chat interface which allows users to send text messages and images in real-time. Users can also initiate voice and video calls directly from the chat interface. By tapping on the profile image of the chat partner, users can view their profile details and an option to block user as shown in Figure 27. The chat menu offers additional options such as searching the conversation, clearing the chat history, and locking the chat for privacy. Additionally, users can delete messages either for themselves or for both participants in the chat.

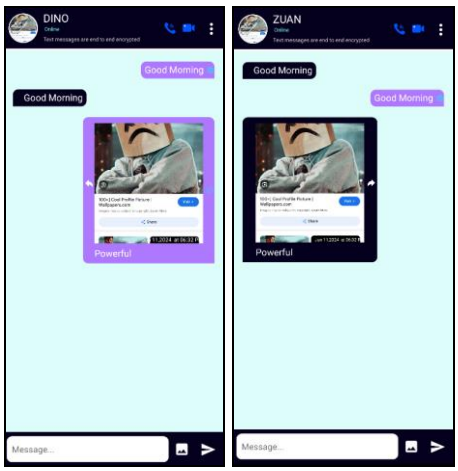

Fig.26 *Chat Interface*

Applied Information Technology and Computer Science Vol. 5 No. 2 (2024) p. 232-253

Fig.27 *About User Interface*

Figure 28 shows the group chat interface which allows real-time messaging within a group. Users can send and receive messages and view participants. In the group info screen, members can see other members and exit the group. Admins, on the other hand, have additional options to add or remove members and change the group name, ensuring smooth group management and communication.

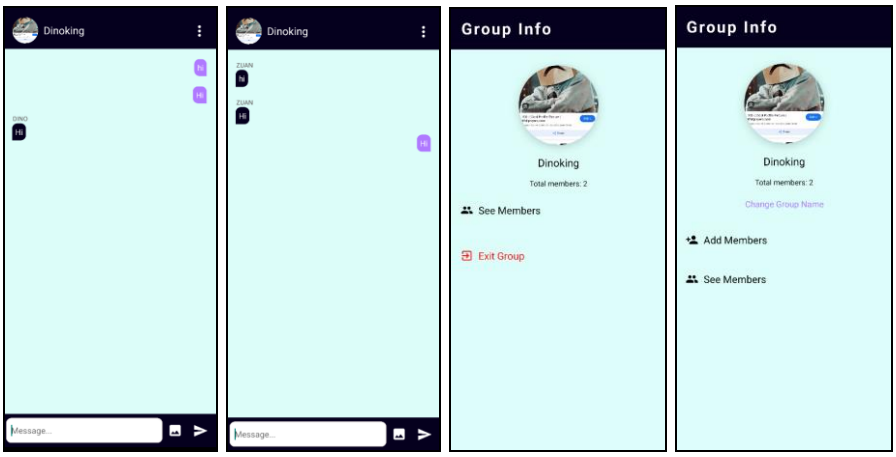

Fig.27 *Group Chat Interface*

4.4.4 Test Plan Result

After the application was developed, functional testing was conducted to evaluate its functionality. The testing aimed to identify any errors occurring during the use of the chat application and to verify if the application met the project's objectives and scope. Tables 5, 6, 7, 8, and 9 provide a summary of the functional testing results for the chat application. Screenshots of the MobSF, AppSweep, and Logcat evaluations are shown in Appendix B.

Table 5 *Test Plan for Phone Authentication Module*

No	Description	Expected Result	Actual Result
1.	Verify phone number input field	The system displays an error message when the phone number field is empty or an invalid phone number is entered, and "Get OTP" is clicked.	PASS
2.	Validate "Get OTP" button functionality	Clicking the "Get OTP" button with a valid phone number navigates to the OTP entry screen.	PASS
3.	Verify OTP input field	The system displays an error message when the OTP field is empty or invalid OTP is entered, and "Submit OTP" is clicked.	PASS
4.	Validate "Submit OTP" button functionality	Clicking the "Submit OTP" button with a valid OTP logs the user in successfully.	PASS
5.	Exceed maximum OTP attempts	The system blocks further attempts for a minute after 3 failed OTP entries and redirects the user to the login page.	PASS
6.	OTP timeout	The system notifies the user that the session has timed out and redirects to the login page.	PASS

Table 5 *Test Plan for Phone Authentication Module (cont.)*

No	Description	Expected Result	Actual Result
7.	Verify OTP for different phone numbers	The system sends different OTPs for different phone numbers.	PASS
8.	Prevent multiple OTP requests	The system prevents multiple OTP requests within a short period.	PASS

Table 6 *Test Plan for Profile Module*

No	Description	Expected Result	Actual Result
1.	Verify default profile view for new user	Displays default avatar and empty fields for name, about, and phone number.	PASS
2.	Verify profile view for existing user with data	Displays user-specific avatar and populated fields correctly.	PASS
3.	Validate "Add" button functionality for new user	Navigates to enter and save profile information correctly.	PASS
4.	Validate "Edit" button functionality for existing user	Allows updating and saving profile information correctly.	PASS
5.	Validate "Logout" button functionality	Logs out and redirects to login screen.	PASS
6.	Verify "View Blocked Users" link functionality	Displays list of blocked users.	PASS
7.	Validate profile picture upload functionality	Allows uploading and displaying the profile picture.	PASS
8.	Error message for empty required fields	Displays error message for empty required fields.	PASS

Table 7 *Test Plan for Message Activity Module*

No	Description	Expected Result	Actual Result
1.	Display of text messages	Text messages are displayed correctly with proper sender identification.	PASS
2.	Sending a text message	Message is sent and appears in the chat window.	PASS
3.	Display of images in chat	Images are displayed correctly within the chat bubbles.	PASS
4.	Sending an image	Image is sent and appears in the chat window.	PASS
5.	Differentiate messages from different users	Messages from different users are displayed in different colors or styles.	PASS
6.	Sending a message with media	Message with media is sent and appears correctly with media preview.	PASS
7.	Read receipt functionality	Messages display correct read/unread status.	PASS
8.	Message deletion	Deleted messages are removed from the chat window.	PASS

Table 8 *Test Plan for Group Activity Module*

No	Description	Expected Result	Actual Result
1.	Display of group chat messages	Group messages are displayed correctly with proper sender identification.	PASS
2.	Sending a text message	Message is sent and appears in the group chat window.	PASS
3.	Group info page	Displays group information, including profile picture, total members, and group name correctly.	PASS
4.	"See Members" link functionality	Clicking "See Members" displays a list of group members.	PASS
5.	"Add Members" link functionality	Clicking "Add Members" navigates to a screen to add new members to the group.	PASS
6.	"Exit Group" button functionality	Clicking "Exit Group" removes the user from the group and redirects to the main chat list.	PASS
7.	"Change Group Name" link functionality	Clicking "Change Group Name" allows updating the group name.	PASS
8.	Validate removal of members	Members can be removed from the group and no longer appear in the member list.	PASS

Table 9 *Security and Performance Evaluation Result*

No.	Tool	Evaluation Result	Remarks
1.	MobSF	Score: 51/100; 2 high, 25 medium, 2 info risks; 2 secure (HIGHT algorithm)	Moderate security level, key high-risk issues related to SSL and Android version vulnerabilities. Passed MASVS-CRYPTO, but not MASVS-STORAGE and MASVS-NETWORK.
2.	AppSweep	3 high, 16 medium, 4 low issues	Highlights areas for improvement in cleartext communication, tap jacking protection, and secure hashing. Passed MASVS-CRYPTO.
3.	Logcat	Encryption time using HIGHT algorithm: 0 ms to 1 ms	Extremely fast encryption, suitable for real-time messaging. Ensures HIGHT algorithm meets cryptographic standards.

5. Conclusion

The proposed chat application has successfully integrated the HIGHT algorithm combined with HMAC-SHA256 to ensure the confidentiality and authenticity of one-to-one text messages. This robust security framework is supported by several key features, including user authentication through phone number and OTP verification via Firebase, and secure key management using ECDH for secure key exchange.

In addition to secure messaging, the application ensures real-time transmission and reception of encrypted messages. Message integrity and authenticity are guaranteed through the generation and verification of MACs using HMAC-SHA256. The application offers a comprehensive set of functionalities aimed at enhancing user experience. These include the ability to register and log in, view, and delete messages, and participate in group chats. Users can also edit their profiles, make voice, and video calls, send images, view other users' profiles, and clear chat history. The application further includes features for searching conversations and displaying user lists. The UTHM Chat application underwent security evaluation using AppSweep and MobSF, certified by OWASP MASTG. The evaluations found several high and medium risk issues, such as debug certificate usage, support for older Android versions, tapjacking and manifest vulnerabilities. Despite these, the application passed encryption tests, demonstrating effective HIGHT algorithm implementation. Also, the results via Logcat indicate that the encryption process is extremely fast, completing in just a fraction of a millisecond. This low latency is crucial for real-time messaging applications, demonstrating the efficiency and scalability of the HIGHT algorithm in terms of resource efficiency.

Currently, the HIGHT encryption algorithm is implemented exclusively for one-to-one text messaging. The application, currently limited to Android, faces scalability and security challenges that could affect user experience and adoption. It might struggle with performance as the user base grows and lacks cross-platform compatibility, which restricts potential users. Vulnerabilities to social engineering and platform-related threats remain, despite using the HIGHT encryption algorithm. Additionally, advanced features like message scheduling and service integrations are absent, potentially limiting appeal. Future improvements should focus on enhancing scalability through backend optimization, expanding to iOS and Windows, and bolstering security with multi-factor authentication and regular checks. Integrating machine learning for predictive text and spam detection, along with continuous user interface enhancements based on feedback, will further improve the application.

Acknowledgement

The authors would like to thank the Faculty of Computer Science and Information Technology, University Tun Hussein Onn Malaysia for its support.

References

[1] Valentin Mulder, Alain Mermoud, Vincent Lenders, and Bernhard Tellenbach, Trends in Data Protection and Encryption Technologies. Springer Nature Switzerland, 2023. doi: 10.1007/978-3-031-33386-6.

[2] R. R. Farashahi, B. Rashidi, and S. M. Sayedi, 'FPGA based fast and high-throughput 2-slow retiming 128-bit AES encryption algorithm', Microelectronics J, vol. 45, no. 8, pp. 1014–1025, 2014, doi: 10.1016/j.mejo.2014.05.004.

[3] M. Masoud, I. Jannoud, A. Ahmad, and H. Alshoubaki, 'The Power Consumption Cost of Data Encryption in Smartphones', Dec. 2015. doi: 10.1109/OSSCOM.2015.7372685.

[4] R. Kousalya and G. A. Sathish Kumar, 'A Survey of Light-Weight Cryptographic Algorithm for Information Security and Hardware Efficiency In Resource Constrained Devices', in 2019 International Conference on Vision Towards Emerging Trends in Communication and Networking (ViTECoN), 2019, pp. 1–5. doi: 10.1109/ViTECoN.2019.8899376.

[5] D. Hong et al., 'HIGHT: A New Block Cipher Suitable for Low-Resource Device', 2006.

[6] Iec, 'INTERNATIONAL STANDARD Information Technology-Security techniques-Encryption algorithms-Part 3: Block ciphers Technologies de l'information-Techniques de sécurité-Algorithmes de chiffrement Partie 3:

Chiffrement par blocs', 2010. [Online]. Available: www.iso.org
[7] B. Dowling and B. Hale, 'Secure messaging authentication against active man-in-the-middle attacks', in Proceedings - 2021 IEEE European Symposium on Security and Privacy, Euro S and P 2021, Institute of Electrical and Electronics Engineers Inc., Sep. 2021, pp. 54–70. doi: 10.1109/EuroSP51992.2021.00015.
[8] Justin Engler and Cara Marie, 'Secure Messaging for Normal People-Version 1.0', 2015.
[9] Talking HealthTech, 'Secure Messaging'. Accessed: Dec. 20, 2023. [Online]. Available: https://www.talkinghealthtech.com/glossary/secure-messaging
[10] Guardian Digital Inc., 'The Importance of Secure Communication in Today's Business World', LinkedIn. Accessed: Dec. 08, 2023. [Online]. Available: https://www.linkedin.com/pulse/importance-secure-communication-todays-business-world/
[11] N. Unger et al., 'SoK: Secure Messaging', 2015.
[12] Statista Research Department, 'Global market share held by mobile operating systems from 2009 to 2023, by quarter', Statista. Accessed: Dec. 08, 2023. [Online]. Available: https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems- since-2009/#:~:text=Android%20maintained%20its%20position%20as,percent%20during%20the%20same%20period.
[13] Android, 'Android Security Features'. Accessed: Dec. 08, 2023. [Online]. Available: https://source.android.com/docs/security/features4
[14] S. R. Swamy and FASNA V, 'Sandbox: A Secured Testing Framework for Applications', ResearchGate, vol. 4, no. 1, Sep. 2020, [Online]. Available: https://www.researchgate.net/publication/344170480
[15] Google Cloud, 'What is encryption?' Accessed: Dec. 08, 2023. [Online]. Available: https://cloud.google.com/learn/what-is-encryption#section-7
[16] IBM, 'What is encryption?' Accessed: Dec. 08, 2023. [Online]. Available: https://www.ibm.com/topics/encryption
[17] S. S. Dhanda, B. Singh, and P. Jindal, 'Lightweight Cryptography: A Solution to Secure IoT', Wirel Pers Commun, vol. 112, no. 3, pp. 1947–1980, Jun. 2020, doi: 10.1007/s11277-020-07134-3.
[18] Meta, 'Facebook Messenger'. Accessed: Dec. 08, 2023. [Online]. Available: https://www.messenger.com/
[19] Emma Bowman, 'After Data Breach Exposes 530 Million, Facebook Says It Will Not Notify Users', npr. Accessed: Dec. 08, 2023. [Online]. Available: https://www.npr.org/2021/04/09/986005820/after-data-breach-exposes-530-million-facebook-says-it-will-not-notify-users
[20] WeChat, 'WeChat Help Center'. Accessed: Dec. 09, 2023. [Online]. Available: https://www.wechat.com/
[21] Ryan, Fergus, Audrey Fritz, and Daria Impiombato, 'WeChat privacy concerns and data collection. In TikTok and WeChat: Curating and controlling global information flows', Australian Strategic Policy Institute, pp. 43–46, 2020, Accessed: Dec. 09, 2023. [Online]. Available: http://www.jstor.org/stable/resrep26120.8
[22] Signal, 'Signal'. Accessed: Dec. 09, 2023. [Online]. Available: https://signal.org

Appendix A: Gantt Chart

Figure A.1 shows the project Gantt chart for Message Encryption Application using HIGHT.

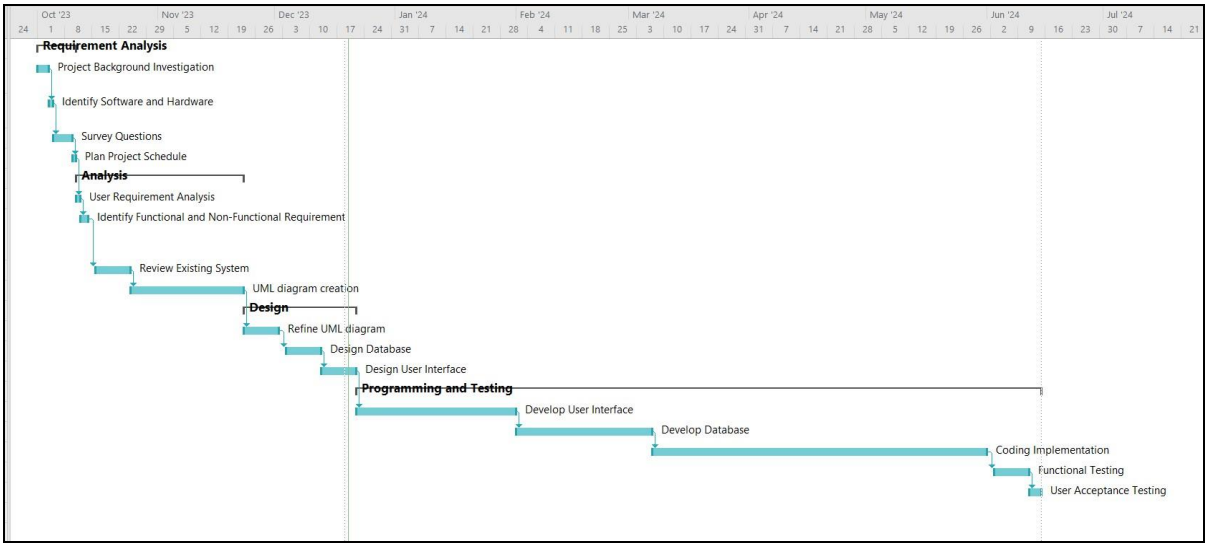

Figure A.1: Project Gantt Chart

Appendix B: Security and Performance Evaluation Result

Applied Information Technology and Computer Science Vol. 5 No. 2 (2024) p. 232-253

Figure B.1, B.2 and B.3 shows the security and performance evaluation result from MobSF, Appsweep, and Logcat respectively.

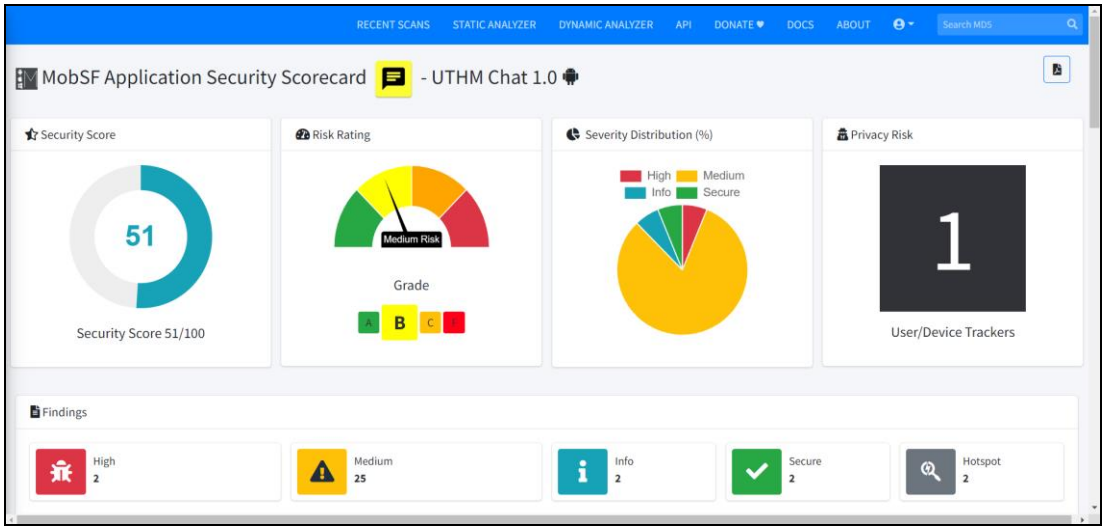

Figure B.1 *Security Evaluation Result from MobSF*

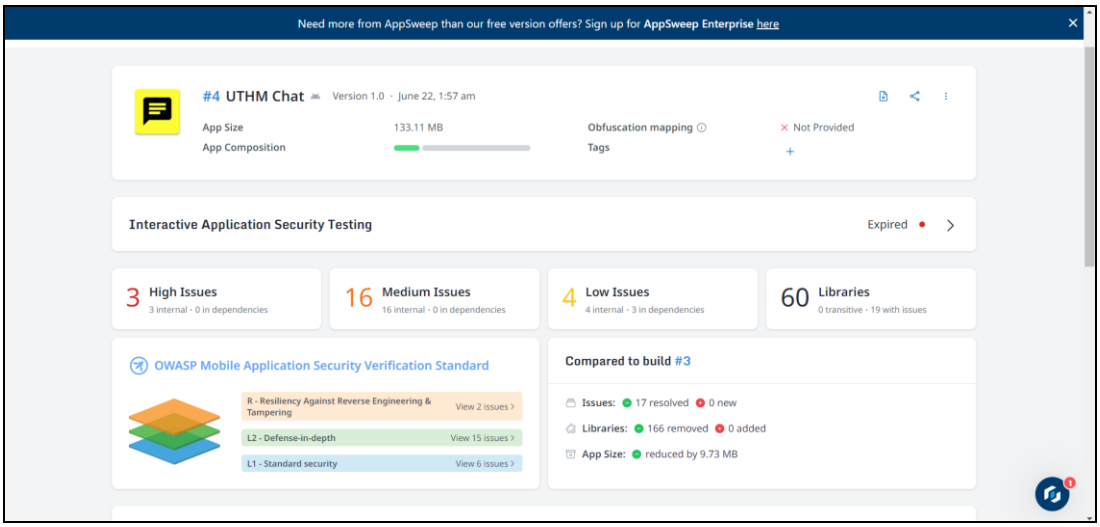

Figure B.2 *Security Evaluation Result from AppSweep*

Figure B.3 *Performance Result from Logcat*