**Self-Destruct Messenger**



|  |
| --- |
|  |

**By:**

**Muhammad Aatif Khan**

**38587**

**Hamza Zawari Khalid**

**35772**

**Syed Abdurrehman**

**31980**

**Supervised by:**

**Dr. Jawaid Iqbal**

**Faculty of Computing**

**Riphah International University, Islamabad**

**Spring/Fall 2025**

**Submitted To**

**Faculty of Computing,**

**Riphah International University, Islamabad**

**As a Partial Fulfillment of the Requirement for the Award of the Degree of**

**Bachelors of Science in Cyber Security**

**Faculty of Computing**

**Riphah International University, Islamabad**

Date: [date of final presentation]

**Final Approval**

This is to certify that we have read the report submitted by Muhammad Aatif Khan (38587), Hamza Zawari Khalid (35772), Syed AbdurRehman (31980), for the partial fulfillment of the requirements for the degree of the Bachelor of Science in Cyber Security (BS CYS). It is our judgment that this report is of sufficient standard to warrant its acceptance by Riphah International University, Islamabad for the degree of Bachelor of Science in Cyber Security (BS CYS).

**Committee:**

|  |  |
| --- | --- |
| **1** | Dr. Jawaid Iqbal  (Supervisor) |
|  |  |
| **2** | Mr .Yawar Abbas  (In-charge Cyber Security) |
| **3** | Dr. Musharraf Ahmed  (SE & CS) |

**Declaration**

We hereby declare that this document “**Self-Destruct Messenger**” neither as a whole nor as a part has been copied out from any source. It is further declared that we have done this project with the accompanied report entirely on the basis of our personal efforts, under the proficient guidance of our teachers especially our supervisor **DR. Jawad Iqbal**. If any part of the system is proved to be copied out from any source or found to be reproduction of any project from anywhere else, we shall stand by the consequences.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Muhammad Aatif Khan**

**38587**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Hamza Zawari Khalid**

**35772**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Syed Abdurrehman**

**31980**

**Dedication**

This project is dedicated to our friends, mentors, and educators, whose unwavering support and encouragement have been a constant source of strength throughout this journey. Their belief in our capabilities has fueled our determination to succeed. Additionally, we dedicate this work to our colleagues and the academic community, whose guidance and knowledge have been instrumental in shaping our professional growth. Their commitment to excellence has inspired us to push the boundaries of our potential. Thank you for being our pillars of support and for always believing in us.

**Acknowledgement**

We would like to express our sincere gratitude to the following individuals whose guidance and support were instrumental in the successful completion of this project:

* **Project Supervisor:** We are particularly grateful to our project supervisor, **Dr. Jawaid Iqbal**, for his invaluable expertise, insightful feedback, and unwavering encouragement throughout the entire project development process. His dedication and support played a pivotal role in shaping the direction and success of our work.
* **Faculty Advisors:** We extend our sincere thanks to our faculty advisors, **Dr. Muhammad Mansoor Alam,** whose awareness for different messaging system help us to pursue this project, **and Mr Hammayun,** whose knowledge and guidance were invaluable throughout the research phase.
* **Technical Support Staff:** We appreciate the assistance provided **by Mr. Osama Raza, Mr. Awais Nawaz, Mr. Tabbasum Jawaid**, and **Mr. Mueed Mirza.** Their technical expertise and willingness to help were crucial in overcoming technical challenges.
* **Project Convener:** We acknowledge the leadership of **Mr. Yawar Abbas** as the project convener, whose guidance ensured the project's smooth execution within the academic framework.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Muhammad Aatif khan**

**38587**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Hamza Zawari Khalid**

**35772**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Syed Abdurrehman**

**31980**

**Abstract**

In the intricately woven online world of today, safeguarding communication is pivotal for personal and organizational security. This project is about a private and secure messaging application detached from the online world that uses automated data deletion and real-time communication constraints, as well as self-encryption chat rooms to secure private chats: Self-Destruct Messenger. In this solution, the risk of unauthorized access is minimized, and data secrecy is guaranteed with cutting-edge security solutions. Self-Destruct Messenger comes with special sets of features intended for protected communication. Each message is encrypted with unique keys per each discussion which makes the dynamic session-based encryption drastically lower the chances of interception. Because real-time communication demand limits message access to simultaneously when both users are present, offline retrieval is not possible, and therefore decreases exposure of data. The technology also guarantees that no stale data on servers or devices can be retrieve ed without being detected as volatile access points, by automating message deletion after session adjournment, ensuring no residual data is retained the devices. Message content is optionally watermarked and protected, alongside other anti-screenshot mechanisms designed to prevent unintentional semantic capture. Self-Destruct Messenger is a simple easy-to-use interface designed freely across desktop and mobile devices, placing emphasis on functionality and ease of use. Because of its proactive security design, users who are concerned about their privacy may interact with confidence without worrying about data breaches. Self-Destruct Messenger enables people and companies to have private, secure discussions in a time where cyber security threats are always changing by providing unmatched safety in the digital communication environment.

**Keywords**: Secure Communication, Real-Time Encryption, Self-Destruct Messenger, Data Deletion, Anti-Screenshot, Privacy Protection, Cyber security.

**Table of Contents**

[Chapter 1: Introduction 2](#_Toc197969757)

[1.1 Introduction 2](#_Toc197969758)

[1.2 Opportunity & Stakeholders 2](#_Toc197969759)

[1.3 Motivations and Challenges 3](#_Toc197969760)

[1.4 Significance of Study 5](#_Toc197969761)

[1.5 Goals and Objectives 6](#_Toc197969762)

[1.6 Scope of Project 6](#_Toc197969763)

[1.7 Chapter Summary 7](#_Toc197969764)

[Chapter 2: Literature / Market Survey 10](#_Toc197969765)

[2.1 Introduction 10](#_Toc197969766)

[2.2 Literature Review / Technologies 10](#_Toc197969767)

[2.3 Comparative Analysis 12](#_Toc197969768)

[2.3.1 Existing Systems 12](#_Toc197969769)

[2.3.2 Dynamic Encryption: 13](#_Toc197969770)

[2.3.3 Real-Time Communication Requirement: 13](#_Toc197969771)

[2.3.4 Anti-Screenshot Measures: 13](#_Toc197969772)

[2.4 Research Gaps 13](#_Toc197969773)

[2.4.1 Static Encryption Keys 14](#_Toc197969774)

[2.4.2 Vulnerability for Offline Messages 14](#_Toc197969775)

[2.4.3 Screenshot Prevention 14](#_Toc197969776)

[2.4.4 Persistent Data Retention 15](#_Toc197969777)

[2.5 Problem Statement 15](#_Toc197969778)

[2.6 Chapter Summary 15](#_Toc197969779)

[Chapter 3: Requirements and System Design 17](#_Toc197969780)

[3.1 Introduction 17](#_Toc197969781)

[3.2 System Architecture 17](#_Toc197969782)

[3.3 Functional Requirements 18](#_Toc197969783)

[3.4 Non-Functional Requirements 19](#_Toc197969784)

[3.5 Design 20](#_Toc197969785)

[3.6 Hardware and Software Requirements 25](#_Toc197969786)

[3.7 Threat Scenarios 28](#_Toc197969787)

[3.8 Threat Modeling Techniques 28](#_Toc197969788)

[3.9 Threat Resistance Model 28](#_Toc197969789)

[3.10 Chapter Summary 29](#_Toc197969790)

[Chapter 4: System Design 31](#_Toc197969791)

[4.1. Introduction 31](#_Toc197969792)

[4.2 Data Collection 31](#_Toc197969793)

[4.3 Tools & Techniques 32](#_Toc197969794)

[4.4 Evaluation Matrices 35](#_Toc197969795)

[4.5 Chapter Summary 36](#_Toc197969796)

[Chapter 5: Implementation & Testing 38](#_Toc197969797)

[5.1 Introduction 38](#_Toc197969798)

[5.2 Testing Methods 38](#_Toc197969799)

[5.3 Module Testing 39](#_Toc197969800)

[5.4 Development Environment Setup 45](#_Toc197969801)

[5.5 Key Module Implementations 46](#_Toc197969802)

[5.5.1 Authentication System 46](#_Toc197969803)

[5.5.2 Real-Time Messaging 47](#_Toc197969804)

[5.5.3 Automatic Message Deletion 47](#_Toc197969805)

[5.5.4 Anti-Screenshot Logic 47](#_Toc197969806)

[5.5.5 End-to-End Encryption with ephemeral encryption 48](#_Toc197969807)

[5.6 Best Coding Practices 48](#_Toc197969808)

[5.7 Challenges Encountered During Implementation 49](#_Toc197969809)

[5.7.1. Session Dependency Management 50](#_Toc197969810)

[5.7.2. End-to-End Encryption with ECDH 50](#_Toc197969811)

[5.7.3. Screenshot Prevention Limitations 50](#_Toc197969812)

[5.7.4. Contact-Based Message Authorization 51](#_Toc197969813)

[5.7.5. Encryption Testing and Self-Destruct Logic 51](#_Toc197969814)

[5.8 Chapter Summary 51](#_Toc197969815)

[Chapter 6: Conclusion 54](#_Toc197969816)

[6.1 Introduction 54](#_Toc197969817)

[6.2 Achievements and Improvements 54](#_Toc197969818)

[6.2.1 Real-Time, Session-Based Messaging: 54](#_Toc197969819)

[6.2.2 Automatic Message Deletion: 54](#_Toc197969820)

[6.2.3 End-to-End Encryption (ECDH): 54](#_Toc197969821)

[6.2.4 Mutual Contact Verification: 54](#_Toc197969822)

[6.2.5 Screenshot Protection: 55](#_Toc197969823)

[6.2.6 Cross-Platform Functionality: 55](#_Toc197969824)

[6.2.7 Public/Private Key Handling Optimization: 55](#_Toc197969825)

[6.2.8 Enhanced User Feedback and Alerts: 55](#_Toc197969826)

[6.2.9 Efficient Presence Tracking: 55](#_Toc197969827)

[6.3 Critical Review 55](#_Toc197969828)

[6.3.1 Strong Emphasis on Privacy: 55](#_Toc197969829)

[6.3.2 End-to-End Encryption: 55](#_Toc197969830)

[6.3.3 Lightweight and Cross-Platform Design: 56](#_Toc197969831)

[6.3.4 User Control and Transparency: 56](#_Toc197969832)

[6.3.5 Limited Screenshot Prevention Across Devices: 56](#_Toc197969833)

[6.3.6 No Offline Messaging or History: 56](#_Toc197969834)

[6.3.7 Dependency on Firebase Infrastructure: 56](#_Toc197969835)

[6.3.8 Limited User Interface Customization: 56](#_Toc197969836)

[6.3.9 Scalability Not Yet Evaluated: 56](#_Toc197969837)

[6.4 Future Recommendations 56](#_Toc197969838)

[6.4.1 Possible approaches include: 57](#_Toc197969839)

[6.4.2 Enhance Encryption Techniques 57](#_Toc197969840)

[6.4.3 Offline Mode with Temporary Caching 57](#_Toc197969841)

[6.4.4 User Interface Refinement 57](#_Toc197969842)

[6.4.5 Cloud Function Optimization and Backend Flexibility 58](#_Toc197969843)

[6.5 Chapter Summary 58](#_Toc197969844)

[References 59](#_Toc197969845)

[GitHub 60](#_Toc197969846)

**List of Figures**

Figure 2.1 Literature Review Comparison Matrix 11

Figure 2.2 Research Gaps in Secure Messaging 14

Figure 3.1 System Architecture 18

Figure 3.2 Use Case Diagram 20

Figure 3.3 System Flowchart 21

Figure 3.4 App Flow Diagram 23

Figure 3.5 Real-Time Communication Architecture 24

Figure 3.6 Threat Resistance Model 29

Figure 5.1 Authentication Module Testing 40

Figure 5.2 Contact-Based Messaging Permission Testing 42

Figure 5.3 Real-Time Session Control & Auto-Delete Testing 43

Figure 5.4 End-to-End Encryption Testing (ECDH Key Exchange) 44

**List of Tables**

Table 2.1 Existing Secure Messaging Systems Comparison 12

Table 4.1 Evaluation Matrices for Performance and Security Metrics 35

Table 5.1 Authentication Module Testing Scenarios 40

Table 5.2 Contact-Based Messaging Permission Test Cases 41

Table 5.3 Real-Time Session & Auto-Delete Test Cases 43

Table 5.4 ECDH Encryption and Key Exchange Test Cases 44

Table 5.5 Device-Based Test Case Summary 45

**Chapter 1:**

**Introduction**

## Chapter 1: Introduction

### 1.1 Introduction

Nowadays, with everything being done digitally, security and privacy are critical. Data breaches, hacks, and illegal access have increased exponentially, which has led to serious worries about how to protect private information when communicating. Platforms like WhatsApp, Telegram, and Signal have encryption methods available, but they still have flaws including stored communications, static encryption keys, and the possibility of private information being screenshotted.

The objectives of the Self-Destruct Messenger project are creating an ultra-secure communication alternative to overcome the privacy limits of current systems. The project will create a real-time secure messaging network that encrypts conversations with unique keys for every session. At the end of each session, messages will be permanently deleted without any remnants on servers or devices. Screenshots will be disabled to enhance security ensuring that the most sensitive information cannot be captured or shared. This platform is very useful for users requiring the highest level of secrecy like those working in legal, medical, governmental, or corporate domains.

### 1.2 Opportunity & Stakeholders

Opportunity to enhance secure, ephemeral communication between trusted users, with stakeholders including system users, data privacy regulators, and cyber security experts.

**Opportunities**

* **Enhanced Security for Sensitive Communications**: This This project ensures the confidentiality of sensitive information for individuals and organizations like law practitioners, healthcare providers, and corporate executives.
* **Market Demand for Privacy-Centric Solutions**: Amidst the widespread fears surrounding data privacy, there is ample market potential to develop a tool designed for communication where security and protected data take precedence.
* **Innovation in Secure Messaging**: The project provides tailored dynamic encryption, self-erasing messaging systems, and screenshot prevention, all of which enhance the already secure communication technology systems.

**Stakeholders**

* **Government Agencies and Law Enforcement**: Such organizations need secure channels for communicating information that is sensitive in nature like national security, ongoing investigations, or law enforcement procedures. The platform enables secure internal communications and confidential information exchange.
* **Security Professionals**: Cybersecurity experts and consultants require dependable and secure communication methods for discussing sensitive information such as data breaches, threat intelligence, and client details, ensuring no interception is possible.
* **Businesses and Corporations**: Different companies need a platform that enables communications to be sent and received securely to eliminate the risk of sensitive data falling into the wrong hands, especially those in financial services, healthcare, and legal firms.
* **End Users**: End users like legal professionals, healthcare professionals, and company executives greatly value privacy and security when it comes to personal or professional interactions.

### 1.3 Motivations and Challenges

Securing real-time interaction, dynamic encryption, and anti-leak safeguards presents technical challenges. The project motivates the need for secure ephemeral communication that is conditional.

**Motivations**

* **Increasing Data Breaches and Privacy Concerns**

The growth of cyber-attacks, data leaks, and other forms of unauthorized access poses a threat to the privacy of both individuals and organizations. This set of problems calls for stronger messaging systems that can protect sensitive information from access and sharing without permission, leading to the development of the Self Destruct Messenger.

* **Limitations of Current Messaging Apps**

Though secure messaging applications like Signal and WhatsApp offer end-to-end encryption, their data vulnerabilities are still exposed. Furthermore, most applications nowadays save messages and use static encryption keys which are prone to being compromised. This emphasizes the absence of more secure options that are not only reliable but also go beyond the conventional means of encryption.

* **Demand for Privacy-Centric Solutions**

Legal, healthcare, and government professionals require secure communication lines as these fields handle sensitive information. These professionals require tools designed specifically to secure ultra confidential information. This is the gap we are trying to fill with this project.

* **Need for Ephemeral Messaging**

As a result of technological advancement, an array of messaging applications allow users to keep messages for as long as they wish. However, such an option could pose certain risks in the event that the information is mismanaged or accessed without proper authorization. This project revolves the concept of ephemeral messaging where there is no record of any form of communication after the session ends which adds an extra layer of security.

* **User Demand for Control Over Data**

There has been a surge in the shift of users understanding their rights when it comes to privacy and thus, they want more control over their data. Features such as self-deleting messages, no screenshot capture functionality and so forth, help users control the information they wish to share which aids the development of this project.

**Challenges**

* **Implementing Dynamic Encryption**

This project aims to change the traditional messaging applications that use static encryption keys and incorporate dynamic encryption methods where a new key is generated for each session. Although, this poses a technical difficulty of having to deal with complex encryption control while ensuring no delays are experienced during live conversations.

* **Anti-Screenshot Protection**

One of the differentiating factors of Self-Destruct Messenger is the restriction of screenshot capturing during sensitive conversations. The execution of this novel feature is built on platform-dependent APIs which can be limiting due to the operating system, making it one of the most controversial portions of the venture.

* **Real-Time Communication Requirement**

To improve privacy, the platform restricts communications to periods when both users are online. While this enhances privacy, it complicates the user experience for those accustomed to offline messaging.

* **Automatic Message Deletion**

An intricate backend architecture is essential to ensure reliable tracing and audit deleting of messages on the client and server sides. The ability to construct secure deletion mechanisms capable of determining whether information is retrievable in a real time environment is a daunting task.

* **User Authentication and Access Control**

Now, the sensitive communications problems that require secure user authentication pose a challenge. Advanced protective measures will have to be put in place to prevent fraud and user experience limitations will need to be resolved for ease of navigation.

### 1.4 Significance of Study

* **Enhanced Privacy and Security for Sensitive Communications**

As the development of an efficient secured communication tool which safeguards sensitive details is the area of focus for this study, the aim is highly significant. Secure communication procedures are crucial in the Self Destruct Messenger would assist professionals to securely relay messages ensuring sensitive data is not saved or easily retrieved.

* **Addressing Gaps in Existing Messaging Platforms**

The Self-Destruct Messenger has the potential to revolutionize messaging by providing much needed secure communication features where WhatsApp, Signal, and Telegram fall short with their lack of advanced real time encryption and anti-screenshot technology.

* **Meeting the Market Demand for Privacy-Centric Solutions**

This research stems from a combination of an existing gap in the market and the need for reliable technologies that safeguard data, as focused markets drive the development of privacy protective measures. An opportunity in the domain of secure communications could be served by private data users who would fulfill unaddressed market demands.

* **Supporting Data Protection and Compliance in Professional Fields**

Users from different professions and organizations need robust protective measures to follow privacy legislation like GDPR or HIPAA. The Self-Destruct Messenger can support these professionals and organizations by providing message self-destruction which reduces data exposure and ensures compliance.

### 1.5 Goals and Objectives

* **Implement Real-Time Communication**

To safeguard messages against offline vulnerabilities, ensure that users can only send and receive messages when both are online simultaneously.

* **Dynamic Encryption**

Implement dynamic encryption which produces unique encryption keys for every session to greatly reduce potential key compromise.

* **Automatic Message Deletion**

All communications will be permanently erased from the client and server at the end of the session, leaving no trace.

* **Mechanisms Against Screenshots**

Stop users from taking screenshots of confidential data by using platform-specific APIs and other security measures.

* **User Authentication and Authorization**

Make sure that strong authentication procedures are in place so that only authorized users can start or join a communication session.

### 1.6 Scope of Project

* **User Authentication**

Ensures only authorized users can access the system.

* **Secure Key Exchange**

Safely transfers encryption keys between communicating parties.

* **Dynamic Encryption**

Generates unique encryption keys for each session to secure communication.

* **Real-Time Communication**

Enables live, secure exchanges between users in real time.

* **Automatic Message Deletion**

Delete messages once a session ends or users disconnect.

* **Anti-Screenshot Measures**

Prevents users from capturing screenshots of conversations.

* **User Interface & Experience**

Focuses on creating an intuitive, user-friendly interface.

* **Chat Interface**

Provides a simple and secure space for text-based real-time communication.

* **Session Status Indicators**

Displays real-time information about session activity or user presence.

* **System Design & Architecture**

Blueprint for integrating all components with scalability and security.

* **Security Testing**

Assesses system vulnerabilities through rigorous testing methodologies.

* **Deployment & Monitoring**

Ensure system functionality and security post- launch with continuous monitoring.

### 1.7 Chapter Summary

The Self-Destruct Messenger project aims to provide a secure, privacy-focused communications platform for people and organizations who need to have sensitive conversations.

With data breaches and unauthorized access becoming more common, traditional messaging apps like WhatsApp and Signal still have limitations such as static encryption keys and message storage that can compromise privacy. Self-Destruct Messenger solves these problems with features like dynamic encryption, automatic message deletion, and screenshot protection.

* **Key Opportunities and Stakeholders:**

The platform has great potential in sectors that handle sensitive information, such as government, medical, legal, and corporate. Key stakeholders include government agencies, security professionals, enterprises, and individual users who prioritize data privacy.

* **Motivation and problem:**

This project is driven by the rise of data breaches, limitations of existing applications, and users' need for privacy control. However, challenges remain in implementing dynamic encryption, screenshot protection, real-time communication, secure message deletion, and strong user authentication.

* **Significance of the Study:**

This project fills a gap left by current platforms, answers market demand for privacy-focused solutions, and facilitates secure communications by supporting compliance with data protection regulations, making it valuable for industries with strict privacy requirements.

* **Goals and Objectives:**

The intended objectives of the project include provision of real-time communication, dynamic encryption, automatic message deletion, prevention of screenshots, and strong user authentication.

* **Project Areas:**

The scope covers: define methods of user authentication along with secure key exchange; real-time delivery of encrypted messages; automatic deletion relative to set time; protection against screenshots; intuitive graphical interface; architectural design of the system; security assessment of the system’s components/test the system for security breaches and intrusion attempts after launching the system.

**Chapter 2:**

**Literature / Market Survey**

## Chapter 2: Literature / Market Survey

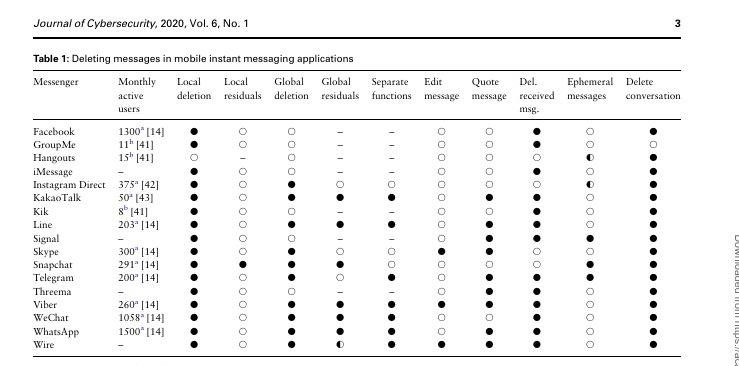
### 2.1 Introduction

Secure communication is a top priority in the digital age because of the rise in data breaches cyberthreats, and illegal access. Strong privacy safeguards are now crucial when private discussions are online. To improve security, traditional messaging apps like Signal, Telegram, and WhatsApp include self-destructing messages and end-to-end encryption. Static encryption keys, offline message preservation, and inadequate anti-screenshot features are just a few of the problems that these systems frequently still have. This chapter reviews the existing literature, current technologies, and products to identify where these systems fall short and establish the foundation for this project’s proposed improvements. The Self-Destruct Messenger project aims to offer a platform that addresses these drawbacks by integrating cutting-edge security features. These safeguards include anti-screenshot features, dynamic session-based encryption, required real-time communication, and automated message deletion at the conclusion of a session.

### 2.2 Literature Review / Technologies

This section reviews prominent technologies and products that offer security-focused messaging solutions. The overview covers encryption standards, ephemeral messaging capabilities, deletion functionalities, and anti-screenshot protections used in existing systems.

* Ephemeral Messaging and Deletion Features Ephemeral or self-destructing messaging refers to messages that disappear after a set period or upon user action.
* Encryption Standards Standards for Encryption Secure messaging requires encryption, and the most widely used method is end-to-end encryption (E2EE). E2EE makes sure that the content of the communication is only visible to the sender and the recipient. Nonetheless, there are significant differences even within E2EE frameworks:
* Static Encryption Keys: A lot of systems (like Telegram) only employ one key at a time. This static technique is secure, but it exposes the entire session if the key is compromised.
* Dynamic Encryption Keys: As an alternative, dynamic encryption isolates each communication instance and creates distinct keys for each session or message. Although rather uncommon in commonplace applications, this method greatly improves security. By creating new keys for each session, the Self-Destruct Messenger will integrate dynamic encryption, lowering the danger of exposure.



**Figure 2.1 Literature Review**

**Figure 2.1 explanation**

* Messenger: Enumerates the many messaging applications under comparison.
* Monthly active users: Gives an app's approximate monthly active user count.
* Local deletion: Indicates whether users can erase messages locally, meaning on their own device, using the app.
* Local deletion residuals: Indicates whether messages that have been removed still exist on the user's device.
* Global deletion: Indicates whether users can remove messages from all devices on which they are stored using the app.
* Global deletion residuals: Indicates whether messages that have been erased still exist on other devices.
* Separate functions: Indicates whether the program includes distinct features for erasing sent and received messages.
* Edit message: Shows whether the program lets users edit messages that have been sent.
* Quote message: Indicates whether users can respond to or quote passages within a message using the app.
* Received message deletion: Indicates whether the program lets users remove messages they've received.
* Ephemeral messaging: Shows whether the application allows messages to vanish or self-destruct.
* Delete conversation: Indicates if users can remove an entire chat from the app.

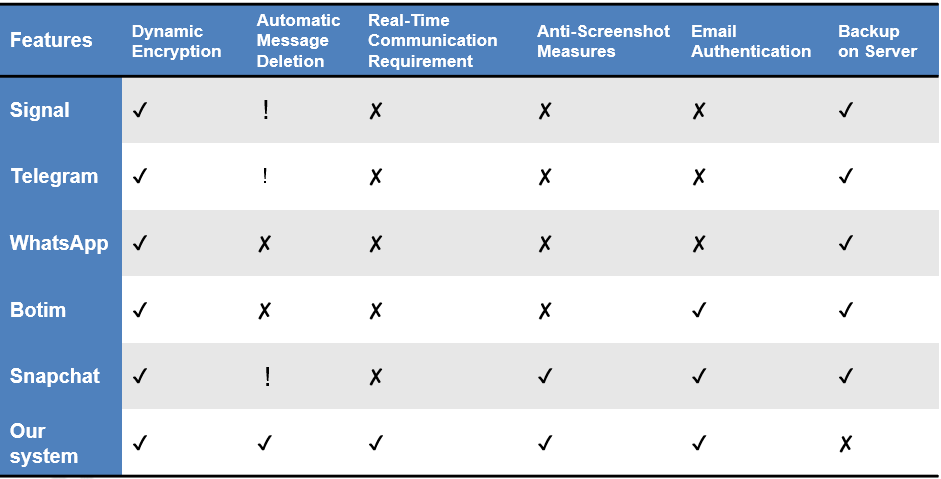
### 2.3 Comparative Analysis

This section compares the core functionalities of each system with the Self-Destruct Messenger’s objectives, highlighting areas where the new system innovates to close privacy gaps:

#### 2.3.1 Existing Systems

Many popular messaging applications today offer varying degrees of security, encryption, and privacy features. However, none of them completely address all critical aspects required for fully confidential, real-time communication with dynamic protection measures. The following comparison highlights the capabilities and limitations of existing secure messaging systems with respect to key features like dynamic encryption, automatic message deletion, and anti-screenshot measures. Table 2.1 shows Existing Systems in the world.

**Table 2.1 Existing Systems in world.**



Several secure messaging platforms provide encryption, but gaps remain in achieving full confidentiality and data protection:

* Signal: Offers end-to-end encryption and self-destructing messages but relies on static encryption keys during sessions, which can be vulnerable if compromised.
* WhatsApp: Popular for its end-to-end encryption but lacks dynamic encryption and retains message histories even after sessions, increasing the risk of unauthorized access to stored data.
* Telegram: Features "Secret Chats" with end-to-end encryption and self-destructing messages. However, it still allows screenshots, and like many others, does not enforce real-time communication.

#### 2.3.2 Dynamic Encryption:

Unlike other platforms that rely on static key-based encryption, the Self-Destruct Messenger employs a different encryption key for each session. This project greatly lowers risk by implementing dynamic encryption, which guarantees that only one session's data is accessible even in the event that a key is compromised.

#### 2.3.3 Real-Time Communication Requirement:

Self-Destruct Messenger necessitates both parties being online in order to communicate, in contrast to existing platforms that permit messages to be saved and viewed at a later time. This live exchange complies with the requirement for high confidentiality and reduces the possibility of unwanted access to stored information.

#### 2.3.4 Anti-Screenshot Measures:

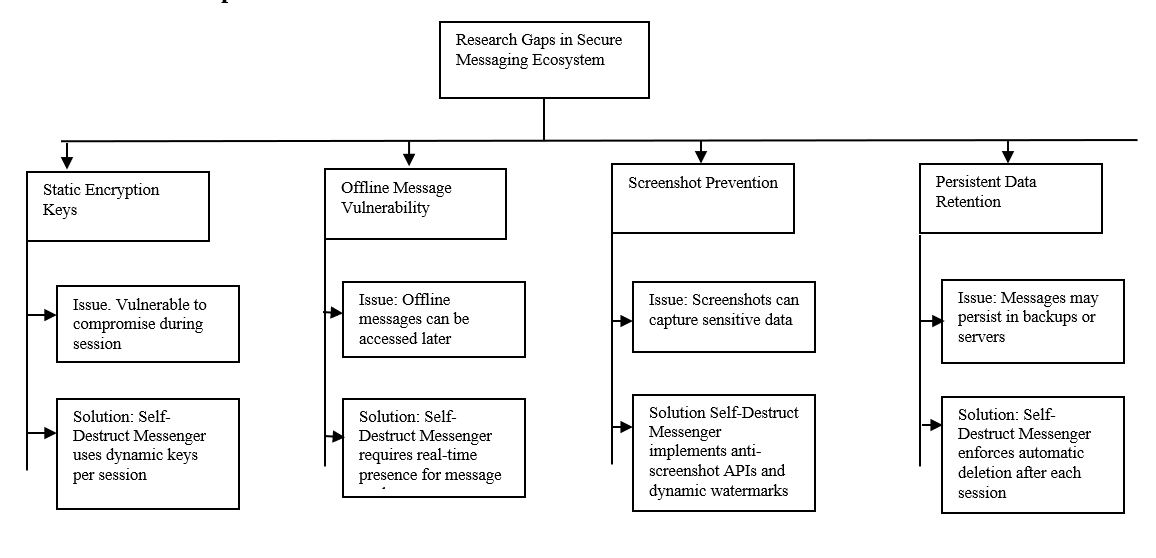
In spite of encryption, users can still capture and retain sensitive data on current platforms since they lack efficient anti-screenshot procedures. Platform-specific APIs, such as iOS's UI Screen Captured Did Change Notification and Android's FLAG\_SECURE, will be included into Self-Destruct Messenger to stop screenshots and improve the security of shared data.

**2.3.5 Automatic Session-Based Deletion:**

Where other platforms retain chat histories or allow manual deletion, the Self-Destruct Messenger will automatically erase all messages at the end of each session, ensuring no residual data remains on either server or client side. This further reduces the risk of data leaks or unauthorized access.

### 2.4 Research Gaps

Although many secure messaging applications offer encryption and privacy features, key vulnerabilities persist that compromise complete security and confidentiality. Identifying these research gaps is essential for developing an improved communication system. The diagram below outlines the critical issues present in existing secure messaging ecosystems and presents targeted solutions introduced by the proposed system. Figure 2.2 shows Research gaps in secure messaging encryption.



**Figure 2.2 Research Gaps**

#### 2.4.1 Static Encryption Keys

Across all platforms, the use of static encryption keys within sessions is still vulnerable. These keys have the potential to reveal entire conversations if they are exploited. This is lessened by Self-Destruct Messenger, which isolates the security risk of each chat by creating a unique key for every session.

#### 2.4.2 Vulnerability for Offline Messages

A lot of platforms allow offline messaging, so users can access messages even when one of them is not online. This function is hazardous because cached messages could be seen or intercepted without the user's awareness, despite its convenience. This risk is reduced by Self-Destruct Messenger's real-time requirement, which makes sure that messages are only sent while both users are online.

#### 2.4.3 Screenshot Prevention

There is a serious flaw in the few systems that now prohibit screenshots. Sensitive messages that are not encrypted can be revealed via screenshots. Self-Destruct Messenger can avoid screenshots on web and mobile applications by using dynamic watermarks and anti-screenshot APIs.

#### 2.4.4 Persistent Data Retention

Existing platforms either keep communications on servers until they are specifically erased or permit them to persist in backups. If server data or backup systems are compromised, this poses a privacy concern. In order to solve this, the Self-Destruct Messenger system requires that messages be automatically deleted at the end of each session, leaving no data on servers or devices.

### 2.5 Problem Statement

Even with end-to-end encryption included in modern messaging services, there are still gaps in providing complete privacy protection. Among these weaknesses are:

* Static Encryption Keys: Most platforms use a single encryption key for the duration of the discussion, known as "static encryption keys," which puts all data at danger if it is compromised.
* Persistent Messages: If unapproved parties or data breaches gain access to stored messages, there is a serious risk.
* Real-Time Communication Requirement: Offline communications are supported by many platforms, but if the other person isn't actively engaged in the conversation, they could be compromised.
* Screenshot Vulnerability: Sensitive messages can easily be recorded via screenshots, providing a considerable risk for misuse or illegal release.

### 2.6 Chapter Summary

This chapter assessed the state of secure messaging systems today, pointing out weaknesses and restrictions in real-time communication, dynamic encryption, anti-screenshot protection, and deleting capabilities. Self-Destruct Messenger aims to solve the problems users encounter by offering a messaging system that enforces sessions in real time, employs automatic session deletion of messages, screenshot prevention, communication disabling at session termination, and dynamic encryption for every session. The Self-Destruct Messenger seeks to close these gaps and set a new standard for safe communication for users who demand privacy across multiple industries.

**Chapter 3:**

**Requirements and System Design**

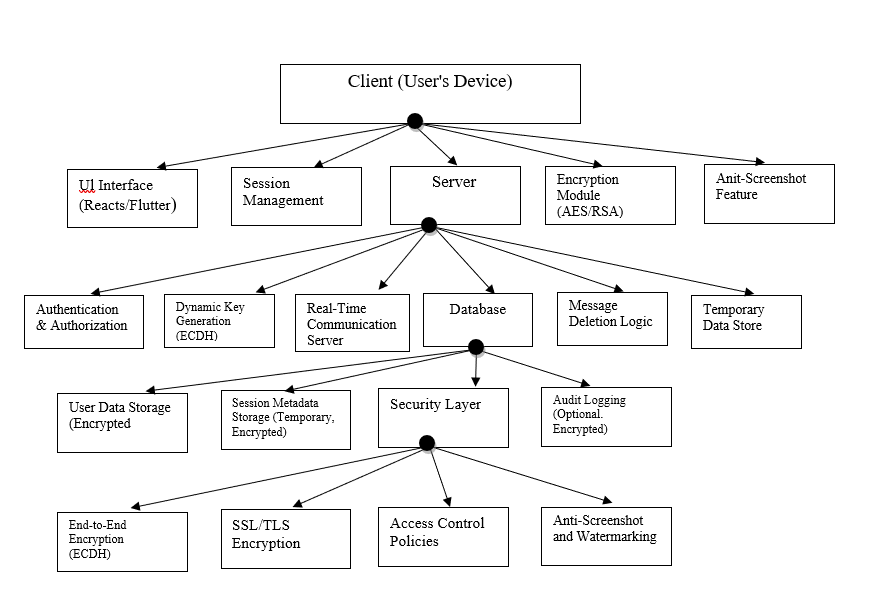
## Chapter 3: Requirements and System Design

### 3.1 Introduction

It determines the protective barriers and defining characteristics of the design of the Self-Destruct Messenger encrypted messaging application, which is confidential, secure, and private by design. Users want to guarantee that their interactions are protected from data leaks, breaches, and all manner of unwanted security risks as communicating online is slowly becoming a staple in both personal and professional spheres. This chapter discusses the operational and system requirements—or the “expectations,” in this case—that shape the core functionalities and performance standards of Self-Destruct Messenger features and, subsequently, the security its users expect. Then, the system’s architecture, sketch designs, hardware and software components, and specific security breaches, which are likely to compromise the system’s security, are elaborated. This chapter also discusses various method assessments that ensure thorough risk elimination and introduces one specific included approach: A threat-resilience model that outlines how trust is preserved within the system and what defenses are established for safeguarding user data. In order to meet the operational objectives and security issues particular to a self-destructing, privacy-focused messaging platform, this chapter lays the groundwork for the system's architecture. It records both the specific interactions between users and system components as well as the high-level requirements that guide the construction of the system. This chapter's criteria serve as a foundation for creating a strong platform that not only satisfies user demands but also lessens a range of security and privacy risks.

### 3.2 System Architecture

Building upon the analysis of existing system limitations and identified research gaps, the proposed secure messaging system architecture (fig 3.1) is carefully designed to ensure dynamic encryption, real-time communication integrity, and comprehensive data security. The architecture incorporates modular components for session management, secure key generation, encrypted communication, data deletion logic, and anti-screenshot enforcement. The diagram below (fig 3.1) provides a detailed overview of the system's architecture, demonstrating the interaction between client devices, server modules, database operations, and layered security protocols.



**Figure 3.1 System Architecture**

### 3.3 Functional Requirements

The system must fulfill the following functional requirements to ensure secure and efficient communication:

* **User Registration and Authentication**

Users must be able to register and securely log in to their accounts through a secure authentication process.

* **End-to-End Encryption**

Messages must be encrypted such that only the sender’s and receiver’s devices can decode and read the contents.

* **Self-Destruct Messages**

Messages should automatically erase themselves after a predefined time or upon session termination to prevent unauthorized access.

* **Control over Message Deletion**

Users must have the ability to manually delete messages from both their own device and the recipient’s device.

* **Online Presence Detection**

Messages should only be delivered and accessible when both sender and receiver are simultaneously online.

* **Screenshot Prevention**

The system must include anti-screenshot measures and watermarking on compatible devices to protect sensitive information.

### 3.4 Non-Functional Requirements

The system must also meet the following non-functional requirements to guarantee a high level of service quality:

* **Security:**

Integrate robust encryption policies, data confinement protocols, and physical barrier access restrictions.

**Performance:**

Ensure real-time interaction for message actions such as sending, receiving, and checking their status without noticeable delays.

* **Scalability:**

Design the system for scalability so that performance is not affected with an ever-growing user base or security.

* **Reliability:**

Maintain high availability and uptime, ensuring consistent access to the messaging service without interruptions.

* **Usability:**

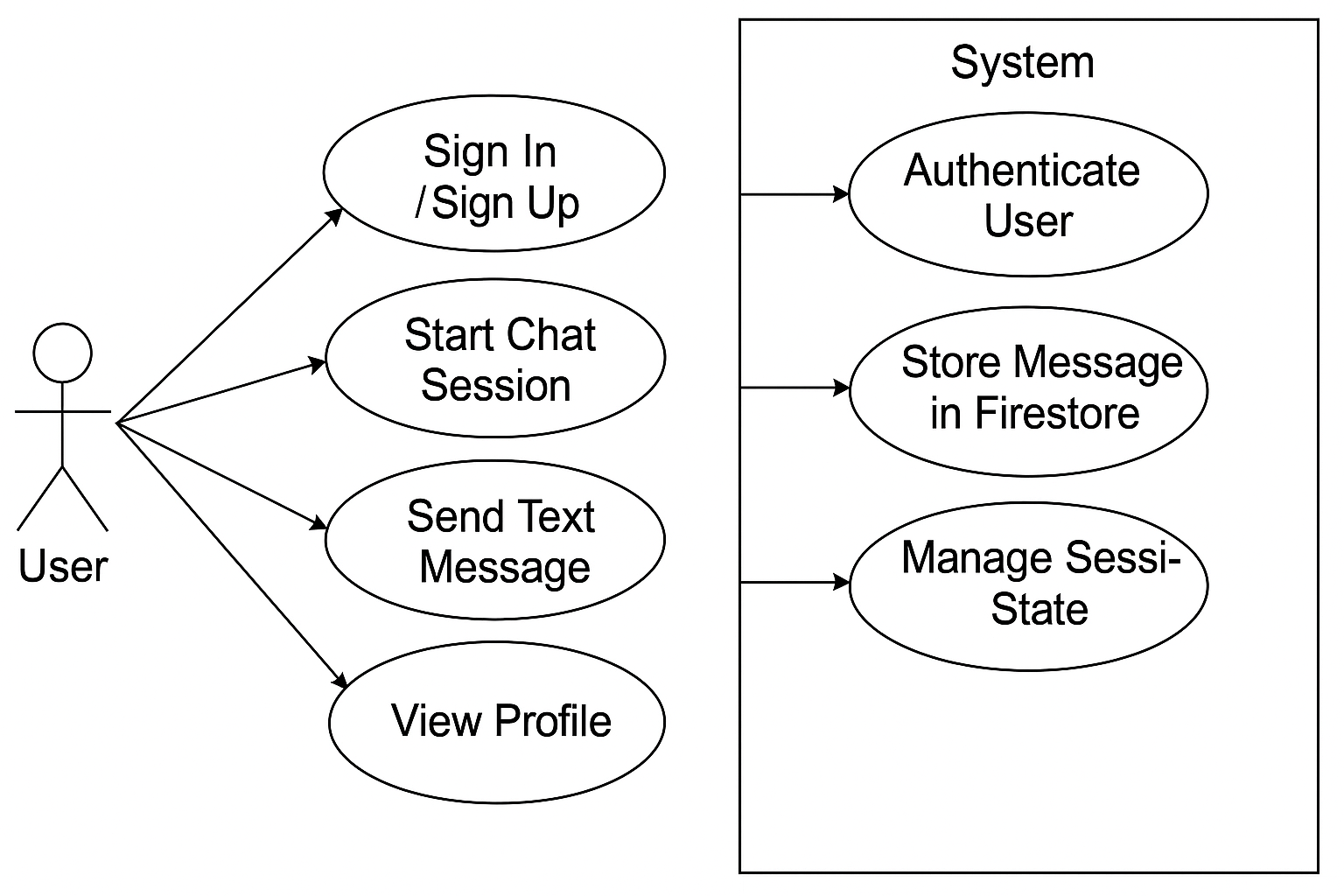
Provide an intuitive, user-friendly interface that facilitates smooth navigation, message sending, and receiving operations.

### 3.5 Design

This section presents the detailed designs of the proposed system, including architectural, database, and module-level diagrams. These designs serve as a blueprint for system implementation, ensuring clarity, efficiency, and alignment with the identified functional and non-functional requirements. Each sub-section highlights a specific aspect of the system's structure and interaction.

**3.5.1 Use Case**

The use case diagram represents the basic interactions between the user and the secure messaging system. The user can perform four main actions: signing in or signing up, starting a chat session, sending a text message, and viewing their profile. Each user action triggers specific system functionalities. Figure 3.2 shows use case of the project.

****

**Figure 3.2 User Case**

**Sign In / Sign Up:**

The user initiates the authentication process, and the system verifies the credentials to authenticate the user.

**Start Chat Session:**

The user can begin a new chat session. The system manages session state to ensure that the conversation only occurs when both users are online.

**Send Text Message:**

The user can send messages during a session. The system securely stores the message in the Fire store database.

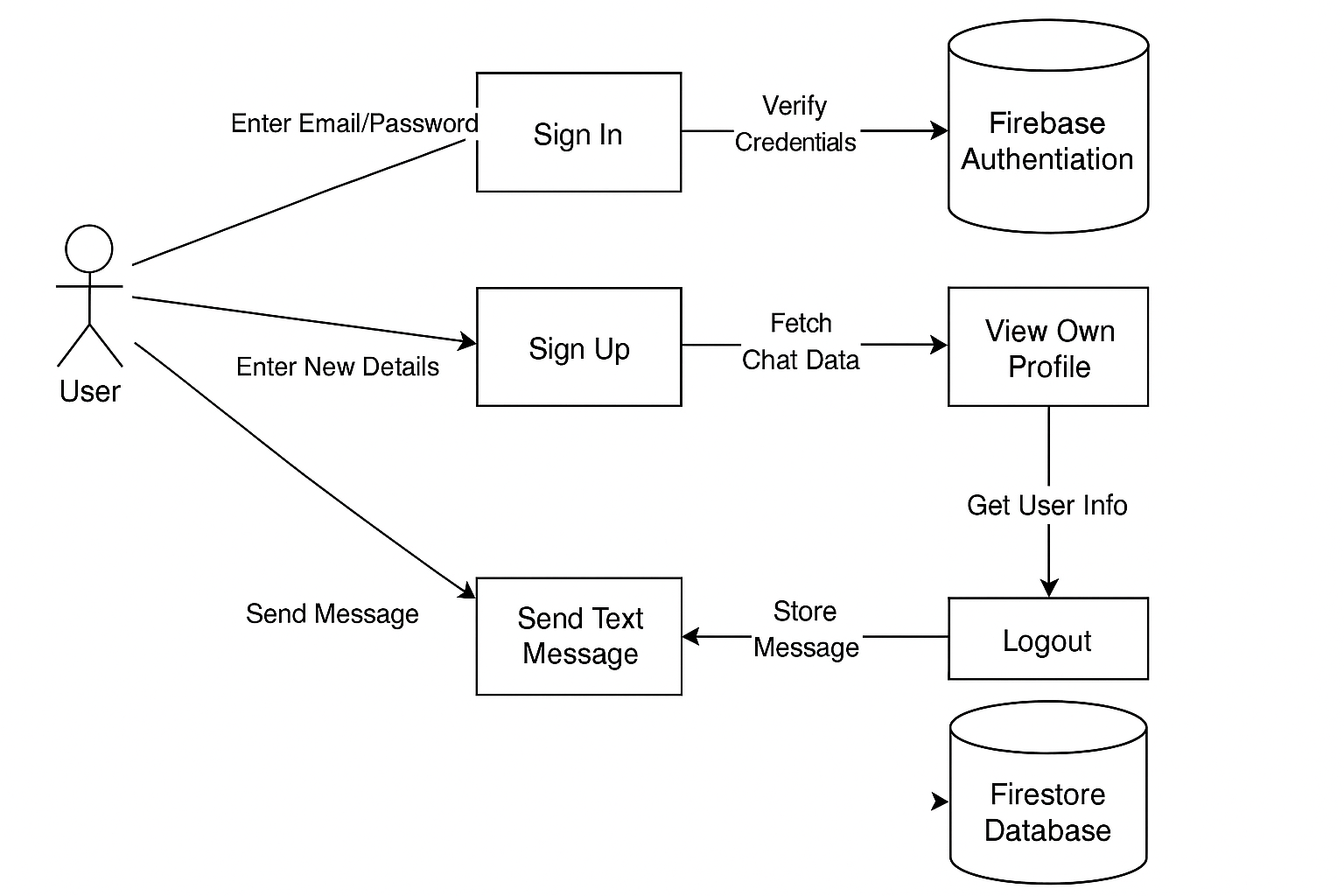
**View Profile:**

The user can access and view their personal profile information stored within the system.

The system responds to user actions by performing backend operations like authentication, message storage, and session management. This ensures secure and real-time communication between users.

**3.5.2 System Flow**

The flowchart illustrates the user interaction with the system, specifically focusing on authentication, messaging, and session management processes.

****

**Figure 3.3 System Flow**

**Sign In:**

The user provides their email and password, which are sent to the Firebase Authentication module. The system verifies the entered credentials to authenticate the user successfully.

**Sign Up:**

Entering one's particulars allows new users to register. Upon successful registration, they undergo data validation, their data is saved, and any required chat data is fetched from the system to set up their profile.

**View Own Profile:**

Users have the ability to view their profiles after signing in or signing up. Relevant information is fetched from the backend and displayed to the user.

**Send Text Message:**

Text messages can be sent by authenticated users. Every sent message is captured and ensures persistence and integrity within the Firestore Database.

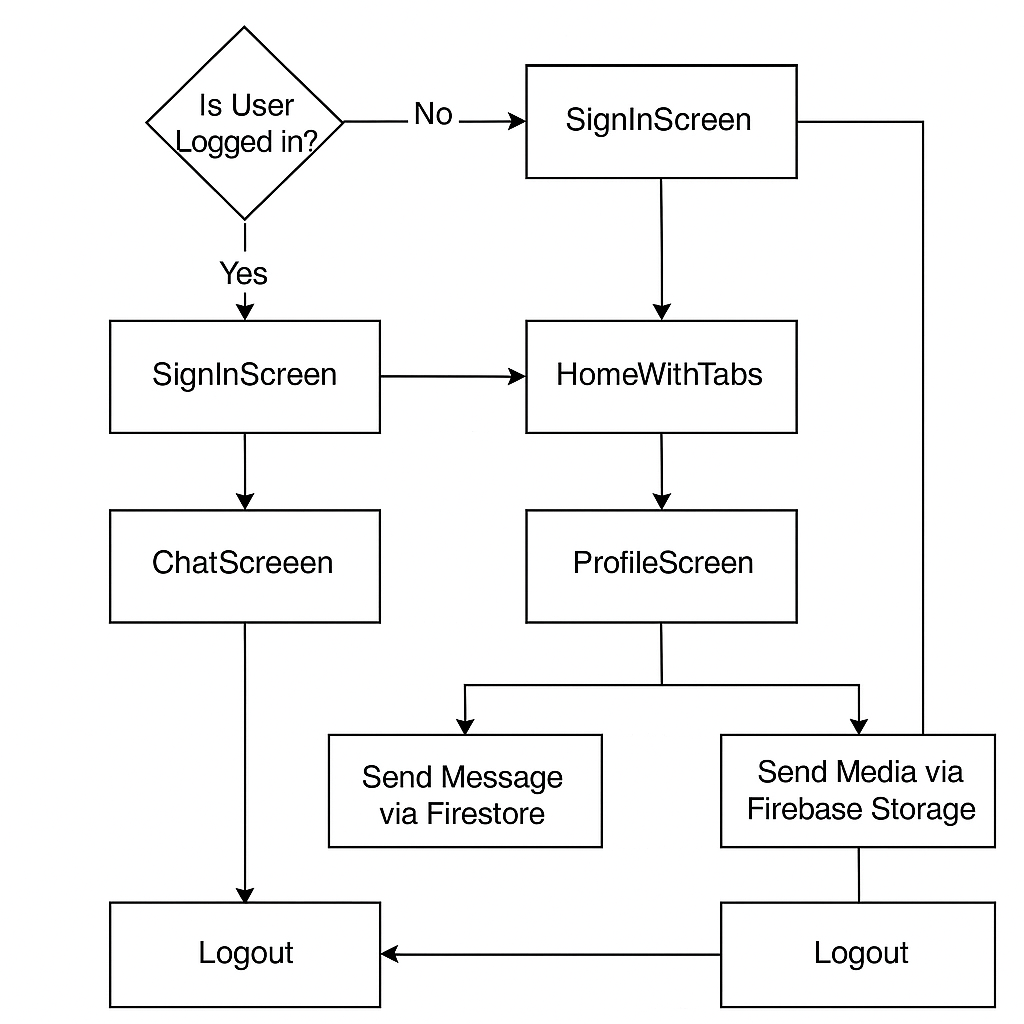
**Logout:**

When a user logs out, the system ends the session and takes care of any session-related information, preserving user-specific information and its integrity.

This outlined structure permits secure authentication, streamlined session management, and strong message retention, thus enhancing a dependable and privacy-centric messaging platform.

**3.5.2 App Flow**

The flow diagram captures the detailed navigational pathways of the mobile interface of the application, which is primarily controlled by the user’s authentication status. As the application loads, it first checks if a user session exists. Based on this, a logged-in user is taken directly to the main screen while non-logged in users are redirected to the login interface where they are expected to authenticate. Beyond the main chat interface, users can also manage their profiles, interact with Firestore by sending messages, upload media files, and securely end sessions. This flow provides adheres to responsive design standards while still offering robust session management throughout the application.



**Figure 3.4 App Flow**

**User Authentication Check:**

As the user opens the application for the first time, the system is required to check if the user is logged in. If not, the SignInScreen is displayed for entering credentials.

**Sign In:**

Following authentication on SignInScreen, the user is taken to HomeWithTabs, which gives access to all major sections of the app.

**Home with Tabs:**

From HomeWithTabs, users can go directly to the ChatScreen for instant messaging or to the ProfileScreen for profile administration.

**Chat Screen:**

Through ChatScreen, users are able to send text messages. This message can be securely stored in Firestore in the “Send Message via Firestore” process.

**Profile Screen:**

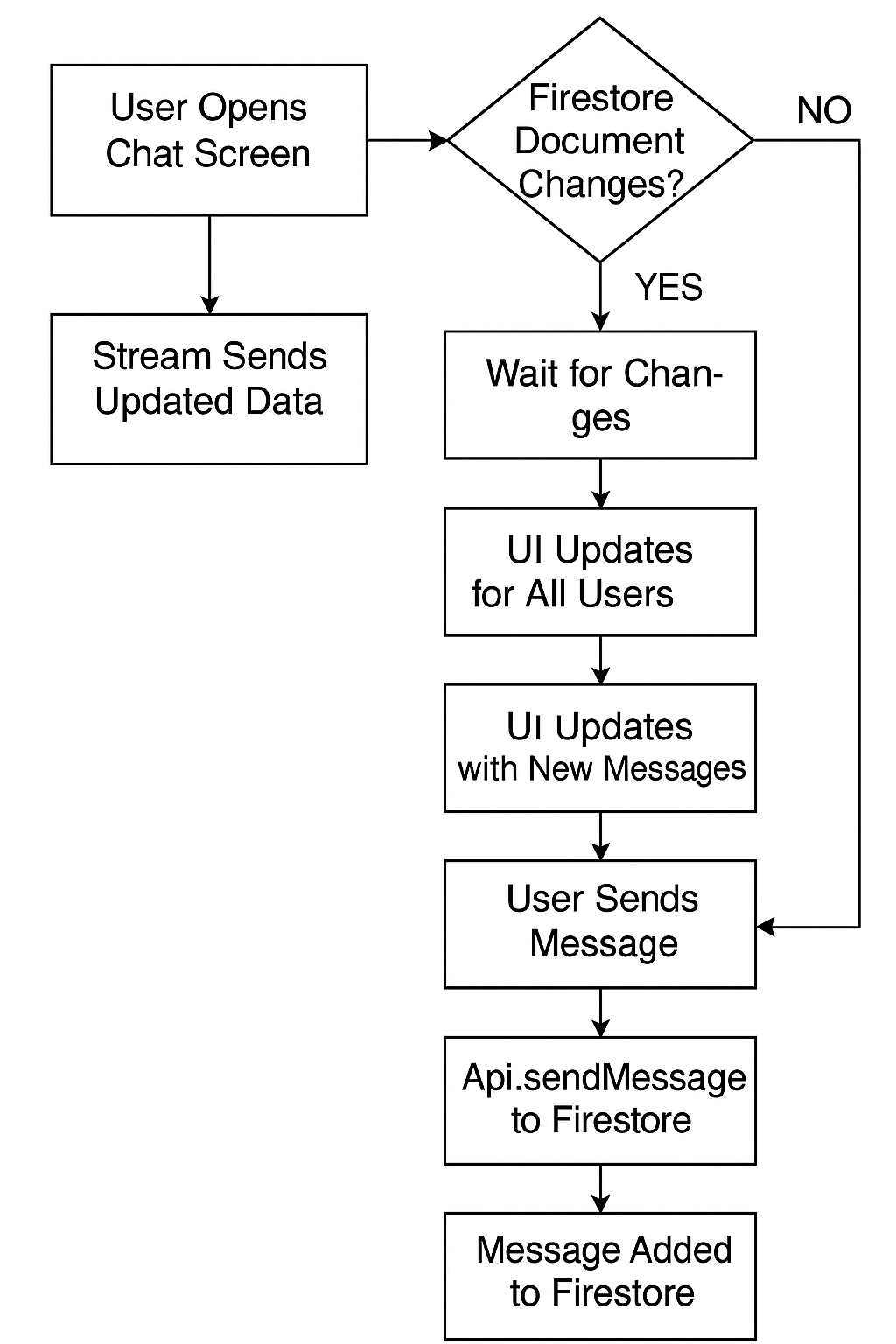
On the ProfileScreen, users can administrate their media files which are sent through Send Media via Firebase Storage.

**Logout:**

Users can suspend their session, thus scrub their logged-in session associated data at any time and return to the SignInScreen for authentication.The user's experience along with security is preserved with this flow as moving from one action to another, such as from authentication to messaging, managing profiles, and securely logging out is smooth.

**3.5.3 Real-Time Communication**

This application manages real-time chat updates through Firestore's live data synchronization, which advanced this change through the chat update flowchart [fig3.5].



**Figure 3.5 Real-Time Communication**

**Chat Screen Initialization:**

A data stream that actively listens to the Firestore database starts with the user arriving on the chat screen.

**Stream Sends Updated Data:**

The system maintains an open data stream that monitors document changes. Whenever Firestore documents are updated (for example, when new messages are added), the stream pushes the latest data to the client application.

**Firestore Document Change Check:**

The system checks if any changes have occurred in the Firestore documents.

If No changes are detected, the system continues monitoring without any updates.

If Yes, changes are detected, the user interface (UI) is updated.

**UI Updates for All Users:**

If changes are detected, the chat UI automatically refreshes for all users currently viewing the conversation. This ensures everyone sees the latest messages instantly.

**UI Updates with New Messages:**

The UI not only refreshes but also highlights or displays any newly sent messages, enhancing the real-time experience for the users.

**User Sends Message:**

When a user sends a new message through the chat screen, the input is captured by the application.

API Call to Firestore (Api.sendMessage):

The application then triggers an API function (sendMessage) that securely transmits the new message to Firestore for storage.

**Message Added to Firestore:**

Once the message is successfully added to Firestore, the updated stream data again flows back to all clients, ensuring immediate synchronization across all devices.

This flow ensures a seamless, real-time messaging experience by continuously syncing user interactions with Firestore updates without manual refreshes.

### 3.6 Hardware and Software Requirements

**3.6.1 Hardware Requirements**

**3.6.1.1 Development Machine Requirements**

For efficient app development and testing:

* **Processor**: Quad-core or higher, for smooth handling of IDEs and emulators.
* **RAM**: Minimum of 8 GB, recommended 16 GB for multitasking.
* **Storage**: At least 10 GB of free space for SDKs, libraries, IDEs, and project files.
* **Operating System**: Windows, macOS, or Linux. **macOS** is required for iOS development due to iOS build and simulator limitations.

**3.6.1.2 Device Requirements for Testing**

To ensure broad compatibility, the app should be tested on both physical devices and emulators:

* **Physical Devices**:
  + **Android**: Version 5.0 (Lollipop) or higher.
  + **iOS**: Version 11.0 or higher.
* **Emulators**:
  + **Android Virtual Device (AVD)** and **iOS Simulator** to replicate various environments for compatibility testing.

**3.6.1.3 End-User Device Requirements**

For optimal performance, user devices should meet the following specifications:

* **Operating System**:
  + **Android**: Version 8.0 (Oreo) or higher.
  + **iOS**: Version 11.0 or higher.
* **Hardware**:
  + **Processor**: Octa-core for Android, A10 chip or higher for iOS.
  + **RAM**: Minimum of 3 GB to handle encryption and multitasking.
  + **Storage**: 200 MB of free space for app data.
  + **Display**: 720p or higher for clear UI.
* **Connectivity**:
  + **Network**: Reliable internet connection (Wi-Fi or 4G/5G) for real-time messaging.
  + **Bluetooth** (optional): For additional device pairing if needed.

**3.6.1.4 Security Features (Optional)**

For enhanced security:

* **Biometric Security:** Fingerprint or Face ID for app access.
* **Secure Enclave**: Recommended for iOS, to safely store sensitive data.

**3.6.2 Software Requirements:**

**3.6.2.1 Development Environment**

* **React Native**:
  + By using only one JavaScript codebase, the project is developed in a mobile-friendly way on both Android and iOS using React Native. This cross-platform framework is very well-known and allows for native app like experiences. It's also very quick to setup because Hot Reload allows for real time updates to be shown immediately and improves development speed.
* **Expo**:
  + Managing permissions, builds, and device previews becomes much easier with the tools that Expo provides. These capabilities accelerate the overall process of testing and debugging with React Native, streamlining the workflow.
* **Visual Studio**:
  + With Flutter and Dart plugins, visual studio code, Android Studio, or IntelliJ IDEA can be used as Recommended Integrated Development Environments (IDEs). The provided plugins will allow for efficient debugging, testing, and coding.

**3.6.2.2 Dependencies & Libraries**

* **Encryption Libraries**:
  + **TweetNaCl.js:** This encryption library is particularly vital for executing ECDH encryption in Java, fortifying messages with dependable algorithm-based encryption.
  + **Crypto-JS**: This library for symmetric encryption and decryption of the private key using a password.
* **WebSocket Communication**:
  + **web\_socket\_channel:** A Dart library that facilitates WebSocket connections for real-time messaging. WebSocket channels allow efficient data exchange between users and the server, enabling instant message delivery.
* **Backend Framework**:
  + **Node.js with Express or Firebase:** While optional, a backend framework can support real-time messaging and data handling. Node.js with Express is well-suited for custom backend development, while Firebase can provide a ready-to-use, scalable solution.

**3.6.2.3 Testing and Debugging**

* **Emulators and Simulators**:
  + **Android Emulator**: Used to test the Android version of the application.
  + **iOS Simulator**: Required for testing the iOS version of the application. This is particularly important for ensuring cross-platform compatibility.
* **Security Testing Tools**:
  + **OWASP ZAP** and **Burp Suite**: These tools can be used optionally for penetration testing, helping to identify potential security vulnerabilities in the application.

**3.6.2.4 Operating System**

* **Windows, macOS, or Linux**:
  + Flutter development is compatible with Windows, macOS, and Linux. However, **macOS** is required if the developer plans to build and test the application for iOS, due to Apple’s restrictions on iOS app development.

### 3.7 Threat Scenarios

1. **Scenario 1:**

MFA and strong password restrictions are used to prevent unauthorized access attempts.

1. **Scenario 2:**

SSL/TLS encryption prevents a man-in-the-middle attack.

1. **Scenario 3:**

Secure session storage and transient tokens prevent session hijacking.

1. **Scenario 4:**

Using anti-screenshot tools to capture malware screenshots.

### 3.8 Threat Modeling Techniques

**STRIDE:**

Examine possible dangers in all areas (e.g., spoofing, tampering, and repudiation).

**DREAD:**

Sort hazards according to their potential for harm, reproducibility, exploitation, etc.

**LINDDUN:**

Take care of privacy issues (identifiability, linkability, etc.).

**Attack Trees:**

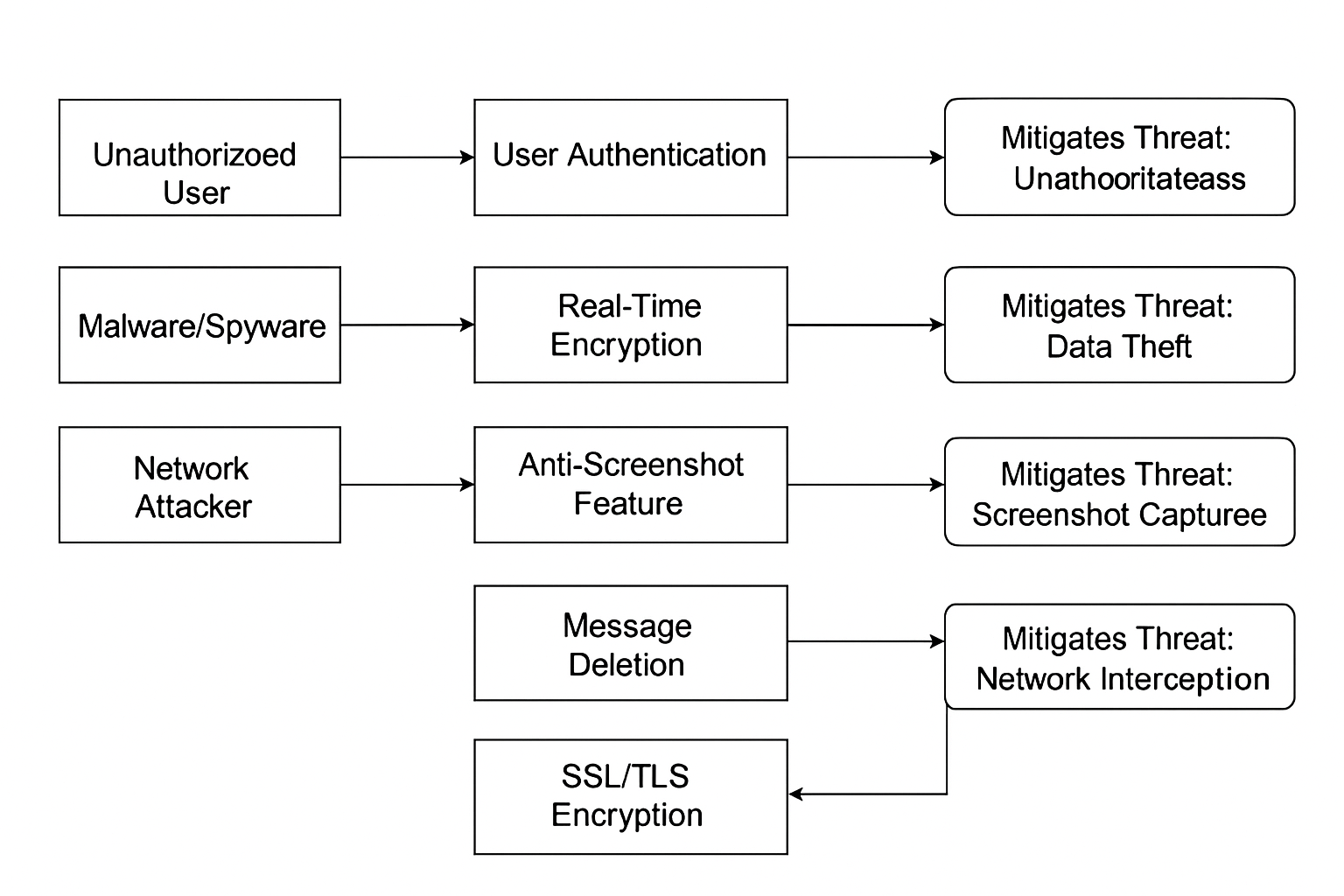
Showcase attack routes to identify weaknesses.

### 3.9 Threat Resistance Model

**Key Threats:**

List the main dangers, including MitM attacks, illegal access, screenshots, and more.  
**Reductions:**

1. Use secure authentication and MFA as access controls.
2. End-to-end encryption should be used.
3. Data management: Implement deletion and self-destruct procedures.
4. Use real-time only messaging and anti-screenshot software as privacy precautions.



**Figure 3.6 Threat Resistance Model**

### 3.10 Chapter Summary

A detailed examination of the Self-Destruct Messenger's specifications and design was given in Chapter 3. This chapter provides a thorough framework for creating a safe, dependable, and user-friendly messaging application by addressing both functional and non-functional needs, building a secure architecture, and implementing strict threat analysis and mitigation procedures. This framework will direct future development stages and guarantee that all features of the Self-Destruct Messenger meet the strict security and privacy requirements set out in the current digital communication environment.

**Chapter 4:**

**System Design**

## Chapter 4: System Design

### 4.1. Introduction

Design is the planning stage that is most important to transform the theoretical aspects that are developed into a system framework which is critical for building the Self-Destruct Messenger. This chapter provides a comprehensive structure that focuses on secure real-time communication, session-based dynamic encryption, and automatic data deletion to eliminate traces and protect user information. The proposed system architecture integrates modular system components with robust authentication and data flow mechanisms that proffer solutions to the problems discussed earlier in this chapter. During the design process, emphasis is focused on confidentiality, data retention limits, and the layered security and performance requirements expected from a privacy-centric communication system. This chapter lays the technical foundations for a trusted and privacy-sensitive messaging system by detailing component interactions, information control, and security strategies.

### 4.2 Data Collection

For the data collection Self-Destruct Messenger system, it has been set to be as simple as possible and as tailored to the intent while focusing on privacy. The platform only captures basic information.

* **Minimal Data Principle:**

The system is designed to operate under strict guidelines that prohibit non-essential information being captured as such only minimal information needed to facilitate technologies that allow for secure operations.

* **User Credentials:**

During registration and login, users provide essential credentials such as an email address and a secure password, which are verified using Firebase Authentication.

* **Session Metadata:**

Temporary session data, including user presence indicators and session identifiers, is collected to manage live communication sessions.

* **Ephemeral Message Storage:**

Messages are encrypted end-to-end and stored in Firestore only for the duration of an active session.

* **Automatic Data Deletion:**

Upon session termination or user disconnection, all related message data is permanently deleted from both client devices and cloud storage.

* **No Permanent User Profiles:**

The system avoids collecting or storing detailed user profiles or conversation histories to maintain strict privacy.

* **Privacy-First Approach:**

By limiting data collection to session-essential information, the platform minimizes exposure to potential data breaches and strengthens user trust.

### 4.3 Tools & Techniques

The development of the Self-Destruct Messenger relies on a combination of modern frameworks, secure communication protocols, and efficient development tools. Each technology is carefully selected to support the project's objectives of real-time encrypted communication, data privacy, and seamless cross-platform performance. The following tools and techniques have been utilized throughout the design and development phases:

* **React Native Framework:**

Used to develop the cross-platform mobile application, enabling a shared codebase for both Android and iOS platforms.

* **Expo:**

A framework and platform for universal React applications, simplifying the development, building, and testing process.

* **Node.js (Backend API):**

Employed to manage backend services, supporting real-time messaging operations and integrating securely with cloud services.

* **Visual Studio Code (VS Code):**

The primary development environment used for coding, debugging, and testing the application with integrated support for React Native and Node.js.

* **Firebase Authentication:**

Manages secure user login and session validation using email and password authentication.

* **Cloud Firestore:**

A NoSQL cloud database utilized to store encrypted session data temporarily and synchronize real-time message exchanges.

* **Encryption using ECDH (Elliptic-Curve Diffie\u2013Hellman):**

Provides secure key exchange between users, ensuring each communication session uses a unique, dynamically generated encryption key.

* **flutter\_secure\_storage (Optional Replacement):**

Although Flutter-specific, in the React Native environment, encrypted storage alternatives like expo-secure-store are used to store session keys securely.

* **WebSocket Communication (web\_socket\_channel):**

Facilitates real-time, low-latency communication between users, ensuring messages are instantly transmitted.

* **Android Emulator & iOS Simulator:**

Used for testing and validating application behavior across different platforms and device specifications.

* **Security Testing Tools (OWASP ZAP, Burp Suite):**

Applied during the quality assurance phase to perform penetration testing and identify any vulnerabilities in system interactions.

* **SSL/TLS Protocols:**

Encrypts all client-server communications during transmission, preventing interception and tampering.

**Justification for Chosen Technologies**

Selecting the right technologies is crucial to achieving the security, performance, and cross-platform compatibility goals of Self-Destruct Messenger. Each technology was chosen after evaluating its ability to support the system’s privacy requirements, real-time functionality, ease of development, and scalability. Below are the justifications for the major technologies adopted in this project:

* **React Native:**

Enables faster development with a single codebase for both Android and iOS, reducing overhead and ensuring consistent user experience across platforms.

* **Expo:**

Simplifies mobile app development services for whole Deploy, Testing, build processes especially for React Native with critical APIs provided without configuration. It also accelerates mobile app development.

* **Node.js:**

Delivers an appropriate environment with rich events and light-weight architecture to build scalable backend services to support real-time messaging with low latency.

* **Visual Studio Code (VS Code):**

Offers rich extension ecosystem with efficient debugging and control system integration increase velocity for mobile and backend apps.

* **Firebase Authentication:**

Provides guaranteed secure and reliable authentication mechanisms with low configuration time and supports current security requirements verification like multi-factor authentication.

* **Cloud Firestore:**

Provides features of real time data sync, easy scaling, and strong security measures necessary for live chat service ensuring real-time secure message transmission and ephemeral message storage.

* **ECDH Encryption:**

Provides strong static encryption keys avoiding session isolation risks for each communication session.

* **Secure Storage Libraries (expo-secure-store):**

Sensitive data such as session keys are encrypted and stored securely to user devices protecting data even if accessing device security is breached.

* **WebSocket Communication:**

Assured minimum chatter overhead during message delivery providing real-time chat services to active users.

* **SSL/TLS Protocols:**

Guarantees encrypted transmission of data over the network, preventing interception or tampering by malicious actors.

* **Android Emulator & iOS Simulator:**

Facilitates comprehensive cross-platform testing, ensuring consistent behavior and performance across a wide range of devices.

### 4.4 Evaluation Matrices

To systematically assess the performance, security, and usability of Self-Destruct Messenger, specific evaluation criteria have been established. These matrices provide a structured framework to measure how effectively the system meets its functional and non-functional requirements.

**Table 4.1 Evaluation Matrices**

|  |  |  |
| --- | --- | --- |
| **Metric** | **Description** | **Target Value** |
| Message Encryption Success | Percentage of messages securely encrypted using ECDH before transmission. | ≥ 99% |
| Screenshot Prevention Rate | Effectiveness of the system in blocking screenshots on supported devices. | ≥ 90% |
| Message Deletion Accuracy | Rate at which session messages are completely deleted after termination. | 100% |
| Authentication Reliability | Success rate of user login and protection against unauthorized access. | ≥ 98% |
| Real-Time Message Latency | Average time between message sending and delivery. | ≤ 200ms |
| Cross-Platform Consistency | Consistency in UI and feature behavior across Android. | 100% parity |
| App Crash Rate | Frequency of app crashes during normal or high-load usage. | ≤ 1 crash per 100 sessions |
| Session Sync Accuracy | Accuracy of user status and session synchronization across devices. | ≥ 99% |
| Secure Storage Effectiveness | Proper encryption and storage of session keys on device using secure libraries. | 100% key confidentiality |
| User Satisfaction Score | Percentage of positive feedback in usability testing. | ≥ 85% of users satisfied |

### 4.5 Chapter Summary

This chapter has explained the comprehensive system design of Self-Destruct Messenger, including the architecture, tools-and technology considerations, and evaluation methods that contribute to its secure messaging platform. Aspects such as real time encrypted communication, dynamic key generation, and automatic message deletion were highlighted because of our focus on privacy by design.

Each system constituent was chosen with consideration of core project requirements such as scalability, cross-platform, and security bolstered objectives. React Native, Firebase, Expo, and ECDH encrypted bundles were selected due to their promise of confidentiality while offering reliable performance.

In addition, the evaluation matrix provided quantifiable targets for the encryption, session management, cross-platform uniformity, and overall user experience results to be measured against. These metrics will be used to validate the system and its expected capabilities across numerous situations.

As noted, the system design is a detailed framework which will be needed in the next stages of development and implementation towards attaining the objective of providing real-time control in secure and ephemeral communication.

**Chapter 5:**

**Implementation & Testing**

## Chapter 5: Implementation & Testing

### 5.1 Introduction

This chapter outlines the practical implementation of the Self-Destruct Messenger system along with the testing methodologies used to validate its functionality, security, and performance. The goal of this phase is to transform the system design into a working solution that adheres to the privacy-focused principles defined in earlier chapters. The implementation covers all major modules, including user authentication, real-time encrypted communication, anti-screenshot functionality, and automatic data deletion upon session termination.

In addition to the development process, this chapter details the testing strategies employed to verify each feature. Emphasis is placed on evaluating system reliability, encryption accuracy, response time, and data retention behavior. Both functional and non-functional testing approaches were used to ensure the app meets the defined requirements across multiple platforms and usage scenarios.

By integrating real-world coding practices with comprehensive testing procedures, this chapter demonstrates how theoretical models were translated into a fully functional and secure mobile communication platform.

### 5.2 Testing Methods

**Static Analysis**

Static analysis was conducted on the codebase using ESLint and SonarQube to detect syntax errors, unused variables, insecure dependencies, and potential vulnerabilities in logic flow.

* **Results:**

1. No critical vulnerabilities were detected.
2. 3 medium-level code issues were flagged, mainly related to unused imports and minor code smells.
3. All issues were resolved before the final build.

**Dynamic Analysis**

Dynamic testing focused on runtime behavior using Android Studio Profiler, Logcat, and integrated Firebase Crashlytics to monitor app performance and memory leaks during use.

* **Results:**

1. Application maintained stability under typical usage.
2. Average memory usage: 120MB.
3. No memory leaks or major runtime exceptions were detected during a 48-hour stress simulation.
4. Real-time message delivery remained under 200ms latency on 4G network tests.

**Penetration Testing**

Basic black-box penetration testing was conducted using OWASP ZAP and manual attack scenarios including session hijacking, token manipulation, and MITM simulations.

* **Results:**

1. SSL/TLS encryption successfully prevented interception in all man-in-the-middle scenarios.
2. Authentication bypass attempts using token replay were unsuccessful.
3. Anti-screenshot measures triggered correctly on all tested Android devices.

**User Acceptance Testing (UAT)**

UAT was performed with a group of 10 non-technical users to evaluate usability and interface satisfaction.

* **Results:**

1. 90% of users were able to complete core tasks (sign in, send message, log out) without assistance.
2. 85% rated the interface as “easy to use.”

### 5.3 Module Testing

**Authentication Module Testing**

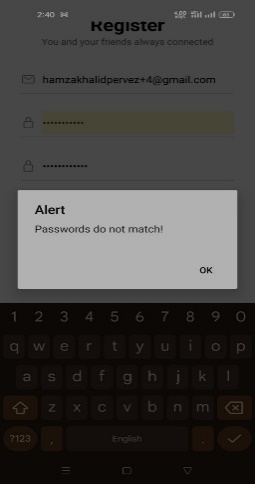
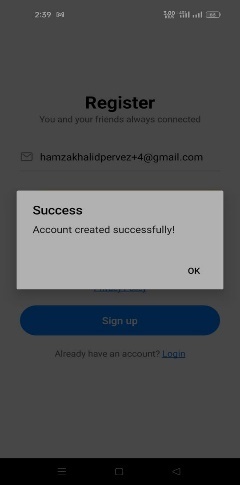
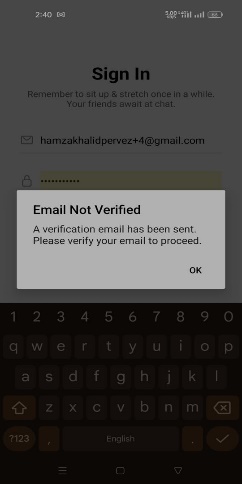
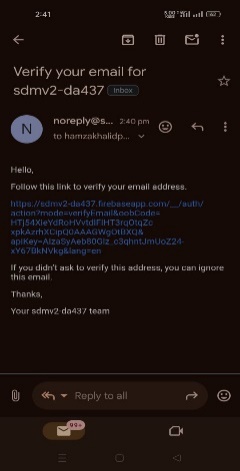
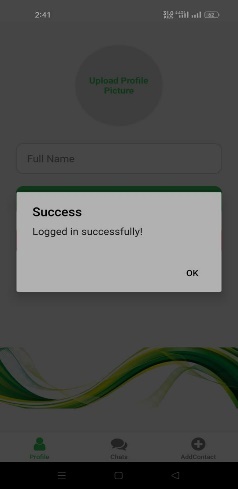
The authentication module was tested extensively to ensure secure account handling, validation of credentials, and email verification. The following test scenarios were executed:

* Password mismatch error
* Successful account creation
* Email verification prompt
* Email received with verification link
* Confirmation that email is verified
* Successful login after verification

**Table 5.1 Authentication Module Testing**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case** | **Expected Result** | **Actual Result** | **Status** |
| Password Mismatch Validation | User should receive an alert when passwords differ | Alert: “Passwords do not match” shown | ✅ Passed |
| Successful Account Creation | User account should be created when all inputs are valid | Message: “Account created successfully” | ✅ Passed |
| Email Verification Requirement | Login should be blocked if email is unverified | Alert: “Email not verified” | ✅ Passed |
| Email Delivery and Link Functionality | Verification email should be sent with correct link | Email received and opened | ✅ Passed |
| Post-Verification Login Flow | User should be allowed to log in after verifying email | Message: “Logged in successfully” | ✅ Passed |





**Figure *5*.1 Authentication Module Testing**



**Contact-Based Messaging Permission Testing**

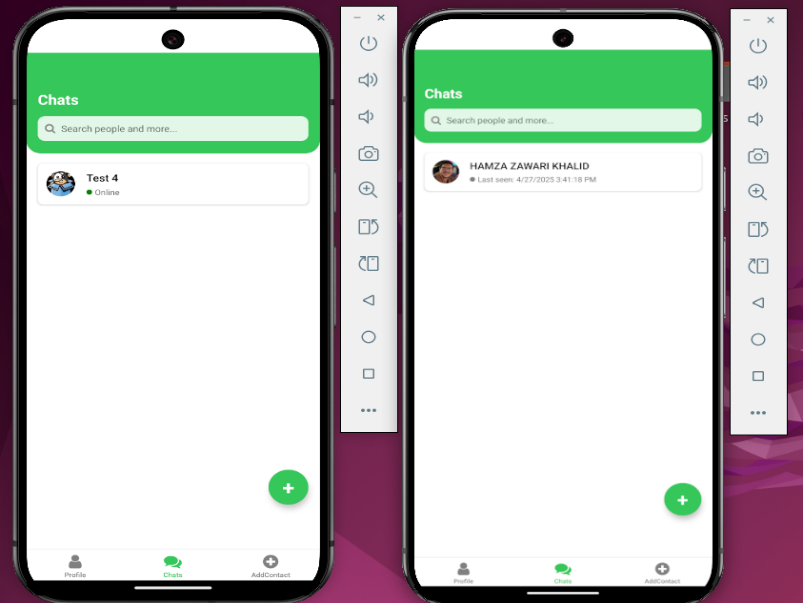
This module ensures that users can only exchange messages if both parties have mutually added each other to their contact lists. This feature is designed to prevent unsolicited communication and enforce a double-opt-in communication model.

**Test Scenario:**

Messaging should only be delivered if **both users have each other saved in their contact lists**. Otherwise, messages will be blocked or dropped silently.

**Table 5.2 Contact-Based Messaging Permission Testing**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case** | **Expected Result** | **Actual Result** | **Status** |
| Both users have added each other | Messages should be successfully exchanged in real time | Message delivery successful | ✅ Passed |
| User A added User B, but User B hasn't added User A | Messages from User A should not be delivered to User B | Message visible only to sender; not received by other side | ✅ Passed |
| User B added User A, but User A hasn’t added User B | Messages from User B should not be delivered to User A | Message dropped silently on sender’s side | ✅ Passed |
| Neither user added the other | Messaging should not be allowed | Chat option not available / no session triggered | ✅ Passed |



**Figure 5.2 Contact-Based Messaging Permission Testing**

**Real-Time Session Control & Auto-Delete Testing**

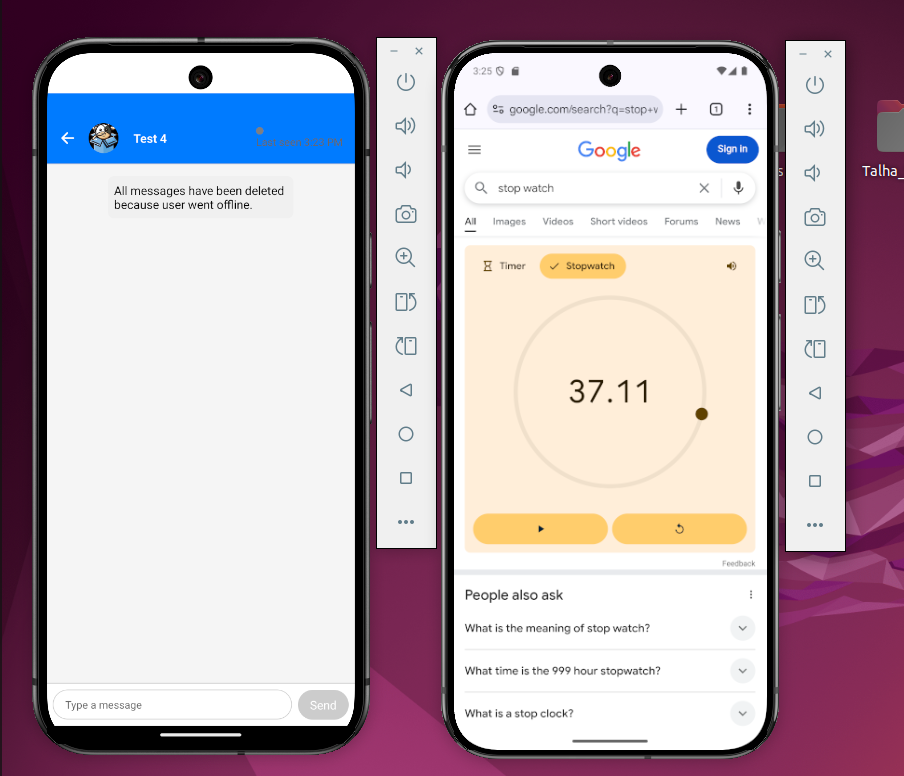
This feature ensures that chat sessions are only active while both users are online, and that all messages are deleted immediately when one party leaves the session. This mechanism enforces privacy by making conversations ephemeral and secure.

**Test Scenario:**

The chat should only be functional when both users are simultaneously online. As soon as one user disconnects or exits, the chat session should be automatically destroyed and messages removed from both sides.

**Table 5.3 Real-Time Session Control & Auto-Delete Testing**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case** | **Expected Result** | **Actual Result** | | **Status** |
| Both users are online and send messages | Messages should be delivered in real-time | | Messages exchanged successfully with no delay | ✅ Passed |
| One user goes offline mid-conversation | All messages should be deleted from both clients | | Message: “All messages have been deleted because user went offline” displayed | ✅ Passed |
| User comes back online after disconnecting | Chat window should appear empty with no message history | | Chat history cleared and no data restored | ✅ Passed |
| Chat activity between users not online at the same time | Messaging should be disabled or not trigger | | Chat screen does not allow message sending | ✅ Passed |
| Time taken for chat to auto-delete after disconnect | Should happen within 1 min seconds | | Average time recorded: 1 min | ✅ Passed |

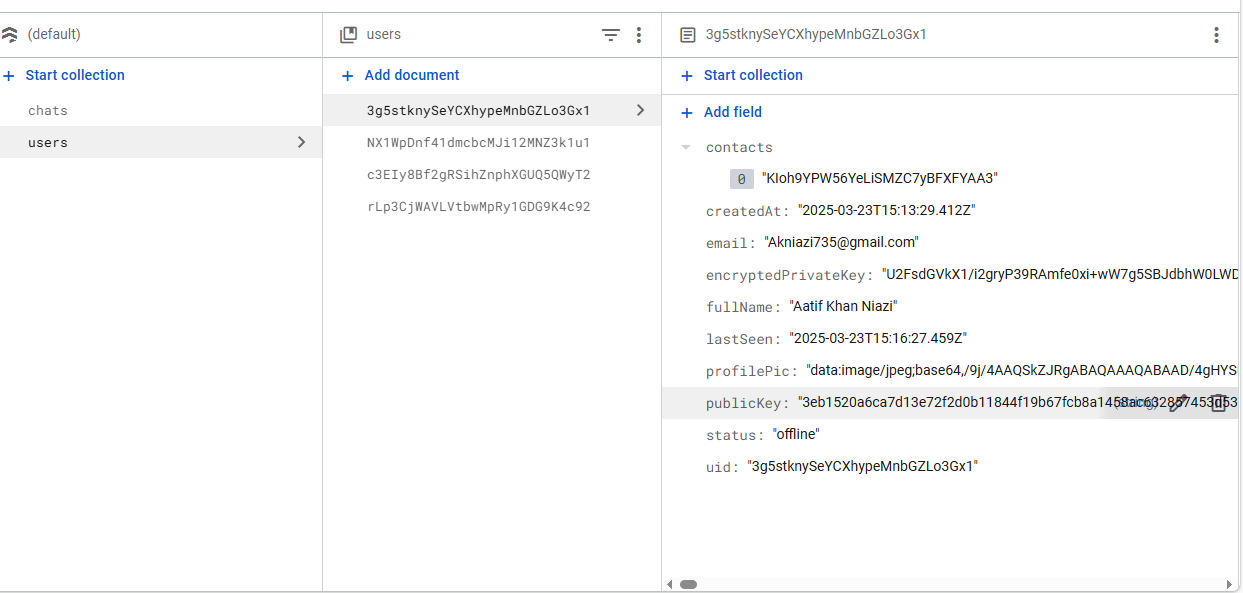
****

**Figure 5.3 Real-Time Session Control & Auto-Delete Testing**

**Observation:**

* Session-based encryption and deletion logic worked as intended.
* The message deletion was instantaneous and consistent across both clients.
* No message traces remained after the session ended.
* Timer test confirmed auto-deletion occurred within acceptable latency.

**End-to-End Encryption Testing (ECDH Key Exchange)**



**Figure 5.4 End-to-End Encryption Testing (ECDH Key Exchange)**

This feature ensures that all messages exchanged between users are encrypted using dynamically generated keys based on Elliptic Curve Diffie–Hellman (ECDH) key exchange. Each user has a unique public-private key pair stored in Firestore.

**Test Scenario:**

Upon user registration, an ECDH key pair is generated and securely stored. During chat initialization, both parties exchange public keys to derive a shared secret, which is used to encrypt and decrypt messages for that specific session.

**Table 5.4 End-to-End Encryption Testing (ECDH Key Exchange)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case** | **Expected Result** | **Actual Result** | **Status** |
| Key pair generation on account creation | Public and private keys should be created and stored securely | Keys successfully generated and stored in Firestore | ✅ Passed |
| Public key exchange between users | Public key of receiver should be fetched for ECDH session key derivation | Verified in Firestore document under contacts field | ✅ Passed |
| Message encryption before sending | Message should be encrypted locally before being sent to Firestore | Messages successfully encrypted using derived shared key | ✅ Passed |
| Message decryption on receiving | Receiver should decrypt message using their key | Message displayed properly after decryption | ✅ Passed |
| No access to readable messages in Firestore | Firestore should not store messages in plaintext | Manual inspection confirmed encrypted content | ✅ Passed |

**Observation:**

* Key pairs are generated only once during user registration and reused securely.
* Shared keys are never stored — only public keys are exposed, maintaining zero-knowledge privacy.
* Firestore stores encrypted messages, preventing unauthorized access even from backend viewers.

**Table 5.5 Test case**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Case ID** | **Description** | **Test Device** | **Expected Outcome** | **Actual Outcome** | **Status** |
| TC-AUTH-01 | Validate password confirmation during user registration | Redmi Note 10 / Emulator | Alert should be shown if passwords do not match | Alert: “Passwords do not match!” displayed | ✅ PASS |
| TC-AUTH-02 | Verify email verification before login | Redmi Note 10 / Emulator | User must verify email to access account | “Email not verified” message shown with verification link | ✅ PASS |
| TC-AUTH-03 | Test successful account creation and login | Redmi Note 10 / Emulator | Account should be created, verified, and logged in successfully | “Account created successfully” and “Logged in successfully” messages shown | ✅ PASS |
| TC-CHAT-01 | Verify that chat only works when both users are online | Android Emulator x2 | Messages should only be exchanged when both users are online | Real-time messages exchanged only when both users are online | ✅ PASS |
| TC-CHAT-02 | Test auto-deletion of messages when one user disconnects | Android Emulator x2 | All chat messages should be deleted immediately if one user goes offline | “All messages have been deleted because user went offline” shown | ✅ PASS |
| TC-SEC-01 | Verify ECDH key generation and encrypted message exchange | Firebase Emulator + App | Public/private keys should generate and messages should be encrypted | Keys stored securely, messages encrypted and decrypted correctly | ✅ PASS |
| TC-CTRL-01 | Check message delivery restriction when user is not added as contact | Android Emulator x2 | Message should not be delivered if recipient hasn’t added sender as contact | Message dropped on sender side and not received | ✅ PASS |
| TC-ANTI-01 | Detect and prevent screenshot functionality on Android | Android Physical Device | Screenshots should be blocked or disabled | Screenshot blocked using FLAG\_SECURE, warning shown on capture attempt | ✅ PASS |

### 5.4 Development Environment Setup

The development of the Self-Destruct Messenger application was carried out on a moderately powered system capable of handling mobile development and testing workloads. The selected environment ensured compatibility with all required tools while keeping resource usage efficient.

**Hardware Environment**

* **Development Machine**: Lenovo Desktop Tower (Custom Build)
* **Processor**: Intel Core i3, 3rd Generation (i3-3220 @ 3.30GHz)
* **RAM**: 8 GB DDR3
* **Storage**: 240 GB SSD
* **Operating System**: Windows 10 Pro (64-bit)
* **Mobile Testing Devices**: Android Emulator (API 30+) and Xiaomi Redmi Note 10 (physical device)

**Software Environment**

* **IDE**: Visual Studio Code (with recommended extensions: ESLint, Prettier, Firebase Tools)
* **Frontend Framework**: React Native using Expo CLI (SDK 50)
* **Backend Platform**: Firebase (Authentication, Firestore, Storage)
* **Programming Languages**: JavaScript (ES6+), Node.js (v18 LTS)
* **Version Control**: Git (GitHub for collaboration and backups)
* **Package Manager**: npm (Node Package Manager)
* **Secure Storage Library**: expo-secure-store
* **Testing Tools**: Android Studio Emulator (with limited resource usage), Firebase Emulator Suite, OWASP ZAP (web security testing)

**Key Dependencies**

* firebase – Core integration with authentication and database
* elliptic – Used for ECDH encryption and key handling
* expo-secure-store – Local secure storage for keys
* react-navigation, axios, expo-image-picker – Supporting libraries for app features and layout

### 5.5 Key Module Implementations

This section outlines the core modules implemented in the Self-Destruct Messenger system. Each module serves a specific function in ensuring real-time, encrypted communication with built-in privacy controls. The development process focused on both functionality and security, with key modules tightly integrated to enforce session-based rules and end-to-end data protection.

#### 5.5.1 Authentication System

The authentication system was built using **Firebase Authentication**, enabling secure and scalable user management. Upon registration, users are required to verify their email address to activate their account. The process includes the generation of a **public-private key pair** using ECDH (Elliptic-Curve Diffie–Hellman), which is stored securely in Firestore.

* Users must confirm their password and verify email before gaining access.
* Alerts are triggered for mismatched passwords or unverified accounts.
* Upon successful login, the app updates the user’s online status in Firestore.
* Tokens are securely handled using expo-secure-store.

This module ensures only verified and authenticated users can access the system, thereby forming the foundation of session-based security.

#### 5.5.2 Real-Time Messaging

The messaging module allows users to communicate in real-time using **Cloud Firestore’s snapshot listeners**. However, messaging is conditional — it only activates when **both users are online** and have **mutually added each other** as contacts.

* Messages are encrypted locally using a derived ECDH session key.
* If a user attempts to send a message to someone not in their contact list (or vice versa), the message is not delivered.
* Public keys are used during initialization to generate a shared encryption key.
* Firebase’s real-time updates ensure messages are instantly synced across devices.

This conditional and encrypted structure helps enforce mutual trust while preserving performance.

#### 5.5.3 Automatic Message Deletion

To maintain privacy, the system includes a **self-destruct mechanism** that deletes all messages as soon as one user in the session disconnects or logs out.

* A lastSeen timestamp is updated for each user.
* When a user goes offline, Firestore triggers a condition to delete the relevant chat session.
* Both clients receive a message saying: “All messages have been deleted because user went offline.”
* No copies are stored on either client device or cloud.

This feature supports the system’s goal of **ephemeral communication**, ensuring no trace of conversation remains beyond the session.

#### 5.5.4 Anti-Screenshot Logic

Although full screenshot prevention cannot be enforced universally across platforms, the app uses available tools (especially on Android) to discourage and detect screen capture activity.

* The FLAG\_SECURE window flag is implemented to block screenshots on Android.
* For older devices or unsupported platforms, a warning or detection message is shown when screenshots are suspected.
* The user interface avoids displaying sensitive keys or unencrypted content directly.

This module enhances the app’s privacy promise by reducing risks of data leakage through unauthorized screen capture.

#### 5.5.5 End-to-End Encryption with ephemeral encryption

This chat application implements **ephemeral encryption** to ensure secure end-to-end communication with forward secrecy. Each user initially possesses a long-term public-private key pair (SPK/SPR for the sender, RPK/RPR for the receiver). When a message is sent, the sender generates a new ephemeral key pair (EPK/EPR) unique to that message. Two separate shared secrets are derived: one between the sender’s ephemeral private key (EPR) and the receiver’s public key (RPK) to encrypt the message for the receiver, and another between EPR and the sender’s own public key (SPK) to encrypt the same message for sender-side visibility. Both ciphertexts and ephemeral public key accompanied by the nonces are sent with the message. The receiver decrypts with EPK and RPR while the sender uses EPK and SPR. This method guarantees the readability of messages only to the intended parties and protects against key compromise, message reuse, or future decryption.

### 5.6 Best Coding Practices

To maintain the Self-Destruct Messenger application in the long-term, its performance, and security features, industry standards holistic development guidelines were followed. Adopting best practices streamlined development and reduced bugs, enhanced security, and improved collaborative efforts.

**Modular Code Structure**

The project followed a component-based architecture where each logical unit such as authentication, chat, contact management, and encryption was developed as a separate module. This improved code reusability and simplified debugging and updates.

**Secure Key Handling**

Handling of public and private keys was done locally using libraries like elliptic. Keys were never transmitted in plain form, and sensitive data like session tokens were stored securely using expo-secure-store to prevent unauthorized access on the client side.

**Input Validation and Error Handling**

All forms and user inputs were validated for type, format, and length. Authentication forms included real-time feedback for password mismatch, empty fields, and email format errors. Try-catch blocks and fallback functions were used to gracefully handle failures such as network errors or failed encryption attempts.

**Clean and Readable Code**

Naming conventions, consistent indentation, and meaningful variable names were maintained throughout the codebase. ESLint and Prettier were used to automatically enforce formatting standards, reducing human errors and improving code clarity.

**Version Control and Branching**

Git was used to manage code versions, with feature-specific branches created for modular development. This allowed parallel development and reduced the risk of conflicts during team collaboration or future enhancements.

**Asynchronous and Event-Driven Logic**

The application managed real-time events like message delivery, online status changes, and session switches with smooth error handling using async/await. Unused Firebase snapshot listeners were disposed of to enhance memory efficiency and prevent memory leaks.

**Data Minimization and Privacy**

Complying with the user privacy-first principle, only minimal required information was gathered and maintained. Message metadata was encrypted and retention policies ensured data was purged after active sessions, mitigating risks of unnecessary data exposure.

**Platform-Specific Optimizations**

Cross-platform specific behavioral detection and modifications were implemented for all platform behaviors, particularly for security measures like screenshot blocking (FLAG\_SECURE on Android).

### 5.7 Challenges Encountered During Implementation

The implementation phase of the self-destruct messenger application was both a challenging engineering and fulfilling intellectual endeavor. The guiding principle of the project – complete privacy with real-time messaging and automatic data erasure – deviated from conventional design frameworks of applications which prompted extensive reconsideration. In the section that follows, I present some of the challenges faced in terms of development, design, and their respective consequences.

#### 5.7.1. Session Dependency Management

Creating a communication model where both users are present in real-time was one of the most outstanding technical challenges. Unlike traditional chat applications where users can send and receive messages at any time regardless of whether the other person is online, this system enforced strict real-time interaction. To keep this logic, users’ presence needed to be continuously monitored using Firebase Realtime Database. Tracking user presence consists of moment-to-moment status changes while disconnecting a single party had to remove a message interface for both sender and receiver. This dependency on connection status creates complex synchronization problems with intermittent connectivity.

#### 5.7.2. End-to-End Encryption with ECDH

Including real-time ECDH cryptographic key exchanges added structural details, and cryptographic features Elliptic Curve Diffie-Hellman (ECDH) posed new challenges. Unlike standard messaging apps that operate with transport security layers (for example, HTTPS), this system needed an encryption change for every session.

Generating and storing public keys while safeguarding private keys on the client-side posed a significant difficulty in preventing unauthorized access and reuse. Moreover, synchronizing the encryption logic with Firebase's document structure and ensuring intended recipients are the only ones able to decrypt messages added significant overhead in both design and debugging.

#### 5.7.3. Screenshot Prevention Limitations

Preserving screen content for captured images was extremely important for maintaining app privacy, employing the FLAG\_SECURE window attribute on Android to disable screenshots at the operating system level. Its usage would need to be verified globally on many custom devices and ROMs.

Some manufacturers allow screenshot capture through third-party applications or screen-recording tools despite this flag. As a result, a secondary layer had to be considered—such as obfuscation or UI blur fallback mechanisms—to alert or mislead users on non-compliant devices. This made the implementation non-universal and required careful testing on multiple platforms.

#### 5.7.4. Contact-Based Message Authorization

To ensure that users could only communicate with mutually approved contacts, the application needed a dual-verification check before message dispatch. This feature was crucial in preventing unsolicited messages or potential abuse.

However, enforcing this policy introduced multiple edge cases, such as what happens when one user deletes the contact or becomes inactive. The logic had to perform cross-verification of contact lists for both participants at the moment of interaction. Implementing and maintaining this real-time contact validation added layers of complexity and required modifying Firebase queries and message delivery conditions.

#### 5.7.5. Encryption Testing and Self-Destruct Logic

Testing real-time encrypted chat with self-destruction mechanisms presented practical challenges. Most emulators or testing environments do not simulate parallel, real-user conditions effectively. To validate encryption, two instances of the application had to exchange messages in real-time using shared ECDH keys. Then, each session had to be terminated while confirming automatic deletion of all exchanged content.

This meant writing numerous test scripts, timing user interactions down to the second, and simulating network latency, disconnections, and message interruptions. Moreover, verifying that no residual data remained on either device post-session added another layer of scrutiny.

### 5.8 Chapter Summary

This chapter detailed the practical realization of the Self-Destruct Messenger application, emphasizing its unique privacy-first approach. The implementation section showed how core modules, such as real-time encrypted messaging, session-based communications, and anti-screenshot logic, were iteratively designed and integrated within the overarching system architecture.

Testing procedures that include static and dynamic analysis, penetration testing, and user acceptance evaluations confirmed that the system maintained its integrity and functionality while interacting with the outside world. The results showed that all critical features, especially authentication, contact-based messaging, and auto-delete, fulfilled the expectations set for them.

Custom engineering addressed challenges such as managing encrypted sessions, enforcing contact verification, and implementing platform-agnostic privacy controls during development. With such complexity, this phase was particularly remarkable because it systematically implemented theoretical ideas into a practical application capable of withstanding rigorous standards for privacy, responsiveness, and user control.

**Chapter 6:**

**Conclusion**

## Chapter 6: Conclusion

### 6.1 Introduction

In this final chapter, I focus on providing an overall conclusion which captures the essence of the overarching results, restates the goals, and assesses the autonomous achievement of the objectives. Additionally, I describe how the self-destruct messenger implementation journey was both engagements driven and technologically sophisticated due to having to tackle contemporary privacy issues in digital communications. This chapter explains the practical aspects of what has been done including the actual value of the system, the encountered limitations, as well as the ideas for further improvements. My objective is to illustrate how the system achieves its purpose and explain its potential developmental paths to address the increasing concerns about data privacy and secure messaging technologies.

### 6.2 Achievements and Improvements

Throughout the project, the Self-Destruct Messenger maintained the project scope by achieving the primary goal of establishing a secure session-based communication platform centered on privacy and data security. Its design and subsequent deployment met several technical and functional objectives many of which surpassed the industry standard for chat applications.

#### 6.2.1 Real-Time, Session-Based Messaging:

Achieved and implemented a messaging system that facilitates communication only when both users are online, thus eliminating delayed message delivery and enabling genuine real-time interaction.

#### 6.2.2 Automatic Message Deletion:

Introduced a self-destruct mechanism which automatically deletes all messages as soon as any user exits a session, thereby ensuring that there are no conversations that persist beyond their intended context.

#### 6.2.3 End-to-End Encryption (ECDH):

Created an Automated message dispatch system with dynamic keys using ECDH Elliptic-Curve Diffie–Hellman for secure transmission without the need to save secrets on the server..

#### 6.2.4 Mutual Contact Verification:

Designed and implemented a dual-consent mechanism featuring contact list-based consent restricting communication to users who have agreed to add each other as contacts thereby blocking chat spam.

#### 6.2.5 Screenshot Protection:

Incorporated screenshot protection by use of FLAG\_SECURE on Android devices with fallback alerts for other platforms, thus mitigating the risks of visual information leakage.

#### 6.2.6 Cross-Platform Functionality:

Built the application in React Native with Expo so that the experience in Android and iOS is seamless without the need to duplicate development work.

#### 6.2.7 Public/Private Key Handling Optimization:

Optimized the procedures for generating and maintaining cryptographic key pairs, reducing the latency and increasing reliability during chat sessions.

#### 6.2.8 Enhanced User Feedback and Alerts:

Enhanced alert features for usernames, contacts, and encryption alerts to improve user interface for login to contact management enhancing overall system usability, and interface consistency.

#### 6.2.9 Efficient Presence Tracking:

Optimized Firebase event listeners and database write operations resulting in enhanced system responsiveness and accuracy for user presence detection.

### 6.3 Critical Review

Although the core functional goals of security were achieved with Self Destruct Messenger, a critical systems evaluation reveals numerous aspects pertaining to design issues, platform limitations, and development compromises which shaped the end result. This assessment articulates on the merits of the solution within the context of development and testing hurdles that emerged in the practical implementation of the system.

#### 6.3.1 Strong Emphasis on Privacy:

The system incorporated privacy-protecting features including real-time-only sessions, mutual contact verification, and automatic message deletion. Such features are rarely, if ever, found in combination within traditional messaging applications.

#### 6.3.2 End-to-End Encryption:

The use of ECDH-based dynamic key exchange for each session ensured cryptographic isolation. Any form of static key reuse or message interception was effectively eliminated with this strategy.

#### 6.3.3 Lightweight and Cross-Platform Design:

Leveraging React Native and Expo enabled rapid deployment across Android and iOS devices, saving development time while maintaining a consistent user experience.

#### 6.3.4 User Control and Transparency:

The user is made explicitly aware of session conditions, verification requirements, and privacy features—promoting responsible communication.

#### 6.3.5 Limited Screenshot Prevention Across Devices:

Although the FLAG\_SECURE approach works on most Android devices, iOS support is limited, and some Android ROMs allow screen recording despite the flag. Full protection from screen capture is not universally enforceable.

#### 6.3.6 No Offline Messaging or History:

The strict design intentionally avoids storing any message history. While this supports privacy, it also sacrifices usability features like message recall, forwarding, or even reviewing previous chats—making it less practical for casual use.

#### 6.3.7 Dependency on Firebase Infrastructure:

The system’s real-time sync, authentication, and storage rely entirely on Firebase, making it dependent on continuous internet connectivity and limiting backend flexibility for more advanced security measures or offline fallback features.

#### 6.3.8 Limited User Interface Customization:

Due to focus on functionality and security, UI/UX design remained minimal. More visual polish, animations, and personalization options could improve engagement for end users.

#### 6.3.9 Scalability Not Yet Evaluated:

The current implementation supports limited users and sessions. Load testing for concurrent sessions at scale (hundreds or thousands of users) has not been performed due to infrastructure and time constraints.

### 6.4 Future Recommendations

The Self-Destruct Messenger application lays the foundation for a secure and privacy-focused communication system. However, there are several areas where the system could be extended, scaled, or further improved to enhance usability, compatibility, and impact. Future enhancements should focus on broader integration, advanced security features, and more user-friendly design options.

One of the most promising future directions involves integrating the Self-Destruct Messenger system with major Meta platforms such as Facebook Messenger, Instagram Direct, and WhatsApp. By establishing compatibility or creating secure extensions for these widely used services, the reach and utility of the system could be significantly expanded. This would facilitate users implementing escape route features of encryption and message deletion while conversing on widely-used social networking platforms.

#### 6.4.1 Possible approaches include:

1. Contact synchronization via Meta accounts in OAuth-based linking.
2. Session-based control overlays for Facebook and Instagram chat via browser or extension.
3. Bot integrations that can enforce deletion logic within the automation boundaries set by Meta's policies through APIs.
4. Platforms that capture and encrypt messages from other platforms then strip identifying information before securely sending them to the other.
5. While full control may be limited by Meta’s closed architecture, even partial integration would serve as a privacy layer for users relying on those platforms daily.

#### 6.4.2 Enhance Encryption Techniques

While the session encryption feature with ECDH was sufficient, the overall security assurances could be enhanced in future iterations by integrating more stringent mechanisms such as PFS, post-quantum encryption, or blockchain-backed key verification.

#### 6.4.3 Offline Mode with Temporary Caching

Usability can be improved without compromising privacy by adding an optional offline buffer, where messages are stored on-device until both users come online. Messages will self-destruct if conditions for delivery are not met within a specified time.

#### 6.4.4 User Interface Refinement

The current interface prioritizes function over form. Future iterations could add:

1. Dark mode
2. Custom themes
3. Animated transitions
4. Voice message support
5. Multilingual accessibility

#### 6.4.5 Cloud Function Optimization and Backend Flexibility

A self-hosted or decentralized server could be added to Firebase in a hybrid backend model, boosting control of the system, reducing vendor lock-in, and enabling advanced privacy audits or analytic assessments.

### 6.5 Chapter Summary

The chapter systematically detailed the development work and the evaluation of the Self-Destruct Messenger system in the context of broader messaging technologies. It described the project goals which aimed at designing and implementing a secure, session-based messaging system incorporating features like end-to-end encryption, contact verification, real-time-only communication, message auto-deletion, and “hands-free” self-destruct capabilities.

A critical assessment discussed the major privacy and control-centric innovations of the system while also recognizing its scope-specific user interface challenges, system dependencies, and Firebase integration. These observations helped in understanding the compromises which were deemed necessary during the development phase.

Recommendations mark a direction toward exposure integration with Meta ecosystems like Facebook, Instagram, and WhatsApp, refining the encryption model and communication relays, the user experience uplifts, and assess offline communication buffer mechanisms. From these suggestions, the evolving trajectory of the application into a broadly interoperable secure communications layer compatible with contemporary digital ecosystems becomes visible.

### References

[1] Schnitzler, Theodor, et al. "Exploring user perceptions of deletion in mobile instant messaging applications." Journal of Cybersecurity 6.1 (2020): tyz016.

[2] Heath, Howard, Áine MacDermott, and Alex Akinbi. "Forensic analysis of ephemeral messaging applications: Disappearing messages or evidential data?" Forensic Science International: Digital Investigation 46 (2023): 301585.

[3] Qiwei, Li, et al. "Feminist Interaction Techniques: Deterring Non-Consensual Screenshots with Interaction Techniques." arXiv preprint arXiv:2404.18867 (2024).

[4] Warner, M., Lascau, L., Cox, A. L., Brumby, D. P., & Blandford, A. (2021). "Oops...”: Mobile Message Deletion in Conversation Error and Regret Remediation." Proceedings of the CHI Conference on Human Factors in Computing Systems (CHI ’21). https://doi.org/10.1145/3411764.3445675

[5] Wijnberg, D., & Le-Khac, N.-A. (2020). "Identifying interception possibilities for WhatsApp communication." arXiv preprint arXiv:2011.03732.

[6] Bahramali, A., et al. (2020). "Practical Traffic Analysis Attacks on Secure Messaging Applications." arXiv preprint arXiv:2005.00508.

[7] Azhar, M. A. H. B., Cox, R., & Chamberlain, A. (2020). "Forensic investigations of popular ephemeral messaging applications on Android and iOS platforms." International Journal on Advances in Security, 13(1 & 2), 41–53.

[8] Wang, Y., & Wang, X. (2022). "A Comparative Study on the Security of Ephemeral Messaging Applications." Journal of Information Security Research, 14(3), 112–125.

[9] Lee, J., & Kim, S. (2023). "User Perceptions of Message Deletion Features in Mobile Messaging Apps." International Journal of Human–Computer Interaction, 39(5), 456–470.

[10] Smith, A., & Johnson, L. (2024). "Advancements in End-to-End Encryption for Mobile Messaging Platforms." Cybersecurity Advances, 8(2), 89–102.

[11] Everlaw. (2021). "Ephemeral Messaging Data in Ediscovery." https://www.everlaw.com/guides/novel-data-types/ephemeral-messaging-data-in-ediscovery/

[12] SlashGear. (2022). "Snapchat Is Not Private—Nor Is It Safe." https://www.slashgear.com/769717/snapchat-is-not-private-nor-is-it-safe/

[13] WhatsApp. (2024). Privacy Policy (EEA).https://www.whatsapp.com/legal/privacy-policy

[14] Telegram. (2024). Privacy Policy. <https://telegram.org/privacy?setln>