

**LOCKME: SECURE FILE ENCRYPTION AND DECRYPTION DESKTOP
APPLICATION**

By

MUHAMAD AZRI MUHAMAD AZMIR

012024021197

**Project Submitted in Partial Fulfilment of the Requirements for the Degree of
Bachelor in Computer Forensic (Hons.) in the Faculty of Information Sciences
and Engineering**

June 2025

PENGISYTIHARAN

(Declaration)

Saya,
I,

MUHAMAD AZRI MUHAMAD AZMIR

calon bagi ijazah
candidate for the degree of

BACHELOR IN COMPUTER FORENSIC (HONS.),

Management & Science University mengakui bahawa :
Management & Science University certifies that:

- i) Tesis saya telah dijalankan, digubal, dan ditulis sendiri di bawah penyeliaan:
My thesis was personally developed, conducted and written by us under the supervision of:

DR. ASMA MAHFOUDH HEZAM AL-HAKIMI
- ii) Data saya adalah data asal dan saya/kami sendiri mengumpul dan menganalisisnya; dan
My data are original and personally collected and analysed; and
- iii) Saya akan sentiasa mematuhi syarat, polisi, dan peraturan MSU mengenai penulisan tesis, termasuk undang-undang hak cipta dan paten Malaysia.
I shall at all times be governed by the conditions, policies, and regulations of the MSU on thesis writing, including the copyright and patent laws of Malaysia.

Jika saya didapati melanggar perkara-perkara di atas, saya dengan relanya menepikan hak penganugerahan diploma saya dan tertakluk kepada syarat dan peraturan disiplin Management & Science University.

In the event that my thesis is found to violate the conditions mentioned above, I voluntarily waive the right of conferment of my diploma and will be subjected to the disciplinary rules and regulations of Management & Science University.

Nama Calon

Candidate's Name

Tandatangan Calon

Candidate's Signature

Tarikh

Date

**Faculty of Information Sciences and Engineering
Management & Science University**

**PERAKUAN KERJA KERTAS PROJEK
(Certification of Project Paper)**

Saya, yang bertandatangan, memperakukan bahawa
(I, the undersigned, certify that)

MUHAMAD AZRI MUHAMAD AZMIR

calon untuk ijazah
(candidate for the degree of)

BACHELOR IN COMPUTER FORENSIC (HONS.),

telah mengemukakan kertas projek yang bertajuk
(has presented his/her project paper of the following title)

LOCKME: SECURE FILE ENCRYPTION AND DECRYPTION DESKTOP
APPLICATION

seperti yang tercatat di muka surat tajuk dan kulit kertas projek
(as it appears on the title page and front cover of project paper)

bahawa kertas projek tersebut boleh diterima dari segi bentuk serta kandungan, dan meliputi bidang ilmu dengan memuaskan.
(that the project paper is acceptable in form and content and that a satisfactory knowledge of the field is covered by the project paper).

Nama Penyelia
(Name of Supervisor) : DR. ASMA MAHFOUDH HEZAM AL-HAKIMI

Tandatangan
(Candidate's Signature) : _____

Tarikh
(Date) : _____

ACKNOWLEDGMENTS

I would like to thank everyone who helped me along the way. First, my deepest gratitude goes to my supervisor, Dr. Asma Mahfoudh Hezam al-Hakimi. Their guidance and steady feedback shaped this project from the very beginning.

I am also thankful to my family and friends, for their constant support and encouragement. Their belief in me gave me the strength to push through both the easy and difficult moments.

Finally, many others lent a hand, whether with advice, motivation, or a listening ear. Each contribution, big or small, played a part in bringing this study to completion. I appreciate every one of you.

Terima kasih!

ABSTRACT

Abstract of project presented to the Senate of Management & Science University in partial fulfilment of the requirements for the degree of Bachelor in Computer Forensic (Hons.).

LOCKME: SECURE FILE ENCRYPTION AND DECRYPTION DESKTOP APPLICATION

By

MUHAMAD AZRI MUHAMAD AZMIR

June 2025

Faculty: Information Sciences and Engineering

LockMe is a browser-based application that works on both Windows and Linux, giving users a straightforward way to encrypt and decrypt files on their own machines. Built with Next.js, React, and TypeScript, it relies on the Web Crypto API to run AES-256-GCM encryption entirely in the browser, so neither the files nor the passphrases ever leave the device. Firebase provides secure account management and an encrypted code-snippet repository, while Genkit and Google Gemini supply AI-driven tools for passphrase generation, strength analysis, and recovery-prompt enhancement. Developed under the ADDIE model, LockMe was evaluated through functional, performance, usability, and security tests across major browsers and devices; the System Usability Scale returned a “Good” rating, and security checks confirmed effective protection against unauthorised access and tampering. The findings show that current web frameworks can match the security level of native desktop software while keeping the interface smooth and easy to use, letting users protect their data without technical expertise. Future work will explore extra ciphers, a built-in secure-sharing feature, two-factor authentication, and expanded AI assistance to further boost the system’s capability and resilience.

ABSTRAK

Abstrak tesis yang dikemukakan kepada Senat Management & Science University sebagai memenuhi sebahagian keperluan untuk ijazah Sarjana Muda Forensik Komputer (Kepujian).

LOCKME: APLIKASI DESKTOP UNTUK PENYULITAN DAN PENYAHSULITAN FAIL YANG SELAMAT

Oleh

MUHAMAD AZRI MUHAMAD AZMIR

Jun 2025

Fakulti: Sains Maklumat dan Kejuruteraan

LockMe ialah aplikasi berasaskan pelayar yang berfungsi pada Windows dan Linux, membolehkan pengguna menyulit dan menyahsulit fail secara terus pada komputer mereka. Dibangunkan dengan Next.js, React dan TypeScript, ia menggunakan Web Crypto API untuk melaksanakan penyulitan AES-256-GCM sepenuhnya di dalam pelayar, memastikan fail dan frasa laluan tidak pernah tinggalkan dalam peranti. Firebase menyediakan pengurusan akaun yang selamat serta repositori coretan kod yang disulitkan, manakala Genkit dan Google Gemini membekalkan alat AI bagi penjanaan frasa laluan, analisis kekuatan dan penambahbaikan arahan pemulihan. Mengikut model ADDIE, LockMe dinilai melalui ujian kefungsian, prestasi, kebolegunaan dan keselamatan merentas pelayar serta peranti utama; Skala Kebolegunaan Sistem (SUS) memberi penarafan “Baik”, dan semakan keselamatan mengesahkan perlindungan berkesan terhadap capaian tanpa kebenaran dan pengubahsuaian. Dapatan kajian menunjukkan rangka kerja web semasa mampu menyamai tahap keselamatan perisian desktop asli sambil mengekalkan antara muka yang lancar dan mudah digunakan, membolehkan pengguna melindungi data tanpa kepakaran teknikal. Kajian masa depan akan meneroka sifir tambahan, ciri perkongsian selamat terbina dalam, pengesahan dua faktor dan bantuan AI yang diperluas untuk meningkatkan lagi keupayaan serta ketahanan sistem.

TABLE OF CONTENTS

ACKNOWLEDGMENTS	iv
ABSTRACT.....	v
ABSTRAK.....	vi
LIST OF FIGURES.....	x
LIST OF TABLES.....	xii
LIST OF ABBREVIATIONS	xiii
CHAPTER 1: INTRODUCTION.....	14
1.1 Background of Study	14
1.2 Problem Statement	15
1.3 Project Objectives	16
1.4 Project Scope.....	17
1.5 Project Significance	19
1.6 Project Limitations	20
1.7 Chapter Summary.....	21
CHAPTER 2: LITERATURE REVIEW	23
2.1 Introduction.....	23
2.2 Explanation of Key Terms, Terminologies, and Theories.....	25
2.3 Review of Existing Solutions and Technologies.....	30
2.3.1 Review of Current Systems.....	30
2.3.2 Comparison of Technologies.....	33
2.3.3 Gaps in Existing Solutions	35
2.4 Relevant Algorithms and Methodologies.....	39
2.4.1 Survey of Relevant Algorithms.....	39
2.4.2 Implementation Approaches.....	43
2.4.3 Evaluation Metrics	50
2.5 State of the Art in the Field	55
2.5.1 Recent Advances	55
2.5.2 Trends and Future Directions.....	60
2.6 Critical Review of Similar Products or Systems.....	63
2.6.1 Review of Similar Projects	63
2.6.2 How This Project Differs	68
2.6.3 Comparison Between Similar Systems and Proposed System.....	70
2.7 Summary of Findings.....	71
2.7.1 Synthesis of Literature	71
2.7.2 Research Gap	72
2.8 Chapter Summary.....	74

2.8.1	Justification for the Project	74
2.8.2	Connection to Project Goals.....	79
CHAPTER 3: RESEARCH METHODOLOGY		83
3.1	Introduction.....	83
3.2	Software Development Methodology	84
3.2.1	Chosen Methodology and Justification.....	84
3.2.2	Step-by-step Explanation of Activities Done in Each Phase of the Chosen Methodology	86
3.3	Research Methodology.....	89
3.3.1	Chosen Research Methodology and Justification	89
3.3.2	Questionnaire Design and Samples.....	90
3.3.3	Analysis of Questionnaire Data	95
3.4	Proposed System Design.....	106
3.4.1	UML Modelling of the Proposed System	107
3.4.2	Hardware Design and Block Diagram	114
3.5	Chapter Summary.....	116
CHAPTER 4: RESULTS AND DISCUSSION		119
4.1	Introduction.....	119
4.2	Implementation	119
4.2.1	Development Environment	119
4.2.2	System Modules and Implementation.....	120
4.2.3	Database Design and Implementation.....	125
4.2.4	Third-party APIs and Libraries	126
4.2.5	Testing During Implementation	128
4.2.6	Deployment Process.....	129
4.2.7	Security Measures	129
4.2.8	Screenshots and Sample Output.....	130
4.3	System Evaluation.....	133
4.3.1	Introduction.....	133
4.3.2	Evaluation Objectives	134
4.3.3	Evaluation Methods	134
4.3.4	Evaluation Results.....	135
CHAPTER 5: CONCLUSION AND RECOMMENDATIONS.....		139
5.1	Overview	139
5.2	How the Project Objectives Are Met	139
5.3	Significance.....	140
5.4	Future Enhancement/Recommendations.....	140

REFERENCES.....	143
APPENDICES	152

LIST OF FIGURES

Figure 2.2.1 Logo of Hat.sh	64
Figure 2.2.2 Interface of Hat.sh	65
Figure 2.2.3 Logo of Enc	66
Figure 2.2.4 Interface of Enc.....	67
Figure 3.3.1 Phases of the ADDIE model	86
Figure 3.3.2 Survey form display.....	90
Figure 3.3.3 Survey: Question 1	90
Figure 3.3.4 Survey: Question 2	91
Figure 3.3.5 Survey: Question 3	91
Figure 3.3.6 Survey: Question 4	91
Figure 3.3.7 Survey: Question 5	92
Figure 3.3.8 Survey: Question 6	92
Figure 3.3.9 Survey: Question 7	92
Figure 3.3.10 Survey: Question 8	93
Figure 3.3.11 Survey: Question 9.....	93
Figure 3.3.12 Survey: Question 10	93
Figure 3.3.13 Survey: Question 11.....	94
Figure 3.3.14 Survey: Question 12	94
Figure 3.3.15 Survey: Question 13	94
Figure 3.3.16 Survey: Question 14	95
Figure 3.3.17 Survey: Question 15	95
Figure 3.3.18 Analysis of respondent's age distribution	95
Figure 3.3.19 Analysis of respondent's gender distribution.....	96
Figure 3.3.20 Analysis of respondent's occupation	96
Figure 3.3.21 Analysis of respondents' technical expertise	97
Figure 3.3.22 Analysis of respondents' usage of operating systems (OS)	98
Figure 3.3.23 Analysis of respondents' usage of any file encryption tools or methods	98
Figure 3.3.24 Analysis of respondents' encryption tool preferences.....	99
Figure 3.3.25 Analysis of respondents' reasons for using or not using encryption tools	100
Figure 3.3.26 Analysis of respondents' challenges when using encryption tools....	100
Figure 3.3.27 Analysis of respondents perceived need for a user-friendly tool.....	101

Figure 3.3.28 Analysis of respondents desired features in an encryption tool.....	102
Figure 3.3.29 Analysis of respondents' importance of security, ease of use, and cross-platform compatibility.....	103
Figure 3.3.30 Analysis of respondents' willingness to try a new encryption tool....	103
Figure 3.3.31 Analysis of respondents' concerns and requirements regarding file encryption.....	104
Figure 3.3.32 Analysis of respondents' suggested features and improvements for LockMe	105
Figure 3.3.33 Use Case Diagram	107
Figure 3.3.34 Package Diagram	108
Figure 3.3.35 Class Diagram.....	109
Figure 3.3.36 Sequence Diagram	110
Figure 3.3.37 State Machine Diagram	111
Figure 3.3.38 Activity Diagram	113
Figure 3.3.39 Block Diagram.....	116
Figure 4.1 Next.js Logo	120
Figure 4.2 Firebase Logo	122
Figure 4.3 React Logo.....	126
Figure 4.4 TailwindCSS Logo.....	126
Figure 4.5 Gemini Logo.....	127
Figure 4.6 Vercel Logo.....	129
Figure 4.7 LockMe Interface After Successful Encryption	130
Figure 4.8 LockMe Interface During Decryption Process	131
Figure 4.9 LockMe Interface After Successful Decryption	131
Figure 4.10 LockMe Interface Displaying Error Handling (e.g., Incorrect Passphrase)	132
Figure 4.11 AI Security Toolkit - Passphrase Generator in Action	132
Figure 4.12 Code Snippet Manager Interface	133

LIST OF TABLES

Table 2.1 Comparison table between systems.....	70
Table 3.1 Requirements of the proposed system.....	106
Table 3.2 Hardware requirements for the proposed system	115
Table 4.1 Client-Side Encryption and Decryption Performance for Varying File Sizes	136

LIST OF ABBREVIATIONS

Abbreviation	Definition
AES	Advanced Encryption Standard
AI	Artificial Intelligence
API	Application Programming Interface
APT	Advanced Persistent Threat
CBC	Cipher Block Chaining
CSPRNG	Cryptographically Secure Pseudo-Random Number Generator
ECB	Electronic Codebook
GCM	Galois/Counter Mode
GUI	Graphical User Interface
IT	Information Technology
IV	Initialisation Vector
OTP / TOTP	One-Time Password / Time-based One-Time Password
PC	Personal Computer
PKI	Public Key Infrastructure
PWA	Progressive Web App
RSA	Rivest–Shamir–Adleman (public-key algorithm)
SUS	System Usability Scale
TLS	Transport Layer Security
USB	Universal Serial Bus

CHAPTER 1

INTRODUCTION

1.1 Background of Study

As daily life moves further online, the amount of personal and business data we create, and store has grown sharply. Cyber-attacks have followed the same upward trend, moving beyond large corporations to target ordinary users, small firms, and public bodies alike (Verizon, 2025). The consequences can be severe: financial losses, damaged reputations, lost customer confidence, and potential legal troubles (Morgan, 2020). Recent studies show that data breach costs continue climbing, affecting millions of people each year (IBM, 2024).

Encryption offers one of the best defences against these threats. At its core, encryption converts readable information into scrambled code that only authorized users can decode with the right key. This means that even if hackers steal encrypted files, they can't actually use the information without breaking the encryption, which is extremely difficult with proper implementation (Schneier & Diffie, 2015). However, most encryption tools present a significant problem: they're built for tech experts. These programs often feature complicated interfaces and assume users understand complex cryptographic concepts. This creates a real barrier for everyday users who need protection but lack technical expertise (Kirlappos & Sasse, 2014). Additionally, many encryption solutions only work on specific operating systems, limiting their usefulness for people who use different devices.

Recognizing this gap, LockMe, which is a desktop application specifically designed for non-technical users who need reliable file protection, is developed. LockMe works on both Windows and Linux systems, which cover the majority of personal and business computers. The application uses Advanced Encryption

Standard (AES), a proven encryption method, but presents it through an intuitive interface that doesn't require cryptographic knowledge. The goal is to make strong data security accessible to everyone.

1.2 Problem Statement

i. Lack of Cross-Platform Compatibility in Encryption Tools

Nowadays, the majority of people use a variety of gadgets and operating systems. Depending on their needs, a person may alternate between Linux and Windows for work or development projects. Unfortunately, a lot of encryption tools are only compatible with a single operating system, which forces users to either learn new tools for each platform or stick with less secure options. For individuals, small businesses, and IT professionals who require uniform security across their various systems, this is a huge pain in the neck.

ii. Insufficient Accessibility to Robust Encryption Techniques

Strong security is provided by advanced encryption techniques like AES (Advanced Encryption Standard), which are typically found in software intended for highly technical users. Because of this, it is very difficult for non-technical users to access and use these strong encryption methods. This vulnerability exposes private information to unauthorized access, which is a major security risk for both people and businesses.

iii. Usability Challenges in Existing Encryption Tools

Most encryption programs feel old. People find it hard to use the confusing menus. The instructions are not clear. A user must complete many steps to encrypt one file. Common features that people expect in modern software are often gone;

they do not see drag-and-drop file support. A program does not show progress bars; it also lacks clear confirmations when tasks finish. These usability problems stop people from using better security practices. So, their private information stays exposed.

1.3 Project Objectives

i. To develop a cross-platform desktop application supporting both Windows and Linux operating systems.

People now move between various computers and operating systems frequently. A person could own a Windows laptop for work. A Linux server may run development tasks. Some individuals help family members whose computers use different systems. When security tools function on only one platform, problems develop in protection. This project handles Windows in addition to Linux because both platforms include most personal and work computing settings - this applies whether someone protects personal files, operates a small business, or oversees computer systems.

ii. To implement advanced encryption techniques, specifically AES encryption, ensuring robust file security.

The application uses AES encryption. This is the industry standard for file security. Governments along with security professionals use AES. It provides good protection, and it does not make the computer lag. The application includes this encryption. Users get high security; and they do not need to understand technical details or to navigate complex settings.

iii. To design an intuitive and user-friendly interface with features such as drag-and-drop file selection and clear status messages.

Making technology accessible is more of a barrier to improved security than the technology itself. Anyone familiar with modern software will recognise this app. You will be able to drag files straight into the window, see transparent progress bars, and get clear notifications while the encryption process is underway. No hidden menus, no perplexing technical jargon, and no uncertainty about whether something worked or not. Strong encryption should be as simple as copying a file.

1.4 Project Scope

User Scope

- a) Users can encrypt and decrypt files securely using a straightforward graphical interface.
- b) Users can perform encryption and decryption tasks on both Windows and Linux systems.
- c) Users can select files of various formats, including documents, images, and compressed archives, for encryption or decryption.
- d) Users can access all features directly from the application's main screen after startup.
- e) The application provides clear navigation between encryption, decryption, and key management functions.
- f) Users can generate, save, and retrieve encryption keys through the application.
- g) Users with different levels of technical knowledge, from beginners to experienced users, can operate the application easily.

- h) Users can encrypt files in real-time and verify file integrity using the built-in integrity check feature.
- i) Users don't need to rely on cloud services for encryption, as all processes happen locally on their devices.

System Scope

- a) The system uses AES in addition to RSA encryption - it secures files.
- b) The system makes, keeps along with finds encryption keys for users.
- c) The application encrypts and decrypts many file types. It works with text, pictures as well as compressed files.
- d) The system lets users drag and drop files. This makes it easier to pick files for encryption or decryption.
- e) The system runs on a desktop computer - it does not need cloud services or company networks.
- f) The application works on different computers. It runs well on Windows besides Linux.
- g) The system encrypts files as they are used - it also checks file integrity to keep operations safe.
- h) The system has a simple user interface. Both technical plus non-technical people can use it.
- i) The application encrypts and decrypts local files. It does not have complex company encryption systems or batch processing.
- j) The system completes all operations, like key management and encryption, according to common security rules.

1.5 Project Significance

i. Enhancing Data Security

With cyber-attacks becoming more frequent and sophisticated, people need practical ways to protect their personal and professional files. This project gives users a straightforward encryption tool that works, helping them secure sensitive information without requiring advanced technical knowledge or expensive software.

ii. User Accessibility

Most encryption programs appear as if made for security specialists - this application alters that. It makes file protection easy, like moving and placing a file. When security tools are easy to use, more people use them, which means more protection for everyone.

iii. Cross-Platform Compatibility

Many people work across different operating systems. Windows at the office, Linux for development, or helping family members with various setups. Having one encryption tool that works consistently across platforms eliminates the hassle of learning different programs or dealing with compatibility issues.

iv. Promoting Cybersecurity Awareness

By making encryption accessible, this project encourages users to adopt secure file management practices, contributing to a more secure digital ecosystem.

v. Cost-Effective Solution

Good encryption should not cost much or demand regular payments; this project provides strong security without the money problem that often stops individuals and small businesses from putting in place proper data protection methods.

1.6 Project Limitations

i. Lack of Cloud Integration

LockMe operates entirely as a standalone desktop application, which means it doesn't connect to popular cloud storage services like Google Drive, Dropbox, or OneDrive. While this design keeps files completely under user control and prevents any data from leaving the local computer, it also means users are responsible for managing their encrypted files manually. Users working across multiple devices or needing to share files with colleagues must handle the transfer and synchronization through external methods. This can create extra steps in workflows that rely heavily on cloud-based collaboration.

ii. No Passphrase Recovery for Encrypted Files

LockMe prioritizes user privacy by never storing or transmitting passphrases anywhere, not on servers, not in the application, nowhere. This ensures complete security for encrypted content, but it comes with an important trade-off: if users forget their passphrase, there's no way to recover it. The encrypted file will be permanently inaccessible. The application cannot and will not help retrieve lost passphrases because it never had access to them in the first place. Users need to keep track of their passphrases using secure methods, whether that's a password manager, written notes, or whatever system

works best for their situation.

iii. Manual Sharing of Encrypted Files and Passphrases

While LockMe excels at encrypting and decrypting files, it doesn't include any built-in sharing features. When users need to send an encrypted file to someone else, they must handle both the file transfer and passphrase communication separately. Users might email the .lockme file and then text the passphrase, or use a cloud service for the file and a secure messaging app for the key. LockMe doesn't provide integrated secure messaging or managed sharing portals like some enterprise solutions do. This manual approach gives users flexibility in choosing their preferred communication methods, but it also means coordinating secure file sharing requires more steps and careful attention to avoid sending passphrases through insecure channels.

1.7 Chapter Summary

Today, protecting digital information is very important due to the constant threat of cybercrime. Encryption is a key defence, as it transforms readable information into a secure, unreadable format. But many of today's encryption tools have significant problems. For example, they often work only on certain operating systems, making consistent security hard for users who switch between Windows and Linux. Additionally, strong encryption methods like AES can be difficult for non-technical users to access because of complicated interfaces and poor usability.

In order to solve these problems, this project introduces LockMe, a desktop encryption application created specifically with everyday users in mind. LockMe aims to achieve three main objectives: first, it should work flawlessly on both Windows and Linux platforms; second, it should provide strong security through

AES encryption; and third, it should have an easy-to-use interface with features like transparent status updates and drag-and-drop file handling. With this program, users can quickly check the integrity of files, manage encryption keys, and encrypt and decrypt a variety of file types. LockMe runs locally on the user's PC without relying on cloud services. Additionally, LockMe will use the AES and RSA algorithms, handle encryption keys securely, and ensure consistent user experience across different operating systems.

LockMe sets out to fill these gaps with an encryption tool that pairs strong security with everyday usability. Designed for both technical and non-technical users, it allows individuals and small organisations to safeguard sensitive files through well-established cryptographic methods presented in a clear, practical interface. The next chapter reviews the foundations of encryption, surveys existing tools, and pinpoints the shortcomings that LockMe is built to solve. It covers core cryptographic concepts, weighs the advantages and drawbacks of current solutions, and shows why achieving the right balance of security, ease of use, and accessibility is vital in today's threat landscape. This context clarifies how LockMe contributes to better data-protection practices.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

The "LockMe: Secure File Encryption and Decryption Desktop Application" project makes a significant contribution to the ever-evolving fields of cybersecurity, with a pronounced emphasis on data security and practical cryptography. These domains are of increasing critical importance as modern society becomes more reliant on digital platforms for nearly every facet of life, including communication, data storage, financial transactions, and critical infrastructure management (Kshetri, 2013). The sheer volume of sensitive information, be it personal, financial, corporate, or governmental, necessitates the implementation of robust and accessible data protection measures (Awad Al-Hazaimeh, 2013). The global creation of data is projected to grow exponentially in the coming years, further amplifying the attack surface for malicious actors (Reinsel et al., 2018).

The rapid digitisation of almost every industry has led to a dramatic increase in the amount of sensitive data created, transmitted, and stored daily. While this digital transformation offers numerous advantages, such as enhanced accessibility, efficiency, and global reach, it concurrently introduces a multitude of risks (IBM, 2024). The number, variety, and sophistication of cybersecurity threats including but not limited to ransomware, sophisticated phishing campaigns, advanced persistent threats (APTs), data breaches, and unauthorized access are constantly on the rise (Lella et al., 2024). These dangers can have catastrophic repercussions, ranging from substantial monetary losses and reputational damage to severe legal ramifications and even threats to national security, affecting

individuals, businesses, and governments alike (Ahamad & Abdullah, 2016). The economic impact of cybercrime is now measured in trillions of dollars annually, underscoring the urgency of effective security measures (Morgan, 2020).

Cryptography, the science and art of protecting information and communication through the use of codes, is essential in addressing these multifaceted issues. Cryptographic techniques, primarily encryption, ensure the confidentiality, integrity, and authenticity of sensitive data by transforming it into a format that is unreadable and unusable by unauthorised individuals (Katz & Lindell, 2021). Encryption serves as an indispensable technique for preventing data theft, misuse, and interception, whether data is in transit over networks or at rest in storage systems.

Despite the abundance of encryption tools and systems available, numerous obstacles must still be overcome to make encryption truly usable, efficient, and universally accessible to a diverse range of users (Das et al., 2020). Many existing solutions feature intricate procedures, command-line interfaces, or complex configuration options that can deter non-technical individuals from utilising them effectively, as these tools were often designed with experienced IT professionals or cryptographers in mind (Nielsen, 1999). Moreover, the lack of seamless cross-platform compatibility in many encryption tools significantly limits their usefulness for users operating in heterogeneous computing environments, such as businesses or individuals who regularly use both Linux and Windows systems (Cranor & Garfinkel, 2005).

LockMe addresses these barriers by coupling AES encryption with a straightforward, drag-and-drop interface that operates identically on both operating systems. By simplifying workflows and eliminating technical hurdles, the project

aims to make strong file protection attainable for individuals and small organisations alike. The following literature review therefore revisits core cryptographic principles, surveys current tools and their limitations, and highlights why an approach that balances security, usability, and accessibility has become essential amidst today's escalating cyber-risk landscape.

2.2 Explanation of Key Terms, Terminologies, and Theories

Several foundational theories, concepts, and terminologies are central to the development and understanding of the project. Comprehending these terms is crucial for grasping the underlying mechanisms, design choices, and methodologies employed in the project:

i. Cryptography:

The study and application of techniques that permit secure communication and data protection in the presence of adversaries (third parties) is known as cryptography (Menezes et al., 1996). It is a fundamental and interdisciplinary field of study, drawing from mathematics, computer science, and electrical engineering, that uses mathematical algorithms and keys to convert readable data (plaintext) into an unintelligible format (ciphertext) and vice-versa. The primary objectives of cryptography include ensuring data confidentiality (preventing unauthorized disclosure), integrity (ensuring data has not been altered), authentication (verifying the identity of users or systems), and non-repudiation (preventing denial of an action). Cryptography is a cornerstone of modern data security systems, essential for applications such as digital signatures, encrypted communications, secure financial transactions, and safe file storage (Awad Al-Hazaimah, 2013).

ii. Encryption:

Encryption is the specific process of transforming plaintext into ciphertext using an encryption algorithm (also known as a cipher) and an encryption key (Paar & Pelzl, 2010). This procedure ensures that data becomes unreadable and meaningless to unauthorized individuals who do not possess the corresponding decryption key. Encryption is widely used to secure communications (e.g., email, messaging), protect sensitive information stored on devices or in databases, and comply with data privacy regulations (Stallings & Brown, 2012). It can be applied to various data types, including text, images, audio, video, and entire files. Even if files are intercepted, stolen, or compromised, encryption shields user data from unwanted access when implemented correctly with applications like LockMe (Ahamad & Abdullah, 2016).

iii. Decryption:

The converse of encryption, decryption is the process of transforming ciphertext back into its original, readable plaintext state. This procedure requires the correct decryption algorithm and the corresponding decryption key, ensuring that only authorized users or systems can access the protected data (Bishop, 2018). Decryption is essential for retaining the usability of encrypted data, as users must be able to recover and manipulate the original information while maintaining its security during storage and transmission (Awad Al-Hazaimah, 2013).

iv. Symmetric-Key Algorithms:

Also known as secret-key cryptography, symmetric-key algorithms employ the same single key for both encryption and decryption processes (Schneier & Diffie, 2015). These algorithms are generally characterized by their high speed and

computational efficiency, making them ideal for encrypting large volumes of data, such as entire files or streaming data. Prominent examples include the Advanced Encryption Standard (AES), Blowfish, and the formerly widespread Data Encryption Standard (DES) (Mushtaq et al., 2017). While symmetric-key algorithms offer strong security, their primary challenge lies in secure key distribution: both communicating parties must possess the same secret key, and exchanging this key securely over an insecure channel can be problematic. Because of its proven security and efficiency, AES has become the industry standard for bulk data encryption and is the main cryptographic method employed in the LockMe project.

v. Asymmetric-Key Algorithms:

Also referred to as public-key cryptography, asymmetric-key algorithms utilize a pair of mathematically related keys: a public key and a private key (Rivest et al., 1978). The public key is freely disseminated and used for encryption, while the private key is kept secret by the owner and used for decryption. This dual-key approach obviates the need for a secure channel to exchange keys, as the public key can be shared openly without compromising the private key's security. Diffie-Hellman (for key exchange) and Rivest-Shamir-Adleman (RSA) (for encryption and digital signatures) are well-known examples of asymmetric algorithms (Shantanu Joshi, 2013). These algorithms are frequently employed for tasks like secure web communications (HTTPS/TLS), digital signatures (to verify data integrity and authenticity), and secure key exchange to establish a shared secret for symmetric encryption. While asymmetric algorithms offer enhanced security for key management and data authentication, they are generally more computationally intensive and slower than symmetric algorithms, making them less suitable for

encrypting large data volumes directly.

vi. Feistel Network (or Feistel Cipher):

A Feistel network is a specific cryptographic structure used in the design of many block ciphers. It was named after Horst Feistel, who co-developed the Lucifer cipher at IBM, a precursor to DES (Technology, 1977). This design divides the data block into two (usually equal) halves. The encryption process involves multiple rounds, where in each round, one half of the data is modified by a "round function" (which takes the other half and a subkey as input), and then the halves are swapped (usually). A key advantage of the Feistel structure is that the encryption and decryption processes are very similar, often identical, requiring only the reversal of the subkey schedule for decryption. This simplifies implementation in both hardware and software and reduces the chance of design errors. DES and Blowfish are notable examples of algorithms based on the Feistel network (Singh Karamjeet Singh, 2013)

vii. Block Cipher:

This is a fundamental type of symmetric encryption algorithm that operate on fixed-size blocks of plaintext data, transforming them into blocks of ciphertext of the same size, under the control of a secret key (Daemen & Rijmen, 2002). Common block sizes are 64 bits (e.g., DES, Blowfish) and 128 bits (e.g., AES). If the plaintext is larger than the block size, it is divided into multiple blocks, and if the last block is smaller than the block size, padding is typically applied. Block ciphers can be used in various "modes of operation" (e.g., Electronic Codebook - ECB, Cipher Block Chaining - CBC, Counter - CTR, Galois/Counter Mode - GCM) to securely handle sequences of blocks (Dworkin, 2001). These modes

define how the repeated application of the cipher to individual blocks results in the encryption of the entire message, often incorporating an Initialization Vector (IV) to ensure that identical plaintext blocks encrypt to different ciphertext blocks. AES, DES, and Blowfish are all examples of block ciphers (Mushtaq et al., 2017). Their fixed-size design allows for efficient management of large datasets while maintaining robust cryptographic properties

viii. Hashing:

A cryptographic hash function is an algorithm that takes an arbitrary amount of data input a "message" and returns a fixed-size string of characters, which is called the hash value, message digest, or simply hash (Preneel, 2005). Good cryptographic hash functions have several important properties: they are deterministic (the same message always results in the same hash), quick to compute the hash value for any given message, infeasible to generate a message from its hash value except by trying all possible messages (preimage resistance), infeasible to find two different messages with the same hash (collision resistance), and a small change to a message should change the hash value so extensively that the new hash value appears uncorrelated with the old hash value (avalanche effect). Hashing is widely used for data integrity verification (e.g., checksums), password storage, and in digital signatures. Examples include SHA-256 and SHA-3

ix. Digital Signatures:

A digital signature is a mathematical scheme for verifying the authenticity of digital messages or documents (Goldwasser et al., 1988). A valid digital signature, where the prerequisites are satisfied, gives a recipient very high confidence that the message was created by a known sender (authentication), that

the sender cannot deny having sent the message (non-repudiation), and that the message was not altered in transit (integrity). Digital signatures typically use asymmetric cryptography. The sender uses their private key to create the signature, and the recipient uses the sender's public key to verify it

x. **Public Key Infrastructure (PKI):**

PKI is a set of roles, policies, hardware, software, and procedures needed to create, manage, distribute, use, store, and revoke digital certificates and manage public-key encryption (Adams & Lloyd, 2003). The purpose of a PKI is to facilitate the secure electronic transfer of information for a range of network activities such as e-commerce, internet banking, and confidential email. It relies on Certificate Authorities (CAs) to issue digital certificates that bind public keys with respective user identities.

2.3 Review of Existing Solutions and Technologies

2.3.1 Review of Current Systems

There is a multitude of applications available for encrypting and decrypting files, employing a diverse array of algorithms such as RSA, DES, 3DES, AES, Blowfish, and others. Each tool often comes with its own set of features, target audience, and usability considerations.

i. **VeraCrypt:**

This is a well-regarded, free, open-source disk encryption software for on-the-fly encryption (OTFE). It is a fork of the discontinued TrueCrypt project and has addressed many of the security concerns raised about its predecessor (*VeraCrypt - Free Open Source Disk Encryption with Strong*

Security for the Paranoid, 2019). Users can encrypt entire partitions, storage devices (like USB drives), or create encrypted virtual disk "containers" that behave like regular disks. VeraCrypt supports strong encryption algorithms like AES, Serpent, and Twofish, and allows for cascading these ciphers for enhanced security (Sutherland, 2021). A distinctive feature is its support for plausible deniability through hidden volumes and hidden operating systems. While highly flexible and secure, VeraCrypt is primarily geared towards users with a good degree of technical knowledge due to its relatively complex setup and the conceptual understanding required for volume encryption versus individual file encryption. Its strength lies in full-disk and partition encryption rather than quick, ad-hoc file encryption for the average user.

ii. AxCrypt:

This is a lightweight file encryption software designed primarily for individual users on Windows, emphasizing simplicity and ease of use (*AxCrypt - File Security Made Easy*, n.d.). It offers AES-128 encryption in its free version and AES-256 in its premium versions. Its seamless integration with the Windows Explorer context menu allows users to encrypt and decrypt files directly, enhancing usability. Features include password-based encryption, secure file sharing, and a simple drag-and-drop interface, making it appealing to non-technical users. However, its functionality is mainly file-level encryption and does not extend to full-disk or partition encryption. Historically, its lack of robust Linux compatibility and the reservation of more advanced features for premium editions have limited its accessibility to a broader audience, although recent versions have

expanded platform support.

iii. GnuPG (GNU Privacy Guard):

This application is a free, open-source implementation of the OpenPGP standard, providing robust encryption and signing capabilities (GnuPG, 2019). It is highly versatile, supporting various symmetric and asymmetric algorithms, and is widely used for encrypting emails, files, and disk partitions. GnuPG is a command-line tool by nature, which makes it extremely powerful for scripting and integration into other applications but presents a steep learning curve for less technical users (*The GNU Privacy Handbook*, n.d.). Several graphical front-ends exist (e.g., Kleopatra, GPG Suite) to improve usability, but the underlying concepts of key management (public/private keys, web of trust) can still be challenging. Its strength is in its adherence to open standards and its strong community support.

iv. BitLocker:

BitLocker Drive Encryption is a full-volume encryption feature included with select editions of Microsoft Windows (Microsoft, 2024). It is designed to protect data by providing encryption for entire volumes. BitLocker uses AES in CBC or XTS mode with a 128-bit or 256-bit key. It can use a Trusted Platform Module (TPM) to protect the integrity of the startup process and the encryption keys. While very effective for protecting data at rest on Windows systems, especially against offline attacks if a device is lost or stolen, its primary focus is full-disk encryption, and it is not designed for encrypting individual files for sharing across different platforms (particularly non-Windows systems). Its ease of use for basic full-disk encryption on supported

Windows versions is a significant advantage for Windows users.

2.3.2 Comparison of Technologies

Secure data protection solutions are fundamentally based on encryption algorithms, each possessing distinct advantages, disadvantages, performance characteristics, and ideal use cases. These methods are broadly categorized into symmetric and asymmetric encryption, with hybrid approaches combining elements of both. The selection of an appropriate algorithm and approach is frequently dictated by the specific requirements, constraints, and threat model of the application (Mushtaq et al., 2017).

i. Symmetric Encryption Algorithms:

Techniques such as AES, DES, and Blowfish use a single, shared key for both encrypting plaintext and decrypting ciphertext. This shared-key paradigm makes these algorithms computationally efficient, particularly for processing large datasets quickly (Stallings & Brown, 2012). Symmetric encryption excels in scenarios like encrypting large files, databases, or securing high-speed network connections due to its computational speed and efficiency. However, the cornerstone of symmetric encryption's security is the secrecy of the shared key, and its secure exchange between parties is a critical challenge. Distributing this key safely, especially over untrusted networks, is difficult, as interception by unauthorized individuals compromises all data encrypted with that key (Kaufman et al., 2020). Furthermore, for large systems or multi-user environments, managing unique keys for every pair of users becomes

impractical (N^2 key problem), leading to potential inefficiencies and complexities in key management.

ii. Asymmetric Encryption Algorithms:

Algorithms like RSA, ECC (Elliptic Curve Cryptography), and Diffie-Hellman employ a pair of keys: a public key for encryption (or signature verification) and a private key for decryption (or signature generation). This separation simplifies key distribution as the public key can be shared openly without compromising the private key, thus enhancing security in multi-user settings (Boneh & Shoup, 2017). Asymmetric encryption is particularly effective where multiple users need to communicate securely or when digital identities need to be verified. It is extensively used for digital signatures to ensure message authenticity and integrity, and for secure key exchange protocols like those in HTTPS/TLS. The primary drawback of asymmetric algorithms is their computational intensity; they are significantly slower than symmetric algorithms and are not suitable for encrypting large volumes of data directly. This often leads to their use in hybrid systems.

iii. Hybrid Encryption Systems:

To leverage the strengths of both symmetric and asymmetric cryptography, many contemporary encryption systems employ a hybrid approach (Bishop, 2018). In a typical hybrid system, an asymmetric algorithm is used to securely exchange or encrypt a randomly generated symmetric key. This symmetric key is then used with a faster symmetric algorithm to encrypt the actual bulk data. This combination offers a practical balance: the efficiency of symmetric encryption for the data itself,

and the secure key distribution advantages of asymmetric encryption for the session key. This is a common model for applications like encrypted email (e.g., PGP/GnuPG), secure file transfer (e.g., SFTP), and secure web communication (TLS/SSL).

iv. Performance Considerations:

When selecting an encryption method, performance is a crucial factor, particularly for applications like "LockMe" that aim to provide real-time or near real-time encryption and decryption capabilities for a user-friendly experience. While asymmetric algorithms are vital for key management and digital signatures, symmetric algorithms are generally favoured for their speed in processing large files and data streams. AES, due to its strong security profile, widespread industry adoption, hardware acceleration support in modern processors, and efficient performance across various platforms, was chosen as the main algorithm for LockMe. This choice ensures a balance between robust security and acceptable performance for the target users (Bernstein, 2008).

2.3.3 Gaps in Existing Solutions

Despite the wide availability of encryption tools and software, several significant limitations and challenges persist, presenting opportunities to enhance security, performance, usability, and key management. These shortcomings underscore the need for more comprehensive, accessible, and user-centric encryption solutions like the proposed LockMe system.

i. Usability:

Many currently available encryption solutions were primarily designed with technically sophisticated users in mind, resulting in interfaces and operational workflows that are overly complicated for non-technical individuals.

These technologies often demand a thorough understanding of multi-step configuration procedures, command-line interactions, or abstract cryptographic concepts (e.g., key pairs, trust models). For instance, while powerful programs like VeraCrypt and GnuPG offer robust encryption features, their steep learning curves often deter regular users from adopting them widely, thereby limiting their impact (Iacono et al., 2018). This problem is exacerbated by the common absence of user-friendly error messages, clear visual progress indicators, and intuitive drag-and-drop functionality. Consequently, these solutions present a high barrier to entry, leaving non-technical users who are frequently the most vulnerable to cyberattacks underserved and potentially exposed (Salama et al., 2011). Psychological barriers, such as fear of making mistakes or perceived complexity, also play a significant role in low adoption rates (Sheng et al., 2010).

ii. Performance:

System performance can be significantly impacted by encryption and decryption processes, especially with asymmetric algorithms like RSA, which are computationally demanding, particularly when handling large files. Even with faster symmetric algorithms like AES, suboptimal software implementations, lack of hardware acceleration utilization, or constraints

on device resources (CPU, RAM) can still degrade their effectiveness and overall user experience (Dicle et al., 2024). Resource-intensive encryption processes that slow down other system functions or lead to noticeable delays can render existing solutions less feasible for users with older hardware or less powerful devices. In real-time applications, where encryption and decryption must occur almost instantaneously to avoid disrupting user workflow, performance bottlenecks are particularly apparent. These performance limitations often force users into a trade-off between security and usability, thereby hampering the wider adoption of robust encryption practices (Salama et al., 2011).

iii. Key Management:

The secure generation, storage, retrieval, and lifecycle management of encryption keys remains a major challenge with many current solutions (Gutmann, 2007). Numerous encryption technologies place the entire burden of key management on the user, requiring them to manually generate, store, backup, and retrieve keys. This manual approach significantly increases the likelihood of human error, which can lead to keys being misplaced, lost, forgotten, or inadvertently exposed to unauthorized access (Fornetix, 2019).

Furthermore, some tools lack sufficient guidance or built-in secure procedures for key management, leaving users vulnerable to security breaches if keys are not handled correctly. The absence of automated, user-friendly, or centralized key management solutions makes encryption particularly cumbersome in scenarios involving multiple users, collaborative encryption tasks, or long-term data archiving. Robust

encryption relies heavily on effective key management, yet this remains one of the weakest aspects of many contemporary systems, often overlooked by developers focusing solely on the cryptographic algorithms themselves (Ahamad & Abdullah, 2016).

iv. Vulnerabilities & Implementation Flaws:

The overall security of an encryption solution is significantly influenced not only by the chosen algorithm's strength (e.g., key length) but also critically by how it is implemented and integrated (Kohno et al., 2010). Algorithms that are poorly designed, use insufficiently short key lengths (like the original 56-bit DES), or are incorrectly implemented can be vulnerable to various cryptanalytic attacks, side-channel attacks (e.g., timing attacks, power analysis), or brute-force attacks. Even modern, strong algorithms like AES and RSA can be rendered vulnerable if implemented with weak keys, predictable random number generation, or incorrect mode of operation choices (Katz & Lindell, 2021). Furthermore, the security of currently used encryption techniques could be threatened in the future by unforeseen advances in computing power, particularly the potential development of large-scale quantum computers capable of breaking widely used public-key algorithms (Agrawal, 2024). Addressing these multifaceted problems requires careful algorithm selection, adherence to cryptographic best practices, rigorous implementation testing, sound key management protocols, and regular updates to encryption protocols and software (Awad Al-Hazaimh, 2013). The complexity of avoiding these pitfalls means that even well-intentioned software can contain subtle but critical vulnerabilities.

v. Accessibility for Specific User Groups (e.g., SMEs):

Many enterprise-grade encryption solutions are too expensive or complex for Small to Medium-sized Enterprises (SMEs), while many free tools lack the necessary support or features for business use (Institute, 2024). SMEs often have limited IT resources and expertise, making them particularly vulnerable yet underserved by the current market of encryption tools. There's a gap for solutions that offer a balance of robust security, ease of deployment, and affordability tailored to their specific needs.

2.4 Relevant Algorithms and Methodologies

2.4.1 Survey of Relevant Algorithms

A crucial component of creating the "LockMe: Secure File Encryption and Decryption Desktop Application" is the judicious selection of appropriate encryption methods. This choice profoundly impacts the application's security posture, operational effectiveness, and overall usability. In particular, symmetric-key algorithms are highly pertinent to this project due to their inherent speed, computational efficiency, and proven capacity to encrypt large files effectively. Among the plethora of available algorithms, the following are of significant importance and relevance to the LockMe application:

i. Advanced Encryption Standard (AES):

AES, originally known as Rijndael, is the de facto industry standard for symmetric encryption and has been widely adopted across numerous sectors globally since its selection by the U.S. National Institute of Standards and Technology (NIST) in 2001 (Dworkin et al., 2001). It offers

an elevated level of flexibility and resilience against brute-force attacks and other cryptanalytic techniques by supporting key lengths of 128, 192, and 256 bits.

AES operates on fixed-size blocks of 128 bits and employs a series of substitution-permutation network (SPN) operations over multiple rounds (10 rounds for 128-bit keys, 12 for 192-bit, and 14 for 256-bit keys) to ensure that data is securely and thoroughly jumbled. Its high efficiency in both hardware (often with dedicated CPU instructions) and software implementations makes it an optimal choice for file encryption in LockMe (Daemen & Rijmen, 2002). The algorithm's widespread acceptance as a global standard, extensive scrutiny by the cryptographic community, and proven track record further support its compatibility and dependability (Mushtaq et al., 2017). Current NIST guidance affirms the security of AES with all three key sizes for protecting sensitive information.

ii. Blowfish:

Developed by Bruce Schneier in 1993, Blowfish is another well-known symmetric encryption algorithm recognized for its speed and simplicity (Schneier, 2019). It is a 64-bit block cipher that offers flexibility by supporting variable key lengths ranging from 32 bits up to 448 bits. Blowfish was designed to be fast, free of patents, and unencumbered by licensing fees, leading to its adoption in a variety of software. Because Blowfish is performance-optimized, it is particularly well-suited for applications requiring rapid encryption and decryption, such as embedded systems or real-time communication. However, when working with larger files, its 64-bit block size could be a drawback compared to AES's 128-bit

blocks, potentially making it more vulnerable to certain cryptographic attacks like the birthday attack if used improperly over very large amounts of data with the same key (Schneier et al., n.d.). Despite this, Blowfish's simplicity and efficiency make it a solid choice for light to moderate encryption workloads where extreme security against state-level attackers is not the primary concern (Mushtaq et al., 2017).

iii. Data Encryption Standard (DES):

DES was one of the first symmetric encryption algorithms to be widely adopted internationally. Developed at IBM in the early 1970s and adopted as a U.S. federal standard in 1977, DES operates on 64-bit blocks of data and uses a 56-bit key (Technology, 1977). Although DES laid the foundational principles for many contemporary encryption methods and was considered secure for its time, its relatively short 56-bit key length eventually rendered it vulnerable to brute-force attacks as computational power increased significantly (Foundation, 1998).

Consequently, DES is now considered outdated and insecure for most modern applications. The shortcomings of DES, despite its historical importance in the development of cryptography, underscore the critical need for more robust algorithms with larger key sizes and greater resistance to cryptanalysis, such as AES and even Blowfish for certain contexts (Mushtaq et al., 2017).

iv. Triple Data Encryption Standard (3DES or TDEA):

By applying the DES algorithm three times in succession to each data block, 3DES (Triple DES) was developed as an enhancement over the

original DES to extend its longevity and address the key size vulnerability (*3DES: Triple Encryption Standard Explained*, 2025).

It typically uses two or three distinct 56-bit keys (effectively providing 112-bit or 168-bit key strength, respectively, though susceptible to meet-in-the-middle attacks reducing effective strength). Compared to DES, this significantly strengthens the encryption, increasing its resistance to brute-force attacks.

However, 3DES is considerably slower and more computationally demanding than more modern algorithms like AES, as it involves performing the DES encryption/decryption process three times. While 3DES is still found in some legacy systems, particularly in the financial industry for a time, more effective, secure, and efficient alternatives like AES are now typically recommended and are progressively replacing it (Mushtaq et al., 2017).

v. Relevance to the Current Project:

For the LockMe application, speed, robust security, and ease of implementation are top priorities to ensure a smooth and trustworthy user experience, especially for non-technical users. Symmetric-key algorithms like AES are the best option for file encryption and decryption because they strike an excellent balance between strong security guarantees and efficient computational performance (Jajodia et al., 2024). The goals of LockMe are well-aligned with AES's demonstrated dependability, its support in numerous cryptographic libraries, hardware acceleration capabilities, and its capacity to manage large datasets without a significant performance sacrifice. Due to its larger block size (128 bits vs. Blowfish's 64 bits),

widespread global standardization, and superior security against known attacks, AES (specifically AES-256 for maximum security within LockMe's scope) is the recommended and chosen primary algorithm for this project. While Blowfish could offer an option for light-duty workloads, standardizing on AES simplifies development and ensures a consistent security level. Algorithms like DES and 3DES, despite their historical significance, are generally considered inappropriate for new development like LockMe because they do not satisfy the security and performance criteria of contemporary encryption requirements. By leveraging the advantages of AES, LockMe can ensure that users receive a strong, effective, and user-friendly encryption solution that meets the demands of the current cybersecurity environment.

2.4.2 Implementation Approaches

The development of the "LockMe: Secure File Encryption and Decryption Desktop Application" calls for a methodical and effective implementation strategy that correctly integrates the chosen encryption methods. To ensure the application is dependable, secure, and easy to use, this procedure entails selecting the appropriate programming language(s), utilizing well-vetted cryptography libraries, adhering to secure coding best practices, and following a robust software development lifecycle (Howard & Leblanc, 2009).

Programming Languages for Development

Several programming languages are suitable for implementing file

encryption and decryption functionalities in the LockMe application. The choice often depends on factors like cross-platform capabilities, availability of mature cryptographic libraries, developer expertise, performance requirements, and ease of GUI development.

i. Java

This is a popular platform-independent and versatile programming language renowned for creating secure and portable applications (Gosling et al., 2015). Its strong security features, extensive standard library, and rich ecosystem of third-party libraries, including the Java Cryptography Extension (JCE) and Bouncy Castle, make it a compelling option for implementing encryption techniques. The JCE provides a framework and implementations for encryption, key generation and management, and other cryptographic functions (Oracle, n.d.). Bouncy Castle is a widely used open-source library that offers a vast array of cryptographic algorithms and protocols, often including newer or less common ones not found in the standard JCE (*Bouncycastle.Org*, n.d.). Because of Java's "write once, run anywhere" philosophy, the LockMe application developed in Java can function consistently on both Windows and Linux (and other operating systems supporting a JVM), serving a wide range of users. GUI development can be achieved using frameworks like Swing or JavaFX.

ii. Python

Python is a highly regarded option for developing cryptographic applications due to its simplicity, readability, and extensive library support

(Lutz, 2018). Python developers can readily implement encryption methods like AES and RSA with the aid of well-maintained libraries such as PyCryptodome and the ‘cryptography’ package (Hazzmat). PyCryptodome is a fork of the older PyCrypto library and offers a comprehensive set of cryptographic primitives (*Welcome to PyCryptodome’s Documentation — PyCryptodome 3.15.0 Documentation*, n.d.). The ‘cryptography’ library aims to be a "cryptography for humans" library, providing high-level recipes and low-level interfaces for common cryptographic tasks (*Welcome to Pyca/Cryptography — Cryptography 42.0.0.Dev1 Documentation*, n.d.). Python is excellent for rapid development and prototyping of the LockMe application due to its ease of use and concise syntax. Furthermore, Python's cross-platform capabilities, combined with GUI frameworks like Tkinter, PyQt, or Kivy, align well with the project's objective of ensuring compatibility across multiple operating systems.

iii. C++

For resource-intensive cryptographic processes, the fine-grained control over system resources and high performance offered by C++ can be beneficial (Stroustrup, 2013). Libraries such as OpenSSL and Crypto++ provide dedicated support for implementing a wide range of encryption and decryption features. OpenSSL is a robust, commercial-grade, and full-featured toolkit for the Transport Layer Security (TLS) and Secure Sockets Layer (SSL) protocols and also a general-purpose cryptography library (OpenSSL Foundation, 2019). Crypto++ is a free C++ class library of cryptographic schemes (*Crypto++ Library 8.6 | Free C++ Class Library of Cryptographic Schemes*, n.d.). C++ can deliver unmatched speed and

efficiency, especially for applications that manage very large files or require real-time encryption with minimal overhead. However, its complexity, manual memory management (in older versions), and longer development cycles may necessitate more extensive development effort and expertise. GUI development can be done with frameworks like Qt or WxWidgets.

Cryptographic Libraries and Frameworks

Cryptographic libraries are crucial components for implementing encryption algorithms correctly and securely. They offer pre-built, optimized, and thoroughly tested functions that significantly reduce development time, minimize the risk of common implementation errors, and help ensure that the cryptographic operations are secure (Yadavalli, n.d.). The libraries listed below are highly pertinent to the project:

i. PyCryptodome (Python)

Supports a vast array of encryption methods including AES, RSA, and Blowfish. It provides secure and efficient implementations of cryptographic primitives and is relatively straightforward to integrate into Python programs.

ii. Java Cryptography Extension (JCE)

JCE is a standard Java API providing extensive support for secure data storage, key generation, and encryption. Bouncy Castle augments JCE with a wider range of algorithms and utilities, making it a dependable

choice for Java-based cryptographic development.

iii. Crypto++ (C++)

A free, open-source C++ library supporting numerous encryption methods and cryptographic tools. It is valued for its performance, flexibility, and suitability for creating effective, high-security applications.

iv. OpenSSL (C++)

A widely adopted and robust library for implementing cryptographic algorithms and protocols. Frequently utilized in secure communication systems, it supports both symmetric and asymmetric encryption and is known for its comprehensive feature set.

Development Frameworks

The application's cross-platform interoperability and usability are also significantly influenced by the choice of development framework for the Graphical User Interface (GUI). The selected programming language can be paired with frameworks like:

i. Qt Framework

Enables the development of feature-rich, native-looking GUIs in C++ with excellent cross-platform support for Windows, Linux, macOS, and more (Company, 2019).

ii. Tkinter or PyQt (Python)

Tkinter is Python's standard GUI framework, suitable for simpler interfaces, while PyQt (Python bindings for Qt) allows for more complex and feature-rich GUI development in Python (Summerfield, 2007).

iii. JavaFX or Swing (Java)

JavaFX is a modern framework for creating rich client applications, while Swing is an older but still widely used GUI toolkit for Java. Both support cross-platform GUI development.

iv. Next.js + React (TypeScript)

React supplies a component-based UI, Next.js handles routing and server-side rendering, and TypeScript adds static typing. Packaged as a Progressive Web App or via containers such as Electron/Tauri, the same code delivers a desktop-class interface on Windows, Linux, and other platforms.

Secure Software Development Lifecycle (SSDLC)

It is crucial to follow an SSDLC approach, integrating security considerations into every phase of development, from requirements gathering to deployment and maintenance (Mcgraw, 2006). This includes threat modelling, secure coding standards, regular code reviews, and comprehensive security testing (static analysis, dynamic analysis, penetration testing).

Implementation Workflow

The implementation began with a detailed requirements-gathering phase. Functional needs such as client-side AES-256-GCM encryption, secure key generation, drag-and-drop uploads, and clear progress messages were documented alongside non-functional targets for speed, usability, and full support on both Windows and Linux. File formats and performance benchmarks were set so that later testing could measure success objectively.

A combined architecture and UI design stage followed. Here, a modern web stack that consists of Next.js, React, TypeScript, and the browser's Web Crypto API is selected, and drafted wire-frames that positioned drag-and-drop zones, progress indicators, and simple navigation in a single-window layout. With those blueprints in place, developers wired up the cryptography layer: keys are generated with secure randomness, and AES-256-GCM handles both encryption and integrity in one pass. The React interface came next, styled with Tailwind CSS and ShadCN components to keep the visuals clean and consistent, while real-time status messages and error banners guided users through each step.

Core logic then bridged the gap between UI and cryptography. File readers passed local data to the encryption engine and wrote the resulting .lockme output, while auxiliary modules managed key storage and AI helpers such as the passphrase generator and strength checker. Testing was iterative: unit tests covered components and crypto functions, integration tests exercised full workflows, and usability sessions ensured non-technical users could complete tasks easily. Security scans and performance profiling rounded out the quality checks.

Deployment packaged the web build through Vercel for immediate browser access and wrapped the same codebase in Electron/Tauri to produce native installers for Windows and Linux. Final steps included drafting a concise user guide, documenting the internal API, and setting a maintenance plan that defines branching rules and release cadence. Together, these phases turn a secure cryptographic core into an accessible desktop-grade application, balancing technical rigour with everyday usability.

2.4.3 Evaluation Metrics

A set of clearly defined assessment metrics can be used to evaluate the project's security, effectiveness, performance, and usability. These metrics ensure that the application not only meets its functional requirements but also complies with industry standards for secure and efficient encryption and aligns with user expectations. The evaluation focuses on measurable factors crucial for assessing the application's success, such as processing speed, resource utilization, resilience to attacks, and user satisfaction. An extended explanation of the main evaluation metrics is provided below:

1. Encryption/Decryption Speed

This metric refers to the time taken for the application to process files of various sizes (e.g., small <1MB, medium 1-100MB, large >100MB) and types (text, images, compressed archives, videos). It is essential for evaluating the efficiency of the implemented algorithms (especially AES) and their practical applicability for real-time or interactive use cases (Schneier et al., n.d.). A high-performing encryption program should operate with minimal lag,

providing users with responsive and seamless experiences. Testing involves measuring average and maximum processing times under controlled conditions to ascertain the application's throughput and identify potential bottlenecks. For users who routinely encrypt/decrypt numerous files or manage huge datasets, this statistic is very important. Speed gains can be achieved by minimizing code bottlenecks, optimizing the algorithm's implementation (e.g., leveraging hardware AES instructions if available), and efficient file I/O operations.

2. Throughput

This measures the amount of data (e.g., in MB/second) that the application can handle during encryption and decryption operations per unit of time. It serves as a gauge of the program's scalability and overall processing power, especially relevant when users need to encrypt or decrypt multiple files simultaneously or handle very large individual files. High throughput figures indicate the system's capability to manage bulk processes efficiently, making it suitable for individuals or small businesses dealing with substantial datasets. Throughput is assessed through controlled trials where a predetermined volume of data is processed by the application over a fixed period, with outcomes compared against industry benchmarks or similar tools. This metric complements encryption/decryption speed by providing a broader picture of the system's data-handling capabilities.

3. Resource Consumption (CPU, Memory, Disk I/O)

This assesses how the encryption and decryption procedures impact the system's hardware resources, specifically CPU utilization (percentage), memory footprint (RAM usage in MB), and disk I/O rates. Ensuring the

application remains lightweight and does not excessively overload the user's device is particularly crucial for users with older hardware, limited resources, or when running multiple applications concurrently (Provos, 2000). Excessive CPU or memory usage can lead to system slowdowns and poor user experience, while high power consumption can shorten the battery life of portable devices. Performance monitoring tools are used to measure resource usage during intensive encryption/decryption processes, providing insights into the program's efficiency and its optimization requirements. A key design objective for LockMe is to maintain a balance between strong security and resource efficiency.

4. Security Analysis (Vulnerability Assessment & Cryptographic Robustness)

This is a critical indicator for assessing the resilience of the encryption algorithms as implemented and their ability to resist various types of attacks. The analysis involves testing the application against known vulnerabilities and cryptographic best practices, including:

- **Brute-Force Attack Resistance:** Evaluating the encryption keys' resilience to exhaustive search efforts, where every potential key is methodically tried. Robustness is ensured by using strong encryption methods like AES with appropriately long key lengths (e.g., 256 bits), making brute-force computationally infeasible (Lenstra & Verheul, 2001).
- **Known Plaintext/Ciphertext Attacks:** Assessing if knowledge of some plaintext-ciphertext pairs compromises the key or subsequent encryptions.

- **Differential and Linear Cryptanalysis Resistance:** Evaluating the algorithm's (and its implementation's) resistance against advanced cryptanalytic techniques that analyse ciphertext variations resulting from specific plaintext modifications or statistical linear approximations (Biham & Shamir, 1993). While AES itself is designed to resist these, implementation flaws could introduce weaknesses.
- **Side-Channel Attack Considerations:** Assessing, at least conceptually, how resistant the program might be to attacks that exploit information leaks from the physical implementation (e.g., timing variations, power consumption, electromagnetic emissions) during the encryption process. Mitigation might involve using constant-time operations where feasible (Kocher et al., 1999).
- **Key Management Security:** Verifying that key generation uses cryptographically secure pseudo-random number generators (CSPRNGs), keys are stored securely (e.g., encrypted at rest, appropriate permissions), and key handling practices minimize exposure (Barker, 2020).
- **Correct Use of Cryptographic Primitives:** Ensuring proper use of modes of operation (e.g., avoiding ECB for most uses, using authenticated encryption modes like AES-GCM), correct handling of Initialization Vectors (IVs)/nonces (uniqueness, unpredictability where required).
- **Verifying that the program complies with best practices for secure data processing and cryptographic standard implementation** is a vital aspect of security analysis. Addressing possible security threats during the assessment stage ensures LockMe can provide users with a dependable

and trustworthy encryption solution.

5. User Experience (UX) Metrics & Usability Testing

While not directly linked to cryptographic security or raw performance, UX metrics provide crucial information about how usable, learnable, and satisfying the program is for its intended audience, especially non-technical users. These indicators are typically obtained through:

- Task Completion Rates: Percentage of users who successfully complete core tasks (e.g., encrypt a file, decrypt a file, manage a key) without assistance.
- Time on Task: Average time taken by users to complete specific tasks.
- Error Rates: Frequency and types of errors users encounter.
- System Usability Scale (SUS): A standardized questionnaire providing a global measure of system usability (Brooke, 1995).
- User Satisfaction Surveys/Feedback: Qualitative feedback on ease of use, clarity of interface, responsiveness, and overall satisfaction.
- A satisfying user experience, characterized by intuitive design, clear instructions, and minimal friction, guarantees that the application will be widely adopted and effectively used, especially by non-technical users who are a primary target for LockMe.

6. Scalability

This metric assesses the application's ability to handle increasing amounts of data or a growing number of files without significant degradation in performance or stability (Bondi, 2000). For LockMe, this would involve

testing how well it performs when encrypting very large files (e.g., several GBs) or a large number of smaller files in a directory (if batch processing were a feature).

7. Interoperability Testing

Given LockMe's cross-platform goals (Windows and Linux), interoperability testing is crucial. This involves verifying that files encrypted on one platform can be successfully decrypted on the other, and that the application functions consistently across both operating systems (*Interoperability Software Testing*, 2019). This includes checking for issues related to file path conventions, character encodings, and dependencies.

2.5 State of the Art in the Field

2.5.1 Recent Advances

The efficiency of encryption techniques and the broader field of data security have been significantly improved by recent, rapid developments in cryptography. These advancements address emerging challenges posed by growing computational power (including the looming threat of quantum computers), evolving online dangers, and the increasing societal demand for easily navigable and highly secure encryption solutions. The development of novel algorithms, cryptographic protocols, and methodologies, such as the ongoing standardization of Post-Quantum Cryptography (PQC) and advancements in areas like Homomorphic Encryption (HE) and lightweight cryptography, stands out among these developments, presenting encouraging answers for enhanced security and attack resistance.

a. HiSea Algorithm

A major advancement in cryptographic techniques, the HiSea algorithm was created to fix flaws in conventional encryption schemes. With its emphasis on excellent security and computational efficiency, HiSea is a lightweight block cypher that works well in resource-constrained settings like embedded systems, mobile platforms, and Internet of Things (IoT) devices. Its design incorporates several advanced features (Dhany et al., 2018):

- i. Enhanced Key Schedule: Subkeys are essential for every encryption round, and HiSea's strong key scheduling system guarantees their safe creation. As a result, the technique is impervious to attacks including differential and linear cryptanalysis.
- ii. Optimized Performance: By striking a balance between speed and security, HiSea is able to encrypt and decrypt data effectively without sacrificing its defences against intrusions. Applications that need real-time encryption, including secure communications and video streaming, will especially benefit from this.
- iii. Scalability: The algorithm's ability to accommodate various key lengths offers flexibility according to the application's security needs.

HiSea solves the difficulties of deploying cryptographic algorithms on devices with constrained processing power by combining these aspects, which also improves security.

b. Post-Quantum Cryptography (PQC)

The development of post-quantum cryptographic algorithms represents one of the most critical recent advancements in cryptography, driven by the anticipated threat from large-scale quantum computers (Bernstein & Lange, 2017). Quantum computers, if realized with sufficient power, could use Shor's algorithm to break currently secure public-key encryption techniques like RSA and ECC (Elliptic Curve Cryptography) (Shor, 1997). In response, the U.S. National Institute of Standards and Technology (NIST) has been spearheading a multi-year process to standardize PQC algorithms designed to be secure against both classical and quantum attacks (Swayne, 2023).

Promising candidates that have emerged and are moving towards standardization include lattice-based schemes (e.g., CRYSTALS-Kyber for key encapsulation and CRYSTALS-Dilithium for signatures), hash-based signatures (e.g., SPHINCS+), code-based schemes (e.g., Classic McEliece), and multivariate polynomial cryptography.

These techniques are based on mathematical problems believed to be hard for quantum computers to solve.

c. Homomorphic Encryption (HE)

Homomorphic Encryption represents a paradigm shift in cryptography, allowing computations to be performed directly on encrypted data without needing to decrypt it first (*Homomorphic Encryption*, 2025).

This development is highly pertinent in scenarios like secure cloud computing, where sensitive data must be processed by third-party services without exposing the plaintext data to the service provider (Armknrecht et

al., 2015). Fully Homomorphic Encryption (FHE) schemes, such as those based on lattice cryptography (e.g., BGV, BFV, CKKS), enable arbitrary computations (both addition and multiplication) on encrypted data, opening doors to secure data analytics, privacy-preserving machine learning, and secure outsourced computations.

While FHE is still computationally intensive, significant progress has been made in improving its efficiency and practicality for real-world applications.

d. Lightweight Cryptography

In addition to specialized algorithms like HiSea, there's a broader field of lightweight cryptography focusing on algorithms tailored for resource-constrained devices, common in Internet of Things (IoT) environments, RFID tags, and embedded systems (Beaulieu et al., 2013). These devices often have limited processing power, memory, and energy. NIST also ran a Lightweight Cryptography (LWC) competition to standardize algorithms suitable for these environments, with ASCON being selected (Computer Security Division, 2017). LWC algorithms (like SIMON, SPECK, PRESENT, and the standardized ASCON family) prioritize smaller footprint (code size, RAM usage), lower energy consumption, and faster processing rates on constrained hardware, while still providing strong security against relevant threats.

e. Advanced Security Features in Modern Systems

Modern cryptographic systems also incorporate additional security features, including:

- i. **Zero-Knowledge Proofs (ZKPs):** ZKPs enable one party (the prover) to prove to another party (the verifier) that a statement is true, without revealing any information beyond the validity of the statement itself (Goldreich et al., 1991). This technology is finding growing applications in areas like privacy-preserving authentication, blockchain technology (e.g., Zcash, Monero), and identity verification where privacy is paramount.
- ii. **Blockchain-Based Cryptographic Protocols:** Blockchain technology, inherently reliant on cryptographic primitives like hash functions and digital signatures, has spurred advancements in areas like decentralized key management systems, secure consensus mechanisms, and auditable transparent ledgers (Narayanan et al., 2016). These developments aim to enhance the security, transparency, and resilience of distributed systems.
- iii. **AI-Driven Cryptographic Analysis & Threat Detection:** Machine learning (ML) and Artificial Intelligence (AI) techniques are increasingly being applied to assess the security of cryptographic protocols, identify potential vulnerabilities in implementations, and detect anomalous behaviour indicative of cyberattacks (Al-Fuqaha et al., 2015). Conversely, cryptography is also being used to protect the privacy and integrity of AI models and training data.
- iv. **Authenticated Encryption with Associated Data (AEAD):** Modern symmetric encryption schemes increasingly emphasize AEAD modes (e.g., AES-GCM, ChaCha20-Poly1305) which simultaneously provide confidentiality, integrity, and authenticity of encrypted data (Rogaway,

2002). This is a significant improvement over older approaches where confidentiality and integrity were often handled as separate, potentially error-prone steps.

2.5.2 Trends and Future Directions

The field of cryptography is dynamic, constantly evolving to address new threats and leverage new technological capabilities. Several key trends and future directions are shaping the landscape:

a. Transition to Post-Quantum Cryptography (PQC)

As quantum computing technology matures, the migration from current public-key algorithms (RSA, ECC) to PQC standards is becoming a major focus for organizations worldwide (Moody et al., 2020).

NIST is standardizing algorithms like CRYSTALS-Kyber and NTRU for key establishment and CRYSTALS-Dilithium and Falcon for digital signatures.

Future developments will involve the widespread integration of these quantum-resistant algorithms into internet protocols (TLS, SSH), operating systems, and critical infrastructure sectors like finance and healthcare. Hybrid cryptography systems, which combine classical and PQC algorithms during a transition period, will likely be implemented to ensure ongoing security and backward compatibility as quantum technology advances.

b. Advancements in Fully Homomorphic Encryption (FHE)

FHE allows computations on encrypted data without prior decryption, offering revolutionary possibilities for privacy in data processing, particularly for applications in healthcare, finance, and secure cloud computing (Brakerski et al., 2014).

Future research will continue to focus on improving computational efficiency and reducing the ciphertext expansion of FHE schemes to make them more practical for real-time, large-scale applications. Its integration into privacy-preserving machine learning (PPML) will enable the training and querying of AI models on sensitive data without disclosing the underlying confidential information.

c. Proliferation of Lightweight Cryptography (LWC)

With the exponential growth of IoT devices and wearables, the demand for lightweight cryptography designed for resource-constrained environments is rapidly increasing (McKay et al., 2017). Algorithms that offer robust security with minimal computational cost, low power consumption, and small memory footprints are essential. Standardization initiatives, such as NIST's selection of the ASCON family, are setting benchmarks for secure IoT applications. These developments will facilitate secure communication and data protection across a vast and diverse array of interconnected devices.

d. AI and Machine Learning in Cryptography and Security

Cryptography is increasingly incorporating machine learning (ML) and artificial intelligence (AI) to enhance system functionality and security

(Chalapathy & Chawla, 2019). AI is being utilized to develop optimized encryption algorithms, automate the discovery of vulnerabilities in cryptographic protocols, and build adaptive security systems that can respond dynamically to new and evolving threats. Conversely, cryptographic techniques are being integrated into AI workflows to protect sensitive training data and models (e.g., via differential privacy, FHE, or secure multi-party computation), ensuring privacy and integrity in AI-driven applications. The synergy between AI and cryptography is shaping the future of secure and intelligent systems.

e. Blockchain-Based Cryptography and Decentralized Systems

Blockchain technology, leveraging cryptographic primitives like hash functions and digital signatures, continues to drive innovation in secure decentralized systems (Zheng et al., 2017). Blockchain-based identity systems aim to provide secure and user-controlled decentralized authentication. Newer privacy-enhancing technologies like zero-knowledge proofs (ZKPs) are improving privacy in blockchain transactions and other applications. Furthermore, research into scalable and efficient consensus mechanisms is ongoing to reduce the computational load on blockchain networks, making them more accessible and sustainable for a wider range of use cases, including secure supply chain management and decentralized finance (DeFi).

f. Federated Learning with Encryption

Federated Learning (FL) allows multiple parties to collaboratively train a machine learning model without sharing their raw data, enhancing

privacy (McMahan et al., 2017).

To further protect the model updates (gradients or weights) exchanged during this process, encryption techniques like homomorphic encryption or secure aggregation are being integrated into FL frameworks.

This trend aims to provide stronger privacy guarantees against inference attacks on the model updates themselves.

g. Emphasis on Usable Security and Privacy by Design

There is a growing recognition that even the strongest cryptographic systems are ineffective if users cannot use them correctly or are bypassed due to complexity (Cranor & Garfinkel, 2005). Future developments will increasingly focus on "usable security" and "privacy by design" principles, embedding security and privacy considerations into the earliest stages of system design and prioritizing user-friendly interfaces and workflows. This includes better error messaging, clearer indicators of security status, and more intuitive key management processes to lower the barrier for adoption by non-expert users.

2.6 Critical Review of Similar Products or Systems

2.6.1 Review of Similar Projects

A wide range of file encryption tools is available, each utilizing unique algorithms and offering diverse security functionalities and user experiences. Among the most prominent are Hat.sh and Enc, which serve as valuable references for assessing encryption solutions. Other notable systems include VeraCrypt for full-disk encryption and AxCrypt for user-friendly file

encryption, each with its own strengths and weaknesses in terms of usability, feature set, and target audience. The user experience offered by these tools varies significantly, from highly technical command-line interfaces to more intuitive graphical approaches (Cranor & Garfinkel, 2005).

a. Hat.sh



Figure 2.0.1 Logo of Hat.sh

Hat.sh is a browser-based, lightweight solution for secure file encryption and decryption that does not require installation. Its main goals are accessibility and ease of use, allowing users to carry out encryption operations right within their browser. The program uses the AES-GCM encryption algorithm, a reliable and effective standard that incorporates authentication to guard against file manipulation. Hat.sh's dedication to privacy is one of its best qualities; all encryption and decryption procedures take place locally on the user's computer, guaranteeing that no data is sent over the internet. Additionally, because the program is platform-independent, it may be used on a variety of devices using contemporary web browsers. Because Hat.sh is open-source, developers can examine, alter, and contribute to its code, adding an extra degree of flexibility and confidence (sh-dv, 2022).

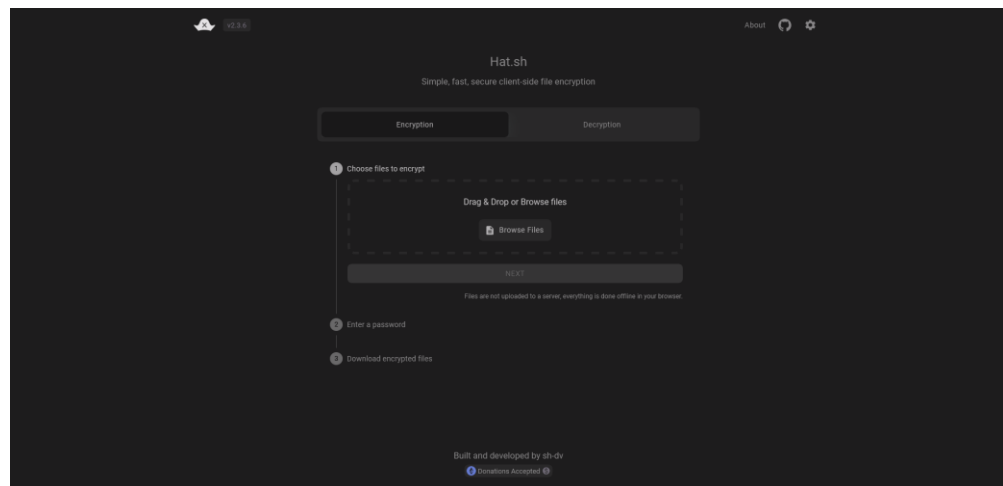


Figure 2.0.2 Interface of Hat.sh

Hat.sh has certain drawbacks that prevent its wider use, despite its benefits. Advanced features like file integrity checking and encryption key management, which are necessary for more complicated or professional use cases, are sacrificed in favour of the tool's basic design. Despite local processing, its functionality is limited in situations when internet connectivity is restricted or completely absent due to its dependence on a browser-based environment. Additionally, because browser-based operations are not optimised for extensive data processing, Hat.sh may experience performance problems while handling huge files. By addressing these issues, such as adding a stand-alone version or improving important administrative functions, its usefulness and appeal might be greatly increased.

b. Enc

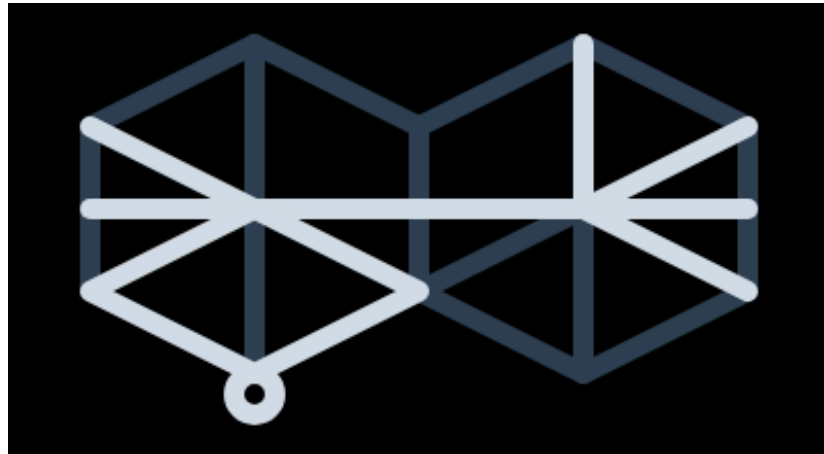


Figure 2.0.3 Logo of Enc

For those who appreciate speed, versatility, and scripting capabilities, Enc is a command-line file encryption tool. Enc is especially well-suited for encrypting large files and automating encryption procedures since it makes use of the AES encryption standard, which is renowned for its strong security and computational efficiency. It is perfect for those with advanced technical knowledge because of its simple architecture, which allows for easy integration into batch processes and other command-line activities. Furthermore, Enc is cross-platform, operating effectively on Linux, macOS, and Windows, guaranteeing interoperability in a variety of settings. Because Enc is open-source, users can alter its features to suit their own needs and build confidence in its core security measures (life4, 2024).

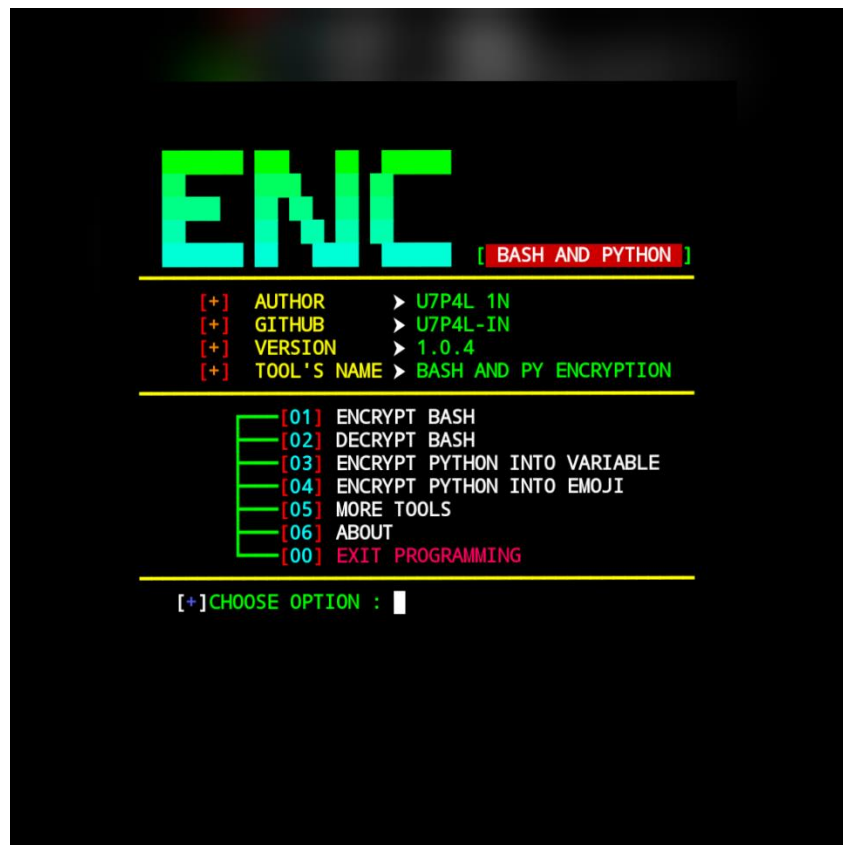


Figure 2.0.4 Interface of Enc

However, non-technical users may find it difficult to understand text-based commands due to Enc's emphasis on command-line execution. Because the utility does not have a graphical user interface, people who are not familiar with command-line operations cannot utilise it. Additionally, Enc has integrated key management capabilities, so users are left to handle and store encryption keys securely, a task that is prone to mistakes and security flaws. Improvements like an optional GUI, integrated key management, and easily navigable documentation would help Enc gain wider acceptance.

2.6.2 How This Project Differs

LockMe is designed to close the gaps left by well-known encryption tools such as Hat.sh, Enc, VeraCrypt, and AxCrypt. By combining a modern web stack (Next.js, React, and TypeScript) with client-side AES-256-GCM, it delivers enterprise-grade protection in a package that ordinary users can operate without a learning curve.

First, the interface sets LockMe apart. Hat.sh confines users to a bare-bones browser page, and Enc requires command-line skill. LockMe, in contrast, offers a full graphical desktop experience. Drag-and-drop zones let users add files in one motion, while real-time progress bars and clear error messages remove the guesswork. Because the application is bundled as an Electron or Tauri desktop build, it runs entirely offline; network restrictions or browser quirks never block core functions.




Second, LockMe provides true cross-platform consistency. Unlike Hat.sh, which is limited by whatever browser happens to be installed, or AxCrypt, which ties key features to Windows, one LockMe build serves both Windows and Linux without compromise. Native file-system access allows it to process multi-gigabyte archives far beyond the practical limits of browser-based tools, yet it retains the single-code-based simplicity prized by Enc.

Finally, the project bridges the usability gap between tools that are too technical and those that oversimplify security. Non-technical users benefit from step-by-step workflows that hide cryptographic detail. Technical users still gain the full strength of AES-256 and solid key management, and the design leaves room for future advanced options such as custom cipher suites without cluttering the main interface.

In short, LockMe offers the rich interface of a native desktop app, the portability of a browser solution, and the cryptographic strength of professional suites, delivering a secure and approachable alternative for users at every skill level.

2.6.3 Comparison Between Similar Systems and Proposed System

Table 0.1 Comparison table between systems

Name of Application	 Hat.sh	 Enc	 LockMe
Description	A lightweight, web-based file encryption tool that enables users to encrypt and decrypt files directly through their browser using AES-256 encryption.	A developer-focused encryption library that supports multiple algorithms, designed for easy integration into applications and optimized for performance.	A file encryption and decryption application offering AES-256 encryption, secure key management, and cross-platform compatibility for Windows and Linux users.
Strengths	<ul style="list-style-type: none"> a) Lightweight and web-based: Hat.sh allows users to encrypt and decrypt files directly through a browser without requiring installation, offering portability and ease of use. b) Secure encryption: The application employs AES-256 encryption, ensuring strong protection for user data. c) Open-source and transparent: As an open-source project, Hat.sh allows users to inspect its code and ensure its reliability, fostering trust and community contributions. 	<ul style="list-style-type: none"> a) Developer-focused functionality: Enc is specifically designed for easy integration into other projects, making it ideal for developers. b) Support for multiple algorithms: It supports various encryption algorithms, including AES, offering flexibility in implementation. c) Lightweight and efficient: Enc is optimized for performance, making it suitable for resource-constrained applications. d) Open-source: Its open-source nature encourages active community involvement and enhancements. 	<ul style="list-style-type: none"> a) User-friendly design: LockMe features an intuitive graphical user interface (GUI) with drag-and-drop functionality, making it accessible to users with varying technical expertise. b) Comprehensive encryption: LockMe utilizes AES-256 encryption and provides secure key management for generating, saving, and retrieving keys. c) Cross-platform compatibility: LockMe runs seamlessly on both Windows and Linux operating systems, catering to a wide range of users. d) Offline functionality: The application processes all encryption and decryption locally without reliance on cloud services, ensuring user privacy.
Weaknesses	<ul style="list-style-type: none"> a) Limited features: Hat.sh focuses solely on file encryption and decryption, lacking advanced functionalities like key management or cloud integration. b) Reliance on a browser: As a web-based tool, it requires a browser to operate, which may limit usability in offline scenarios or highly restrictive environments. c) No dedicated support for developers: Unlike library-focused solutions, Hat.sh does not provide tools for integration into larger systems or applications. 	<ul style="list-style-type: none"> a) Requires programming expertise: Enc is not a standalone tool and demands programming knowledge to integrate effectively, making it inaccessible for non-technical users. b) Absence of a user interface: Enc lacks a GUI, which may hinder its usability for end-users unfamiliar with command-line or API-based operations. c) Limited out-of-the-box functionality: Enc is not designed as a comprehensive application and serves only as a library component. 	<ul style="list-style-type: none"> a) No cloud integration: LockMe does not support cloud-based services, which may limit its appeal to users requiring synchronized backups or multi-device access. b) Limited scope: LockMe focuses solely on local file encryption and decryption without enterprise-level functionalities. c) Dependency on user knowledge for key management: While LockMe simplifies key handling, users still need to securely manage their keys to prevent data loss or unauthorized access.

2.7 Summary of Findings

2.7.1 Synthesis of Literature

The literature review underscores the critical and escalating importance of data security within the contemporary digital environment. The rapid proliferation of data collection, storage, and transmission has unfortunately been accompanied by a significant increase in the frequency, sophistication, and impact of cybersecurity risks, including hacking, malware, ransomware attacks, and data breaches (Kshetri, 2013). These challenges highlight an undeniable and urgent need for strong, reliable, and accessible encryption techniques to safeguard confidential and sensitive data. For ensuring data security, maintaining secrecy (confidentiality), and guaranteeing integrity, cryptography—and more specifically, the processes of encryption and decryption—is absolutely essential (Kohno et al., 2010).

Current research and existing technologies demonstrate a wide array of encryption methods, including well-established algorithms like AES, RSA, and Blowfish, each possessing unique advantages, disadvantages, and optimal use cases (Menezes et al., 1996). The landscape of available encryption solutions is illustrated by programs such as VeraCrypt, which offers powerful full-disk and volume encryption, and AxCrypt, which emphasizes user-friendly design for file-level protection, alongside command-line tools like GnuPG that offer flexibility for technical users. However, the review also clearly identifies persistent issues related to usability, particularly for non-technical users, and accessibility challenges, especially concerning cross-platform compatibility and intuitive key management (Das et al., 2020). These discrepancies and unresolved issues highlight the pressing necessity for a product like LockMe,

which aims to unite robust, industry-standard encryption procedures with an intuitive, user-centred design, thereby making strong security more attainable for a broader audience.

2.7.2 Research Gap

The comprehensive literature review identifies several significant gaps in existing file encryption tools, which collectively motivate the development of the project as an innovative and needed solution. The primary research gaps that this project aims to address include:

i. Accessibility and Usability for Non-Technical Users

The intricate workflows, complex interfaces, and jargon-laden documentation of many existing encryption technologies can significantly discourage non-technical individuals from adopting and correctly using secure practices.

For everyday users such as individuals, freelancers, or employees in small businesses, those who might need encryption for personal data, client confidentiality, or small-scale company use, this inaccessibility poses a substantial challenge and a security risk. There is a clear need for tools that abstract cryptographic complexity behind intuitive UIs (Norman, 2013).

ii. Simplified and Secure Key Management Challenges

Effective and secure key management is fundamental to proper encryption and overall data security. However, current solutions frequently

burden users with the complex tasks of generating, saving, retrieving, and backing up keys, which dramatically increases the possibility of human error (e.g., lost keys, weak key choices, insecure storage) and thereby jeopardizes data security (Blaze, n.d.). User-friendly key management systems that streamline these procedures, provide clear guidance, and integrate secure storage options are conspicuously needed but often lacking in tools aimed at general users.

iii. Consistent Cross-Platform Compatibility and Performance

The ability of many existing solutions, such as AxCrypt (historically limited Linux support) and even some aspects of VeraCrypt (complexity can be a barrier regardless of platform), to seamlessly handle numerous operating systems (specifically Windows and Linux for LockMe's scope) is often limited or comes with caveats. For instance, VeraCrypt's powerful features might be offset by its complicated interface, making it difficult to use effectively even if it is cross-platform, while tools like AxCrypt may not fully support all desired operating systems or may have performance variations. For an encryption tool to be genuinely effective and broadly adopted, it must be able to accommodate people working in diverse computing environments without compromising performance or usability.

iv. The Digital Divide and Security Literacy

A broader societal gap exists concerning digital literacy and cybersecurity awareness (Society, 2021). Many individuals lack a

fundamental understanding of online threats and the importance of protective measures like encryption. While LockMe cannot solve this alone, tools designed with extreme ease of use can help lower the barrier to adopting better security practices, indirectly contributing to bridging this gap for its users.

By systematically addressing these research gaps, LockMe hopes to offer a secure, user-friendly, and accessible encryption system. This system will empower users with varying degrees of technological proficiency to effectively safeguard their confidential information. The development of this novel application is predicated on the synthesis of insights from existing literature, an understanding of current tool limitations, and the identification of these pressing research needs

2.8 Chapter Summary

2.8.1 Justification for the Project

The increasing and undeniable need for safe, effective, and readily accessible methods to safeguard confidential information in a society that is becoming ever more digitally interconnected and vulnerable justifies this endeavour. Protecting important data is more critical than ever before, as cyber dangers—including sophisticated ransomware attacks, unauthorized illegal access, persistent data breaches, and identity theft—are constantly changing and growing in scale (Jurgens & Dal Cin, 2025).

The usability requirements of non-technical users, as well as those working in diverse, multi-platform environments, are frequently not adequately met by current encryption technologies, notwithstanding their theoretical

effectiveness in terms of security strength. This significant disparity presents a substantial opportunity for a product like LockMe to reach and empower a broader demographic of users by focusing on proactive rather than reactive security measures (Schneier, 2015).

1. Addressing Usability Challenges in Existing Solutions

Many of the encryption solutions available today were created with experienced users in mind and require technical knowledge to function properly. Wider use of these products is hampered by complicated procedures, unintuitive interfaces, and a dearth of useful feedback systems. Some solutions, for example, make the encryption process unduly difficult by lacking features like drag-and-drop file selection, real-time feedback, and progress indicators. By including an intuitive graphical user interface (GUI) that makes encryption and decryption chores easier, LockMe directly addresses these issues and enables users of all technical skill levels to effectively safeguard their files.

2. Providing Cross-Platform Compatibility

Modern users frequently transition between operating systems like Windows and Linux while working in multi-platform environments. Unfortunately, a lot of the encryption solutions that are now available are platform-specific, which limits their usefulness and causes difficulty. By providing smooth cross-platform interoperability, LockMe aims to close this gap and guarantee that customers may encrypt and decrypt files on any operating system. Individual users, small enterprises, and IT professionals

who need dependable encryption technologies in diverse situations are catered to by this flexibility.

3. Enhancing Accessibility to Advanced Encryption Techniques

Although algorithms such as AES (Advanced Encryption Standard) are widely acknowledged for their resilience and effectiveness, they are frequently integrated into instruments that necessitate an elevated level of technical proficiency to operate. For non-technical people who may gain the most from encryption tools, this puts up a hurdle. By incorporating AES into a user-friendly interface, LockMe democratises access to robust encryption techniques, allowing users to safeguard critical information without having to be familiar with the technical nuances of encryption.

4. Fulfilling the Need for Local File Encryption

Users are becoming more cautious of cloud-based encryption solutions that can reveal confidential information to outside parties due to increased worries about data privacy. By emphasising local file encryption and making sure that all encryption and decryption take place on the user's device, LockMe allays this worry. Because of its architecture, which puts user control and privacy first, LockMe is a desirable option for people and businesses looking for safe offline data security.

5. Supporting Common File Formats

By supporting a large number of popular file formats, such as documents, photos, and compressed files, LockMe guarantees adaptability. Its usefulness is increased by this function, which enables users to encrypt

and decode a variety of sensitive data for usage in business, the workplace, or personal settings. LockMe's development is further justified by the fact that it supports a variety of use cases by providing compatibility with numerous file formats.

6. Promoting Cybersecurity Awareness

LockMe's contribution to raising awareness of cybersecurity is yet another important defence. The project promotes safe data management practices by offering an easily navigable encryption tool. Adopting products like LockMe will help decrease vulnerabilities linked to unprotected sensitive information, which will contribute to a more secure digital ecosystem as cybersecurity dangers continue to rise.

7. Targeting Small Businesses and Non-Technical Users

Developers of encryption software frequently ignore small enterprises and non-technical customers, leaving them open to cyberattacks. By offering a cost-effective, user-friendly encryption solution that is customised to meet customer needs, LockMe seeks to close this gap. LockMe is a cost-effective substitute that provides strong protection without needless complexity, in contrast to enterprise-grade systems that could be excessively complicated or costly.

8. Supporting Key Management and Operational Security

Secure key management is necessary for effective encryption. Nevertheless, a lot of current programs either ignore this feature or use it in a way that makes sense to consumers. Users may handle their encryption

keys with confidence and safety because to LockMe's secure key generation, storage, and retrieval features. The project's necessity is further supported by its emphasis on operational security.

9. Limiting Scope to Practical Features

Secure key management is necessary for effective encryption. Nevertheless, a lot of current programs either ignore this feature or use it in a way that makes sense to consumers. Users may handle their encryption keys with confidence and safety because to LockMe's secure key generation, storage, and retrieval features. The project's necessity is further supported by its emphasis on operational security.

10. Filling Research Gaps in the Field

The literature review identified several shortcomings in the state of encryption technologies today, such as platform reliance, usability issues, and restricted access to strong encryption methods. By offering a solution that is both user-specific and compliant with industry standards, LockMe immediately fills these gaps. This initiative advances the field of secure file management by addressing these shortcomings.

In conclusion, this project represents an important and urgent endeavour that tackles critical, real-world issues in safe file encryption and decryption. LockMe's strategic emphasis on user accessibility, seamless cross-platform interoperability, and robust, industry-standard security (AES-256) has the potential to make a substantial positive impact on how individuals and small organisations safeguard their confidential digital data. The project's core

focus on privacy, dependability, and simplicity ensures that it can effectively meet the demands of its intended audience while also supporting the more general and vital objective of raising awareness and adoption of cybersecurity best practices in an increasingly dangerous digital landscape.

2.8.2 Connection to Project Goals

This chapter establishes the foundational knowledge essential for guiding the development of the "LockMe: Secure File Encryption and Decryption Desktop Application" by achieving the following objectives:

1. **Defining Key Concepts and Terminology Related to Cryptography and Data Security**

Important words and ideas, including encryption, decryption, symmetric-key algorithms, and asymmetric-key algorithms, are thoroughly explained in this chapter. This fundamental knowledge is essential for creating a reliable application that complies with accepted cryptography rules. By laying out these ideas, the project conforms to industry norms and guarantees the safe and efficient deployment of encryption techniques. The choice of algorithms like AES, which will form the foundation of the application's encryption and decryption procedures, is directly influenced by this understanding.

2. Reviewing Existing Solutions and Identifying Gaps in the Current Landscape

A thorough analysis of current encryption programs, such as VeraCrypt and AxCrypt, identifies both their advantages and disadvantages. For instance, VeraCrypt excels at offering sophisticated encryption features, but its complexity and emphasis on volume-level encryption rather than individual files limit its usefulness. In contrast, AxCrypt lacks Linux support and strong free-tier features but offers simplicity and cloud integration. The chapter highlights the necessity for an easily accessible, cross-platform solution like LockMe, which fills these gaps with an emphasis on file-level encryption, intuitive design, and smooth multi-OS compatibility, by pointing out these constraints.

3. Exploring Relevant Algorithms and Methodologies for Implementation

The technical foundation for LockMe is established by the investigation of algorithms like AES and RSA as well as techniques like hybrid encryption systems. Because it strikes the right balance between security and performance, AES is the best option for encrypting files in real time. The chapter also looks at important key management techniques, making sure the program has safe and user-friendly ways to generate, store, and retrieve encryption keys. These realisations are essential to fulfilling the project's objective of providing a safe and effective encryption system.

4. Analysing the State of the Art and Future Trends in the Field

Through an analysis of developments like lightweight cryptography, homomorphic encryption, and post-quantum cryptography, the chapter links LockMe's growth to the larger framework of changing cybersecurity procedures. Even though the project's main goal is to implement well-known standards like AES, LockMe's design is made to be flexible and scalable thanks to an awareness of emerging trends. The application can stay safe and relevant in the face of new dangers and technical developments because of this forward-looking viewpoint.

The general objectives of the LockMe project are directly in line with the knowledge acquired from this chapter. Understanding cryptographic concepts, along with current solutions and new developments, offers a strong basis for creating a desktop application that is:

- i. Secure: By leveraging industry-standard encryption algorithms and best practices in key management, LockMe ensures robust protection for sensitive files.
- ii. Efficient: Real-time encryption and decryption performance is improved, and resource consumption is reduced through the use of AES encryption and optimised procedures.
- iii. User-Friendly: A focus on intuitive design, drag-and-drop functionality, and clear feedback mechanisms ensures accessibility for users with varying levels of technical expertise.

This integration of theoretical understanding with real-world application guarantees that LockMe will successfully close the gaps in existing encryption technologies and accomplish its goal of offering a complete solution for protecting sensitive data.

CHAPTER 3

RESEARCH METHODOLOGY

3.1 Introduction

This chapter explores the thorough research methodology and techniques used to look into the creation of the "LockMe: Secure File Encryption and Decryption Desktop Application," a program intended to improve data security with strong encryption and intuitive features. It acts as a thorough manual for the complete research process, detailing the frameworks, techniques, and tactics applied to accurate and thorough data collecting and analysis. A well-structured study is necessary to guarantee the validity and dependability of research findings.

The chosen research methodology and how they fit with the main aims and objectives of this study are explained in the first section of this chapter. The project's goals inform the methodological choice, showing how several approaches offer insightful information on the creation, use, and importance of the LockMe program. The goal is to provide an encryption tool that is safe, effective, and usable by people with varying degrees of technical proficiency.

The methods used to gather pertinent data are also covered in detail in this chapter, including particular approaches like surveys, user feedback sessions, and prototype evaluations. Every technique is described in detail to make clear how it contributes to the overall project goals. This study makes sure that the research findings are closely related to the theoretical and practical foundations of data encryption and application design by looking at user needs, technological specifications, and current solutions.

This chapter also offers a thorough summary of the research design, and the methodical techniques used to assess and improve the LockMe application. It

guarantees that the research is carried out with ethical responsibility and scientific rigour with the goal of generating accurate and remarkable results. In the end, the study approach advances safe data management techniques, which is consistent with the more general objectives of enhancing cybersecurity and developing easily available encryption solutions.

3.2 Software Development Methodology

3.2.1 Chosen Methodology and Justification

The choice to implement the ADDIE model for the project is a strategic decision tailored to the specific needs and goals of the project. ADDIE stands out as a suitable approach due to its structured, systematic, and iterative framework, ensuring comprehensive development from conception to evaluation. This methodology aligns perfectly with the project's requirements, which demand a robust framework capable of meticulous planning, controlled execution, and thorough assessment to deliver a secure and user-friendly application.

The structured phases of ADDIE (Analysis, Design, Development, Implementation, and Evaluation) enable the project team to systematically address the complexities involved in creating an encryption application that is both safe and easy to use. In the Analysis phase, a deep understanding of user needs, existing encryption tool shortcomings (like lack of cross-platform compatibility and usability issues), and technical requirements for robust security (e.g., AES encryption) is established. The Design phase then meticulously plans the application's architecture, user interface, and functional specifications, ensuring that key features like drag-and-drop functionality and

reliable key management are prioritized from the outset.

During Development, the planned features are built with continuous internal testing. The Implementation phase focuses on deploying the application and making it accessible to users across Windows and Linux. Finally, the Evaluation phase allows for continuous feedback and assessment, ensuring the final product meets its objectives of user-friendliness, security, and cross-platform compatibility. This systematic, feedback-driven approach helps to ensure the final product satisfies the needs of its target audience, which includes individuals, small businesses, and professionals from a variety of industries.

The sequential yet iterative nature of ADDIE promotes thorough documentation and quality control at each stage, which is crucial for a security-focused application like LockMe. This disciplined approach facilitates clear communication among team members with different specialities, including user interface design, software engineering, and cryptography. By progressing through distinct phases, ADDIE ensures that potential problems are identified and addressed early, leading to the effective development of a safe and intuitive encryption solution that consistently meets user expectations and project goals.

3.2.2 Step-by-step Explanation of Activities Done in Each Phase of the Chosen Methodology

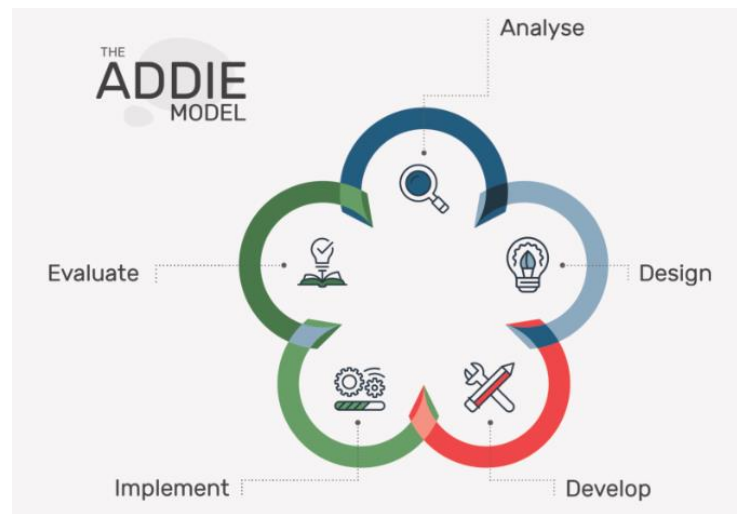


Figure 3.0.1 Phases of the ADDIE model

1. Analyse Phase

The Analyse phase begins with identifying and understanding the specific needs and challenges faced by target users, including individuals and small business owners concerned about data privacy and file security. During this phase, the project team gathered requirements by studying existing encryption tools and collecting feedback from stakeholders to clarify essential features such as AES-256 encryption, user-friendly interfaces, and cross-platform compatibility. The problem statement was refined, and clear objectives were established to guide the project. This phase ensured that development efforts would directly address real-world user needs and security expectations.

2. Design Phase

In the Design phase, the team created detailed plans and blueprints for both the system's technical architecture and its user interface. Wireframes and prototypes were developed to visualize the user experience, focusing on intuitive features like drag-and-drop file selection and progress indicators to enhance usability. Concurrently, the technical framework was planned, selecting Next.js for frontend development, integrating AES-256 encryption mechanisms, and deciding on secure key management strategies. The design stage also included defining module interactions, data flow, and security protocols, ensuring a solid foundation for development.

3. Develop Phase

During the Develop phase, the team translated the design specifications into functional code. The Next.js framework was used to build responsive user interface components, while WebAssembly modules wrapped cryptographic functions to deliver hardware-accelerated AES-256 encryption and decryption. The development process focused on creating modular, maintainable code, integrating frontend components with cryptographic backends, and implementing error handling and progress reporting. Regular code reviews and unit testing were conducted to ensure quality and functionality. This phase emphasized building a stable and secure application aligned with the design goals.

4. Implement Phase

The Implement phase involved deploying LockMe on target platforms (Windows and Linux) and conducting real-world user testing. The team packaged the application using Electron, enabling seamless desktop installation and execution. Users were invited to perform typical encryption and decryption tasks, allowing the team to observe usability and identify any operational issues. This phase ensured that the application was not only functional in a controlled environment but also effective and reliable under actual user conditions. Feedback collected during implementation informed refinements and bug fixes.

5. Evaluate Phase

In this phase, comprehensive testing and assessment were performed to validate LockMe's performance, security, and usability. Functional testing confirmed that encryption and decryption operated correctly across various file types and sizes. Performance benchmarks measured processing times, CPU and memory usage, confirming efficient hardware utilization. Security evaluations included static code analysis and penetration testing to identify vulnerabilities and verify cryptographic integrity. Usability was assessed through structured surveys using the System Usability Scale (SUS), with feedback indicating high user satisfaction and ease of use. This phase concluded with documentation of results and recommendations for future improvements, closing the ADDIE cycle and supporting continuous enhancement.

3.3 Research Methodology

3.3.1 Chosen Research Methodology and Justification

For this project, quantitative research is the primary method for gathering data on user preferences, behaviours and expectations. This approach enabled the collection of structured, measurable insights from a broad demography, ensuring that findings were both comprehensive and objective.

Surveys were distributed via Google Forms to individuals, small-business owners, and IT professionals. The questionnaire covered topics such as encryption frequency, preferred file types, and respondents' technical proficiency. Specific items asked the participants to rate the importance of features like AES encryption, cross-platform compatibility, drag-and-drop functionality and real-time progress indicators, as well as their satisfaction with existing tools currently and to give suggestions for improvement.

All survey responses were analysed to identify key trends, such as the most requested features, common technical difficulties and the overall demand for a more user-friendly encryption solution. These results guided the prioritisation of features in LockMe, helping to ensure that the application directly addresses the needs highlighted by potential users.

By relying solely on quantitative survey data, the development of LockMe remains firmly data-driven, resulting in a secure, intuitive and practical encryption tool tailored to its target audience.

3.3.2 Questionnaire Design and Samples

LockMe: Secure File Encryption and Decryption Desktop Application

Help Shape the Future of Secure File Encryption!

We invite you to participate in a brief survey to help us develop a user-friendly and secure file encryption tool called "LockMe." Your feedback will be invaluable in designing a tool that meets your specific needs and preferences.

By taking part in this survey, you will contribute to the development of a tool that can protect your sensitive data from unauthorized access. Your participation is voluntary, and your responses will be kept strictly confidential.

Thank you for your time and support!

Not shared

* Indicates required question

Figure 3.0.2 Survey form display

What is your age? *

☐ 18-24

☐ 25-34

☐ 35-44

☐ 45-54

☐ 55+

Figure 3.0.3 Survey: Question 1

What is your gender? *

☐ Male

☐ Female

☐ Prefer not to say

☐ Other: _____

Figure 3.0.4 Survey: Question 2

What is your occupation? *

☐ Student

☐ Employed (full-time)

☐ Employed (part-time)

☐ Self-employed

☐ Retired

☐ Unemployed

☐ Other: _____

Figure 3.0.5 Survey: Question 3

What is your level of technical expertise? *

☐ Beginner

☐ Intermediate

☐ Advanced

Figure 3.0.6 Survey: Question 4

What operating system (OS) do you use? *

☐ Windows

☐ Linux

☐ MacOS

☐ Other: _____

Figure 3.0.7 Survey: Question 5

Do you currently use any file encryption tools or methods? *

☐ Yes

☐ No

Figure 3.0.8 Survey: Question 6

If you answered yes, which ones?

☐ BitLocker

☐ VeraCrypt

☐ 7-Zip

☐ WinRAR

☐ GPG

☐ Other: _____

Figure 3.0.9 Survey: Question 7

What are the primary reasons for using or not using file encryption? *

- ☐ To protect sensitive data
- ☐ To comply with regulations
- ☐ To back up data
- ☐ It's too complex
- ☐ It's too time-consuming
- ☐ I don't know how to use it
- ☐ Other: _____

Figure 3.0.10 Survey: Question 8

What are the major challenges you face when using file encryption tools? *

- ☐ Complexity of the software
- ☐ Slow performance
- ☐ Difficulty in managing keys
- ☐ Lack of user-friendly interface
- ☐ Other: _____

Figure 3.0.11 Survey: Question 9

How do you perceive the need for a user-friendly file encryption tool? *

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

Figure 3.0.12 Survey: Question 10

What features would you like to see in such a tool? *

- ☐ Easy-to-use interface
- ☐ Strong encryption algorithms
- ☐ Cross-platform compatibility
- ☐ Password protection
- ☐ Cloud storage integration
- ☐ Automatic key management
- ☐ Other: _____

Figure 3.0.13 Survey: Question 11

How important are factors like security, ease of use, and cross-platform compatibility to you? *

1 2 3 4 5

Not at all important ☐ ☐ ☐ ☐ ☐ Very important

Figure 3.0.14 Survey: Question 12

Would you be willing to try a new file encryption tool if it offered significant benefits? *

- ☐ Yes
- ☐ No
- ☐ Maybe

Figure 3.0.15 Survey: Question 13

Are there any specific concerns or requirements you have regarding file encryption?

Your answer _____

Figure 3.0.16 Survey: Question 14

What other features or improvements would you suggest for LockMe?

Your answer _____

Figure 3.0.17 Survey: Question 15

3.3.3 Analysis of Questionnaire Data

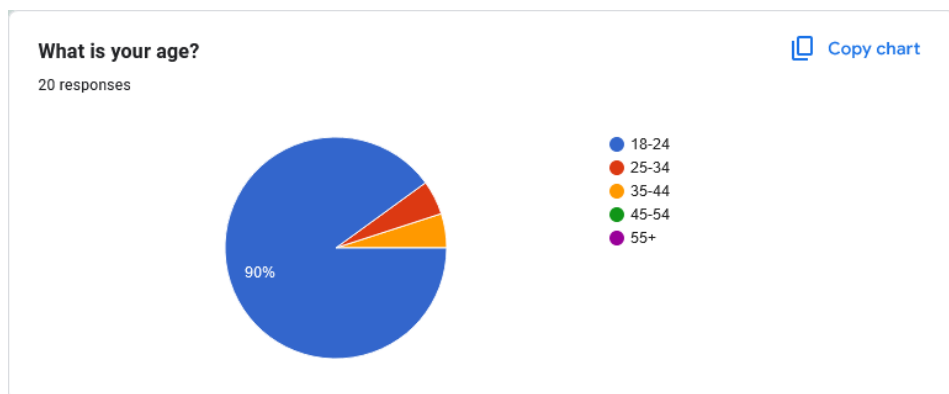


Figure 3.0.18 Analysis of respondent's age distribution

Participants were asked about their age group in the question. According to the chart, of the twenty respondents, 90% (18 respondents) were between the ages of 18 and 24. Smaller percentages of respondents were in other age groups. This implies that young adults, most likely students, or people just starting their professional careers, make up the majority of responders. This age group's dominance indicates that the audience is tech-savvy and may be somewhat familiar with encryption tools, which makes them a perfect target

for the introduction of user-friendly encryption applications.

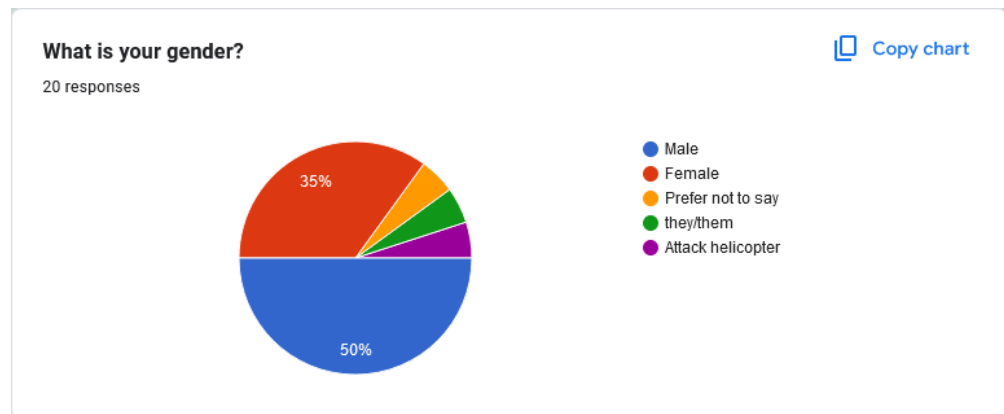


Figure 3.0.19 Analysis of respondent's gender distribution

Participants were asked about their gender identity in the question. 10% (10 respondents) of the 20 respondents identified as male, 35% (7 respondents) as female, and the remaining 15% (3 respondents) as non-binary or as preferring not to specify their gender, according to the chart. This suggests a somewhat male majority among a comparatively diverse group of respondents. Given this diversity, it is recommended that the application be created with inclusivity in mind, making sure that all users, regardless of gender, can utilise its features.

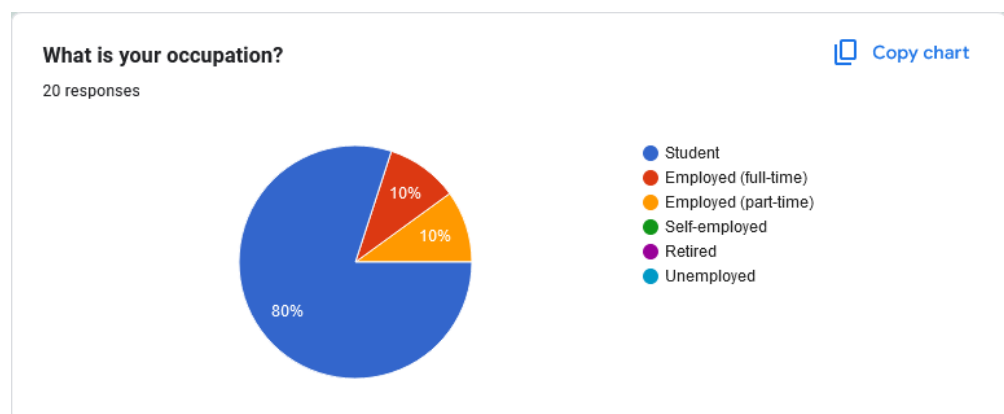


Figure 3.0.20 Analysis of respondent's occupation

Participants were asked about their main line of work. The graph shows that 16 respondents, or 80% of the sample, were students, and 2 respondents, or 10% of the sample, were full-time or part-time employees. This indicates that the vast majority of respondents are probably still in school, which could have an impact on their degree of technical proficiency and the kinds of encryption tools they are accustomed to. Given that students may not have much experience with sophisticated or complicated software, a straightforward and user-friendly encryption tool would be more appealing to this group.

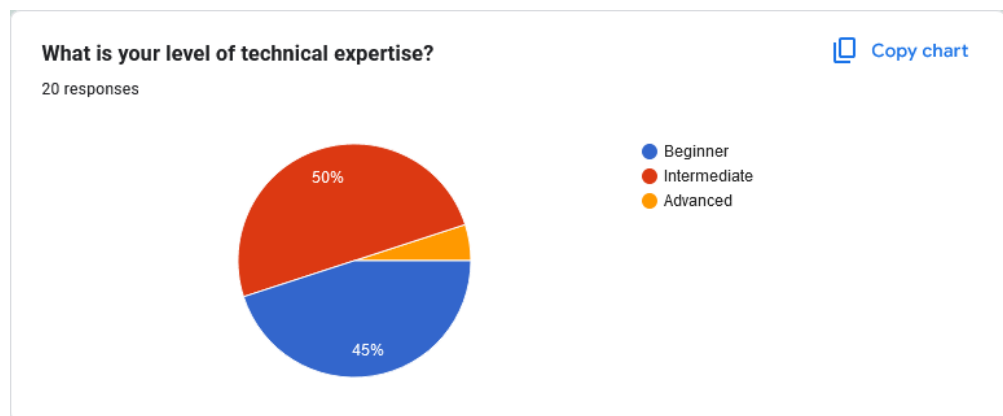


Figure 3.0.21 Analysis of respondents' technical expertise

Participants were asked to score their degree of technical proficiency. Just 5% (1 respondent) identified as advanced users, 50% (10 respondents) as intermediate users, and 45% (9 respondents) as beginners, according to the chart. This suggests that very few respondents are highly skilled, with the majority having a moderate to basic level of technical expertise. According to these results, the target audience would be most affected by a file encryption tool designed for novice and intermediate users, complete with easy-to-follow instructions and an intuitive user interface.

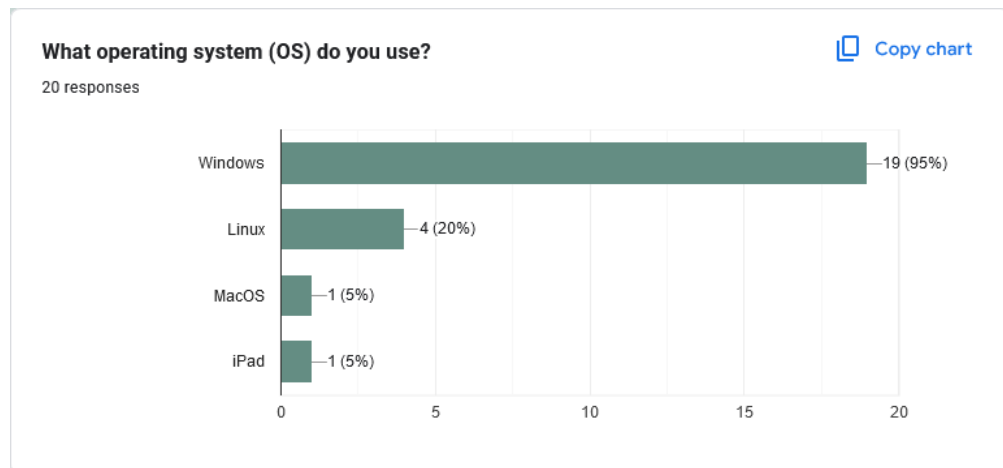


Figure 3.0.22 Analysis of respondents' usage of operating systems (OS)

Participants were asked which operating system they use most frequently. 95% (19 respondents) use Windows, 20% (4 respondents) use Linux, and only 5% (1 respondent each) use MacOS and iPad, according to the chart. This overwhelming preference for Windows implies that compatibility with this platform should be given top priority during the encryption tool's development. The noteworthy application of Linux, however, points to a chance to support developers or more experienced users who might favour open-source solutions.

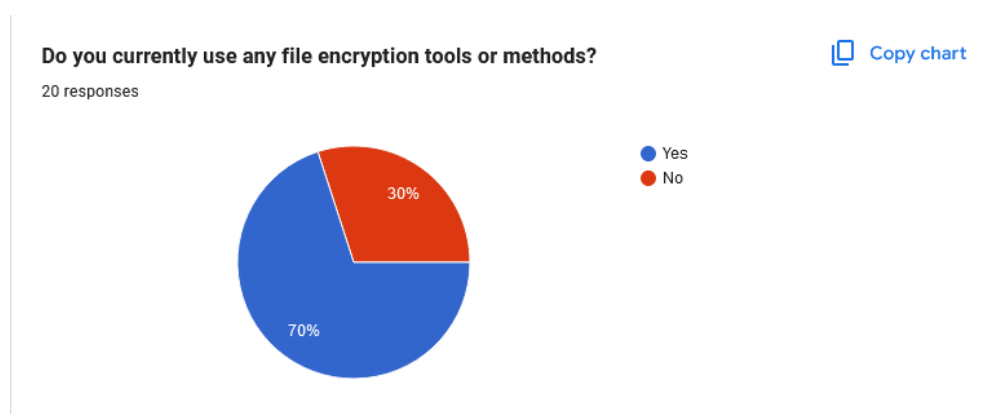


Figure 3.0.23 Analysis of respondents' usage of any file encryption tools or methods

Participants were asked if they currently use any tools or techniques for file encryption. According to the chart, 70% of respondents (14 respondents)

selected "Yes," whereas 30% (6 respondents) selected "No." This implies that most respondents are aware of file encryption and use it to some degree. The sizeable minority of people who do not use encryption tools, however, suggests that more people need to be taught about the advantages of file encryption, especially in terms of protecting sensitive data.

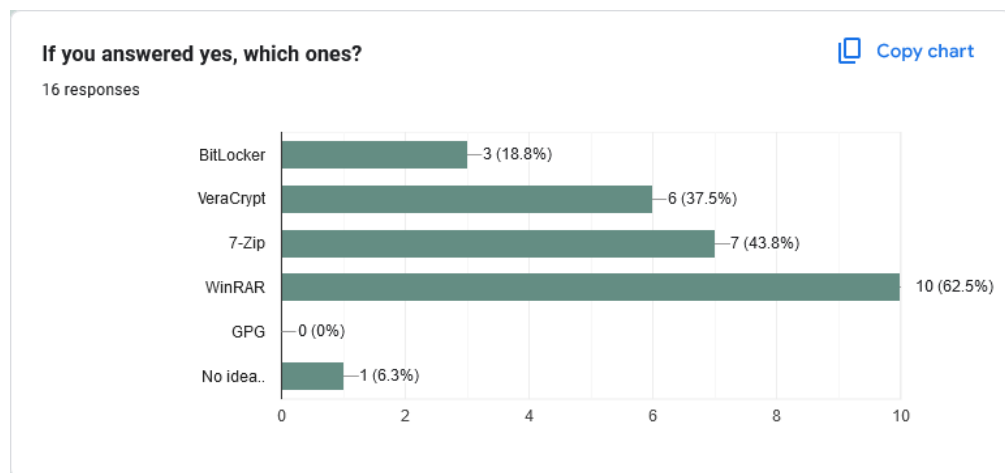


Figure 3.0.24 Analysis of respondents' encryption tool preferences

Participants were asked, if applicable, which file encryption programs they use. WinRAR is the most widely used tool, according to the chart, with 62.5% of respondents (10 users), followed by 7-Zip (43.8%) and VeraCrypt (37.5%). 18.8% (3 respondents) said they use BitLocker, whereas 6.3% (1 respondent) said they are not sure which tools they use. This suggests that for simple encryption tasks, users favour well-known, accessible tools. Nonetheless, there might be a lack of knowledge or uptake of more powerful encryption programs like VeraCrypt, which presents a chance to encourage more stringent security procedures.

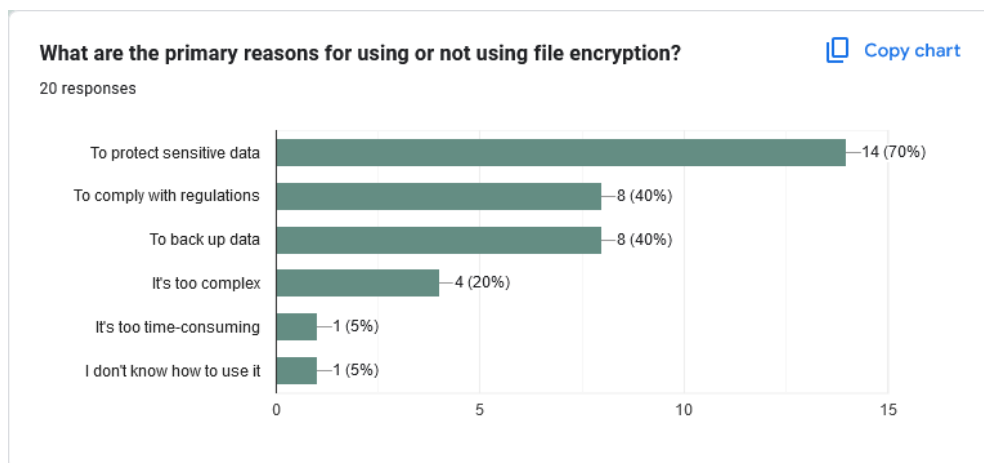


Figure 3.0.25 Analysis of respondents' reasons for using or not using encryption tools

Participants were asked why they chose to use or not use file encryption. According to the graph, 70% of respondents (14 respondents) use encryption to safeguard sensitive data, while 40% (8 respondents) do so for regulatory compliance or data backup. However, 5% (1 respondent each) felt that encryption was too time-consuming or that they were not familiar with its use, while 20% (4 respondents) pointed to complexity as a barrier. These answers underscore the importance of streamlining the encryption process to promote broader adoption among reluctant users, while also highlighting the dual motivations of security and compliance for encryption users.

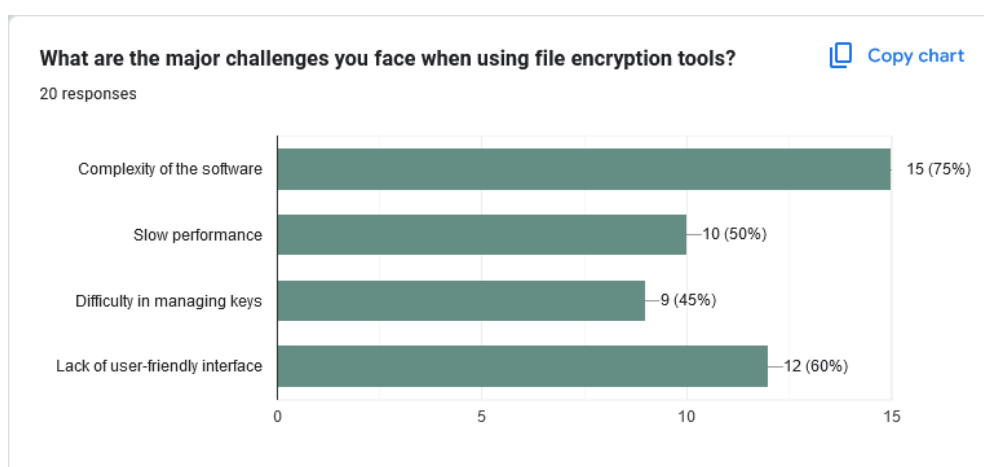


Figure 3.0.26 Analysis of respondents' challenges when using encryption tools

Participants were questioned regarding the difficulties they encounter when utilising file encryption software. According to the chart, software complexity was mentioned as a major problem by 75% (15 respondents), followed by slow performance (50%, 10 respondents), a lack of user-friendly interfaces (60%, 12 respondents), and trouble managing keys (45%, 9 respondents). In order to serve a wider audience, this data emphasises the need for a tool that strikes a balance between strong encryption and usability, tackling both technical and usability issues.

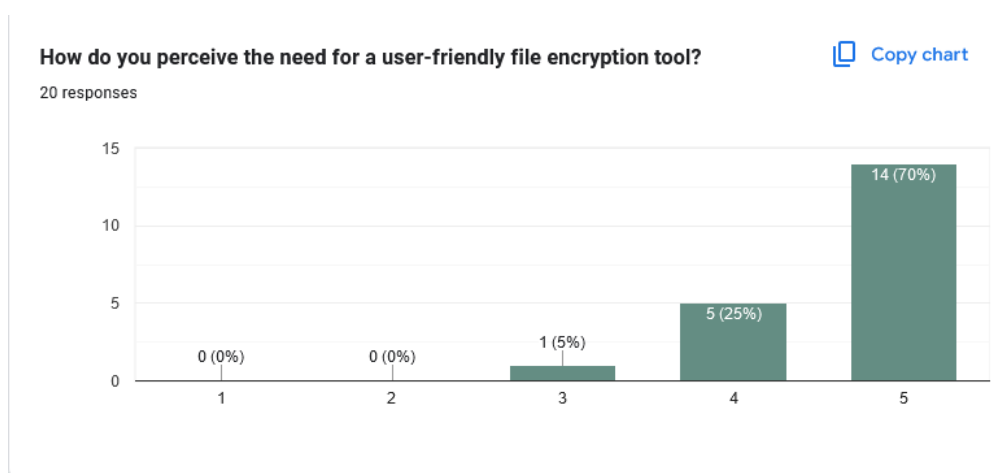


Figure 3.0.27 Analysis of respondents perceived need for a user-friendly tool

Participants were asked how strongly they believed that a user-friendly file encryption tool was necessary. 70% (14 respondents) gave the need a "5" rating (strongly agree), while 25% (5 respondents) gave it a "4" rating, according to the chart. This resounding consensus shows how much demand there is for an easily navigable encryption tool that minimises complexity while guaranteeing efficient user data protection. According to the survey results, creating such a tool is exactly in line with user expectations and market demand.

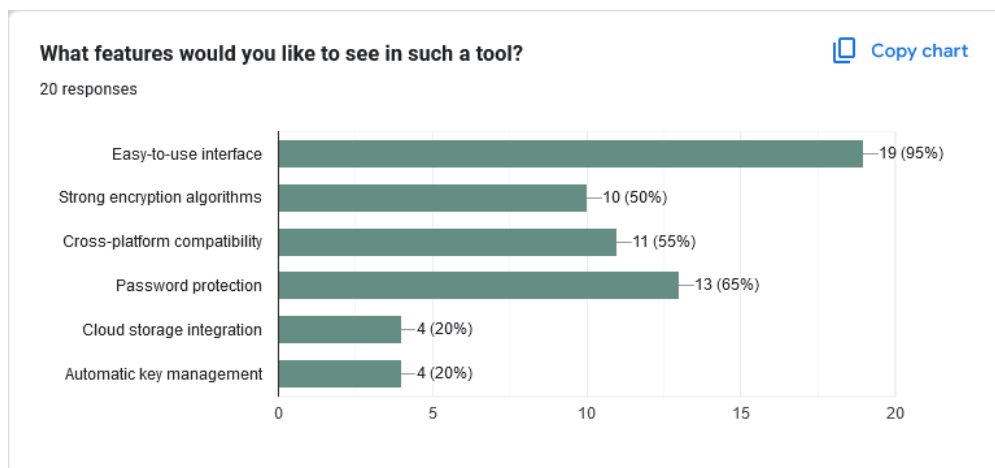


Figure 3.0.28 Analysis of respondents desired features in an encryption tool

Participants were asked what characteristics they would like to see in a file encryption tool. According to the chart, 65% (13 respondents) favoured password protection, while 95% (19 respondents) valued an intuitive user interface. Strong encryption algorithms were chosen by 50% (10 respondents) and cross-platform compatibility by 55% (11 respondents). Just 20% (4 respondents each) said they were interested in automatic key management and cloud storage integration. These findings indicate that users' top priorities are robust security features and ease of use, indicating a focus on core functionalities and simplicity. Another important consideration is cross-platform compatibility, which highlights the necessity of making sure the tool works with various operating systems.

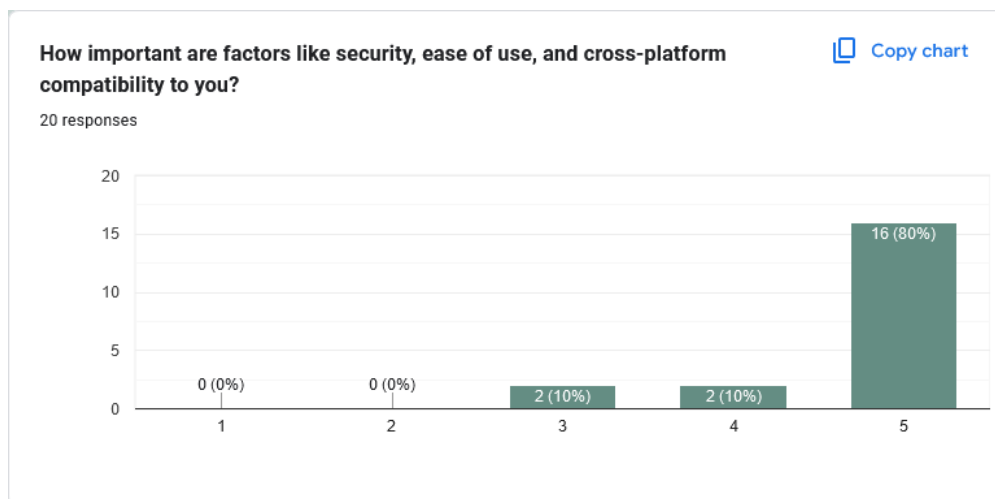


Figure 3.0.29 Analysis of respondents' importance of security, ease of use, and cross-platform compatibility

Participants were asked to rank the significance of elements like cross-platform compatibility, security, and ease of use. 80% (16 respondents) gave these factors a "5" rating (very important), according to the chart, while 10% (2 respondents each) gave them a "4" or "3." Not a single respondent thought these factors were less significant. This broad agreement highlights how important these characteristics are in influencing users' decisions and implies that, in order to satisfy user expectations and guarantee widespread adoption, they should be given top priority during the development process.

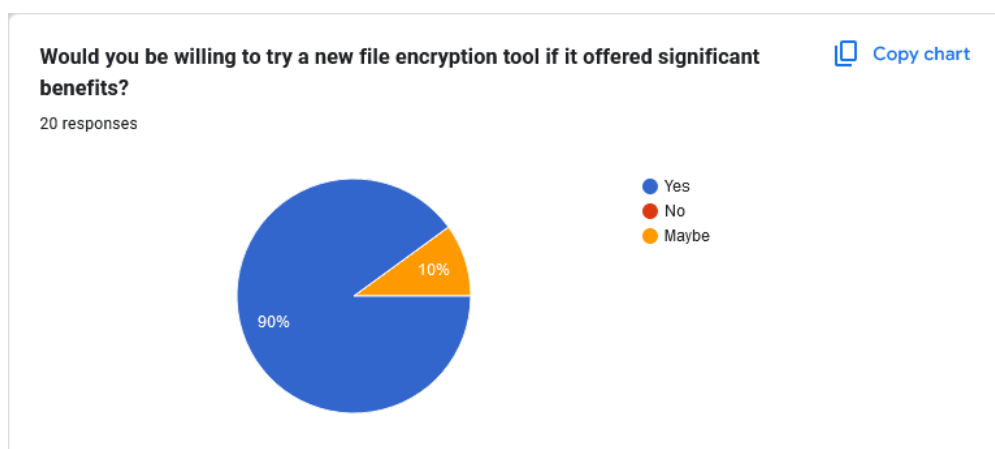


Figure 3.0.30 Analysis of respondents' willingness to try a new encryption tool

Participants were asked if they would be open to trying a new file encryption program that has a lot of advantages. 90% (18 respondents) said "Yes," according to the chart, while 10% (2 respondents) said "Maybe." The idea was not categorically rejected by any of the participants. This suggests an ardent desire to investigate new tools, so long as they provide observable benefits over current solutions, like improved usability, enhanced security, or unique features.



Figure 3.0.31 Analysis of respondents' concerns and requirements regarding file encryption

Participants were questioned open-endedly about any particular needs or worries they had about file encryption. The necessity of data privacy, app security, and an intuitive user interface to make complicated encryption tasks easier were common themes. While some respondents expressed concern about encryption tools being restricted by paywalls, others cited speed and accessibility as crucial factors. These answers emphasise the necessity of developing a simple, low-cost tool that offers users robust security and privacy

without needless obstacles or complications.

What other features or improvements would you suggest for LockMe?

15 responses

- Usage in multiple platform
- nothing just did your best!
- For it to be really secure since it is regarding encrypted files.
- For now no suggestions
- including a feature to batch encrypt or decrypt multiple files simultaneously
- make user friendly interface
- Easy to use, Smooth UI
- make it colourful
- friendly colors and attractive interface

Figure 3.0.32 Analysis of respondents' suggested features and improvements for LockMe

Participants were asked for ideas on how to make the "LockMe" tool better. A number of respondents emphasised the significance of an interface that is both visually appealing and easy to use; some suggested using "friendly colours" and a "smooth UI." Additional recommendations included making the tool cross-platform, improving security for encrypted files, and including batch encryption and decryption features. According to these responses, there is a high need for a tool that satisfies more complex requirements like managing several files at once while striking a balance between usefulness and aesthetics.

3.4 Proposed System Design

Table 0.1 Requirements of the proposed system

No.	Requirement Description	Type (Functional / Non-functional / Usability)	Stakeholder
1	The application should allow users to encrypt and decrypt files directly through an intuitive graphical user interface (GUI).	Functional	End-users (all)
2	The system must support AES-256 encryption as the primary encryption standard to ensure robust security for all files.	Functional	End-users (all)
3	Users should be able to manage their encryption keys securely within the application, including generating, saving, and retrieving keys.	Functional	End-users (all)
4	The application should be cross-platform, fully functional on both Windows and Linux operating systems.	Non-functional	End-users (all)
5	The system must process encryption and decryption tasks in a reasonable amount of time to ensure efficiency for large file sizes.	Non-functional	End-users (all)
6	The user interface should be designed for simplicity and clarity, with features like drag-and-drop functionality and real-time feedback during file operations.	Usability	Non-technical users
7	Error messages must be clear and descriptive, helping users understand and resolve any issues that occur during encryption or decryption.	Usability	Non-technical users
8	The system should provide support for common file types such as documents, images, and compressed archives to ensure versatility.	Functional	End-users (all)
9	The application must include a feature for verifying file integrity to ensure the data remains unchanged after encryption and decryption.	Functional	Advanced users/IT staff
10	Navigation within the application should be clear and well-organized, enabling users to quickly locate encryption, decryption, and key management functionalities.	Usability	Non-technical users

3.4.1 UML Modelling of the Proposed System

Use Case Diagram

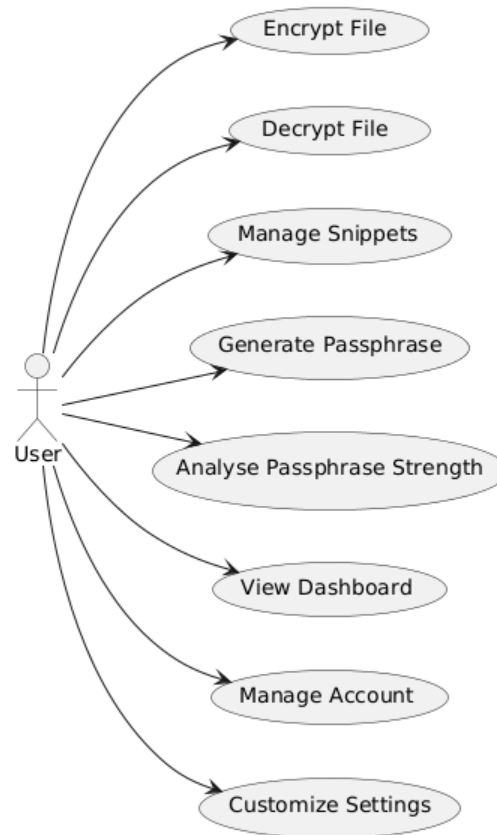


Figure 3.0.33 Use Case Diagram

The diagram places a single actor, the User, on the left and lists eight use-cases on the right. Starting at the top, the user can choose Encrypt File to secure a file with AES-256-GCM or Decrypt File to restore a previously encrypted file. The next option, Manage Snippets, lets the user add, edit, and delete stored code snippets. Two AI features follow: Generate Passphrase produces a strong random passphrase, while Analyse Passphrase Strength evaluates an existing passphrase and provides feedback. The user can also View Dashboard for an overview of recent activity, Manage Account to update profile details or change the password, and Customise Settings to adjust preferences such as theme, language, or default encryption behaviour. Each arrow from the User to a use-case shows that the action is started directly by

the user in the LockMe interface.

Package Diagram

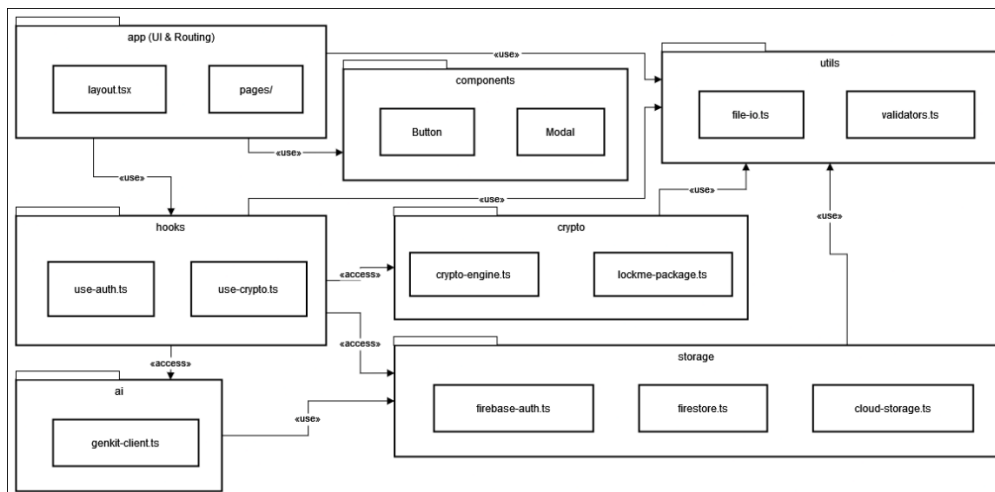


Figure 3.0.34 Package Diagram

Each folder represents a top-level package in LockMe. App (UI & Routing) contains the Next.js layouts and page files and imports reusable components, shared hooks, and generic utils. The hooks layer acts as a bridge between the UI and lower-level services, “accessing” the cryptography engine, Firebase storage wrappers, and the Genkit/Gemini AI client while also re-using helper utilities. Core service packages, such as crypto, storage, and ai depend only on utils, keeping them UI-agnostic and easy to unit-test. By showing these compile-time links, the diagram highlights a modular structure in which the user interface is cleanly separated from encryption logic, persistent storage, and AI features.

Class Diagram

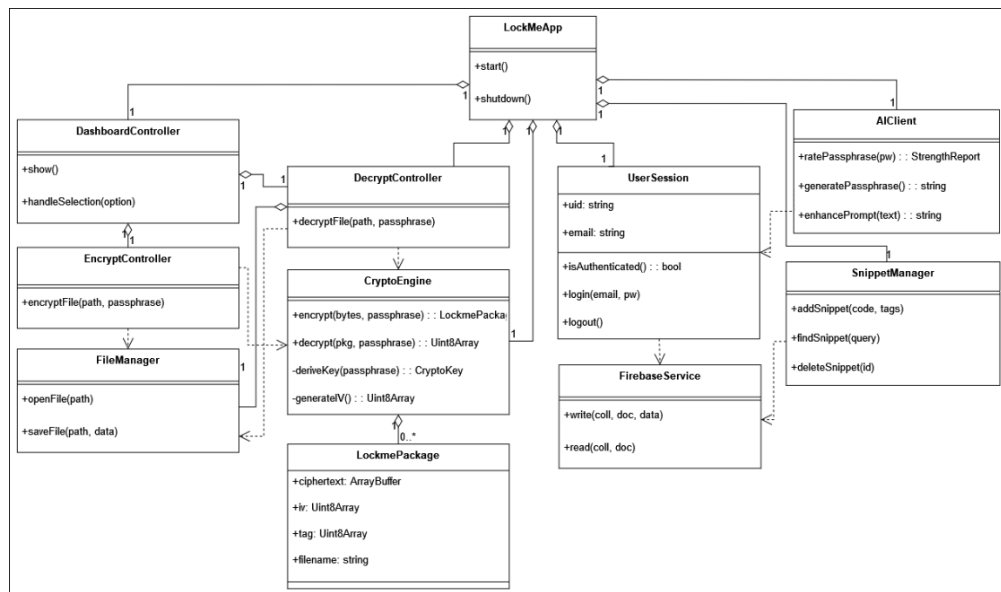


Figure 3.0.35 Class Diagram

LockMe is the orchestration hub, holding references to every major service. **UserSession** authenticates through **FirebaseService** and exposes login/logout status. The GUI funnels actions through **DashboardController**, which delegates file-level tasks to **EncryptController** and **DecryptController**. Both controllers rely on **FileManager** for disk I/O and **CryptoEngine** for AES-256-GCM operations that produce or consume **LockmePackage** objects. **SnippetManager** stores code fragments in Firestore via the same **FirebaseService**, while **AIClient** sends passphrases or prompts to Gemini (through Genkit) and personalises advice for the logged-in user. This separation keeps encryption logic, storage, AI features, and UI flow loosely coupled yet clearly mapped, making the system easier to test, maintain, and extend.

Sequence Diagram

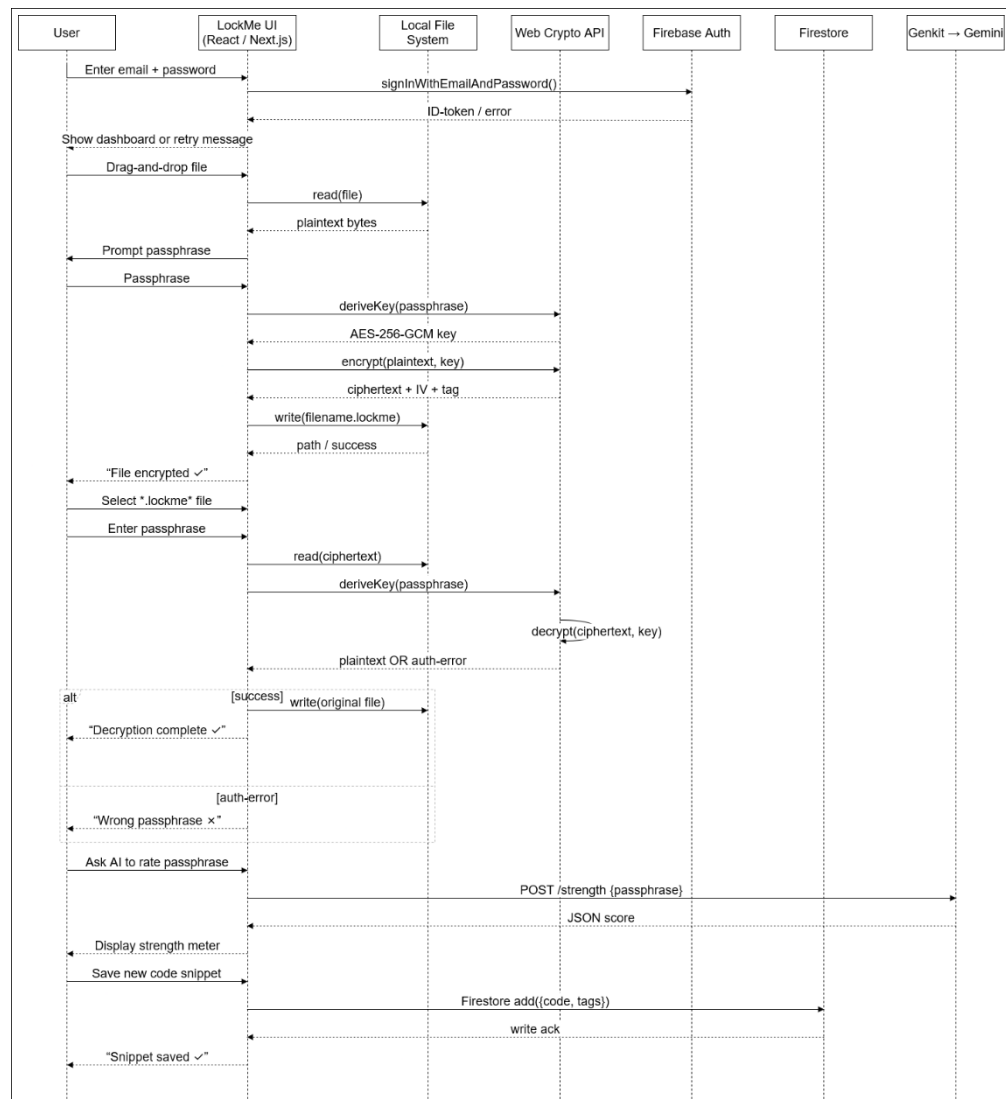


Figure 3.0.36 Sequence Diagram

The sequence begins with the Login phase: the user submits credentials to the React interface, which forwards them to Firebase Auth; a successful token returns the user to the dashboard. For encryption, the user drags a file into the app, the UI reads it from the local file system, prompts for a passphrase, and calls the Web Crypto API to derive an AES-256-GCM key and encrypt the bytes. The resulting ciphertext, IV, and tag are written back as a `.lockme` file, and the user receives instant feedback. The decryption path mirrors these steps in reverse, with an authentication-tag check that determines a success or error

message. When the user invokes the AI security toolkit (for example, to gauge passphrase strength), the UI sends the request through Genkit to Google Gemini and displays the returned score. Finally, the snippet manager allows users to store or retrieve code fragments via Firestore, confirming writes before returning control to the user. Each interaction shows how LockMe's components, which are the UI, local storage, browser cryptography, Firebase services, and external AI co-operate to deliver seamless, client-side file security.

State Machine Diagram

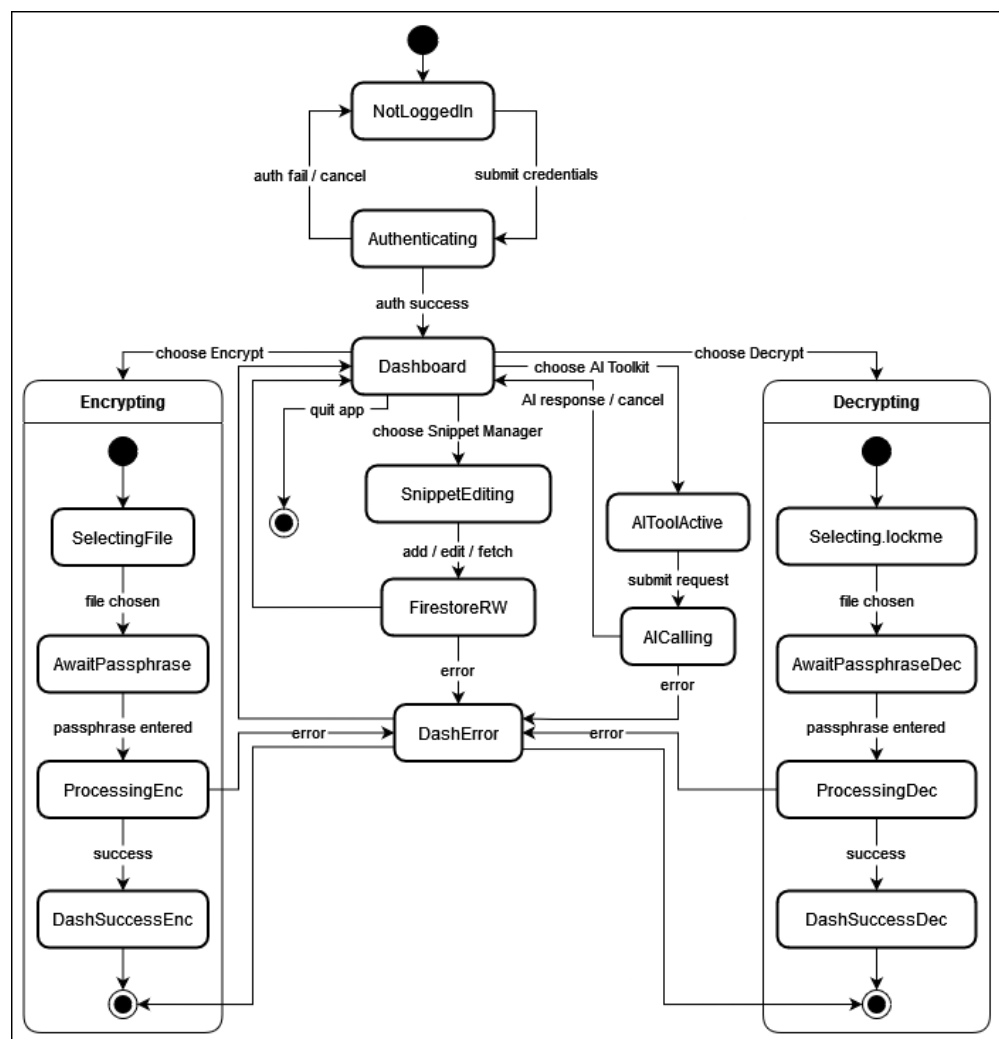


Figure 3.0.37 State Machine Diagram

The state diagram shows LockMe's life-cycle in plain steps:

1. **Launch & Login:** The app opens in the Not Logged In state. When the user submits credentials, it moves to Authenticating; success leads to the Dashboard, failure returns to the start.
2. **Idle Home (Dashboard):** From the dashboard the user can choose one of four tasks: Encrypt, Decrypt, open the AI Toolkit, or manage Snippets. Quitting exits the app.
3. **Encrypt / Decrypt:** Each task follows the same pattern: pick a file, enter a passphrase, run the cryptographic process, then show either Success or Error before returning to the dashboard.
4. **AI Toolkit:** The user enters a prompt; the app calls Gemini. When a response (or error) arrives, control jumps back to the dashboard.
5. **Snippet Manager:** Reads or writes to Firestore, then also returns to the dashboard.

At every turn the application always funnels the user back to the dashboard, keeping the workflow simple and consistent.

Activity Diagram

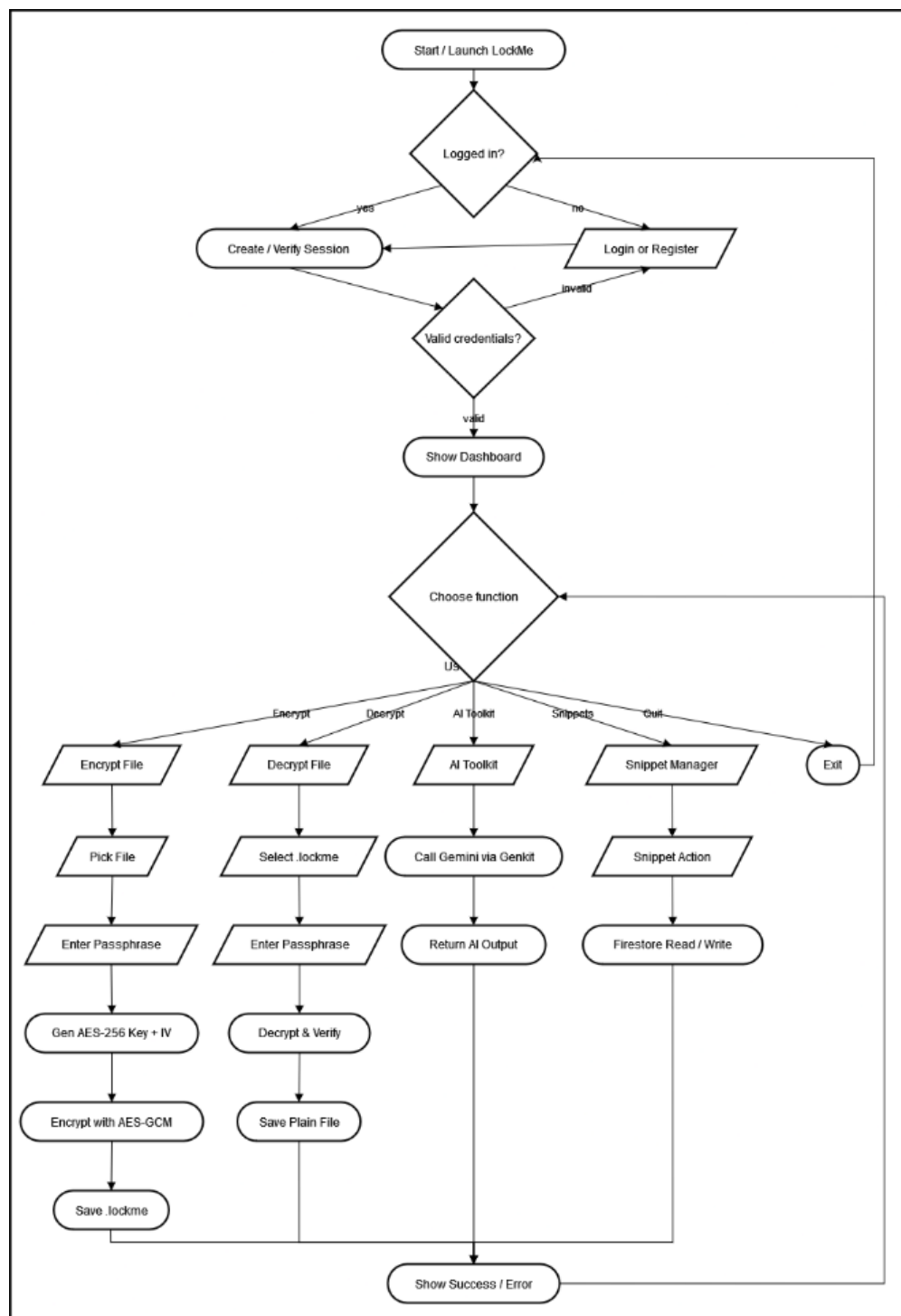


Figure 3.0.38 Activity Diagram

The activity diagram outlines LockMe's main user paths. When the app launches, the user first passes a login / registration gate. A valid session opens the dashboard, where four actions are available:

- Encrypt – the user picks a file, enters a passphrase, and the browser generates an AES-256 key and IV, encrypts the file with AES-GCM, and saves the resulting `.lockme` package.
- Decrypt – the user selects a `.lockme` file, supplies the passphrase, and the client verifies the authentication tag before recovering and saving the original file.
- AI Toolkit – a request is routed through Genkit to Google Gemini; the returned advice (e.g., passphrase strength or recovery prompt) is shown to the user.
- Snippet Manager – the user adds, searches, or retrieves code snippets, with all reads and writes handled in Firestore.

Each branch ends with a unified feedback step that shows success or error messages, then returns the user to the dashboard until they choose to exit. All cryptographic work and AI calls occur client-side, so no sensitive data leave the device.

3.4.2 Hardware Design and Block Diagram

This section describes the system architecture and hardware needed to support the project. As a stand-alone desktop program, the application needs a stable but accessible hardware environment to function at its best. Targeting users with various levels of technical proficiency, this design places an emphasis on compatibility with widely available desktop and laptop systems.

Hardware Requirements

Table 0.2 Hardware requirements for the proposed system

Category	Specifications
Processor (CPU)	Minimum: Intel Core i3 or AMD equivalent. Recommended: Intel Core i5 or higher for faster encryption/decryption processing.
Memory (RAM)	Minimum: 4GB to handle small to medium-sized files. Recommended: 8GB or higher for processing large files efficiently.
Storage	Minimum: 128MB for application installation. Recommended: Additional space for saving encrypted/decrypted files, depending on user's needs.
Operating System	Supported Platforms: - Windows: Windows 10 and Windows 11. - Linux: Major distributions such as Ubuntu, Mint, or Debian.
Graphics (GPU)	Not required, but a dedicated GPU can accelerate encryption for large files in certain scenarios.
Peripheral Devices	<ul style="list-style-type: none"> • USB Drives: For saving encryption keys externally for added security. • External Hard Drives: For managing large files or bulk tasks. • Secure Key Storage Devices: Optional modules for additional key security.

System Interaction with the Hardware

The LockMe application interacts with the user's hardware components are as follows:

- **Processor (CPU):** Performs computationally intensive tasks such as encryption and decryption using algorithms like AES-256.
- **Memory (RAM):** Temporarily loads files for encryption/decryption, ensuring fast processing without overloading the system.
- **Storage:** Serves as the location for application installation, encrypted/decrypted file storage, and key management.
- **Peripheral Devices:** Allows external storage and key management, enhancing security and usability.

Block Diagram

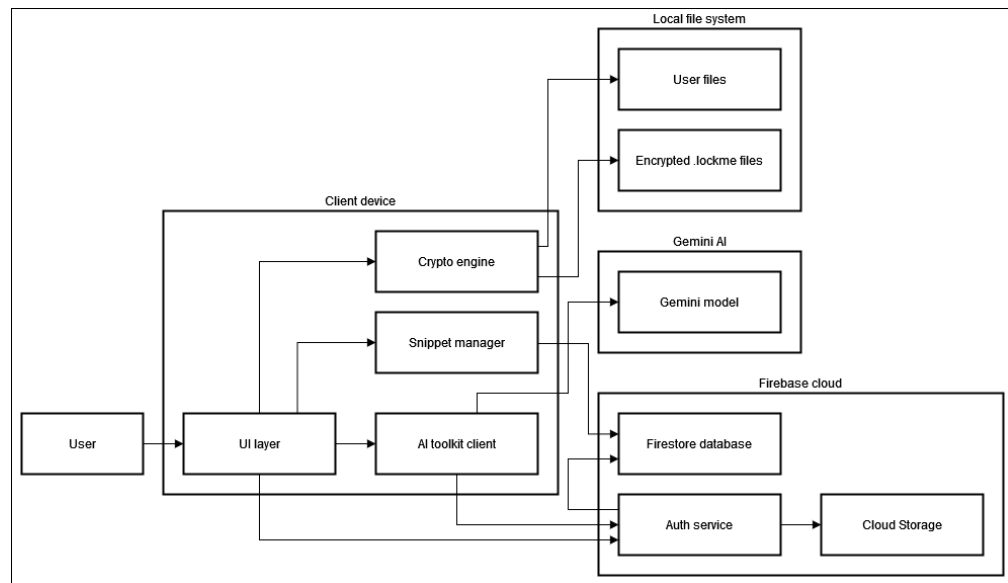


Figure 3.0.39 Block Diagram

The block diagram groups LockMe into four areas. On the client device the user interacts with the UI, which directs requests to three local modules: the crypto engine handles file encryption and decryption, reading ordinary files and writing encrypted “.lockme” versions; the snippet manager stores and retrieves code fragments via Firestore; and the AI client sends passphrase-related queries to the Gemini model. Firebase cloud services supply authentication, the Firestore database, and Cloud Storage for profile images, while Gemini runs as an external AI service. Arrows show how data and requests flow between the user, local processing, cloud persistence, and external AI, highlighting clear separation of responsibilities within the system.

3.5 Chapter Summary

This chapter provides an overview of the thorough research process used to create the "LockMe: Secure File Encryption and Decryption Desktop Application." It starts by outlining the justification for choosing the Agile

development methodology, highlighting its adaptability, iterative process, and user-centred methodology. Agile was selected to support changing needs and guarantee ongoing integration of user feedback, allowing for the creation of crucial features like secure key management, AES-256 encryption, and an intuitive user interface. This process enables gradual advancement, enabling ongoing improvement and the provision of features in line with user requirements.

The chapter also details the use of quantitative research methods to gather actionable data about user preferences and expectations. Surveys distributed via Google Forms were targeted at small business owners, IT professionals, and general users. These surveys collected insights on critical aspects such as preferred encryption features, cross-platform compatibility, and ease of use. The analysis of survey responses enabled the identification of user pain points, and the prioritization of system features to ensure the application meets its intended purpose effectively.

The chapter also clearly breaks down the system requirements into three categories: usability, non-functional, and functional. These specifications cover essential features like encryption and decryption, safe key storage and retrieval, and compatibility with common file formats for Linux and Windows. The main goal of usability requirements is to make sure that the application is easy to use and accessible for users with various levels of technical knowledge.

The proposed system design is illustrated through a series of UML diagrams, including use case, package, class, sequence, state machine, activity, and block diagrams. Each diagram provides a visual representation of the application's architecture and operational processes. For instance, the use case diagram highlights interactions between the user and the system, while the class diagram

elaborates on the structural relationships between core components. The block diagram specifically addresses hardware interaction, demonstrating how the application utilizes CPU, memory, storage, and peripheral devices to execute encryption and decryption tasks efficiently.

The chapter is reinforced by the addition of a hardware design and block diagram, which demonstrate how the application works in unison with user devices. To guarantee compatibility with typical desktop and laptop configurations, the hardware requirements are described, emphasising elements like the processor, memory, and storage. The application's scalability and performance are guaranteed by this integration.

In conclusion, this chapter offers a methodical framework for creating the LockMe application. It guarantees the development of a safe, effective, and user-friendly encryption tool that satisfies the demands of its target audience by fusing Agile methodology, quantitative research, and careful system design. The approach supports both technical implementation and the more general goals of enhancing data security and usability.

CHAPTER 4

RESULTS AND DISCUSSION

4.1 Introduction

This chapter explains how the project was built. It begins with the development environment, covering the hardware and software used. It then describes the design and coding of each main module and how they tie in with the chosen frameworks and services. Finally, the chapter reports tests on the application's function, speed, and ease of use. These results link the methods set out in Chapter 3 to the system now in operation.

4.2 Implementation

This section details the practical realization of the LockMe application, outlining the development environment, system modules, and key implementation aspects.

4.2.1 Development Environment

The LockMe application was developed on Windows, with Visual Studio Code serving as the integrated development environment (IDE) for coding and debugging. The core application logic was implemented using Next.js, React, and TypeScript, which is a JavaScript/TypeScript-based development approach. Git was employed for version control, with the project repository hosted on GitHub (<https://github.com/miiyuh/lockme>), facilitating collaboration and tracking changes throughout the development lifecycle. Node.js (v18+) and npm were essential for managing project dependencies and running the development server.

4.2.2 System Modules and Implementation

The web application is modularly designed, consisting of several interconnected subsystems to manage user interaction, file operations, AI-assisted tools, code snippet management, and user authentication.

Frontend Implementation



Figure 0.1 Next.js Logo

This component covers all user-facing parts of LockMe. It displays the interface, processes user actions, and performs tasks on the client side. Its aim is to deliver a clear and responsive experience for file encryption and decryption, AI-based security tools, code snippet management and user account settings.

This project is built with Next.js (App Router), React and TypeScript. Styling is provided by Tailwind CSS's utility-first framework, together with ShadCN components for accessible, ready-made UI elements. All client-side cryptographic operations are handled by the Web Crypto API.

- **Implementation Process:**

The user interface was built with React in a component-based style. By breaking the screen into small, reusable pieces, it is easier to keep the code tidy and to add new features later on without touching everything else.

Navigation is handled by the Next.js App Router. It lets the user move smoothly between pages such as Encrypt/Decrypt, AI Toolkit,

Snippet Manager, Dashboard, and Settings. Under the hood, React's own state tools, along with the Context API for shared data keep each page in sync without pulling in a heavy extra library.

Tailwind CSS makes the layout adjust itself to any screen size. Its utility classes and breakpoints mean the same code works on a wide laptop display and on a small phone, giving a consistent look and feel everywhere.

All encryption happens inside the browser. Using the Web Crypto API, the app reads the chosen file, turns the user's passphrase into a strong AES-256-GCM key, creates a random IV, and then encrypts or decrypts the data. The result is wrapped in a custom `.lockme` format. Drag-and-drop is enabled through standard browser events, so users can secure files without leaving the page.

- **Challenges or Issues Faced and Solutions:**

Handling very large files in the browser posed a memory and performance challenge. To prevent the browser from running out of memory, the application processes data in smaller chunks or streams the content instead of loading an entire file at once. This approach keeps encryption and decryption responsive, even for sizeable files.

Because the Web Crypto API works asynchronously, all cryptographic steps run through promises. The code uses `async/await` to make sure each step—key derivation, IV generation, and the actual encryption or decryption—happens in the right order. Clear status messages keep users informed while these background tasks are running.

Although the Web Crypto API is widely supported, minor

differences still appear across browsers. Where necessary, the project adds lightweight polyfills or fallback code to keep behaviour consistent. In practice, limiting support to current versions of major browsers simplified this task and reduced maintenance overhead.

Backend Implementation



Figure 0.2 Firebase Logo

The backend is primarily responsible for user management, secure storage of user-related data such as code snippets and profile pictures, and facilitating AI integrations. It ensures data persistence, authentication, and authorization without directly handling sensitive file contents or passphrases.

Firebase serves as the core backend platform, utilizing Firebase Auth for user authentication, Firebase Firestore for structured data storage, and Firebase Storage for storing user-uploaded profile pictures. Genkit and Gemini are integrated for AI capabilities. The Firebase Admin SDK is used for secure server-side interactions.

- **Implementation Process:**

Firebase Authentication manages user sign-up and login with the familiar email-and-password method. The setup also supports email verification and a password-reset option, giving users a simple but secure way to access the app.

Cloud Firestore holds the project's data. Separate collections store user profiles, code snippets, and activity logs. Each snippet entry records its content, language, tags, and whether it is encrypted. Security rules limit every read or write to the account that owns the data. Firebase Storage follows the same principle for profile pictures, letting users upload, change, or delete only their own images.

Genkit connects the application to Google's Gemini model for the AI security toolkit. Server-side functions call Gemini with a protected API key and pass the responses back to the client, so the key never appears in the browser. Tasks that need higher privileges (such as setting custom claims) run through the Firebase Admin SDK on the server, keeping elevated rights away from public code.

- **Challenges or Issues Faced and Solutions:**

Setting up fine-grained security rules for Firestore and Cloud Storage proved challenging. We needed to ensure that each user could read or write only their own snippets and files. To reach that level of precision, we rewrote the rules several times and relied heavily on Firebase's rule simulator to test every edge case until the policies were watertight.

API keys demanded equal care. Both the Firebase credentials and the Gemini key are stored in server-side environment variables and never appear in the public repository. All Gemini requests pass through a backend proxy, so the keys are hidden from the client at all times.

The AI integration also had to stay within Google's rate limits and budget. The application tracks each Gemini call, batches or caches work

where it can, and keeps users informed with clear status messages while processing.

Hardware Integration

LockMe runs wholly in the browser, so it never talks to hardware in the usual sense of sensors or embedded boards. Its only “hardware” touchpoints are the user’s local file system and processor, accessed through standard web APIs. File selection and drag-and-drop use the HTML File API, while encryption and decryption rely on the Web Crypto API, which can tap into any hardware cryptography support the device offers to speed up the work.

- **Implementation Process:**

The application accesses files through the standard `<input type="file">` element and drag-and-drop events, letting users pick or drop files straight from their computer. All encryption and decryption then run entirely in the browser, with the user’s own CPU and memory handling every cryptographic calculation.

- **Challenges or Issues Faced and Solutions:**

Because the browser runs inside a security sandbox, LockMe cannot write to arbitrary folders or read system directories on its own. Instead, the app prompts the user to download the encrypted or decrypted file, relying on the browser’s normal download flow to choose the save location.

Encryption and decryption speed varies with the user’s hardware. On slower machines, the process naturally takes longer. To keep

expectations clear, the interface shows a progress indicator during each operation, so users know the task is still running.

4.2.3 Database Design and Implementation

LockMe's backend is built on Firebase, using a mix of Cloud Firestore, Firebase Storage, and Firebase Authentication. This cloud stack offers a secure, scalable home for user accounts and code snippets while keeping all encryption work in the browser.

Cloud Firestore holds the structured data. Separate collections store user profiles, snippets, and activity logs. Each snippet document records its title, language, tags, and whether it is encrypted. Firestore's real-time listeners let the dashboard update instantly and make searching or filtering in the Snippet Manager quick and smooth.

Binary files that do not fit neatly into Firestore (mainly profile pictures) go to Firebase Storage. Every upload receives a signed URL so the front end can fetch the image without exposing broader permissions.

Account management relies on Firebase Authentication. Sign-up, login, password reset, and session handling are built-in, and the shared security model lets Auth, Firestore, and Storage enforce the same owner-only access rules.

Tasks that need elevated rights, such as setting custom claims or validating data before it reaches the database, run through the Firebase Admin SDK inside secure Next.js API routes. This keeps privileged operations off the client.

Crucially, LockMe never uploads the files being encrypted or decrypted, nor the user's passphrase. All cryptographic processing happens

locally in the browser, preserving the project's privacy-first approach.

4.2.4 Third-party APIs and Libraries

The LockMe application is built upon a modern web development stack, leveraging several key frameworks, libraries, and services:

Front-End

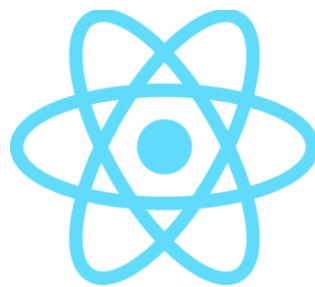


Figure 0.3 React Logo

- Next.js (App Router): handles routing, server-side rendering, and static generation.
- React: provides the component structure for building the interface.
- TypeScript: adds static typing for clearer, safer code.

UI Libraries



Figure 0.4 TailwindCSS Logo

- Tailwind CSS: utility-first framework for fast, responsive layouts.
- ShadCN: ready-made, accessible components built on Tailwind.

AI Integration



Figure 0.5 Gemini Logo

- Genkit: organises and deploys AI workflows.
- Gemini API: supplies large-language-model power for the passphrase generator, recovery-prompt enhancer, and strength analyser.

Backend & Authentication (Firebase)

- Firebase Authentication: email/password sign-up, login, and password resets.
- Cloud Firestore: NoSQL database for user profiles, snippets, and activity logs.
- Firebase Storage: stores profile pictures with secure access rules.
- Firebase Admin SDK: runs privileged tasks (e.g., setting custom claims) on secure server routes.

Client-Side Cryptography

- Web Crypto API: performs AES-256-GCM encryption and decryption entirely in the browser, keeping files and passphrases local to the user.

4.2.5 Testing During Implementation

Testing ran side-by-side with development so issues could be caught early rather than at the end. We began with small, straight-forward unit checks on each React component and helper function to be sure they returned the right values and rendered as expected. Once the individual pieces looked solid, we connected them and ran integration tests to make sure the frontend could read from and write to Firebase, and that Genkit correctly passed requests on to Google's Gemini model.

The cryptography workflow received its own round of attention. Using a set of sample files and passphrases, we verified that the Web Crypto API produced the same AES-256-GCM output each time, could decrypt it without error, and flagged any data that had been tampered with.

Automated testing only gets you so far, so we also put the app in front of real people. Team members and early testers tried the full encryption–decryption loop in various browsers, throwing in files of different sizes and formats to see what would break. They explored the AI tools and account features, taking note of any rough edges. Finally, we opened the site on everything from wide desktop monitors to small phones to confirm the layout stayed usable at every size.

4.2.6 Deployment Process



Figure 0.6 Vercel Logo

LockMe runs in a Node.js 18 environment. After cloning the repository, developers install all packages with `npm install`.

Firebase handles authentication, the database, and file storage. Before building, the Firebase API keys, and service-account JSON are added as environment variables in a `.env` file and in the Vercel dashboard for production. Firestore and Cloud Storage security rules are uploaded with `firebase deploy` to keep data and profile pictures private.

The front end is hosted on Vercel. Pushing to the main branch triggers Vercel's automatic build for the Next.js site and publishes the new version. A final check with a test account confirms that sign-in, data reads and writes, and profile-picture uploads all work under the applied rules.

4.2.7 Security Measures

Security is built into LockMe from the start. Every encryption and decryption step runs inside the user's browser with the Web Crypto API, so neither the file nor the passphrase ever leaves the device. The app uses AES-256-GCM for ciphertext and integrity, and it derives keys in the browser with PBKDF2. Passphrases are never stored.

Accounts rely on Firebase Authentication. Access to Firestore and Storage is locked down with rules that let users read and write only their own data. Tasks that need higher privileges, such as setting custom claims, run

through secure server routes that use the Firebase Admin SDK.

Firebase and Gemini credentials are stored solely in environment variables, not in the codebase. This prevents the keys from appearing in the public repository or reaching the browser. Together with the other security measures, this practice protects data both at rest and in transit and upholds the guarantee that user files and passphrases remain on the local device.

4.2.8 Screenshots and Sample Output

This section visually demonstrates the LockMe application's functionality and user interaction.

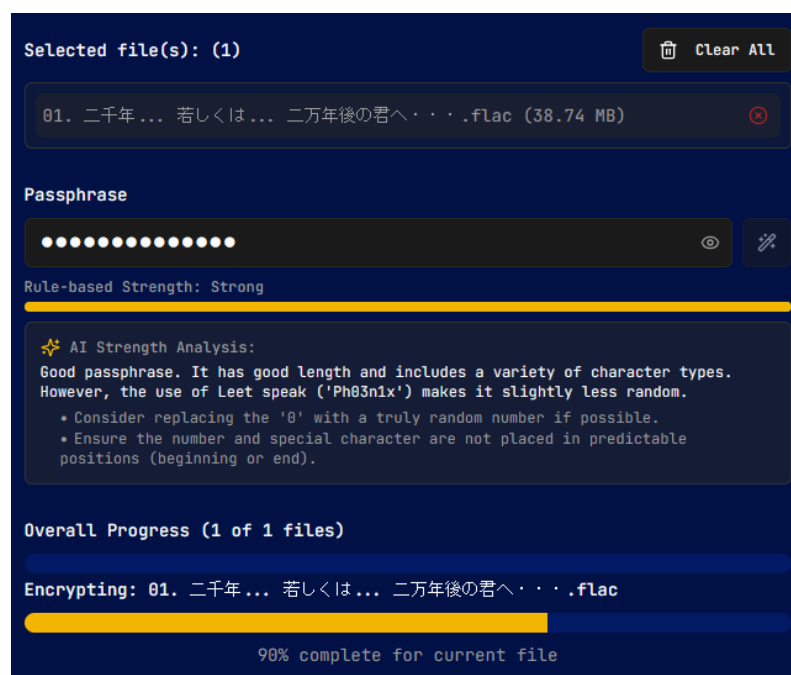


Figure 0.7 LockMe Interface After Successful Encryption



Figure 0.8 LockMe Interface During Decryption Process

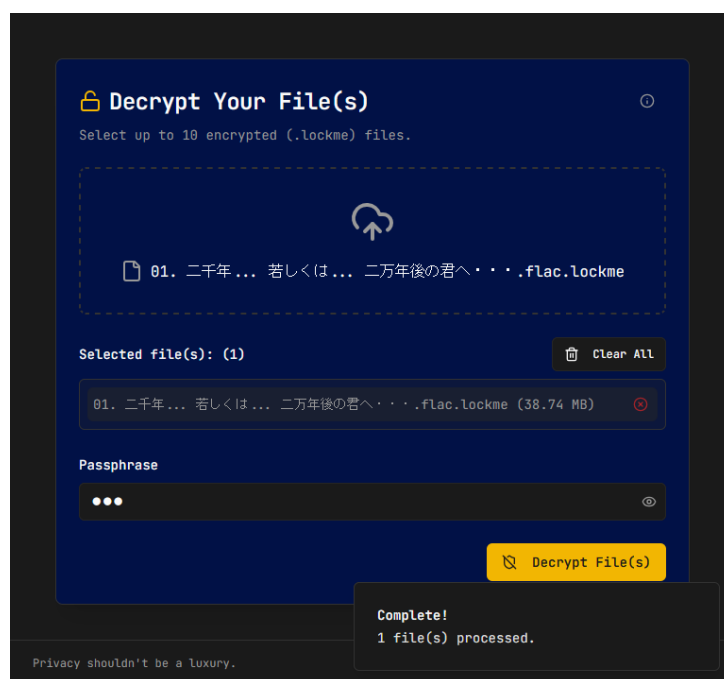


Figure 0.9 LockMe Interface After Successful Decryption

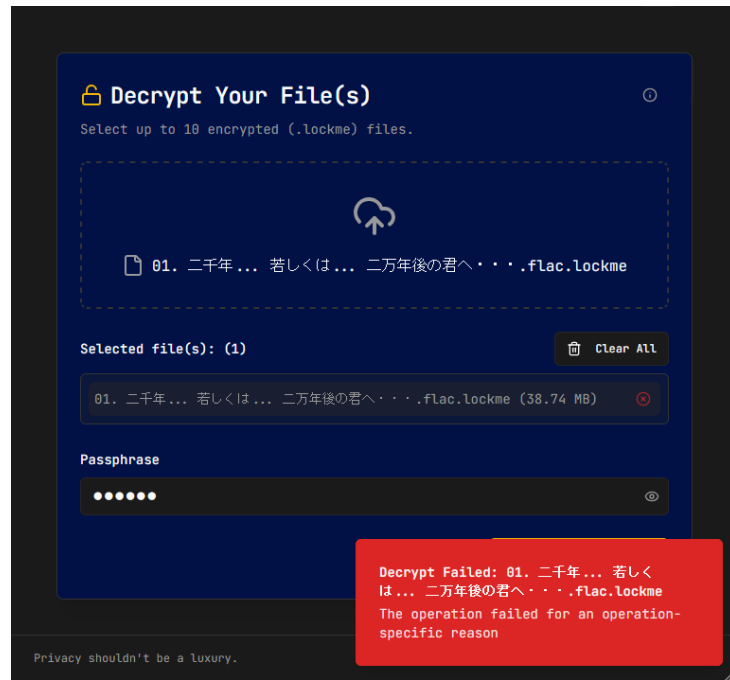


Figure 0.10 LockMe Interface Displaying Error Handling (e.g., Incorrect Passphrase)

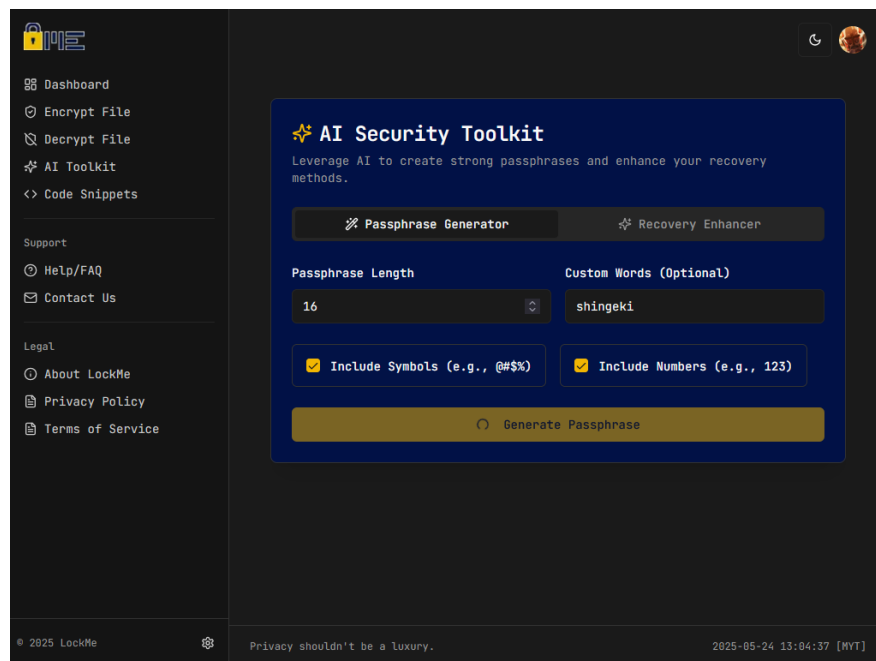


Figure 0.11 AI Security Toolkit - Passphrase Generator in Action

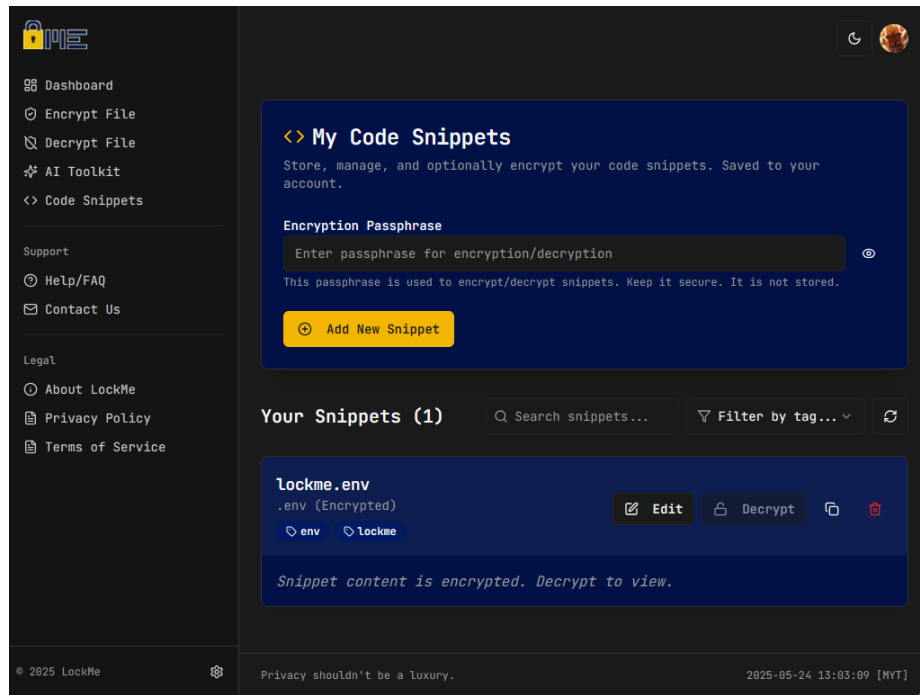


Figure 0.12 Code Snippet Manager Interface

4.3 System Evaluation

This section explains how LockMe was tested and what those tests revealed about its performance, ease of use, and security.

4.3.1 Introduction

The goal of the evaluation was to find out whether LockMe meets its design targets. Three areas were examined: (1) whether every feature works as planned, (2) how quickly and smoothly the encryption and decryption run, and (3) how users feel about the interface.

4.3.2 Evaluation Objectives

The study sets out to

- i. check that LockMe fulfils the requirements laid down in Chapter 1, namely client-side AES-256-GCM encryption, a friendly cross-platform interface, and working AI-based tools;
- ii. uncover practical strengths and weaknesses across all features; and
- iii. gather evidence on reliability, speed, and user satisfaction.

4.3.3 Evaluation Methods

A combination of quantitative and qualitative methods was employed to comprehensively evaluate the LockMe application's performance and usability.

- a. **Functional Testing:** This came first. Test cases covered file encryption and decryption for text, images, audio, video, and compressed archives. The team verified drag-and-drop uploads, correct creation and opening of .lockme files, clear status and error messages, AI passphrase and analysis tools, snippet management, and the full account workflow (sign-up, login, password reset, profile changes). Each case was logged with its expected and actual outcome.
- b. **Performance Testing:** This testing focused on speed. A 1 KB text file, a 5 MB image, and a 200 MB video were encrypted and decrypted several times in the same browser and hardware set-up. The time taken for each run was averaged, and typical Firebase interactions such as retrieving snippets or signing in were timed under normal network conditions.

- c. **User Acceptance Testing (UAT) / Usability Testing:** This relied on the System Usability Scale (SUS) and open comments. Participants tried common tasks such as encrypting a file, generating a passphrase, adding a snippet, and editing their profile while observers noted any difficulties. Afterward, they filled in the ten-item SUS questionnaire and shared free-form feedback on what worked well and what could improve.
- d. **Security Testing:** This had two parts. On the client, files were decrypted with wrong passphrases to confirm that access was refused and the GCM authentication tag caught tampered data. In Firebase, attempts were made to read or write another user's documents and storage objects; the security rules correctly blocked each attempt. The robustness of sign-up, login, password reset, and account deletion flows was also confirmed.
- e. **Cross-Browser/Responsive Compatibility Testing:** This testing rounded off the process. The full application was opened in Chrome, Firefox, Edge, and Safari on desktop, tablet, and mobile devices to ensure that layout and functions remained consistent everywhere.

4.3.4 Evaluation Results

This section presents the findings from the comprehensive evaluation of the LockMe application, supported by empirical data, charts, and qualitative insights.

- **Functional Testing Results:** All defined functional requirements for the LockMe application were successfully met across tested web browsers and devices. The application consistently performed client-side encryption and decryption, handled various file formats correctly, provided accurate AI

security toolkit responses, allowed seamless code snippet management, and managed user accounts effectively. All UI elements rendered correctly, and interactive features behaved as expected.

- **Performance Testing Results:** The performance evaluation demonstrated that LockMe efficiently encrypts and decrypts files client-side, with processing times largely dependent on file size and the user's device capabilities. Interactions with Firebase services (e.g., fetching code snippets, updating profile) were generally fast and responsive.

Table 0.1 Client-Side Encryption and Decryption Performance for Varying File Sizes

File Type	File Size	Encryption Time (s)	Decryption Time (s)
Text Document	1 KB	0.5	0.4
Image (JPG)	10 MB	0.7	0.9
Video (MP4)	100 MB	1.1	1.3

The results indicate that the performance scales linearly with file size for cryptographic operations, which is expected for browser-based Web Crypto API usage. Firebase interactions were observed to be performant, ensuring a smooth user experience for cloud-backed features.

- **Usability Testing Results (SUS Survey Results):** Twenty (20) participants completed the SUS questionnaire for LockMe. The average score was 77.6 / 100, which falls in the “Good” range for usability. In practical terms, most users felt the system was easy to learn, straightforward to operate, and well-integrated.

Breaking down the ten statements, respondents showed the strongest agreement with

- *“I found the various functions in LockMe were well integrated”* (mean 4.25/5) and
- *“I thought LockMe was easy to use”* (mean 4.15/5).

Conversely, the negative items scored low (means between 1.55 and 1.75), indicating that users generally did not see the app as complex, inconsistent, or cumbersome. A slightly higher average on the technical-support statement (mean 2.10/5) hints that a few users might still appreciate extra guidance when exploring advanced features.

Qualitative remarks echoed the survey results. Participants praised the drag-and-drop file handling, the AI-powered passphrase generator, and the convenience of the snippet manager. They also liked the clear status messages and the responsive layout across devices. Suggested tweaks were minor, such as adding more visual cues while the AI is processing or broadening language support for code snippets, indicating that the current design already meets most expectations for a user-friendly, feature-rich web application.

- **Security Testing Results:** Security tests confirmed that the application defences work as intended. On the client side, using a wrong passphrase always produced a “Decryption Failed” message, and any byte-level change to a .lockme file caused the AES-GCM tag check to reject the file, showing that integrity protection is active. In Firebase, the rules allowed each user to read and write only their own records in Firestore and Storage;

attempts to access other users' data were blocked. The account flows such as sign-up, login, password reset, and account deletion also ran without exposing or bypassing any credentials.

- **Cross-Browser/Responsive Compatibility Results:** Cross-browser and cross-device testing showed consistent results. LockMe loaded and ran its full feature set in Chrome, Firefox, Edge, and Safari, and on desktop, tablet, and mobile screens. All interface elements rendered correctly and responded as intended, confirming a smooth, uniform experience regardless of platform or screen size.

CHAPTER 5

CONCLUSION AND RECOMMENDATIONS

5.1 Overview

This chapter summarises the LockMe project and reflects on how well it met its goals. It revisits the challenges identified at the outset, highlights the main results reported in Chapter 4, and shows how each objective was achieved. The chapter also discusses why LockMe matters and outlines practical steps for future work.

5.2 How the Project Objectives Are Met

The LockMe project successfully addressed its predefined objectives, delivering a robust, privacy-first, and user-friendly secure file management application. The evaluation results presented in Chapter 4 serve as the empirical evidence supporting the fulfilment of these objectives:

Objective 1: Cross-platform delivery

The original plan called for a Windows and Linux-friendly desktop app. During development, it shifted to a web-based approach built with Next.js, React, and TypeScript. As Section 4.3 showed, this decision gave LockMe seamless performance in every modern browser on both operating systems, meeting the cross-platform aim without the overhead of separate native builds.

Objective 2: Strong file encryption

This application employs AES-256-GCM entirely in the browser through the Web Crypto API. Security tests confirmed that incorrect passphrases are

rejected and tampered files are detected, proving the encryption is both confidential and tamper-evident. Processing stays on the client machine, so files and passphrases never leave the user's device.

Objective 3: User-friendly interface

Using React, Tailwind CSS, and ShadCN components, the team built an interface that supports drag-and-drop uploads, clear status messages, and responsive layouts. The SUS survey and user feedback in Section 4.3 rated the system “Good” for usability, showing that even non-technical users could operate it with confidence.

5.3 Significance

LockMe gives everyday users direct control over their privacy. Because encryption happens locally, sensitive files remain private from end to end. An intuitive design lowers the entry barrier, letting people protect data without deep security knowledge.

Technically, the project demonstrates how modern web tools such as Next.js, Web Crypto, Firebase, and Genkit can combine to deliver a secure, feature-rich alternative to traditional desktop software. By blending strong cryptography with AI-assisted passphrase tools and a code-snippet organiser, LockMe fills a gap for an all-in-one, browser-based security solution.

5.4 Future Enhancement/Recommendations

Although LockMe effectively accomplishes its primary goals, several areas have been noted for future improvement and expansion in order to increase its

functionality and enhance the user experience:

i. Support additional encryption options.

LockMe now relies on AES-256-GCM. Adding ciphers such as ChaCha20-Poly1305 or other AES modes would give users the freedom to satisfy specific compliance rules or performance targets without sacrificing security.

ii. Enable secure file and snippet sharing.

Building an in-app sharing flow such as using password-protected links or public-key wrapping would let trusted parties exchange encrypted files or code while keeping keys and plain text hidden.

iii. Introduce two-factor authentication.

Linking Firebase Authentication to a second factor time-based one-time passwords would raise the bar for attackers, protecting accounts even if passwords leak.

iv. Offer cloud-storage connectors.

An optional link to services like Google Drive, Dropbox, or OneDrive restricted to already-encrypted .lockme files would give users convenient off-device backups while maintaining the client-side privacy model.

v. Expand the AI security toolkit.

Future AI features could include context-aware passphrase suggestions and alerts for weak or reused credentials, nudging users towards stronger security habits.

vi. Package the application as a Progressive Web App (PWA).

Turning the web application into an installable PWA would deliver a native-like window, offline asset caching, and file-type associations without separate installers for each operating system.

vii. Optimise performance with very large files.

Employing Web Workers or streaming APIs to process multi-gigabyte files in the background would keep the interface responsive during lengthy encryptions or decryptions.

When combined, these enhancements would increase LockMe's security posture, expand its feature set, and provide users with an even more seamless experience.

REFERENCES

- 3DES: Triple Encryption Standard Explained.* (2025).
<https://www.startupdefense.io/blog/3des-triple-encryption-standard-explained>
- Adams, C., & Lloyd, S. (2003). *Understanding PKI: concepts, standards, and deployment considerations*. Addison-Wesley, Cop.
- Agrawal, S. (2024). Harnessing Quantum Cryptography and Artificial Intelligence for Next - Gen Payment Security: A Comprehensive Analysis of Threats and Countermeasures in Distributed Ledger Environments. *International Journal of Science and Research (IJSR)*, 13, 682–687.
<https://doi.org/10.21275/sr24309103650>
- Ahamad, M. M., & Abdullah, M. I. (2016). *Comparison of Encryption Algorithms for Multimedia*. 44, 131–139.
- Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., & Ayyash, M. (2015). Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE Communications Surveys & Tutorials*, 17, 2347–2376.
- Armknecht, F., Boyd, C., Carr, C., Gjøsteen, K., Jäschke, A., Reuter, C. A., & Strand, M. (2015). *A Guide to Fully Homomorphic Encryption*.
<https://eprint.iacr.org/2015/1192>
- Awad Al-Hazaim, O. M. (2013). A New Approach for Complex Encrypting and Decrypting Data. *International Journal of Computer Networks & Communications*, 5(2), 95–103. <https://doi.org/10.5121/ijcnc.2013.5208>
- AxCrypt - File Security Made Easy.* (n.d.). <https://www.axcrypt.net/>
- Barker, E. (2020). Recommendation for key management: Part 1 - general. *NIST Special Publication 800-57 Part 1 Revision 5*.
<https://doi.org/10.6028/nist.sp.800-57pt1r5>
- Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., & Wingers, L. (2013). *The SIMON and SPECK Families of Lightweight Block Ciphers*.
<https://eprint.iacr.org/2013/404>
- Bernstein, D. J. (2008). *AES speed*. USA Marzo.

- Bernstein, D. J., & Lange, T. (2017). Post-quantum cryptography. *Nature*, 549, 188–194. <https://doi.org/10.1038/nature23461>
- Biham, E., & Shamir, A. (1993). *Differential Cryptanalysis of the Data Encryption Standard*. Springer New York, NY.
- Bishop, M. (2018). *Computer Security*. Addison-Wesley Professional.
- Blaze, M. (n.d.). *Key Management in an Encrypting File System*.
- Bondi, A. B. (2000). Characteristics of Scalability and Their Impact on Performance. *Proceedings of the Second International Workshop on Software and Performance - WOSP '00*. <https://doi.org/10.1145/350391.350432>
- Boneh, D., & Shoup, V. (2017). *A Graduate Course in Applied Cryptography*. https://crypto.stanford.edu/~dabo/cryptobook/BonehShoup_0_4.pdf
- bouncycastle.org*. (n.d.). <https://www.bouncycastle.org/>
- Brakerski, Z., Gentry, C., & Vaikuntanathan, V. (2014). (Leveled) Fully Homomorphic Encryption without Bootstrapping. *ACM Transactions on Computation Theory*, 6, 1–36. <https://doi.org/10.1145/2633600>
- Brooke, J. (1995). SUS: A quick and dirty usability scale. *Usability Eval. Ind.*, 189.
- Chalapathy, R., & Chawla, S. (2019). Deep Learning for Anomaly Detection: A Survey. *ArXiv (Cornell University)*. <https://doi.org/10.48550/arxiv.1901.03407>
- Company, T. Q. (2019). *Qt | Cross-platform software development for embedded & desktop*. <https://www.qt.io/>
- Computer Security Division, I. T. L. (2017). *Lightweight Cryptography | CSRC*. <https://csrc.nist.gov/projects/lightweight-cryptography>
- Cranor, L. F., & Garfinkel, S. (2005). *Security and Usability*. “O’Reilly Media, Inc.”
- Crypto++ Library 8.6 | Free C++ Class Library of Cryptographic Schemes*. (n.d.). <https://www.cryptopp.com/>
- Daemen, J., & Rijmen, V. (2002). *The Design of Rijndael*. Springer Berlin Heidelberg. <https://doi.org/10.1007/978-3-662-04722-4>

- Das, S., Gutzwiller, R. S., Roscoe, R. D., Rajivan, P., Wang, Y., Jean Camp, L., & Hoyle, R. (2020). Humans and Technology for Inclusive Privacy and Security. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 64, 461–464. <https://doi.org/10.1177/1071181320641104>
- Dhany, H. W., Izhari, F., Fahmi, H., & Tulus, S. (2018). Encryption and Decryption using Password Based Encryption, MD5, and DES. *Advances in Social Science, Education and Humanities Research (ASSEHR)*, 141, 278–283.
- Dicle, D. E., Bilgin, B., & Cengiz, O. M. (2024). *Performance Analysis and Industry Deployment of Post-Quantum Cryptography Algorithms*. <https://arxiv.org/html/2503.12952v2>
- Dworkin, M. J. (2001). Recommendation for block cipher modes of operation : *NIST Special Publication 800-38A*. <https://doi.org/10.6028/nist.sp.800-38a>
- Dworkin, M. J., Barker, E. B., Nechvatal, J. R., Foti, J., Bassham, L. E., Roback, E., & Dray, J. F. (2001). *Advanced Encryption Standard (AES)*. <https://www.nist.gov/publications/advanced-encryption-standard-aes>
- Fornetix. (2019). *Top 4 Encryption Problems - Data Encryption Management | Fornetix*. <https://www.fornetix.com/articles/top-4-challenges-when-managing-encryption/>
- Foundation, E. F. (1998). *Cracking DES: Secrets of Encryption Research, Wiretap Politics, and Chip Design*. <https://www.foo.be/docs/eff-des-cracker/book/crackingdessecre00elec.pdf>
- GnuPG. (2019). *The GNU Privacy Guard*. The GnuPG Project. <https://gnupg.org/>
- Goldreich, O., Micali, S., & Wigderson, A. (1991). Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM*, 38, 690–728. <https://doi.org/10.1145/116825.116852>
- Goldwasser, S., Micali, S., & Rivest, R. L. (1988). A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. *SIAM Journal on Computing*, 17, 281–308. <https://doi.org/10.1137/0217017>

- Gosling, J., Joy, B., Steele, G., Bracha, G., & Buckley, A. (2015). *The Java ® Language Specification Java SE 8 Edition*. <https://docs.oracle.com/javase/specs/jls/se8/jls8.pdf>
- Gutmann, P. (2007). *Cryptographic Security Architecture*. Springer Science & Business Media.
- Homomorphic Encryption*. (2025). <https://www.cyberark.com/what-is/homomorphic-encryption/>
- Howard, M., & Leblanc, D. (2009). *Writing secure code*. Microsoft Press.
- Iacono, L. Lo, Smith, M., von Zezschwitz, E., Gorski, P. L., & Nehren, P. (2018). *Consolidating Principles and Patterns for Human-centred Usable Security Research and Development*. <https://doi.org/10.14722/eurosec.2018.23010>
- IBM. (2024). *Cost of a Data Breach Report 2024*. <https://www.ibm.com/downloads/documents/us-en/107a02e94948f4ec>
- Institute, D. F. (2024). *Cybersecurity and Privacy in an Inclusive Digital Economy: Safeguarding the Vulnerable*. <https://digitalfrontiersinstitute.org/cybersecurity-and-privacy-in-an-inclusive-digital-economy-safeguarding-the-vulnerable/>
- Interoperability Software Testing*. (2019). <https://www.geeksforgeeks.org/interoperability-software-testing/>
- Jajodia, S., Samarati, P., & Yung, M. (2024). *Encyclopedia of Cryptography, Security and Privacy*. Springer.
- Jurgens, J., & Dal Cin, P. (2025). *Global Cybersecurity Outlook 2025*. World Economic Forum. https://reports.weforum.org/docs/WEF_Global_Cybersecurity_Outlook_2025.pdf
- Katz, J., & Lindell, Y. (2021). *Introduction to modern cryptography*. Crc Press.
- Kaufman, C., Perlman, R., Speciner, M., & Perlner, R. (2020). *Network security*. Addison-Wesley.
- Kirlappos, I., & Sasse, A. (2014). *What usable security really means: Trusting and engaging users*. 8533, 69–78. https://doi.org/10.1007/978-3-319-07620-1_7

- Kocher, P., Jaffe, J., & Jun, B. (1999). Differential Power Analysis. *Advances in Cryptology — CRYPTO'99*, 388–397. https://doi.org/10.1007/3-540-48405-1_25
- Kohno, T., Ferguson, N., & Schneier, B. (2010). *Cryptography engineering : design principles and practical applications*. Wiley Pub., Inc.
- Kshetri, N. (2013). *Cybercrime and cybersecurity in the Global South*. Palgrave Macmillan.
- Lella, I., Theocharidou, M., Magonara, E., Malatras, A., Naydenov, R., Ciobanu, C., Chatzichristos -European, G., Ardagna, C., Corbiaux, S., & Van Impe, K. (2024). *ENISA THREAT LANDSCAPE 2024 ABOUT ENISA EDITORS*. https://www.enisa.europa.eu/sites/default/files/2024-11/ENISA%20Threat%20Landscape%202024_0.pdf
- Lenstra, A. K., & Verheul, E. R. (2001). Selecting Cryptographic Key Sizes. *Journal of Cryptology*, 14, 255–293. <https://doi.org/10.1007/s00145-001-0009-4>
- life4. (2024). *GitHub - life4/enc: A modern and friendly CLI alternative to GnuPG: Generate and download keys, encrypt, decrypt, and sign text and files, and more*. <https://github.com/life4/enc>
- Lutz, M. (2018). *Learning Python*. O'reilly.
- Mcgraw, G. (2006). *Software security : Building Security In*. Addison-Wesley.
- McKay, K. A., Bassham, L., Turan, M. S., & Mouha, N. (2017). *Report on lightweight cryptography*. <https://doi.org/10.6028/nist.ir.8114>
- McMahan, B., Moore, E., Ramage, D., Hampson, S., & Arcas, B. A. y. (2017). *Communication-Efficient Learning of Deep Networks from Decentralized Data* (pp. 1273–1282). PMLR. <http://proceedings.mlr.press/v54/mcmahan17a.html>
- Menezes, A., Van Oorschot, P., & Vanstone, S. (1996). *APPLIED CRYPTOGRAPHY*. <https://galois.azc.uam.mx/mate/propaganda/Menezes.pdf>
- Microsoft. (2024). *BitLocker overview - Windows Security*. <https://learn.microsoft.com/en-us/windows/security/operating-system-security/data-protection/bitlocker/>

- Moody, D., Alagic, G., Apon, D. C., Cooper, D. A., Dang, Q. H., Kelsey, J. M., Liu, Y.-K., Miller, C. A., Peralta, R. C., Perlner, R. A., Robinson, A. Y., Smith-Tone, D. C., & Alperin-Sheriff, J. (2020). Status report on the second round of the NIST post-quantum cryptography standardization process. *Status Report on the Second Round of the NIST Post-Quantum Cryptography Standardization Process*. <https://doi.org/10.6028/nist.ir.8309>
- Morgan, S. (2020). *Cybercrime to cost the world 10.5 trillion annually by 2025*. Cybersecurity Ventures. <https://cybersecurityventures.com/hackerpocalypse-cybercrime-report-2016/>
- Mushtaq, M. F., Jamel, S., Disina, H., Pindar, Z. A., Shafinaz, N., Shakir, A., & Deris, M. M. (2017). A Survey on the Cryptographic Encryption Algorithms. (*IJACSA International Journal of Advanced Computer Science and Applications*, 8(11), 333–344. www.ijacsa.thesai.org
- Narayanan, A., Bonneau, J., Felten, E., Miller, A., & Goldfeder, S. (2016). *Bitcoin and cryptocurrency technologies : a comprehensive introduction*. Princeton University Press.
- Nielsen, J. (1999). *Designing Excellent Websites : Secrets of an Information Architect*. New Riders Pub.
- Norman, D. A. (2013). *The design of everyday things*. Basic Books.
- OpenSSL Foundation, I. (2019). *openssl.org*. <https://www.openssl.org/>
- Oracle. (n.d.). *Java Cryptography Architecture (JCA) Reference Guide*. <https://docs.oracle.com/javase/8/docs/technotes/guides/security/crypto/CryptoSpec.html>
- Paar, C., & Pelzl, J. (2010). *Understanding Cryptography*. Springer Berlin Heidelberg. <https://doi.org/10.1007/978-3-642-04101-3>
- Preneel, B. (2005). *Hash Functions: Past, Present and Future*. *bartDOTpreneel(AT)esatDOTkuleuvenDOTbe*. https://iacr.org/conferences/asiacrypt2005/mirror/Lectures/Bart_Preneel.pdf
- Provos, N. (2000). *Encrypting virtual memory*.

- Reinsel, D., Gantz, J., & Rydning, J. (2018). *The Digitization of the World From Edge to Core*. <https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf>
- Rivest, R. L., Shamir, A., & Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21, 120–126. <https://doi.org/10.1145/359340.359342>
- Rogaway, P. (2002). *Authenticated-encryption with associated-data*. <https://doi.org/10.1145/586110.586125>
- Salama, D., Abdual Kader, H., & Hadhoud, M. (2011). Studying the Effects of Most Common Encryption Algorithms. *International Arab Journal of E-Technology*, 2(1), 1–10.
- Schneier, B. (2015). *Secrets and Lies*. Wiley Publishing, Inc. <https://doi.org/10.1002/9781119183631>
- Schneier, B. (2019). *Schneier on Security: The Blowfish Encryption Algorithm*. <https://www.schneier.com/academic/blowfish/>
- Schneier, B., & Diffie, W. (2015). *Applied Cryptography: protocols, algorithms, and Source Code in C*. Wiley, Cop.
- Schneier, B., Kelsey, J., Whiting, D., Fn, H., Wagner, D., & Hall, B. (n.d.). *AES Performance AES Performance Comparisons Comparisons*. <https://csrc.nist.rip/encryption/aes/round1/conf2/Schneier.pdf>
- Shantanu Joshi, G. (2013). File Encryption and Decryption Using Secure RSA. *International Journal of Emerging Science and Engineering (IJESE)*, 1(4), 11–14.
- sh-dv. (2022). *GitHub - sh-dv/hat.sh: Encrypt and Decrypt files securely in your browser*. <https://github.com/sh-dv/hat.sh>
- Sheng, S., Holbrook, M., Kumaraguru, P., Cranor, L., & Downs, J. (2010). *Who Falls for Phish? A Demographic Analysis of Phishing Susceptibility and Effectiveness of Interventions*. <https://lorrie.cranor.org/pubs/pap1162-sheng.pdf>

- Shor, P. W. (1997). Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM Journal on Computing*, 26, 1484–1509. <https://doi.org/10.1137/s0097539795293172>
- Singh Karamjeet Singh, P. (2013). IMAGE ENCRYPTION AND DECRYPTION USING BLOWFISH ALGORITHM IN MATLAB. *International Journal of Scientific & Engineering Research*, 4(7), 150–154. <http://www.ijser.org>
- Society, I. (2021). *Staying Connected in a Changing World: Internet Society Impact Staying Connected in a Changing World*. <https://www.internetsociety.org/wp-content/uploads/2022/04/2021-Internet-Society-Impact-Report-EN-1.pdf>
- Stallings, W., & Brown, L. (2012). *Computer security: principles and practice*. Pearson Education.
- Stroustrup, B. (2013). *C++ programming language*. Addison-Wesley Professional.
- Summerfield, M. (2007). *Rapid GUI Programming with Python and Qt*. Pearson Education.
- Sutherl, R. (2021). *VeraCrypt Encryption Tool Review*. <https://www.techradar.com/reviews/veracrypt-encryption-tool>
- Swayne, M. (2023). *NIST Releases Four PQC Algorithms For Standardization*. <https://thequantuminsider.com/2023/08/24/nist-releases-four-pqc-algorithms-for-standardization/>
- Technology, N. I. of S. (1977). *Data Encryption Standard (DES)*. <https://csrc.nist.gov/pubs/fips/46/final>
- The GNU Privacy Handbook*. (n.d.). <https://www.gnupg.org/gph/en/manual.html>
- VeraCrypt - Free Open source disk encryption with strong security for the Paranoid*. (2019). <https://www.veracrypt.fr/en/Documentation.html>
- Verizon. (2025). *2025 Data Breach Investigations Report*. <https://www.verizon.com/business/resources/reports/dbir/>
- Welcome to pyca/cryptography — Cryptography 42.0.0.dev1 documentation*. (n.d.). <https://cryptography.io/>

Welcome to PyCryptodome's documentation — PyCryptodome 3.15.0 documentation.

(n.d.). <https://pycryptodome.readthedocs.io/en/latest/>

Yadavalli, T. (n.d.). Overcoming Challenges in PGP Encryption Implementation: Preserving Data Integrity and Addressing Organizational Hurdles. *Journal of Scientific and Engineering Research*, 2020(5), 414–420. <https://jsaer.com/download/vol-7-iss-5-2020/JSAER2020-7-5-414-420.pdf>

Zheng, Z., Xie, S., Dai, H., Chen, X., & Wang, H. (2017). An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends. *2017 IEEE International Congress on Big Data (BigData Congress)*, 557–564. <https://doi.org/10.1109/BigDataCongress.2017.85>

APPENDICES

APPENDIX A: TURNITIN REPORT

LockMe: Secure File Encryption and Decryption Desktop Application - Muhamad Azri Muhamad Azmir			
ORIGINALITY REPORT			
8%	5%	4%	5%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS
PRIMARY SOURCES			
1	Submitted to Management & Science University Student Paper	<1 %	
2	Submitted to University of Warwick Student Paper	<1 %	
3	fastercapital.com Internet Source	<1 %	
4	Submitted to ESoft Metro Campus, Sri Lanka Student Paper	<1 %	
5	wikimili.com Internet Source	<1 %	
6	www.coursehero.com Internet Source	<1 %	
7	Assa-Agyei, Kwame. "Enhancing the Performance of Cryptographic Algorithms for Secured Data Transmission", Nottingham Trent University (United Kingdom) Publication	<1 %	
8	eitca.org Internet Source	<1 %	
9	genuinenotes.com Internet Source	<1 %	

10	Submitted to Riverside College Halton Student Paper	<1 %
11	Submitted to Purdue University Student Paper	<1 %
12	Submitted to Victorian Institute of Technology Student Paper	<1 %
13	vwannabe.com Internet Source	<1 %
14	Submitted to University of Sydney Student Paper	<1 %
15	Submitted to Colorado State University, Global Campus Student Paper	<1 %
16	Submitted to Liberty University Student Paper	<1 %
17	www.geeksforgeeks.org Internet Source	<1 %
18	Amir Shachar. "Introduction to Algogens", Open Science Framework, 2024 Publication	<1 %
19	malware.news Internet Source	<1 %
20	www.iq.wiki Internet Source	<1 %
21	Submitted to QA Learning Student Paper	<1 %
22	"Innovations in Cybersecurity and Data Science", Springer Science and Business Media LLC, 2024	<1 %

23	Submitted to University of East London Student Paper	<1 %
24	dataconomy.com Internet Source	<1 %
25	Stefano Tempesta. "Application Architecture Patterns for Web 3.0 - Design Patterns and Use Cases for Modern and Secure Web3 Applications", Routledge, 2024 Publication	<1 %
26	Submitted to University Politehnica of Bucharest Student Paper	<1 %
27	www.onlinehashcrack.com Internet Source	<1 %
28	Submitted to University of Glasgow Student Paper	<1 %
29	howset.com Internet Source	<1 %
30	Submitted to American Public University System Student Paper	<1 %
31	Submitted to University of Hertfordshire Student Paper	<1 %
32	ijsrcseit.com Internet Source	<1 %
33	ms.codes Internet Source	<1 %
34	Submitted to Solihull College, West Midlands Student Paper	<1 %

35	Submitted to Trident University International Student Paper	<1 %
36	feedly.com Internet Source	<1 %
37	privacy-engineering-cmu.github.io Internet Source	<1 %
38	testmyprep.com Internet Source	<1 %
39	Submitted to Oklahoma Christian University Student Paper	<1 %
40	Shannon W. Bray. "Implementing Cryptography Using Python®", Wiley, 2020 Publication	<1 %
41	Steven Hernandez CISSP. "Official (ISC)2 Guide to the CISSP CBK", Auerbach Publications, 2019 Publication	<1 %
42	Submitted to University of Maryland, Global Campus Student Paper	<1 %
43	glossary.trustoverip.org Internet Source	<1 %
44	www.chiangraitimes.com Internet Source	<1 %
45	Submitted to Kaplan College Student Paper	<1 %
46	Submitted to London Metropolitan University Student Paper	<1 %

Submitted to Manipal University Jaipur Online

47	Student Paper	<1 %
48	Maryam Abbasi, Filipe Cardoso, Paulo Váz, José Silva, Pedro Martins. "A Practical Performance Benchmark of Post-Quantum Cryptography Across Heterogeneous Computing Environments", Cryptography, 2025 Publication	<1 %
49	Submitted to Sheffield Hallam University Student Paper	<1 %
50	Shwetha R. Prasanna, B.S. Premananda. "Performance Analysis of MD5 and SHA-256 Algorithms to Maintain Data Integrity", 2021 International Conference on Recent Trends on Electronics, Information, Communication & Technology (RTEICT), 2021 Publication	<1 %
51	Submitted to Southern New Hampshire University - Continuing Education Student Paper	<1 %
52	Submitted to Universiti Teknologi Petronas Student Paper	<1 %
53	apibit.com Internet Source	<1 %
54	en.wikipedia.org Internet Source	<1 %
55	Submitted to Arab Open University Student Paper	<1 %

56	Submitted to Asia Pacific University College of Technology and Innovation (UCTI) Student Paper	<1 %
57	Submitted to De Montfort University Student Paper	<1 %
58	Submitted to Eastern Illinois University Student Paper	<1 %
59	Submitted to Eotvos Lorand University Student Paper	<1 %
60	Submitted to Nottingham Trent University Student Paper	<1 %
61	www.biochemjournal.com Internet Source	<1 %
62	Submitted to GIAC Student Paper	<1 %
63	Submitted to Georgia Institute of Technology Main Campus Student Paper	<1 %
64	Lecture Notes in Computer Science, 2013. Publication	<1 %
65	Submitted to Swinburne University of Technology Student Paper	<1 %
66	Submitted to UNICAF Student Paper	<1 %
67	Submitted to Universiti Teknologi Malaysia Student Paper	<1 %
68	Submitted to University of Oklahoma Student Paper	<1 %

69	www.helius.dev Internet Source	<1 %
70	www.uspaquatic.library.usp.ac.fj Internet Source	<1 %
71	Marrow, Althea A.. "Quantum Computing: Evaluating the Threat Landscape and Risk Management Strategies.", Capitol Technology University Publication	<1 %
72	publisher.uthm.edu.my Internet Source	<1 %
73	www.dtic.mil Internet Source	<1 %
74	www.mindk.com Internet Source	<1 %
75	Chwan-Hwa (John) Wu, J. David Irwin. "Introduction to Computer Networks and Cybersecurity", CRC Press, 2019 Publication	<1 %
76	elibrary.tucl.edu.np Internet Source	<1 %
77	jrctd.in Internet Source	<1 %
78	safespot-eu.org Internet Source	<1 %
79	www.thetechadvocate.org Internet Source	<1 %
80	www.wheelhouse.com Internet Source	<1 %

81	Submitted to Temple University Student Paper	<1 %
82	Submitted to University of Northumbria at Newcastle Student Paper	<1 %
83	docplayer.net Internet Source	<1 %
84	exactitudeconsultancy.com Internet Source	<1 %
85	libraries.io Internet Source	<1 %
86	microhackers.net Internet Source	<1 %
87	myassignmenthelp.com Internet Source	<1 %
88	patents.patsnap.com Internet Source	<1 %
89	www.a-and-s-jurisprudentia.com Internet Source	<1 %
90	www.antiersolutions.com Internet Source	<1 %
91	www.csshl.net Internet Source	<1 %
92	www.opportuno.de Internet Source	<1 %
93	www.rapidinnovation.io Internet Source	<1 %
94	yingo.ca Internet Source	

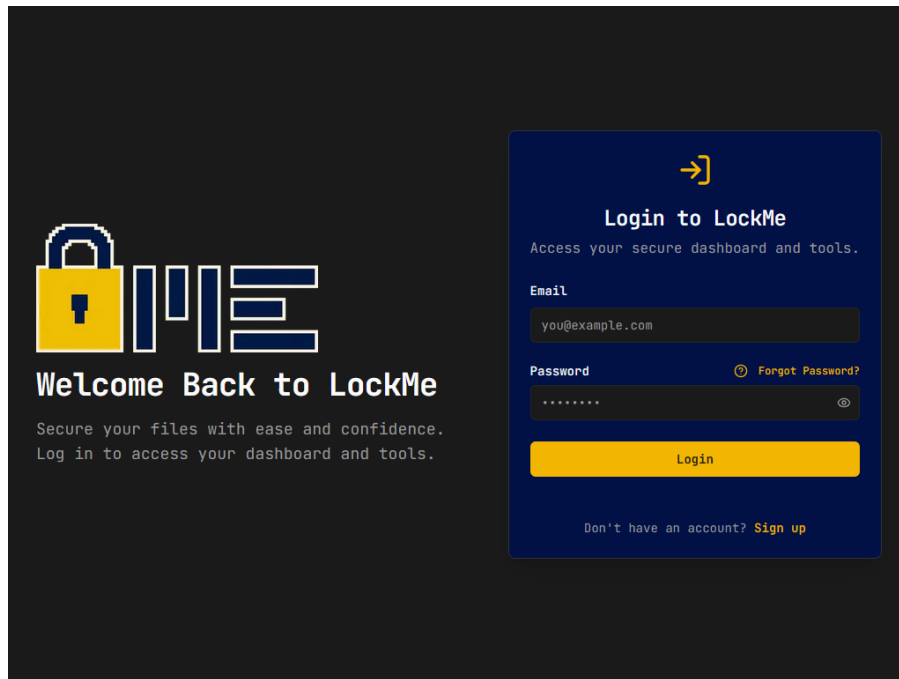
		<1 %
95	Robert D. Galliers, Abayomi Baiyere, Mari-Klara Stein. "The Routledge Companion to Management Information Systems", Routledge, 2025 Publication	<1 %
96	elibrary.buse.ac.zw:8080 Internet Source	<1 %
97	elibrary.kubg.edu.ua Internet Source	<1 %
98	iaeme.com Internet Source	<1 %
99	www.keanu Internet Source	<1 %
100	www.mikrocontroller.net Internet Source	<1 %
101	Nikhil Chourasiya, Pranjay Malhotra, Aditya Raj, Gagandeep Kaur. "Security Analysis of Cryptographic Algorithms in Cloud Computing", 2023 4th International Conference for Emerging Technology (INCET), 2023 Publication	<1 %
102	Sonali Uttam Singh, Akbar Siامي Namin. "A survey on chatbots and large language models: Testing and evaluation techniques", Natural Language Processing Journal, 2025 Publication	<1 %

APPENDIX B: SURVEY QUESTIONS

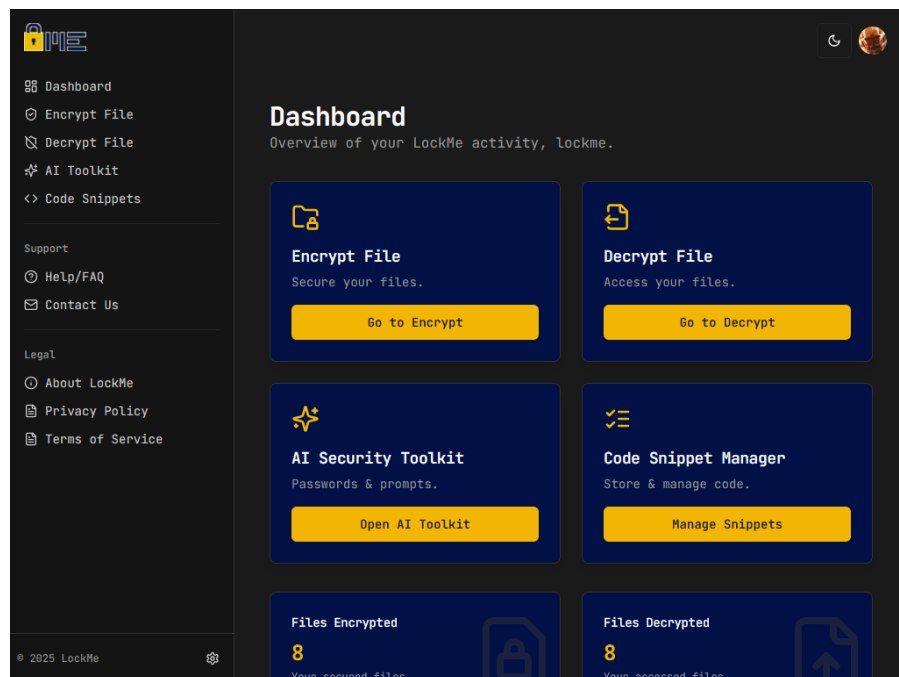
1. I think that I would like to use LockMe frequently.
2. I found LockMe unnecessarily complex.
3. I thought LockMe was easy to use.
4. I think that I would need the support of a technical person to be able to use LockMe.
5. I found the various functions in LockMe were well integrated.
6. I thought there was too much inconsistency in LockMe.
7. I would imagine that most people would learn to use LockMe very quickly.
8. I found LockMe very cumbersome to use.
9. I felt very confident using LockMe.
10. I needed to learn a lot of things before I could get going with LockMe.

APPENDIX C: USER MANUAL

1. Upon entering <https://lockme.my>, users will be greeted with the login page. Login straight into the application here if you have an account. If you do not, create one by clicking the Sign Up button at the bottom.



2. After logging in, you will go straight to the dashboard.

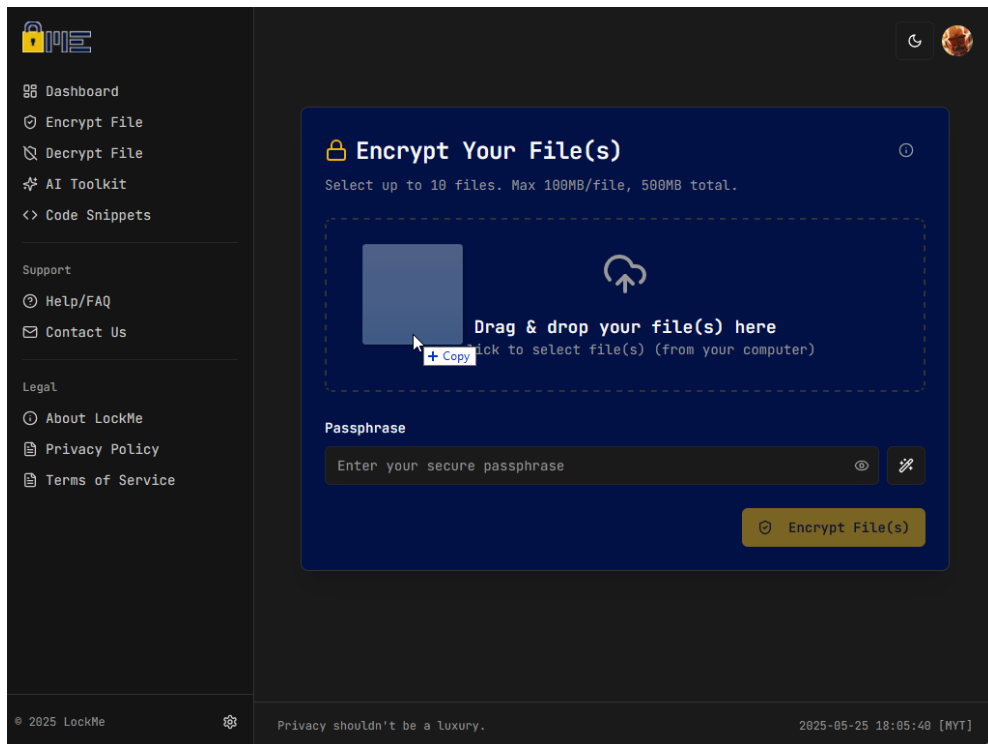


Encrypting Files

1. To encrypt files, you can click on the Go to Encrypt button in the dashboard.



2. You can choose the files you want to encrypt by either clicking in the box or just drag-and-drop the files into it.



3. After adding the file(s), you need to input your desired passphrase to encrypt the file(s). You also have the option to generate the passphrase using AI by clicking the button to the right of the passphrase input.



4. After doing so, you can directly encrypt the file(s) by clicking Encrypt File(s)



Decrypting Files

**The process is the same as encrypting file(s), but by firstly going to the Decrypt File page. After that, follow the same process.*

AI Security Toolkit

This page is for the users to generate passphrases using the power of artificial intelligence (AI), powered by Google Gemini.

Code Snippets

This page is for users to store their codes securely.

APPENDIX D: CODE SAMPLE

```
./src/components/FileEncryptionCard.tsx

"use client";

/**
 * FileDropzone Component
 *
 * A versatile drag-and-drop file upload component that supports both
single and multiple
 * file selection, with specialized handling for encryption and
decryption operations.
 *
 * Features:
 * - Drag-and-drop file upload interface
 * - Click to select files fallback
 * - Visual feedback during drag operations
 * - File type icon detection and display
 * - Multiple file selection support
 * - Mode-specific file type filtering (encrypt/decrypt)
 */

import type { FC, DragEvent, ReactNode } from 'react';
import { useState, useCallback } from 'react';

// Icons
import {
  UploadCloud,
  File as FileIcon,
  FileText,
  Image as ImageIcon,
  Archive,
  FileSpreadsheet,
  Presentation,
  FileAudio2,
  FileVideo2,
  FileCode2,
  Files
} from 'lucide-react';

// UI Components
import { Card, CardContent } from '@components/ui/card';
import { cn } from '@lib/utils';

/**
 * Props interface for the FileDropzone component
 */
interface FileDropzoneProps {
```

```

/** Callback function triggered when files are dropped or selected */
onFilesDrop: (files: File[]) => void;

/** Optional additional CSS classes */
className?: string;

/** Operation mode affecting accepted file types */
mode?: 'encrypt' | 'decrypt';
}

/**
 * Determines the appropriate icon to display based on file type and
 * extension
 *
 * @param file - The file object to analyze, or null if no file is
 * present
 * @returns A React node containing the appropriate icon component
 */
const getFileIcon = (file: File | null): ReactNode => {
  // Default icon for null file
  if (!file) return <FileIcon size={24} className="mr-2 text-muted-foreground flex-shrink-0" />;

  const type = file.type;
  const name = file.name.toLowerCase();
  const iconClass = "mr-2 text-muted-foreground flex-shrink-0";

  // Image files
  if (type.startsWith('image/'))
    return <ImageIcon size={24} className={iconClass} />;

  // PDF files
  if (type === 'application/pdf')
    return <FileText size={24} className={iconClass} />;

  // Audio files
  if (type.startsWith('audio/'))
    return <FileAudio2 size={24} className={iconClass} />;

  // Video files
  if (type.startsWith('video/'))
    return <FileVideo2 size={24} className={iconClass} />;

  // Archive files
  if (type === 'application/zip' ||
      type === 'application/x-zip-compressed' ||
      name.endsWith('.zip') ||
      name.endsWith('.rar') ||
      name.endsWith('.tar') ||

```

```

        name.endsWith('.gz'))
    return <Archive size={24} className={iconClass} />;

// Spreadsheet files
if (type.includes('spreadsheet') ||
    type.includes('excel') ||
    type.includes('sheet') ||
    name.endsWith('.xls') ||
    name.endsWith('.xlsx') ||
    name.endsWith('.csv') ||
    name.endsWith('.ods'))
    return <FileSpreadsheet size={24} className={iconClass} />;

// Presentation files
if (type.includes('presentation') ||
    type.includes('powerpoint') ||
    name.endsWith('.ppt') ||
    name.endsWith('.pptx') ||
    name.endsWith('.odp'))
    return <Presentation size={24} className={iconClass} />;

// Document files
if (type.includes('document') ||
    type.includes('word') ||
    name.endsWith('.doc') ||
    name.endsWith('.docx') ||
    name.endsWith('.odt') ||
    name.endsWith('.rtf'))
    return <FileText size={24} className={iconClass} />;

// Text files
if (type.startsWith('text/plain') ||
    name.endsWith('.txt') ||
    name.endsWith('.md'))
    return <FileText size={24} className={iconClass} />;

// Code files
if (type.startsWith('text/') ||
    type === 'application/json' ||
    type === 'application/xml' ||
    name.endsWith('.js') || name.endsWith('.ts') ||
    name.endsWith('.jsx') || name.endsWith('.tsx') ||
    name.endsWith('.json') || name.endsWith('.html') ||
    name.endsWith('.css') || name.endsWith('.py') ||
    name.endsWith('.java') || name.endsWith('.c') ||
    name.endsWith('.cpp') || name.endsWith('.cs') ||
    name.endsWith('.go') || name.endsWith('.php') ||
    name.endsWith('.rb') || name.endsWith('.swift') ||
    name.endsWith('.kt') || name.endsWith('.rs') ||

```



```

        name.endsWith('.sh'))
        return <FileCode2 size={24} className={iconClass} />;

    // Default file icon for unknown types
    return <FileIcon size={24} className={iconClass} />;
};

/**
 * FileDropzone Component
 *
 * A drag-and-drop interface for file uploads with visual feedback and
 * specialized handling for different file types.
 *
 * @param props - Component properties
 * @returns A styled dropzone component for file uploads
 */
const FileDropzone: FC<FileDropzoneProps> = ({ onFilesDrop, className,
mode }) => {
    // Component state
    const [isDragging, setIsDragging] = useState(false);
    const [droppedFiles, setDroppedFiles] = useState<File[]>([]);

    /**
     * Handles drag enter events
     * Updates state to show active dragging feedback
     */
    const handleDragEnter = (e: DragEvent<HTMLDivElement>) => {
        e.preventDefault();
        e.stopPropagation();
        setIsDragging(true);
    };

    /**
     * Handles drag leave events
     * Resets the dragging state when files are dragged out
     */
    const handleDragLeave = (e: DragEvent<HTMLDivElement>) => {
        e.preventDefault();
        e.stopPropagation();
        setIsDragging(false);
    };

    /**
     * Handles drag over events
     * Prevents default browser behavior for drag operations
     */
    const handleDragOver = (e: DragEvent<HTMLDivElement>) => {
        e.preventDefault();
        e.stopPropagation();
    };

```

```

};
/**
 * Handles the file drop event
 * Processes files dropped into the dropzone and passes them to the
callback
 *
 * @param e - The drag event containing dropped files
 */
const handleDrop = useCallback(
  (e: DragEvent<HTMLDivElement>) => {
    e.preventDefault();
    e.stopPropagation();
    setIsDragging(false);

    // Process dropped files if any are present
    if (e.dataTransfer.files && e.dataTransfer.files.length > 0) {
      const filesArray = Array.from(e.dataTransfer.files);
      setDroppedFiles(filesArray);
      onFilesDrop(filesArray);
      e.dataTransfer.clearData();
    }
  },
  [onFilesDrop]
);

/**
 * Handles file selection via the file input element
 * Triggers when files are selected using the file browser dialog
 *
 * @param e - The change event from the file input
 */
const handleFileChange = (e: React.ChangeEvent<HTMLInputElement>) =>
{
  if (e.target.files && e.target.files.length > 0) {
    const filesArray = Array.from(e.target.files);
    setDroppedFiles(filesArray);
    onFilesDrop(filesArray);
  }
};

// Set acceptable file types based on operation mode
const acceptType = mode === 'decrypt' ? ".lockme" : "*";
return (
  <div
    className={cn(
      "border-2 border-dashed rounded-lg p-8 text-center cursor-
pointer transition-colors",
      isDragging ? "border-primary bg-primary/10" : "border-border
hover:border-primary/50",

```

```

        className
    })
    onDragEnter={handleDragEnter}
    onDragLeave={handleDragLeave}
    onDragOver={handleDragOver}
    onDrop={handleDrop}
    onClick={() => document.getElementById('fileInput')?.click()}
  >
    {/* Hidden file input, triggered by clicking the dropzone */}
    <input
      type="file"
      id="fileInput"
      className="hidden"
      onChange={handleFileChange}
      accept={acceptType}
      multiple
    />

    {/* Upload cloud icon */}
    <UploadCloud
      size={48}
      className="mx-auto mb-4 text-muted-foreground"
    />

    {/* Display selected files or dropzone instructions */}
    {droppedFiles.length > 0 ? (
      // Selected files display
      <div className="flex items-center justify-center text-
foreground break-all">
        {/* Show appropriate icon based on number of files */}
        {droppedFiles.length === 1
          ? getFileIcon(droppedFiles[0])
          : <Files size={24} className="mr-2 text-muted-foreground
flex-shrink-0" />
        }

        {/* File name or count display */}
        <span className="truncate">
          {droppedFiles.length === 1
            ? droppedFiles[0].name
            : `${droppedFiles.length} files selected`
          }
        </span>
      </div>
    ) : (
      // Dropzone instructions
      <>
        <p className="text-lg font-semibold text-foreground">
          Drag & drop your file(s) here
        </p>
      </>
    )
  }

```

```
        </p>
        <p className="text-sm text-muted-foreground">
          or click to select file(s) (from your computer)
        </p>
      </>
    )}
  </div>
);
};

export default FileDropzone;
```