1. Introduction

1. Introduction to File Shield and Data Protection

This introductory chapter establishes the foundational importance of file security in the contemporary digital landscape. It delineates the overarching objectives that drive the implementation of robust file protection systems, the complex problems they are designed to solve, their essential components, and the contextual considerations within a project framework, including key assumptions, constraints, advantages, and limitations.

1.1. Objective of the New System

The primary objective of deploying a robust file security system is to safeguard digital assets from a spectrum of threats, including unauthorized access, illegal tampering, accidental deletion, intentional damage, and data degradation.[1] This comprehensive protection necessitates the implementation of stringent measures, controls, and governance policies. At its core, the system aims to uphold the fundamental principles of the CIA Triad: ensuring **Confidentiality** by preventing unauthorized access to sensitive information such as Personally Identifiable Information (PII), Intellectual Property (IP), and corporate secrets; maintaining **Integrity** by preserving data accuracy and preventing unauthorized alteration or corruption; and guaranteeing **Availability** by ensuring that files remain accessible to authorized users whenever required.

Beyond these technical safeguards, the system's objectives extend to critical business and ethical considerations. A well-implemented file security solution is instrumental in preserving individual privacy, protecting an organization's intellectual assets, ensuring adherence to a myriad of regulatory compliance mandates (e.g., HIPAA, GDPR, PCI DSS, ISO 9001, FDA 21 CFR Part 11), and maintaining a positive brand reputation by proactively preventing costly data breaches and associated legal repercussions.[1] This broader scope underscores that file security systems are not merely an IT overhead but represent strategic investments that protect an organization's core values, market standing, and long-term viability. The value proposition of such a system transcends technical functionality, directly contributing to an organization's resilience and trustworthiness in an increasingly data-dependent world.

1.2. Problem Definition

The central challenge that file security systems are designed to mitigate is unauthorized access to files and sensitive data. This phenomenon occurs when an individual or a program gains entry to or uses a system or resource without explicit permission from the owner or administrator, often leading to severe consequences such as data breaches, the compromise of confidential information, and disruptions to critical services.

Unauthorized access can manifest through a diverse array of attack vectors, highlighting the multi-dimensional nature of the problem. These include:

- **Exploiting software vulnerabilities:** Attackers frequently leverage known weaknesses in outdated or unpatched software to gain illicit entry.
- **Credential compromise:** This involves the theft of user credentials through methods like phishing, keylogging, or brute force attacks, where automated scripts rapidly try numerous username and password combinations.
- **Social engineering:** Threat actors manipulate individuals within an organization to divulge sensitive information or perform actions that facilitate unauthorized access.
- **Insider threats:** Risks can originate from within the organization, encompassing non-malicious insiders who cause accidental harm due to negligence or lack of awareness, malicious insiders who intentionally steal data or cause damage, and compromised insiders whose legitimate accounts or credentials have been taken over by external attackers.
- **Physical security breaches:** Although less common than digital vectors, physical access to servers, network devices, or critical infrastructure components can also lead to unauthorized digital access.

The repercussions of neglecting file security are substantial, often resulting in significant financial losses, severe damage to an organization's brand reputation, and considerable regulatory penalties.[2] The diverse attack surface, encompassing technical, human, and physical vectors, necessitates a comprehensive, multi-faceted defense strategy. This approach must combine robust technological safeguards with strong policies, continuous employee training, and appropriate physical security measures, recognizing that a purely technical solution is insufficient against such a broad spectrum of threats.

1.3. Core Components

A robust file security system is not a singular product but rather an integrated ecosystem of interoperating solutions, each contributing a distinct layer of protection to achieve comprehensive defense. This layered defense principle is fundamental, as the failure of one security mechanism does not immediately compromise the entire system, due to the presence of subsequent safeguards. Key components include:

- **File Shield (Real-time Protection):** This serves as the primary layer of active antivirus protection. It continuously scans programs and files stored on a device for malicious threats as they are opened, run, modified, and saved, aiming to prevent malware infection. File Shield offers extensive configurability, allowing for adjustments in scan sensitivity, selection of specific file types to scan (e.g., common infected packers or all files), and defining automated actions upon detection (e.g., quarantining, repairing, or deleting threats). It also incorporates advanced techniques like heuristics and code emulation to detect unknown malware.

- **Sensitive Data Shield (Content-Aware Protection):** Operating as a specialized layer, this component is designed to identify and protect documents containing sensitive information, such as banking details, passwords, IDs, payslips, and other PII, typically found in formats like.pdf,.doc,.docx,.xls, and.xlsx. It specifically guards against malware and unauthorized access to these critical files, offering granular control over which applications and users can access them. The existence of this specialized shield alongside general file scanning highlights that generic file-level protection is often insufficient for highly sensitive data, necessitating a content-aware layer.

- **Data Encryption:** This foundational security measure converts plaintext data into unreadable ciphertext, ensuring confidentiality both when data is at rest (stored on devices or in the cloud) and in transit (shared across networks).[2] It employs strong cryptographic algorithms, such as Advanced Encryption Standard (AES) and Rivest-Shamir-Adleman (RSA), alongside robust encryption keys.

- **Access Control & Identity Management:** These mechanisms define and enforce who can access files and what specific actions they are permitted to perform. This includes Access Control Lists (ACLs) at the operating system level, Role-Based Access Control which assigns permissions based on user roles, and Attribute-Based Access Control which grants access based on user, resource, and environmental attributes. These are typically coupled with strong authentication methods like multi-factor authentication to verify user identity. The ability to block other user accounts on a shared PC from accessing protected documents further emphasizes the importance of user-level access management.

- **Data Loss Prevention (DLP):** DLP tools are designed to monitor and analyze data flows across endpoints, networks, and cloud services to prevent the unauthorized sharing or exfiltration of sensitive data.

- **File Integrity Monitoring (FIM):** This component continuously monitors critical file attributes, including volume, access rights, creation, and modification dates, along with hash functions, to detect any unauthorized modifications, deletions, or corruption. Alerts are triggered when divergences between the original and current states are detected.

- **Backup & Recovery:** Essential for business continuity, these solutions involve automated processes to create and store previous versions of files. This enables rapid restoration in the event of data loss due to accidental deletion, corruption, or cyberattacks, adhering to best practices like the 3-2-1 rule (three copies of data, on two different types of media, with one copy offsite).

- **Continuous Monitoring & Threat Detection:** This involves leveraging tools such as Security Information and Event Management (SIEM) systems to aggregate and analyze logs, Intrusion Detection/Prevention Systems (IDS/IPS) to monitor network and system activities, and Endpoint Detection and Response (EDR) solutions to monitor and respond to threats on endpoints. These tools work in real-time to identify suspicious patterns, anomalies, and potential malicious activities.

- **Digital Rights Management (DRM):** This extends control beyond basic encryption, regulating what can be done with document content *after* it has been opened and decrypted. DRM can prevent actions like copying, editing, screen grabs, or unauthorized printing, and allows for remote revocation of document access regardless of its location. This addresses the crucial "last mile" problem of content misuse, where encryption alone is insufficient for complete control over sensitive information once legitimately accessed.

The integration of these diverse components forms a robust defense-in-depth strategy, acknowledging that no single mechanism can provide absolute security. This comprehensive approach necessitates careful architectural design, deployment, and ongoing management to ensure seamless interoperability and effective protection across the entire data lifecycle.

1.3.1 Key File Security Mechanisms and Their Functions

Mechanism	Primary Function	Key Features/Examples
File Shield	Real-time malware detection and prevention at the file system level.	Scans files on open, run, modify, save; configurable sensitivity; heuristic analysis; code emulation; auto-run scan for removable media.
Sensitive Data Shield	Content-aware protection for sensitive documents.	Scans for PII, financial data in specific file types (.pdf,.doc,.xls); blocks unauthorized application/user access; manual document protection.
Data Encryption	Renders data unreadable without a decryption key.	AES, RSA algorithms; protects data at rest (storage) and in transit (network); software like Boxcryptor, NordLocker, Virtru.
Access Control (ACLs, RBAC, ABAC)	Defines and enforces user/system permissions to files and actions.	Granular control (read, write, execute); role-based permissions; attribute-based policies; Principle of Least Privilege (PoLP); strong authentication (MFA).
Data Loss Prevention (DLP)	Prevents unauthorized exfiltration of sensitive data.	Monitors data flows across endpoints, networks, cloud; classifies sensitive data; enforces protection policies; detects anomalous transfers.
File Integrity Monitoring (FIM)	Detects unauthorized modifications or deletions of critical files.	Tracks changes to file attributes (hash, size, date, permissions); real-time alerts; routine audits.
Backup & Recovery	Ensures data availability and rapid restoration after incidents.	Automated backups; off-site storage; versioning; adherence to 3-2-1 rule.
Continuous Monitoring & Threat Detection	Real-time analysis of security events and patterns.	SIEM (log aggregation), IDS/IPS (network/system anomaly detection), EDR (endpoint activity monitoring), NTA (network traffic analysis).
Digital Rights Management (DRM)	Controls usage of content after decryption.	Prevents copying, editing, screen grabs, unauthorized printing; remote access revocation; dynamic watermarks.

1.4. Project Profile

Data security within the context of project management is paramount, extending beyond mere technical implementation to become a fundamental aspect of ensuring overall project success. This involves the diligent protection of all sensitive project information, ranging from financial data and project plans to internal communications and progress reports, against both unauthorized access and potential breaches.

The scope of data security in project management encompasses mitigating risks from both external and internal sources. While external threats such as hackers and cyberattacks are a constant concern, significant risks also arise internally, including accidental data loss due to human error or the misuse of information by employees. A robust data security strategy therefore aims to create a secure environment where project teams can collaborate and share information without fear of breaches or leaks.

Key practices that project managers must integrate into their methodologies include:

- **Access Control:** Implementing role-based access to ensure that only individuals with a legitimate need-to-know can access specific information. This is complemented by regular audits of access rights and the mandatory use of two-factor authentication (2FA) for all systems storing sensitive project data

- **Data Encryption:** Employing end-to-end encryption for all communications and data storage. It is crucial to ensure that sensitive files are encrypted prior to sharing, adding a vital layer of protection even if files are intercepted

- **Regular Backups:** Establishing automated backup routines for all project data and storing these backups in secure, off-site locations to protect against data loss from technical failures or physical disasters.

- **Employee Training:** Recognizing that human error is a significant factor in data breaches, project managers must implement comprehensive security awareness programs. These programs educate team members on best practices, including recognizing phishing attempts and securely handling sensitive information, and reinforce clear data security policies.

Project management software and secure collaboration platforms that offer built-in security features—such as data encryption, access control, and real-time monitoring—are invaluable tools in this endeavor. The overarching goal of such cybersecurity projects is to effectively combat cyber threats, protect personal data, secure transactions, preserve customer confidence, and continuously identify system vulnerabilities. This strategic integration of security throughout the project lifecycle, from planning to execution, influences budget allocation, timelines, and team responsibilities, underscoring its pivotal role in successful project delivery.

1.5. Assumptions and Constraints

In the context of software development and cybersecurity projects, assumptions and constraints serve as foundational elements for project planning, bridging the gap between established facts and inherent uncertainties.[30] Assumptions represent beliefs or presuppositions about the project, its environment, or other relevant factors that are presumed to be true in the absence of absolute certainty. Conversely, constraints are defined as restrictions or limitations that must be considered throughout the development process.

The primary purpose of clearly documenting these assumptions and constraints is to foster mutual understanding among project teams and stakeholders, facilitate effective risk management, and enable informed decision-making throughout the software development lifecycle. This transparency ensures that the software solution is developed within its predefined scope and limitations.

Assumptions and constraints can be categorized into several types:

- **Business Assumptions:** These relate to the broader business environment in which the software solution operates, including market conditions, adherence to specific regulatory requirements (e.g., ISO 27001, HIPAA, PCI DSS), or internal business policies that may influence the system's design or functionality.

- **Environmental Assumptions:** These consider external factors that could influence the software solution, such as the availability of a reliable power supply, consistent network connectivity, or cultural and social factors that impact software usage and deployment.

- **Resource Constraints:** These refer to practical limitations in terms of time, allocated budget, available personnel, or the specific hardware and software resources required for project implementation.

Effective management of assumptions and constraints involves several best practices. It is crucial to clearly differentiate between assumptions (beliefs yet to be confirmed) and constraints (unavoidable limitations). All statements should be specific, clear, and verifiable, allowing for confirmation or invalidation as the project progresses. Project teams must continuously review, update, and validate these factors at key project milestones to ensure their accuracy and relevance. Assumptions should be viewed with a degree of skepticism and challenged to determine the potential impact if they prove incorrect. For unavoidable constraints, developing appropriate workarounds is essential.

Ultimately, both assumptions and constraints must be incorporated into the project plan, influencing tasks, schedules, budgets, and resource assignments. Their management should also be reviewed as part of a post-project evaluation. This explicit connection between assumptions/constraints and risk management highlights that these elements are not merely administrative details but critical inputs for proactive risk identification and mitigation in cybersecurity projects. Unvalidated assumptions or unaddressed constraints can directly translate into project failures, security vulnerabilities, or budget overruns, underscoring the need for continuous monitoring and adaptation in a dynamic security landscape.

1.6. Advantages and Limitations of the Proposed System

Implementing a comprehensive file security system offers substantial benefits while also presenting certain challenges. A clear understanding of both is essential for informed decision-making and effective deployment.

Advantages of Robust File Protection:

- **Data Security:** Core file protection mechanisms, including encryption, access control lists, and granular file permissions, provide robust data security by preventing unauthorized access to sensitive information. This is critical for preventing data breaches and other security incidents.

- **Regulatory Compliance:** File protection mechanisms are indispensable for adhering to a wide array of regulatory requirements, such as GDPR, HIPAA, and PCI-DSS, as well as quality management standards like ISO 9001 and FDA 21 CFR Part 11. Failure to comply can result in significant financial penalties and reputational damage.

- **Business Continuity:** By preventing data loss due to accidental or malicious deletion, corruption, or other forms of damage, file protection mechanisms (e.g., backup and recovery, auditing, logging) are crucial for ensuring business continuity. They enable rapid data recovery in the event of an incident, allowing business operations to resume quickly.

- **Increased Productivity:** Ensuring that files are consistently available and securely accessible to authorized users minimizes the risk of downtime and data loss incidents, thereby contributing to increased organizational productivity.

- **Enhanced Collaboration:** File protection mechanisms facilitate secure file sharing and access among authorized users, which helps prevent conflicts and misunderstandings that can arise in collaborative environments.

- **Reputation:** Demonstrating a strong commitment to data security and compliance through robust file protection enhances an organization's reputation. This builds trust with customers, partners, and stakeholders, positively impacting the organization's standing and bottom line.

- **Intellectual Property (IP) Protection:** Safeguarding documents containing proprietary information and corporate secrets is vital for maintaining an organization's competitive advantage.

Limitations of File Protection Systems:

- **Performance Overhead:** The implementation of certain file protection mechanisms, such as real-time encryption, extensive access control lists, and continuous auditing, can introduce additional processing requirements. This may consume significant system resources and potentially slow down file access and overall system performance.[5]

- **Complexity:** File protection systems can be inherently complex, often requiring specialized knowledge for their proper implementation, configuration, and ongoing management. This complexity increases the risk of errors and misconfigurations, which can inadvertently compromise data security.[5]

- **Compatibility Issues:** Some file protection mechanisms may not be universally compatible with all types of files, applications, or operating system environments, leading to potential limitations in file usage or integration challenges.

- **Cost:** The investment required for implementing robust file protection mechanisms, including software licenses, specialized hardware, and the hiring or training of skilled personnel, can be substantial. This can pose a particular challenge for smaller organizations with limited budgets.

- **User Frustration:** Overly stringent file protection measures, such as requirements for complex passwords, frequent authentication prompts, or highly restricted access, can lead to user frustration. If not designed with usability in mind, these measures may inadvertently impact productivity or even encourage users to seek insecure workarounds.

- **Human Factor Vulnerabilities:** Despite the most advanced technical controls, file security remains susceptible to human factors. Social engineering tactics, accidental human error, or malicious insider actions can still lead to data breaches, highlighting that technology alone is not a complete solution.

Table 1.6.1: Comparative Analysis of File Protection Advantages and Limitations

Category	Advantage	Limitation
Security Posture	Prevents unauthorized access, data breaches, and tampering.	Human factors (error, social engineering, insider threats) remain vulnerabilities.
Compliance & Governance	Essential for meeting regulatory mandates (GDPR, HIPAA, PCI-DSS, ISO 9001).	Complexity of compliance across diverse regulations.
Operational Resilience	Ensures business continuity through rapid data recovery and minimized downtime.	Performance overhead impacting system speed and resource consumption.
Productivity & Collaboration	Facilitates secure access and sharing, enhancing efficiency.	User frustration from stringent controls or complex processes.
Organizational Impact	Protects intellectual property and enhances brand reputation.	High implementation and ongoing maintenance costs.
Technical Implementation	Provides granular control over data access and usage.	Compatibility issues with certain file types, applications, or OS.

2. Requirement Determination & Analysis

2. Requirement Determination & Analysis for File Security

This chapter delves into the systematic methodologies employed for identifying and analyzing the specific needs and expectations pertinent to a file security system. It details the processes for gathering requirements, identifies the diverse user groups and threat actors that the system must accommodate or protect against, and introduces the various tools and techniques essential for successful implementation, alongside a critical examination of their respective benefits and drawbacks.

2.1. Requirement Determination

The process of requirement determination, often referred to as requirements gathering, is a foundational and critical phase in any project, particularly in the development of complex cybersecurity systems.[33] It involves systematically identifying, meticulously documenting, and effectively managing the diverse needs and expectations of all stakeholders involved.[33] This comprehensive understanding forms the bedrock upon which the entire system design, development, and subsequent implementation phases are built.

Several key methodologies are employed to ensure a thorough and accurate determination of requirements:

- **Stakeholder Identification:** The initial and most crucial step involves identifying and engaging all parties with a vested interest in the project's success. This includes project managers, end-users who will interact with the system, subject matter experts who possess domain knowledge, and compliance officers who ensure adherence to regulations. Collecting perspectives from all these roles from the outset helps ensure that all diverse needs are considered.

- **Stakeholder Interviews:** Once identified, individual or group interviews are conducted using open-ended questions to elicit detailed information about their goals, needs, and expectations. These sessions also serve to uncover any potential conflicts or differing priorities among stakeholders.

- **Workshops and Group Sessions:** Facilitating collaborative environments, such as brainstorming sessions or agile workshops, encourages creative thinking and allows for the capture of a wide range of ideas and requirements through interactive techniques like mind mapping or affinity diagrams.

- **Surveys and Questionnaires:** For projects with a large or geographically dispersed stakeholder base, surveys and questionnaires provide an efficient method for gathering quantitative data and insights, ensuring that meaningful responses translate into meaningful requirements.
- **Existing Documentation Analysis:** A thorough review of existing documentation, including business plans, user manuals, and technical specifications, provides valuable background information. This process can reveal requirements that might have been overlooked and helps identify potential conflicts or gaps in the already gathered information.

Requirements are typically categorized into distinct types to ensure clarity and precision:

- **Business Requirements:** These define the fundamental objectives the project aims to accomplish for the organization, such as boosting market share or enhancing customer lifetime value.

- **User Requirements:** These outline the specific goals that end-users can achieve with the product, often articulated through user stories, use cases, or scenarios.

- **Product Specifications:** These describe precisely how the system must operate to satisfy both user and business needs, and they are further subdivided into:

  - **Functional Requirements:** These specify the concrete capabilities and behaviors the system must exhibit (e.g., "The system shall encrypt files upon saving," or "The user shall be able to log in"). These must be precise, concise, and explicit.

  - **Non-Functional Requirements:** Often referred to as "quality attributes," these impose constraints on *how* the system performs its functions. This category critically includes **security**, alongside performance, reliability, scalability, and portability. The classification of security as a non-functional requirement signifies that it is not an isolated feature but rather a pervasive quality that dictates how functional requirements are implemented, influencing architectural choices, performance considerations, and usability from the outset.

2.2. Targeted Users

File security solutions are meticulously designed to protect data for a broad spectrum of users and entities within an organization, with the overarching aim of preventing data loss through unauthorized access. Understanding these diverse user groups and the potential threat actors is crucial for tailoring effective security measures.
Primary Categories of Targeted Users/Entities (those for whom access is legitimate and protected):

- **Legitimate Users:** This category encompasses employees, external partners, and customers who require authorized access to specific files and systems to perform their roles or access services. The file security system ensures their data privacy, particularly concerning Personally Identifiable Information (PII) and Protected Health Information (PHI), and facilitates secure collaboration among them.

- **Administrators/Security Personnel:** These are privileged users responsible for managing the security system itself, configuring policies, monitoring activity logs, and responding to security incidents. Their access is highly sensitive and requires the most stringent controls, often incorporating role-based access control (RBAC) and multi-factor authentication (MFA).

Threat Actors (those against whom protection is implemented):

The evolving threat landscape necessitates a comprehensive understanding of adversaries. The identification of various "insider threat" categories alongside external hackers demonstrates a mature understanding of this landscape. This implies that file security systems must incorporate robust internal controls, user behavior analytics, and continuous monitoring, moving beyond a sole focus on external perimeter defenses to address risks originating from within.

- **External Attackers/Hackers:** These are individuals or organized groups outside the organization attempting to gain unauthorized entry through a variety of sophisticated methods, including exploiting software vulnerabilities, brute force attacks, cross-site scripting (XSS), and advanced persistent threats (APTs).

- **Non-malicious Insiders:** These are legitimate users who, through negligence, accidental actions, or a lack of security awareness, inadvertently cause harm or create vulnerabilities. This highlights the critical need for ongoing employee training and clear security policies.

- **Malicious Insiders:** These are legitimate users who intentionally attempt to steal data, sabotage systems, or cause harm to the organization for personal gain. Their intimate knowledge of internal systems and processes makes them particularly dangerous.

- **Compromised Insiders:** In these scenarios, legitimate user accounts or credentials are stolen or compromised by external attackers, who then impersonate the legitimate user to gain unauthorized access and perform malicious activities.

Specific User Contexts Requiring Tailored Security:

- **Shared Personal Computers:** The Sensitive Data Shield feature, for instance, explicitly allows for blocking other user accounts on the same PC from accessing protected documents, which is a crucial control in shared computing environments.

- **Internet of Things (IoT) Devices:** As IoT devices proliferate, security solutions must target their authentication and authorization processes, ensuring that only authorized devices can connect to a network, alongside encrypting data transmitted to and from these devices.

- **Database Users:** For various database types (relational, NoSQL, document, graph, time series), security solutions define granular access controls, leverage policy tags, implement column-level security, and utilize role-based access control (RBAC) to restrict data access.

The comprehensive approach to file security acknowledges that threats can originate from multiple vectors and actors, both external and internal. This necessitates a design that integrates robust internal controls, user behavior analytics, and continuous monitoring, thereby extending protection beyond traditional perimeter defenses to address the full spectrum of risks.

2.3. Details of tools and techniques used / implemented

The implementation of robust file security necessitates a strategic combination of specialized software tools, technical mechanisms, and established best practices. This integrated approach creates a layered defense, providing comprehensive protection across the data lifecycle. The sheer number and diversity of these tools and techniques underscore that a comprehensive file security system is not a single product but an integrated ecosystem of interoperating solutions. This implies inherent complexity in deployment, configuration, and ongoing management, requiring a holistic security architecture.

- **Antivirus and Real-time Protection:**

  - **File Shield:** This core component continuously scans files as they are accessed (opened, run, modified, saved) to detect and prevent malicious threats from infecting the system.[9] It is highly configurable, allowing administrators to adjust sensitivity levels, specify file types for scanning (e.g., common infected packers or all files), and define automated actions upon detection (e.g., moving to quarantine, repairing, or deleting infected files).[10] Advanced features like heuristics and code emulation are employed to identify unknown malware by analyzing code for suspicious behavior.

  - **Sensitive Data Shield:** This specialized tool specifically identifies and protects documents containing sensitive information, such as PII, banking details, and payslips, typically in common document formats like.pdf,.doc,.docx, and.xls.[12] It guards against unauthorized access and malware, allowing for manual protection of specific documents and granular application-level access control.

  - **Anti-Exploit Monitor:** This component scans and protects against known exploits targeting popular software, adding another layer of defense against vulnerabilities.

- **Encryption:**

  - **File Encryption Software:** These tools convert plaintext data into unreadable ciphertext using strong cryptographic algorithms such as AES, RSA, or Blowfish, and secure encryption keys.[15] Examples include Boxcryptor for cloud storage encryption, NordLocker for personal file encryption, and Virtru, which integrates seamlessly with email and cloud applications to encrypt files and attachments.

  - **Data at Rest & In Transit:** Encryption is applied to data both when it is stored on devices or in cloud services (data at rest) and when it is being transmitted over networks (data in transit) to ensure confidentiality throughout its lifecycle.

- **Access Control Mechanisms:**

  - **Access Control Lists (ACLs):** These are operating system-level controls that define which users or system processes can access specified files or directories and what actions they are permitted to perform (e.g., read, write, execute).

  - **Role-Based Access Control (RBAC):** This model assigns permissions based on predefined user roles within an organization, simplifying access management by granting access rights to roles rather than individual users.

  - **Attribute-Based Access Control (ABAC):** This more dynamic model grants access based on a combination of attributes associated with the user (e.g., department, clearance level), the resource (e.g., file type, sensitivity), and the context (e.g., time of day, location).

  - **Principle of Least Privilege (PoLP):** A fundamental security principle, PoLP dictates that users should be granted only the minimum access rights necessary to perform their required tasks, thereby limiting the potential damage from a compromised account.

  - **Authentication:** This process verifies the identity of a user or system, typically through strong passwords, Multi-Factor Authentication (MFA), biometric data, or security tokens.

  - **Authorization:** Once a user's identity is authenticated, authorization mechanisms define precisely what actions that user is permitted to perform within the system.

- **Identity and Access Management (IAM) Systems:** These centralized systems are used to manage user identities and their access rights across an organization's various IT resources, streamlining the administration of access controls.

- **Data Loss Prevention (DLP):** DLP solutions continuously monitor and analyze data flows across endpoints, network perimeters, and cloud services to identify and prevent the unauthorized sharing, transfer, or exfiltration of sensitive data.[1] They often include monitoring, alert systems, and remediation options like blocking data transfers or quarantining files.

- **File Integrity Monitoring (FIM):** FIM tools track changes to critical file attributes, including content, characteristics (e.g., name, size, dates), and access permissions, generating alerts when discrepancies are detected.[2] FIM can operate through uninterrupted surveillance or periodic audits.

- **Backup and Recovery Solutions:** These automated systems are vital for data resilience, storing previous versions of files to allow for restoration in the event of data corruption, accidental deletion, or cyberattacks.[2] Best practices include the 3-2-1 backup rule.

- **Secure Document Management Systems (DMS):** Centralized platforms that provide comprehensive document control, including standardization through templates and naming conventions, robust version control and revision tracking, approval workflows, secure storage and retrieval, and detailed audit trails.

- **Network Security:** This foundational layer includes firewalls to control network traffic, intrusion detection and prevention systems (IDS/IPS) to identify and block malicious activities, network segmentation to isolate sensitive data, and secure communication channels like Virtual Private Networks (VPNs) and Secure File Transfer Protocols (SFTP).

- **Secure Configuration & Hardening:** This involves minimizing vulnerabilities by systematically securing server configurations, operating system settings, and application parameters, often following industry best practices.

- **Continuous Monitoring & Logging:** This involves the aggregation and analysis of security-relevant information from various sources. Security Information and Event Management (SIEM) platforms centralize log data, Endpoint Detection and Response (EDR) systems monitor endpoint activity, and Network Traffic Analysis (NTA) products inspect network traffic for suspicious patterns and threats. Metadata analysis plays a critical role in this, providing context and aiding in threat detection and incident response.

- **Digital Rights Management (DRM):** DRM systems extend control over documents even after they have been opened and decrypted. They can prevent actions such as copying, editing, screen grabs, or unauthorized printing, and allow for remote revocation of access, ensuring persistent control over intellectual property.

The sheer number and diversity of these tools and techniques highlight that a comprehensive file security system is not a single product but an integrated ecosystem of interoperating solutions. This implies inherent complexity in deployment, configuration, and ongoing management, requiring a holistic security architecture that ensures seamless interoperability and effective protection across the entire data lifecycle.

2.4. Advantages and Limitations of the used security tools

The effectiveness of file security tools and techniques must be critically evaluated against their practical implications, recognizing that while they offer significant protective capabilities, they also introduce challenges.
General Advantages of File Protection Mechanisms:

- **Robust Data Security:** These tools are instrumental in preventing unauthorized access, data breaches, and tampering, thereby safeguarding sensitive information effectively.

- **Regulatory Compliance:** Implementing these mechanisms is essential for meeting a wide array of regulatory requirements, including GDPR, HIPAA, PCI-DSS, ISO 9001, and FDA 21 CFR Part 11. Adherence helps organizations avoid substantial financial penalties and reputational damage.

- **Business Continuity:** By providing capabilities for rapid data recovery and minimizing downtime, file protection mechanisms contribute significantly to an organization's operational resilience in the face of data loss incidents.

- **Increased Productivity & Collaboration:** Secure and controlled access to files ensures that authorized users can work efficiently and collaborate without compromising data integrity or confidentiality.

- **Enhanced Reputation & IP Protection:** A demonstrated commitment to data security builds trust with customers and partners, while also safeguarding an organization's intellectual property and competitive advantage.

General Limitations of File Protection Systems:

- **Performance Overhead:** Many file protection mechanisms, such as real-time encryption, extensive access control checks, and continuous auditing, demand significant processing power and system resources. This can lead to noticeable impacts on system performance, potentially slowing down file access and overall processing times.
- **Complexity:** The implementation, configuration, and ongoing management of robust file security systems often require specialized knowledge and expertise. This complexity increases the potential for errors and misconfigurations, which can inadvertently create security vulnerabilities.

- **Compatibility Issues:** Certain security mechanisms may not be fully compatible with all file types, legacy applications, or diverse operating system environments, leading to limitations in their applicability or requiring costly workarounds.

- **Cost:** The financial investment required for file protection, encompassing software licenses, specialized hardware, and the recruitment or training of skilled cybersecurity personnel, can be substantial, posing a particular challenge for smaller organizations with constrained budgets.

- **User Frustration:** Overly stringent security controls, such as requirements for complex passwords, frequent multi-factor authentication prompts, or highly restricted access, can lead to user frustration. If not balanced with usability, these measures can negatively impact productivity or even encourage users to bypass security protocols for convenience

- **Human Factor Vulnerabilities:** Despite advanced technical controls, file security remains susceptible to human elements. Social engineering attacks, accidental human error, or malicious insider actions continue to pose significant risks, underscoring that technology alone cannot provide complete immunity.

Specific Tool/Technique Considerations:

- **File Shield/Antivirus:** While providing essential real-time protection, choosing to "Scan all files" (as opposed to recommended extensions) can significantly impact system performance due to increased resource consumption.[10] Effective deployment necessitates careful tuning of sensitivity and exclusions to balance security with operational efficiency.

- **Access Control Lists (ACLs):** While ACLs offer the advantage of easily determining who can access a specific object and simplifying the revocation of all access to that object, they present scalability challenges. It can be difficult to ascertain a specific user's overall access rights across numerous objects, and similarly challenging to revoke a user's rights across all objects they have access to. This highlights a practical limitation for granular management in large-scale environments.

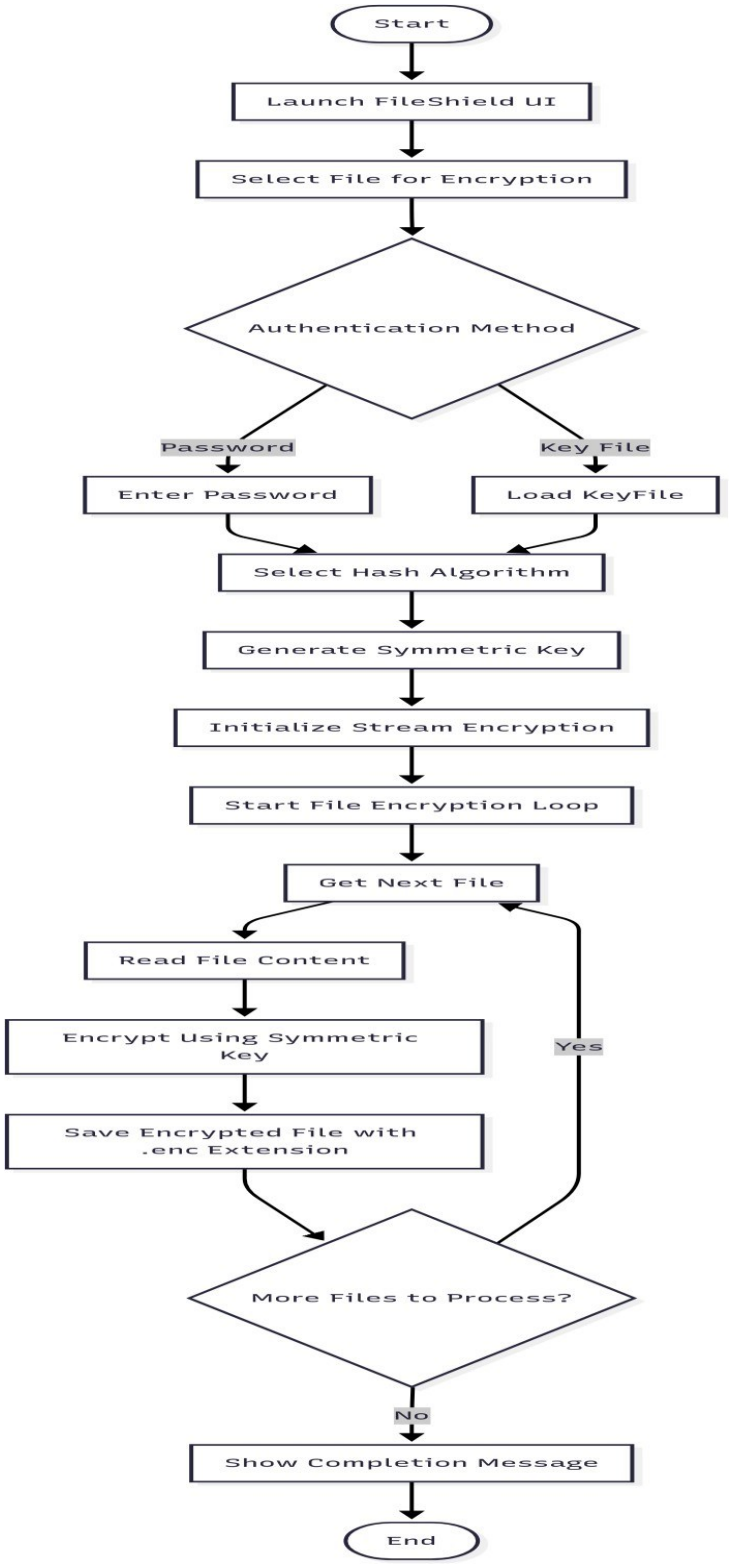
- **Encryption:** While foundational for confidentiality, pure file encryption does not inherently control what happens to the data *after* it has been decrypted by an authorized recipient. This limitation necessitates the use of Digital Rights Management (DRM) to enforce post-decryption controls. Furthermore, the secure management of encryption keys remains a critical and complex challenge.

- **AI/ML in Security:** The integration of Artificial Intelligence and Machine Learning offers advanced capabilities for threat detection and automation. However, this also introduces new challenges such as algorithmic bias, transparency concerns in decision-making, and the emergence of new AI-specific security vulnerabilities that could be exploited.

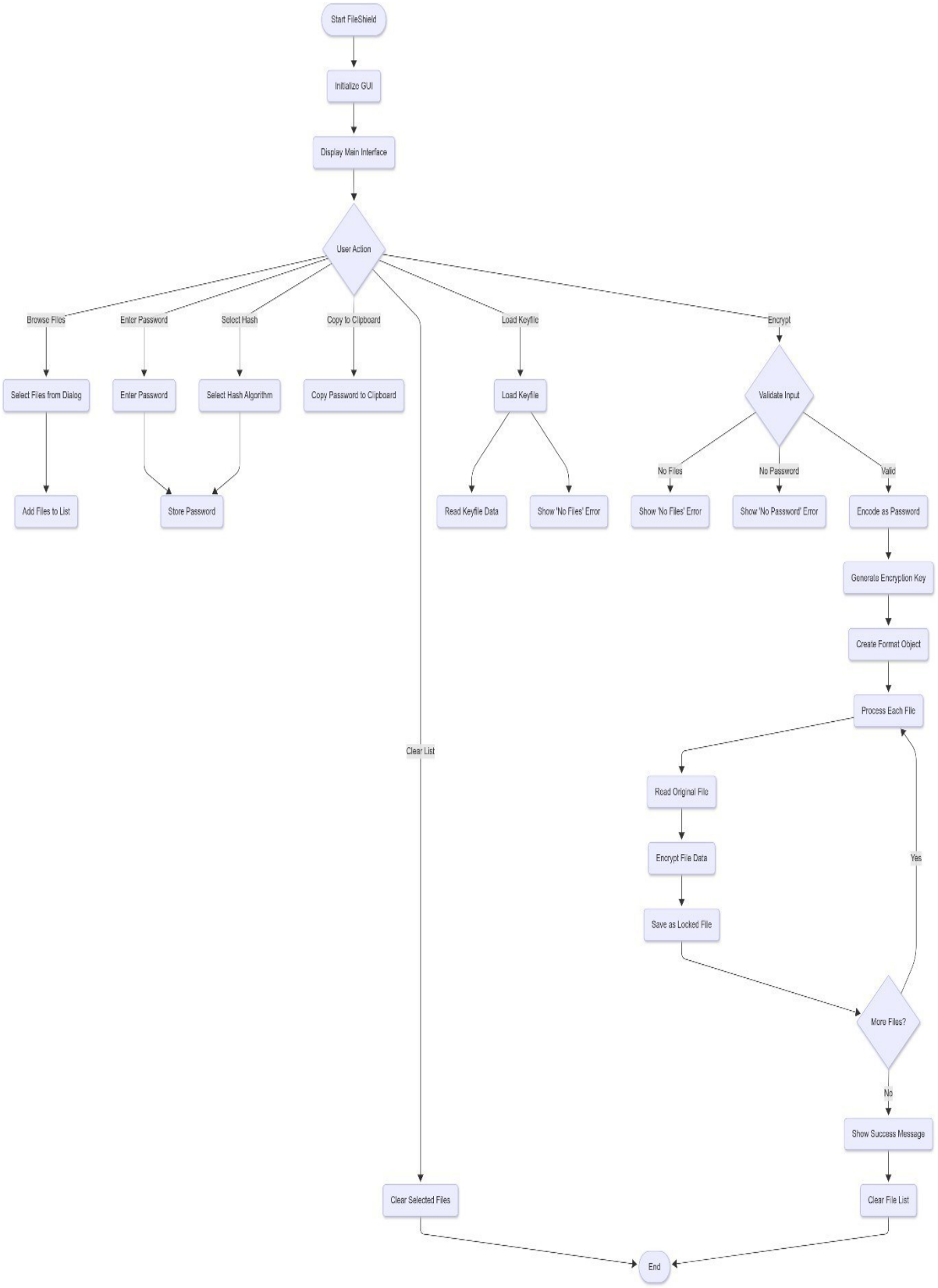
3. System Design

3. System Design

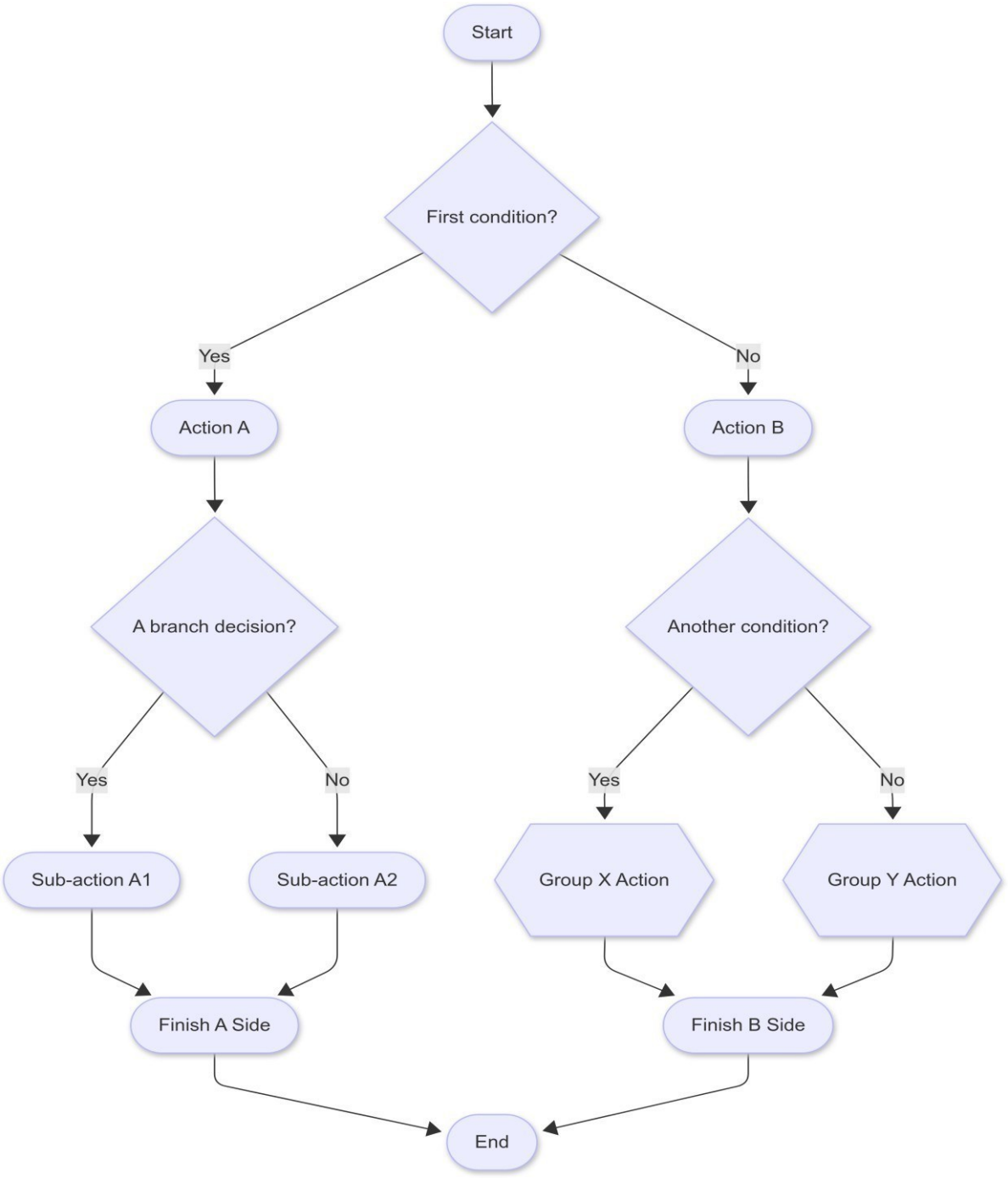
3.1. Flow chart



3.11 Admi Panel



3.12 Client Panel



3.13 Admin & User panle

3.2 Dataset details

1. Users Data Structure (users_data.json)

Purpose: Stores user account information with role-based access control

Schema:

json

```
{
  "username": {
    "password": "hashed_password_sha256",
    "role": "admin|user",
    "timestamp": "YYYY-MM-DD HH:MM:SS"
  }
}
```

Example

```
{
  "sagar": {
    "password": "50e8830d12c25d0587249805ddd7ff4916bcd094e6604133b8702c7f9f886c3f",
    "role": "admin",
    "timestamp": "2025-08-08 10:20:55"
  }
```

Field Descriptions:

- username: Unique identifier (string)
- password: SHA-256 hash of user password (64-character hex string)
- role: User privilege level ("admin" or "user")
- timestamp: Account creation date and time

2. System Logs Structure (system_logs.json)

Purpose: Records all system activities for security monitoring and audit trails

Schema:

json

```
  {
    "timestamp": "YYYY-MM-DD HH:MM:SS",
    "username": "string",
    "action": "ACTION_TYPE",
    "details": "action_description"
  }
```

Sample Dataset:

json

```
  {
    "timestamp": "2025-08-08 10:29:20",
    "username": "testuser",
    "action": "CLIENT_LOGIN",
    "details": "User accessed client interface"
  }
```

Action Types:

- Authentication: CLIENT_LOGIN, CLIENT_LOGOUT, ADMIN_LOGIN, ADMIN_LOGOUT
- File Operations: FILE_SELECTED, FILE_ENCRYPTED, FILE_DECRYPTED, FILE_SELECT_ERROR
- File Management: ENCRYPTED_FILE_SAVED, DECRYPTED_FILE_SAVED, FILE_INFO_VIEWED
- Security: KEY_GENERATED, ENCRYPTION_FAILED, DECRYPTION_FAILED
- Administration: DATA_REFRESH, DATA_EXPORT, LOGS_CLEARED

3. Files Data Structure (files_data.json)

Purpose: Tracks user file operations and encryption metadata

Schema:

json

```
{
  "username": {
    "last_file": "filename",
    "last_accessed": "ISO_timestamp",
    "encrypted_data": "truncated_encrypted_content",
    "encryption_timestamp": "ISO_timestamp",
    "last_decryption": "ISO_timestamp",
    "key_generated": "ISO_timestamp"
  }
}
```

Sample Dataset:

json

```
{
  "testuser": {
    "last_file": "sagar.enc",
    "last_accessed": "2025-08-08T09:35:10.391514",
    "encrypted_data": "gAAAAABolXdIqjxBYWFxiuCjr0BGoaWsnENMsQLhLLFMxth6C0-
rGvZtvLzkGF1HWkLBckL-NdERegBFk4vREfBlsdv0t7FB1hXC...",
    "encryption_timestamp": "2025-08-08T09:34:24.907446",
    "last_decryption": "2025-08-08T09:35:16.670046"
  }
```

Field Descriptions:

- last_file: Name of the most recently processed file
- last_accessed: ISO timestamp of last file access
- encrypted_data: First 100 characters of Fernet-encrypted content
- encryption_timestamp: When file was last encrypted
- last_decryption: When file was last decrypted
- key_generated: When user's encryption key was generated

4. Active Sessions Structure (active_sessions.json)

Purpose: Tracks currently logged-in users and session information

Schema:

json
```
{
  "username": {
    "user_type": "string",
    "login_time": "ISO_timestamp",
    "last_activity": "ISO_timestamp",
    "session_id": "unique_session_identifier"
  }
}
```

Sample Dataset:

json

```
{
  "ADMIN": {
    "user_type": "Administrator",
    "login_time": "2025-08-08T10:37:29.123456",
    "last_activity": "2025-08-08T10:54:29.876543",
    "session_id": "admin_20250808_103729"
  }
```

Field Descriptions:

- user_type: Role designation ("Administrator" or "Client User")
- login_time: Session start time (ISO format)
- last_activity: Most recent user action timestamp
- session_id: Unique identifier for tracking session

Application Architecture Dataset

User Interface Components

python

```
# Login System Components
LOGIN_FIELDS = {
    "username_entry": "CTkEntry widget for username input",
    "password_entry": "CTkEntry widget with password masking",
    "login_button": "Authentication trigger",
    "register_button": "New user account creation"
}

# Client Interface Tabs
CLIENT_TABS = {
    "original_content": "Display selected file content",
    "encrypted_content": "Show Fernet-encrypted data",
    "decrypted_content": "Display decrypted file content"
}

# Admin Interface Tabs
ADMIN_TABS = {
    "dashboard": "System statistics and recent activity",
    "users_data": "User account information with password hashes",
    "system_logs": "Complete activity audit trail",
    "files_data": "User file operations metadata",
    "active_sessions": "Currently logged-in user tracking"
```

```
}
```

Encryption Implementation Dataset

```
python
# Cryptography Configuration
ENCRYPTION_SETTINGS = {
    "algorithm": "Fernet (AES 128 in CBC mode)",
    "key_derivation": "PBKDF2HMAC with SHA-256",
    "iterations": 100000,
    "salt_method": "Username-based deterministic salt",
    "encoding": "URL-safe base64"
}

# File Type Support
SUPPORTED_FILES = [
    "text_files": ["*.txt", "*.md", "*.log"],
    "documents": ["*.doc", "*.docx", "*.pdf"],
    "spreadsheets": ["*.xlsx", "*.xls", "*.csv"],
    "presentations": ["*.pptx", "*.ppt"],
    "images": ["*.jpg", "*.jpeg", "*.png", "*.gif", "*.bmp"],
    "videos": ["*.mp4", "*.avi", "*.mkv", "*.mov"],
    "audio": ["*.mp3", "*.wav", "*.flac", "*.aac"],
    "archives": ["*.zip", "*.rar", "*.7z", "*.tar"],
    "code": ["*.py", "*.js", "*.html", "*.css", "*.java", "*.cpp"]
]
```

System Statistics Dataset

```
python

# Dashboard Metrics
SYSTEM_METRICS = {
    "total_registered_users": 4,
    "active_sessions_count": 3,
    "total_system_logs": 147,
    "files_processed_today": 12,
    "encryption_operations": 23,
    "decryption_operations": 18,
    "admin_actions": 5,
    "failed_login_attempts": 2
}

# Security Events
SECURITY_EVENTS = {
    "successful_authentications": 15,
    "failed_authentications": 2,
    "admin_panel_accesses":  3,
    "file_encryptions": 23,
    "file_decryptions": 18,
    "key_generations": 4,
    "suspicious_activities": 0
}
```

Usage Patterns and Sample Workflows

Typical User Session Flow

1. Authentication: User enters credentials → System validates against users_data.json
2. Session Creation: Active session recorded in active_sessions.json
3. File Selection: User chooses file → Logged in system_logs.json
4. Encryption/Decryption: Cryptographic operations → Metadata saved to files_data.json
5. Logout: Session terminated → Cleanup of active session data

Admin Monitoring Workflow

1. Admin Login: Privileged access → Session tracking begins
2. Data Refresh: Load all system data files → Display current metrics
3. User Oversight: Review user activities and file operations
4. Log Analysis: Examine system logs for security events
5. Data Export: Generate system backup → Create timestamped export file

4.
Development

4. Development

This chapter delves into the practical aspects of implementing a file protection system, covering specific script examples for core functionalities, user interface design considerations for optimal usability, and the crucial methodologies for rigorously testing the system's security posture.

4.1 Script details / Source code

It is divided into three main components:

1. **Authentication & User Management** (login1.py)
2. **Client Interface** for users to encrypt/decrypt files (client.py)
3. **Admin Interface** for monitoring system activity (admin.py)

Additionally, it uses **JSON files** (users_data.json, system_logs.json, files_data.json) for storage.

---

1. Authentication System – login1.py

Purpose: Handles user registration, login, and launching of role-based interfaces.

Key Features:

- **User Management**
  - Stores users in users_data.json.
  - Passwords hashed with **SHA-256**.
  - Automatically creates a default **admin user** (sagar/devprit).
- **Registration**
  - Validates username, password strength, and uniqueness.
  - Saves user data (username, password hash, role, timestamp).
- **Login**
  - Verifies credentials.
  - Routes users based on role:
    - **Admin → AdminInterface**
    - **User → ClientInterface**
- **UI**
  - Built with **CustomTkinter**.
  - Cybersecurity-themed design with dark mode.

Code :-

Core Application Structure

python

```
class CyberSecurityApp:
    def __init__(self):
        # File-based user storage
        self.users_file = "users_data.json"
        self.load_users_from_file()

        # Add admin user if not exists
        self.create_admin_user()

        # UI Setup
        ctk.set_appearance_mode("dark")
        ctk.set_default_color_theme("green")
```

```
self.root = ctk.CTk()
        self.root.title("CyberSec Authentication System")
        self.root.geometry("500x600")
        self.root.configure(fg_color="#0a0a0a")

 Current user info
        self.current_user = None
        self.current_role = None
```

Data Persistence Methods

```
python
def load_users_from_file(self):
    """Load users from JSON file"""
    try:
        if os.path.exists(self.users_file):
            with open(self.users_file, 'r') as f:
                self.users_data = json.load(f)
            print(f"Debug - Loaded {len(self.users_data)} users from file")
        else:
            self.users_data = {}
            print("Debug - Created new users data structure")
    except Exception as e:
        print(f"Debug - Error loading users file: {e}")
        self.users_data = {}

def save_users_to_file(self):
    """Save users to JSON file"""
    try:
        with open(self.users_file, 'w') as f:
            json.dump(self.users_data, f, indent=2)
        print("Debug - Users data saved to file successfully")
    except Exception as e:
        print(f"Debug - Error saving users file: {e}")
```

Security Functions

```
python
def hash_password(self, password):
    return hashlib.sha256(password.encode()).hexdigest()

def create_admin_user(self):
    """Create admin user 'sagar' with password 'devprit' if not exists"""
    admin_username = "sagar"
    admin_password = "devprit"

    if admin_username not in self.users_data:
        hashed_pw = self.hash_password(admin_password)
        timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")

        self.users_data[admin_username] = {
            "password": hashed_pw,
            "role": "admin",
            "timestamp": timestamp
        }
```

```
 self.save_users_to_file()
        print("Debug - Admin user 'sagar' created successfully")
```

User Registration

python

```
def register_user(self):
    """Handle user registration"""
    username = self.reg_username.get().strip()
    password = self.reg_password.get()
    confirm = self.reg_confirm.get()

    # Validation
    if not username or not password or not confirm:
        messagebox.showerror("Error", "All fields are required")
        return

    if password != confirm:
        messagebox.showerror("Error", "Passwords do not match")
        return

    if len(password) < 3:
        messagebox.showerror("Error", "Password must be at least 3 characters long")
        return

    if username in self.users_data:
        messagebox.showerror("Error", "Username already exists")
        return

    try:
        hashed_pw = self.hash_password(password)
        timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")

        # Add user to data structure
        self.users_data[username] = {
            "password": hashed_pw,
            "role": "user",
            "timestamp": timestamp
        }

        # Save to file
        self.save_users_to_file()

        messagebox.showinfo("Success", "Registration successful!\nPlease login with your credentials.")
        self.show_login_page()

    except Exception as e:
        messagebox.showerror("Error", f"Registration failed: {str(e)}")
```

Login Authentication

python

```
def login_user(self):
    """Handle user login and route to appropriate interface"""
    username = self.login_username.get().strip()
    password = self.login_password.get()

if not username or not password:
        messagebox.showerror("Error", "Username and password are required")
        return

    try:
        # Check if user exists in file data
        if username not in self.users_data:
            messagebox.showerror("Error", "User not found")
            return

        user_data = self.users_data[username]
        hashed_pw = self.hash_password(password)

        if user_data["password"] == hashed_pw:
            self.current_user = username
            self.current_role = user_data["role"]

            # Route to appropriate interface based on role
            if self.current_role == "admin":
                messagebox.showinfo("Admin Login", f"Admin login successful!\nWelcome, {username}")
                self.launch_admin_interface()
            else:
                messagebox.showinfo("User Login", f"User login successful!\nWelcome, {username}")
                self.launch_client_interface(username)

        else:
            messagebox.showerror("Error", "Invalid password")

    except Exception as e:
        messagebox.showerror("Error", f"Login failed: {str(e)}")
```

Interface Launching

python

```
def launch_client_interface(self, username):
    """Launch the client interface for regular users"""
    if ClientInterface is None:
        messagebox.showerror("Error", "Client interface is not available. Make sure client.py exists.")
        return

    try:
        # Hide the main login window
        self.root.withdraw()



 # Launch client interface
        client = ClientInterface(self, username)
        client.run()

        # Show the main window again when client closes
        self.root.deiconify()

    except Exception as e:
        messagebox.showerror("Error", f"Failed to launch client interface: {str(e)}")
        self.root.deiconify()
```

```
def launch_admin_interface(self):
    """Launch the admin interface for admin users"""
    if AdminInterface is None:
        messagebox.showerror("Error", "Admin interface is not available. Make sure admin.py exists.")
        return

    try:
        # Hide the main login window
        self.root.withdraw()

        # Launch admin interface
        admin = AdminInterface(self)
        admin.run()

        # Show the main window again when admin closes
        self.root.deiconify()

    except Exception as e:
        messagebox.showerror("Error", f"Failed to launch admin interface: {str(e)}")
        self.root.deiconify()
```

2. Client Interface – client.py

Purpose: Allows authenticated users to **encrypt, decrypt, and manage files**.

Core Functionalities:

- **Encryption/Decryption**
  - Uses **Fernet (AES-128 in CBC mode with HMAC)**.
  - Keys derived deterministically from username using **PBKDF2-HMAC-SHA256**.
- **File Operations**
  - Select files (supports text, images, PDFs, media, etc.).
  - Encrypt/Decrypt content.
  - Save encrypted/decrypted files.
  - View file metadata (size, modification date, status).
- **Logs & Tracking**
  - Logs every action in system_logs.json.
  - Maintains per-user file activity in files_data.json.
- **UI**
  - Split into:
    - **Control Panel** → File selection, encryption, decryption, logout.
    - **File Panel** → Tabs for Original, Encrypted, and Decrypted content.
    - **Status Bar** → Shows real-time status.

Code: -

Client Initialization

python

```
class ClientInterface:
    def __init__(self, main_app, username):
        self.main_app = main_app
        self.current_user = username
        self.users_file = "users_data.json"
        self.logs_file = "system_logs.json"
        self.files_data_file = "files_data.json"

        self.user_key = None
        self.selected_file_path = None

        # Load data
        self.load_logs()
        self.load_files_data()
        self.generate_user_key()

        # UI Setup
        ctk.set_appearance_mode("dark")
        ctk.set_default_color_theme("green")
```

```
self.root = ctk.CTk()
        self.root.title(f"SecureVault Client - {username}")
        self.root.geometry("1200x800")
        self.root.configure(fg_color="#1a1a2e")
```

Encryption Key Generation

```
python
def generate_user_key(self):
    """Generate encryption key for user (deterministic based on username using PBKDF2)"""
    from cryptography.hazmat.primitives import hashes
    from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC

    # Create a user-specific key based on username (deterministic)
    salt = self.current_user.encode()[:16].ljust(16, b'0')
    kdf = PBKDF2HMAC(
        algorithm=hashes.SHA256(),
        length=32,
        salt=salt,
        iterations=100000,
    )
    key = base64.urlsafe_b64encode(kdf.derive(self.current_user.encode()))
    self.user_key = key
```

File Selection with Multi-Format Support

```
python
def select_file(self):
    """Select file for encryption/decryption"""
    file_path = filedialog.askopenfilename(
        title="Select file to encrypt/decrypt",
        filetypes=[
            ("All files", "*.*"),
            ("Text files", "*.txt"),
            ("Document files", "*.doc *.docx"),
            ("PDF files", "*.pdf"),
            ("Image files", "*.jpg *.jpeg *.png *.gif *.bmp"),
            ("Video files", "*.mp4 *.avi *.mkv *.mov"),
            ("Audio files", "*.mp3 *.wav *.flac *.aac"),
            ("Archive files", "*.zip *.rar *.7z *.tar"),
            ("Code files", "*.py *.js *.html *.css *.java *.cpp"),
            ("Excel files", "*.xlsx *.xls"),
            ("PowerPoint files", "*.pptx *.ppt")
```

```
        ]
    )


if file_path:
        self.selected_file_path = file_path
        filename = os.path.basename(file_path)
        self.selected_file_label.configure(text=f"File:  {filename}")

        # Load file content
        try:
            # Try to read as text first, fall back to binary for non-text files
            try:
                with open(file_path, 'r', encoding='utf-8') as file:
                    content = file.read()
            except UnicodeDecodeError:
                # For binary files, read as binary and encode to base64 for display
                with open(file_path, 'rb') as file:
                    binary_content = file.read()
                    content = f"[BINARY FILE - {len(binary_content)} bytes]\n\nBase64
representation:\n{base64.b64encode(binary_content).decode()[:500]}..."

            self.original_text.delete("1.0", "end")
            self.original_text.insert("1.0", content)

            # Switch to original content tab
            self.tabview.set("Original Content")

            self.status_label.configure(text=f"File loaded: {filename}")

        except Exception as e:
            messagebox.showerror("Error", f"Failed to load file: {str(e)}")
```

File Encryption

```
python
def encrypt_file(self):
    """Encrypt the selected file content"""
    content = self.original_text.get("1.0", "end").strip()

    if not content:
        messagebox.showerror("Error", "No content to encrypt. Please select a file first.")
        return
```

```
    try:
        # Create Fernet cipher with user's key
        cipher = Fernet(self.user_key)



    # Encrypt the content - Fernet returns a URL-safe base64 token (bytes)
        encrypted_token = cipher.encrypt(content.encode())
        encrypted_text = encrypted_token.decode()  # store/display as string

        # Display encrypted content
        self.encrypted_text.delete("1.0", "end")
        self.encrypted_text.insert("1.0", encrypted_text)

        # Switch to encrypted tab
        self.tabview.set("Encrypted  Content")

         self.status_label.configure(text="File  encrypted  successfully!")

        # Save encryption info
        if self.current_user not in self.files_data:
            self.files_data[self.current_user] = {}

        self.files_data[self.current_user]['encrypted_data'] = encrypted_text[:100] + "..." if len(encrypted_text) > 100 else
encrypted_text
        self.files_data[self.current_user]['encryption_timestamp'] = datetime.datetime.now().isoformat()
        self.save_files_data()

        self.log_action(self.current_user, "FILE_ENCRYPTED", "File content encrypted successfully")

        messagebox.showinfo("Success", "File encrypted successfully!")

    except Exception as e:
        messagebox.showerror("Error", f"Encryption failed: {str(e)}")
        self.log_action(self.current_user, "ENCRYPTION_FAILED", f"Encryption failed: {str(e)}")
```

File Decryption

python

```
def decrypt_file(self):
    """Decrypt the encrypted content"""
    encrypted_content = self.encrypted_text.get("1.0", "end").strip()
```

```
    if not encrypted_content:
        messagebox.showerror("Error", "No encrypted content to decrypt.")
        return

try:
        # Create Fernet cipher with user's key
        cipher = Fernet(self.user_key)

        # encrypted_content is a URL-safe base64 token string from Fernet
        decrypted_content = cipher.decrypt(encrypted_content.encode()).decode()

        # Display decrypted content
        self.decrypted_text.delete("1.0", "end")
        self.decrypted_text.insert("1.0", decrypted_content)

        # Switch to decrypted tab
        self.tabview.set("Decrypted Content")

        self.status_label.configure(text="File decrypted successfully!")

        # Save decryption info
        if self.current_user not in self.files_data:
            self.files_data[self.current_user] = {}

        self.files_data[self.current_user]['last_decryption'] = datetime.datetime.now().isoformat()
        self.save_files_data()

        self.log_action(self.current_user, "FILE_DECRYPTED", "File content decrypted successfully")

        messagebox.showinfo("Success", "File decrypted successfully!")

    except Exception as e:
        messagebox.showerror("Error", f"Decryption failed: {str(e)}")
        self.log_action(self.current_user, "DECRYPTION_FAILED", f"Decryption failed: {str(e)}")
```

File Save Operations

python

```
def save_encrypted_file(self):
    """Save encrypted content to file"""
    encrypted_content = self.encrypted_text.get("1.0", "end").strip()

    if not encrypted_content:
        messagebox.showerror("Error", "No encrypted content to save.")
```

```
        return

    file_path = filedialog.asksaveasfilename(
        title="Save encrypted file",
        defaultextension=".enc",
        filetypes=[("Encrypted files", "*.enc"), ("Text files", "*.txt"), ("All files", "*.*")]
    )



 if file_path:
        try:
            with open(file_path, 'w', encoding='utf-8') as file:
                file.write(encrypted_content)

            filename = os.path.basename(file_path)
            self.status_label.configure(text=f"Encrypted file saved: {filename}")
            self.log_action(self.current_user, "ENCRYPTED_FILE_SAVED", f"Encrypted file saved: {filename}")
            messagebox.showinfo("Success", "Encrypted file saved successfully!")

        except Exception as e:
            messagebox.showerror("Error", f"Failed to save file: {str(e)}")

def save_decrypted_file(self):
    """Save decrypted content to file"""
    decrypted_content = self.decrypted_text.get("1.0", "end").strip()

    if not decrypted_content:
        messagebox.showerror("Error", "No decrypted content to save.")
        return

    file_path = filedialog.asksaveasfilename(
        title="Save decrypted file",
        defaultextension=".txt",
        filetypes=[("Text files", "*.txt"), ("All files", "*.*")]
    )

    if file_path:
        try:
            with open(file_path, 'w', encoding='utf-8') as file:
                file.write(decrypted_content)
```

```
            filename = os.path.basename(file_path)
            self.status_label.configure(text=f"Decrypted file saved: {filename}")
            self.log_action(self.current_user, "DECRYPTED_FILE_SAVED", f"Decrypted file saved: {filename}")
            messagebox.showinfo("Success", "Decrypted file saved successfully!")

        except Exception as e:
            messagebox.showerror("Error", f"Failed to save file: {str(e)}")
```

System Logging

```
python
def log_action(self, username, action, details):
    """Log user actions"""
    timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    log_entry = {
        "timestamp": timestamp,
        "username": username,
        "action": action,
        "details": details
    }
    # ensure logs_data exists
    if not hasattr(self, 'logs_data'):
        self.logs_data = []
    self.logs_data.append(log_entry)
    self.save_logs()
```

3. Admin Interface – admin.py

Purpose: Provides administrators with **system monitoring and management tools**.
Capabilities:

- **Dashboard**
  - Displays total users, logs, active sessions, and recent activities.
- **Users Data**
  - Shows registered users (with role and password hashes).
- **System Logs**
  - Displays chronological logs of user/admin activities.
- **Files Data**
  - Shows encryption/decryption activity and metadata for all users.
- **Controls**
  - Refresh all data.
  - Export system data snapshot (admin_export_YYYYMMDD_HHMMSS.json).
  - Clear logs.
  - Return to main login.

Code: -

Admin Interface Initialization

```
python
class AdminInterface:
    def __init__(self, main_app):
        self.main_app = main_app
        self.users_file = "users_data.json"
        self.logs_file = "system_logs.json"
        self.files_data_file = "files_data.json"

        # Load data
        self.load_logs()
        self.load_files_data()

        # UI Setup
        ctk.set_appearance_mode("dark")
        ctk.set_default_color_theme("green")

        self.root = ctk.CTk()
        self.root.title("SecureVault Admin Dashboard")
        self.root.geometry("1400x900")
        self.root.configure(fg_color="#0a0a0a")
```

Data Management Methods

```
python
def load_logs(self):
    """Load system logs from file"""
```

```
try:
 if os.path.exists(self.logs_file):
            with open(self.logs_file, 'r') as f:
                self.logs_data = json.load(f)
        else:
            self.logs_data = []
    except Exception as e:
        print(f"Error loading logs: {e}")
        self.logs_data = []

def save_logs(self):
    """Save logs to file"""
    try:
        with open(self.logs_file, 'w') as f:
            json.dump(self.logs_data, f, indent=2)
    except Exception as e:
        print(f"Error saving logs: {e}")

def load_files_data(self):
    """Load files data from file"""
    try:
        if os.path.exists(self.files_data_file):
            with open(self.files_data_file, 'r') as f:
                self.files_data = json.load(f)
        else:
            self.files_data = {}
    except Exception as e:
        print(f"Error loading files data: {e}")
        self.files_data = {}
```

User Data Display

```
python
def load_users_data(self):
    """Load and display users data"""
    try:
        if os.path.exists(self.users_file):
            with open(self.users_file, 'r') as f:
                users_data = json.load(f)


            self.users_text.delete("1.0",  "end")
```

```
            header = "=" * 80 + "\n"
            header += "USERS DATABASE - ADMIN VIEW (INCLUDING PASSWORD HASHES)\n"

header += "=" * 80 + "\n\n"
self.users_text.insert("end", header)

            for username, user_info in users_data.items():
                user_display = f"USERNAME: {username}\n"
                user_display += f"ROLE: {user_info.get('role', 'user').upper()}\n"
                user_display += f"PASSWORD HASH: {user_info.get('password', 'N/A')}\n"
                user_display += f"REGISTERED: {user_info.get('timestamp', 'N/A')}\n"
                user_display += "-" * 60 + "\n\n"
                self.users_text.insert("end",  user_display)

            return len(users_data)
        else:
            self.users_text.delete("1.0", "end")
            self.users_text.insert("end", "No users data file found.")
            return 0
    except Exception as e:
        self.users_text.delete("1.0", "end")
        self.users_text.insert("end", f"Error loading users data: {str(e)}")
        return 0
```

System Logs Display

```
python
def load_logs_display(self):
    """Load and display system logs"""
    self.logs_text.delete("1.0", "end")

    if not self.logs_data:
        self.logs_text.insert("end", "No system logs available.")
        return 0

    header = "=" * 80 + "\n"
    header += "SYSTEM ACTIVITY LOGS\n"
    header += "=" * 80 + "\n\n"
    self.logs_text.insert("end", header)
    # Sort logs by timestamp (newest first)
    sorted_logs = sorted(self.logs_data, key=lambda x: x.get('timestamp', ''), reverse=True)
    for log in sorted_logs:
        log_display = f"[{log.get('timestamp', 'N/A')}] "
        log_display += f"{log.get('username', 'SYSTEM')} - "
```

```
            log_display += f"{log.get('action', 'UNKNOWN')}: "
            log_display += f"{log.get('details', 'No details')}\n"
            self.logs_text.insert("end", log_display)
        return len(self.logs_data)
```

Dashboard Statistics

```
python
def update_dashboard_stats(self):
    """Update dashboard statistics"""
    # Update user count
    total_users = self.load_users_data()
    self.total_users_label.configure(text=str(total_users))

    # Update logs count
    total_logs = len(self.logs_data)
    self.total_logs_label.configure(text=str(total_logs))

    # Update recent activity
    self.activity_text.delete("1.0",  "end")

    activity_header = "=== RECENT SYSTEM ACTIVITY ===\n\n"
    self.activity_text.insert("end", activity_header)

    if self.logs_data:
        # Show last 10 logs
        recent_logs = sorted(self.logs_data, key=lambda x: x.get('timestamp', ''), reverse=True)[:10]

        for log in recent_logs:
            activity_line = f"[{log.get('timestamp', 'N/A')}] "
            activity_line += f"{log.get('username', 'SYSTEM')} - "
            activity_line += f"{log.get('action', 'UNKNOWN')}: "
            activity_line += f"{log.get('details', 'No details')}\n"
            self.activity_text.insert("end", activity_line)
    else:
        self.activity_text.insert("end", "No recent activity recorded.")
```

Data Export Functionality

```
python
def export_data(self):
    """Export system data"""
    try:
        timestamp = datetime.datetime.now().strftime("%Y%m%d_%H%M%S")
```

```
        # Load current users data
        users_data = {}
        if os.path.exists(self.users_file):
            with open(self.users_file, 'r') as f:
                users_data = json.load(f)

        export_data = {
            "export_timestamp": datetime.datetime.now().isoformat(),
            "users": users_data,
            "logs": self.logs_data,
            "files": self.files_data
        }

        export_filename = f"admin_export_{timestamp}.json"
        with open(export_filename, 'w') as f:
            json.dump(export_data, f, indent=2)

        messagebox.showinfo("Export Complete", f"Data exported successfully to:\n{export_filename}")
        self.log_action("ADMIN", "DATA_EXPORT", f"Admin exported system data to {export_filename}")

    except Exception as e:
        messagebox.showerror("Export Error", f"Failed to export data: {str(e)}")
```

System Maintenance

```
python
def clear_logs(self):
    """Clear system logs"""
    if messagebox.askyesno("Clear Logs", "Are you sure you want to clear all system logs?"):
        self.logs_data = []
        self.save_logs()
        self.load_logs_display()
        self.update_dashboard_stats()
        messagebox.showinfo("Success", "System logs cleared successfully!")
        self.log_action("ADMIN", "LOGS_CLEARED", "Admin cleared all system logs")

def refresh_all_data(self):
    """Refresh all data"""
    self.load_logs()
    self.load_files_data()
    self.update_dashboard_stats()
```

```
self.load_logs_display()
self.load_files_display()
messagebox.showinfo("Success", "All data refreshed successfully!")
self.log_action("ADMIN", "DATA_REFRESH", "Admin refreshed all system data")
```

Key Libraries and Dependencies

python

```
# Core UI Framework
import customtkinter as ctk
from tkinter import messagebox, filedialog

# Security and Cryptography
import hashlib
from cryptography.fernet import Fernet
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC

# Data Management
import json
import datetime
import os
import base64
```

---

4. Data Files
   - **users_data.json**
     Stores user credentials and metadata.

     Example:

```
{
  "sagar": {
    "password": "50e8830d12c25d0587249805ddd7ff4916bcd094e6604133b8702c7f9f886c3f",
    "role": "admin",
    "timestamp": "2025-08-08 10:20:55"
  }
}
```

system_logs.json

Stores system-wide activity logs.

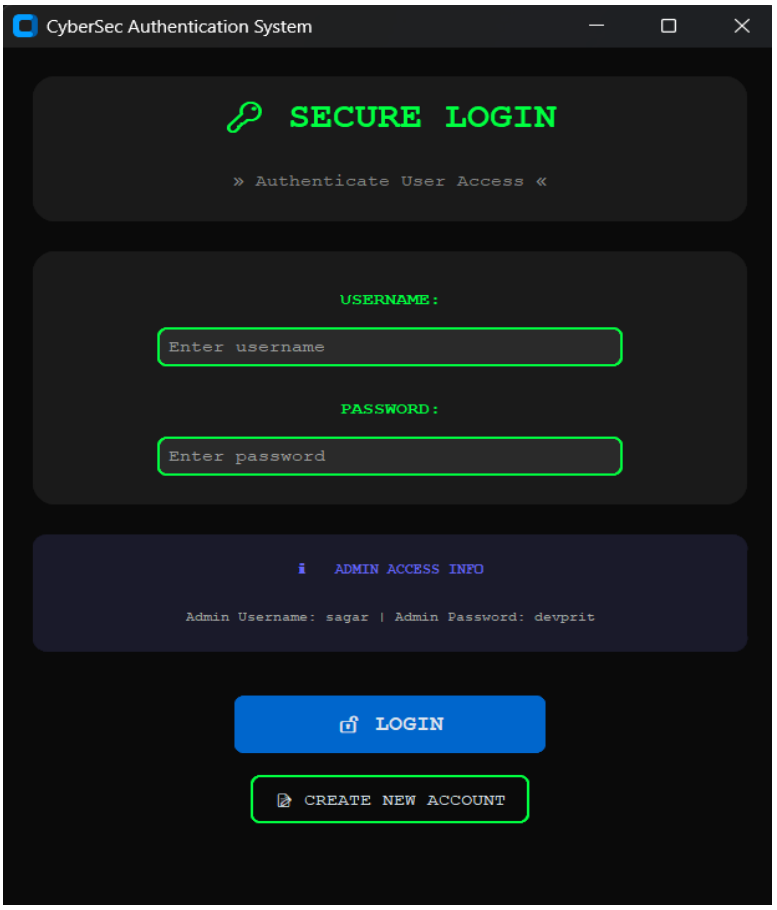
Example:

```
{
  "timestamp": "2025-08-29 09:11:48",
  "username": "ADMIN",
  "action": "DATA_REFRESH",
  "details": "Admin refreshed all system data"
}
```

files_data.json
Stores per-user file operation history (last file accessed, encryption/decryption logs, etc.).

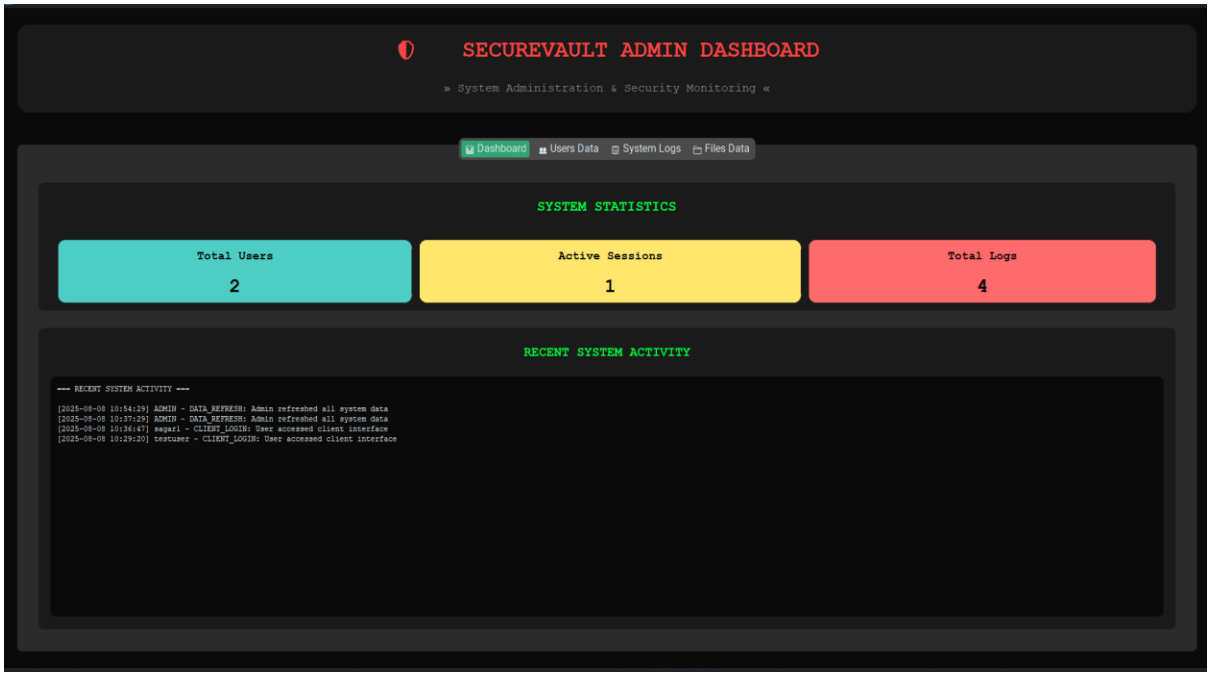
4.2. Screen Shots / UI Design of simulation (if applicable)

- Login Page



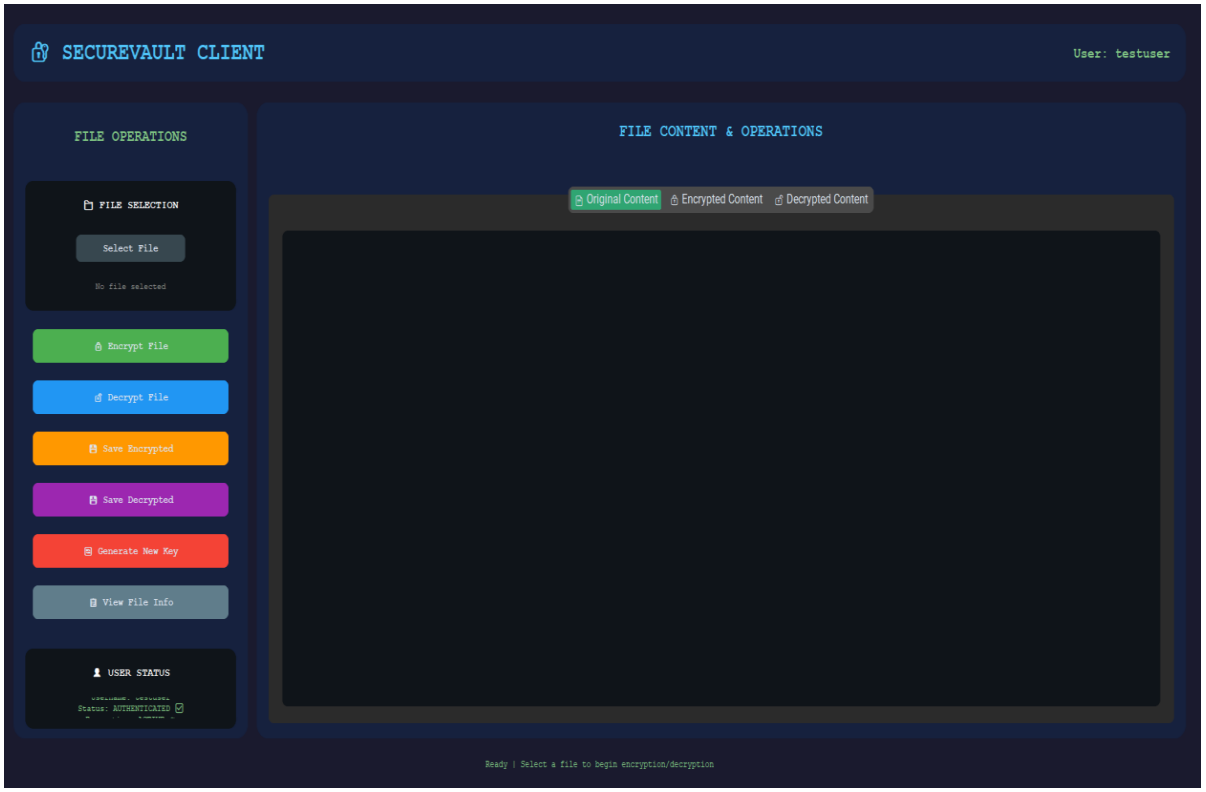
5.11 Login Page

- **Admin Panel**



5.12 Admin Panel

- **Client Panel**



5.13 Client Panel

4.3. Test reports

Rigorous security testing is an indispensable phase in the development lifecycle of any file protection system, ensuring its resilience against evolving cyber threats. This process is not a one-time activity but an ongoing cycle of assessment, identification, and remediation, crucial for adapting to new attack vectors and maintaining a strong security posture. The comprehensive array of security testing methodologies and the structured approach to reporting indicate that file security is a continuous improvement process.
Security Software Testing Methodologies:

An efficient combination of various testing methods is employed to ensure software security:

- **Vulnerability Scanning:** This involves using automated tools to scan software, networks, and applications for known vulnerabilities or weaknesses, such as outdated components, weak passwords, or insecure configurations.
- **Penetration Testing (Pen Testing):** Simulates real-world attacks on the software to identify exploitable vulnerabilities. Conducted by ethical hackers, it probes for flaws related to authentication, authorization, network configurations, and application logic.
- **Risk Assessment:** A systematic process to identify potential risks and threats, evaluate their likelihood and impact, prioritize them based on severity, and develop strategies for mitigation.
- **Ethical Hacking:** Similar to penetration testing, it involves simulating attacks to uncover vulnerabilities that might be missed by other testing methods.
- **Security Scanning:** Utilizes automated tools to scan for common vulnerabilities like SQL injection, cross-site scripting (XSS), and buffer overflow attacks.
- **Posture Assessment:** Involves analyzing the overall security posture by reviewing security policies, procedures, and configurations to identify loopholes.
- **Security Auditing:** A systematic review of the software's architecture, design, and implementation against security policies and regulatory standards (e.g., HIPAA, PCI DSS, ISO 27001) to identify weaknesses and gaps in controls.

Key Attributes for Security Testing:

Effective security testing must validate several critical attributes, often guided by the CIA Triad and extended principles:

- **Confidentiality:** Verifies that sensitive information is protected from unauthorized access through proper encryption and access controls.
- **Integrity:** Ensures data remains accurate and unaltered, preventing unauthorized modification or tampering during storage, transmission, and processing.
- **Authentication:** Evaluates the strength of mechanisms that confirm user identities to prevent unauthorized access.
- **Authorization:** Assesses the effectiveness of access controls, ensuring users can only perform actions permitted by their roles.
- **Availability:** Verifies that systems and applications are operational and accessible when needed, with defenses against denial-of-service attacks.
- **Non-Repudiation:** Ensures that actions taken by users cannot be denied later, typically through audit logs and digital signatures.
- **Resilience:** Assesses the system's ability to withstand and recover from security incidents, minimizing impact and restoring normal operations promptly.

Structure of Security Test Reports:

Security assessment reports, including vulnerability assessment and penetration testing reports, are meticulously structured documents providing a snapshot of the organization's security environment and guiding remediation efforts. They are designed to be comprehensive and actionable, catering to both technical and non-technical stakeholders.

A typical report includes:

- **Executive Summary:** A brief, non-technical overview of the assessment's purpose, scope, key findings, and the overall security posture. This section highlights critical vulnerabilities and their potential impact for company executives.

- **Engagement Summary/Methodology:** Details the scope, timeline, targets of the test, and the specific steps taken to identify and assess vulnerabilities. This includes the tools and techniques used (e.g., vulnerability scanning, penetration testing, network scans, interviews) and the testing environment.

- **Scan Results/Key Findings:** A detailed explanation of how vulnerabilities are organized and categorized. This section lists all identified vulnerabilities, often indexed, with their severity levels (e.g., critical, high, medium, low).

- **Risk Assessment:** Evaluates the potential impact of each vulnerability, assigning a risk level based on the likelihood of exploitation and potential damage. Common Vulnerability Scoring System (CVSS) scores are frequently used for this purpose

- **Recommendations and Remediations:** This crucial section provides actionable recommendations for addressing identified vulnerabilities. These may include technical fixes, software updates, configuration changes, or suggestions for implementing additional security measures.[55] Evidence and examples, such as detailed walkthroughs demonstrating exploits, screenshots, logs, or vulnerability scan reports, are often included to support findings and recommendations.

The structured nature of these reports facilitates a continuous improvement cycle. By systematically identifying weaknesses, prioritizing risks, and providing clear remediation steps, organizations can adapt to evolving threats and continuously enhance their file security posture.

5. Proposed Enhancement

5. Proposed Enhancements

The landscape of file security is continuously evolving, driven by the increasing sophistication of cyber threats and advancements in technology. Future enhancements in file protection systems will largely be shaped by the integration of Artificial Intelligence (AI) and Machine Learning (ML), moving towards more proactive, adaptive, and intelligent security solutions. This represents a significant paradigm shift from traditional signature-based or rule-based security approaches.

Transformative Role of AI/ML in File Security:

AI and ML are poised to revolutionize how file security is managed and automated, addressing the increasing volume and complexity of cyber threats, particularly in real-time detection and automated response.

- **Advanced Threat Detection and Prevention:** AI-driven systems can analyze vast volumes of diverse data—including logs, network traffic, and user behavior—to identify meaningful security incidents.[24] These systems excel at detecting both known threats (e.g., malware, phishing) and previously unseen attacks (zero-day exploits) by identifying subtle anomalies or suspicious patterns that traditional methods might miss.

- **Behavioral Analytics:** AI-powered behavioural analytics solutions continuously monitor user and entity behavior within a network. By detecting deviations from normal patterns, these tools can proactively identify insider threats, compromised accounts, and other security risks. This involves analyzing factors such as user access patterns, device usage, and application behaviour.

- **Automated Incident Response:** AI tools can significantly enhance incident response processes by providing real-time alerts, prioritizing security incidents based on severity, and orchestrating automated response actions. This capability helps organizations mitigate the impact of security breaches more efficiently, reducing response times and minimizing downtime.

- **Vulnerability Management:** AI-powered vulnerability management solutions can autonomously scan networks, systems, and applications to identify potential security vulnerabilities. They prioritize remediation efforts based on risk levels, leveraging machine learning algorithms to analyze historical data, security advisories, and threat intelligence feeds to identify emerging vulnerabilities and recommend appropriate patches or mitigations.

- **Adaptive Access Control:** AI-driven adaptive access control systems represent a significant enhancement to traditional access models. They dynamically adjust user access privileges based on contextual factors such as real-time user behavior, device posture, geographic location, and time of access. By continuously evaluating risk factors, these tools can enforce granular access policies to protect sensitive data and resources from unauthorized access in a highly dynamic environment.

- **Enhanced Malware Detection and Prevention:** AI-powered malware detection solutions leverage machine learning algorithms to analyze file attributes, code behavior, and network traffic patterns. This enables them to detect and block malware infections in real-time, identifying both known malware variants and previously unseen threats, thereby defending against advanced persistent threats (APTs) and zero-day attacks.

- **Advanced Data Loss Prevention (DLP):** AI-driven DLP solutions will become more sophisticated in preventing sensitive data leakage. They will monitor and analyze data flows across endpoints, networks, and cloud services with greater precision, identifying and classifying sensitive data, enforcing data protection policies, and detecting anomalous data transfer activities indicative of potential data exfiltration attempts.

Addressing AI-Specific Challenges:

While AI offers immense potential, its integration also introduces new challenges that must be addressed:

- **Algorithmic Bias:** Ensuring that AI models are fair and do not introduce biases in security decisions is crucial.
- **Transparency Concerns:** Understanding how AI models arrive at their conclusions can be challenging, necessitating efforts towards explainable AI in security.
- **AI Security Vulnerabilities:** AI systems themselves can become targets for attacks, requiring robust security measures to protect the AI models and their training data.

Future enhancements will also focus on privacy-preserving AI techniques, such as differential privacy and homomorphic encryption, which allow for secure data analysis without exposing personal information This ensures that as security systems become more intelligent, they also uphold the highest standards of data privacy and ethical considerations. The continuous evolution of file security will therefore be characterized by a symbiotic relationship between advanced AI capabilities and a renewed focus on privacy and ethical deployment.

6. Conclusion

5. Conclusion

The comprehensive analysis of file shield and advanced file security systems reveals a critical imperative in the contemporary digital landscape: robust file protection is not merely a technical feature but a fundamental pillar of organizational resilience and strategic success. The primary objective of such systems, rooted in the principles of Confidentiality, Integrity, and Availability, extends beyond preventing malware to safeguarding privacy, intellectual property, and brand reputation, while ensuring regulatory compliance.

The problem of unauthorized access is multi-faceted, stemming from a complex interplay of technical vulnerabilities, human factors (including both malicious and non-malicious insiders), and physical security gaps. This necessitates a layered defense strategy, integrating diverse components such as real-time file shields, sensitive data protection, advanced encryption, granular access controls (ACLs, RBAC, ABAC), data loss prevention, file integrity monitoring, and robust backup and recovery solutions. The sheer number and specialized functions of these components underscore that effective file security is an integrated ecosystem, not a monolithic solution.

Project implementation of file security must be viewed as a strategic endeavor, deeply embedded within project management methodologies from inception. This requires meticulous requirement determination, distinguishing between functional and critical non-functional aspects like security, and a proactive approach to managing assumptions and constraints that directly influence project risks.

While the advantages of comprehensive file protection are substantial—ranging from enhanced data security and regulatory adherence to improved business continuity and organizational reputation—these benefits come with inherent trade-offs. Performance overhead, system complexity, compatibility challenges, significant costs, and potential user frustration are practical limitations that must be carefully balanced. This signifies that optimal file security is a dynamic equilibrium, requiring continuous optimization tailored to an organization's specific risk appetite and operational context. The practical challenges associated with managing granular access controls, such as ACLs, further highlight the need for scalable and user-friendly solutions.

Looking forward, the future of file security is inextricably linked with the transformative potential of Artificial Intelligence and Machine Learning. AI/ML will drive advancements in real-time threat detection, behavioral analytics, automated incident response, adaptive access control, and sophisticated malware prevention. However, the successful integration of these technologies will also demand careful consideration of new challenges, including algorithmic bias, transparency, and the security of AI systems themselves.

Ultimately, effective file security is a continuous journey of adaptation, driven by the evolving threat landscape and technological innovation. It demands a holistic approach that combines cutting-edge technical solutions with strong organizational policies, ongoing employee training, and a deep understanding of data flows and their inherent vulnerabilities. By prioritizing these elements, organizations can build resilient defenses that protect their most valuable digital assets and foster enduring trust.

7.
Bibiliography

1. **selimyilbas. (2025).** ***Secure-Vault*** **[Python project]. GitHub. https://github.com/selimyilbas/secure-vault**

2. **gabfl. (n.d.).** ***Vault*** **[Python password manager]. GitHub. https://github.com/gabfl/vault**

3. **Maneesh-Pradeep. (n.d.).** ***secret_vault*** **[Python project]. GitHub. https://github.com/Maneesh-Pradeep/secret_vault**

4. **anthonynsimon. (n.d.).** ***secrets-vault*** **[Python project]. GitHub. https://github.com/anthonynsimon/secrets-vault**

5. **scientific-python. (n.d.).** ***vault-template*** **[Project template]. GitHub. https://github.com/scientific-python/vault-template**

6. **GeeksforGeeks. (2025, June 20).** ***Encrypt and decrypt files using Python*****. GeeksforGeeks. https://www.geeksforgeeks.org/python/encrypt-and-decrypt-files-using-python/**

7. **Fadheli, A. (2024, April).** ***How to encrypt and decrypt files in Python*****. ThePythonCode. https://thepythoncode.com/article/encrypt-decrypt-files-symmetric-python**

8. **Johns, R. (2025, February 21).** ***Build a Python file encryption tool with PyQt (step-by-step)*****. Hackr.io.**

9. **Ali, M. A. (2023, December 5).** ***Unlocking cryptography: A hands-on guide to encrypting and decrypting files using Python*****. Medium.**

10. **mr rockstar. (2024, August 10).** ***Encrypting Python (.py) files: A comprehensive guide*****. ClientArea Blog.**

11. **Malik, K. (2023, November 22).** ***File encryption in Python: An in-depth exploration of symmetric and asymmetric techniques*****. Snyk Blog.**

12. **Titus, A. J., et al. (2018, March 5).** ***PySEAL: A Python wrapper implementation of the SEAL homomorphic encryption library*****. arXiv. https://arxiv.org/abs/1803.01891**

13. **w3resource. (2024, October 15).** ***Python project – Encrypt and decrypt files*****. w3resource. https://www.w3resource.com/projects/python/python-file-encryption-decryption-project.php**