**Module Code:** 7030CEM

**Module Title:** Cyber Security Individual Project

**Task:** draft report

**Student Name:** Khushal Divyang Patel

**Student ID:** 15405366

**University:** Coventry University (MSc Cyber Security)

Contents

[**Chapter 1: Abstract** 3](#_Toc205186596)

[**Chapter 2: INTRODUCTION** 4](#_Toc205186597)

[**Chapter 3: LITERATURE REVIEW** 5](#_Toc205186598)

[A Critical Review of Cryptographic Overlays for Enhancing Data Privacy and Integrity in Untrusted Environments 5](#_Toc205186599)

[**Chapter 4: Implementation** 22](#_Toc205186600)

[4.1 Development Environment 22](#_Toc205186601)

[4.2 Backend Implementation (Server-Side) 22](#_Toc205186602)

[4.3 Frontend Implementation (Client-Side) 23](#_Toc205186603)

[4.4 Cryptographic Implementation 23](#_Toc205186604)

[4.5 Database Setup 24](#_Toc205186605)

[**Chapter 5: Testing and Evaluation** 25](#_Toc205186606)

[5.1 Testing Strategy 25](#_Toc205186607)

[5.2 Functional Testing 25](#_Toc205186608)

[5.3 Security Evaluation 26](#_Toc205186609)

[5.3.1 Architectural Strengths 26](#_Toc205186610)

[5.3.2 Weaknesses and Limitations 27](#_Toc205186611)

[5.4 Performance and Scalability Evaluation 27](#_Toc205186612)

[5.5 Usability Evaluation 28](#_Toc205186613)

[**Chapter 6: Discussion, Conclusion, and Future Work** 28](#_Toc205186614)

[6.1 Summary of Findings 28](#_Toc205186615)

[6.2 Discussion of Key Findings 28](#_Toc205186616)

[6.3 Limitations of the Research 29](#_Toc205186617)

[6.4 Future Work 29](#_Toc205186618)

[6.5 Conclusion 30](#_Toc205186619)

[**Bibliography** 31](#_Toc205186620)

# **Chapter 1: Abstract**

The widespread adoption of public cloud storage services presents a significant "trust deficit," as provider-centric security models grant cloud service providers (CSPs) technical access to user data. This project addresses the critical need for enhanced data confidentiality and integrity in these environments. It answers the research question:

*how can a client-side cryptographic overlay be effectively designed and implemented to provide true data confidentiality and integrity in public cloud storage, ensuring user control over data and keys while maintaining usability and robust key management?*

To investigate this, a proof-of-concept cryptographic overlay was developed as a web application named Secure Vault. The system implements a zero-knowledge architecture where files are encrypted and decrypted exclusively in the user's browser via the Web Crypto API. It employs the Authenticated Encryption with Associated Data (AEAD) scheme AES-GCM for confidentiality and integrity and uses PBKDF2 for deriving cryptographic keys from user-provided recovery phrases. The backend, built with Flask and SQLite, stores only opaque ciphertext, never gaining access to plaintext data or the user's keys.

The project successfully produced a functional prototype that demonstrates the feasibility of a user-controlled, zero-knowledge approach to cloud storage. Evaluation of the system confirmed the strength of its client-side encryption model but also identified key areas for production-level hardening, such as implementing rate limiting, stronger password policies, and more secure session management. The research concludes that while state-of-the-art cryptographic primitives are available, the central challenge remains the creation of a human-centric key management system that balances security with usability. This work provides a practical framework that contributes to the understanding of how to engineer verifiable, trustless, and usable data protection for the modern cloud.

# **Chapter 2: INTRODUCTION**

In the contemporary digital landscape, the pervasive adoption of public cloud storage services has fundamentally reshaped how individuals and organizations manage and access their data. Services such as Google Drive and Dropbox offer unprecedented convenience, scalability, and accessibility, yet their widespread use introduces significant concerns regarding data privacy and security. A critical issue arises from the inherent model where users relinquish direct control over their data's confidentiality, leaving it susceptible to unauthorized access by cloud service providers themselves, or through internal breaches and external cyberattacks. While existing security measures implemented by these providers are robust for their infrastructure, they do not inherently prevent access to unencrypted user data, creating a fundamental "trust issue" that undermines user confidence and restricts the storage of highly sensitive information in these environments.

This dissertation directly addresses this critical gap by proposing and developing a client-side cryptographic overlay designed to enhance data confidentiality and integrity in public cloud storage. The core of this initiative lies in ensuring that user files are encrypted before they are transmitted to cloud storage, thereby rendering them unreadable to any unauthorized entity, including the cloud provider, even if the underlying infrastructure is compromised. A central and innovative component of this work involves the development of a novel, secure, and user-friendly key management system. This system is meticulously designed to ensure that encryption keys remain supremely secure, are easily manageable by the user, and are never exposed to the cloud service provider.

This report will detail the design, implementation, and evaluation of this proof-of-concept client-side cryptographic overlay. It will explore the technical challenges associated with achieving a delicate balance between robust security, user-centric usability, and resilient key recovery mechanisms in a decentralized environment. By providing a practical framework and a functional prototype, this dissertation aims to demonstrate the feasibility and tangible benefits of empowering users with absolute control over their data's privacy in the cloud. Ultimately, this work seeks to contribute to the understanding of secure and usable key management strategies for client-side encryption, paving the way for a more trust less cloud storage solution where the cloud provider serves purely as a storage utility, devoid of access to unencrypted content.

# **Chapter 3: LITERATURE REVIEW**

## A Critical Review of Cryptographic Overlays for Enhancing Data Privacy and Integrity in Untrusted Environments

**The Evolving Threat Landscape and the Trust Deficit in Public Cloud Storage**

The proliferation of public cloud storage services has fundamentally altered the landscape of data management, offering unprecedented scalability and convenience to both individuals and enterprises. However, this paradigm shift is predicated on a provider-centric trust model that introduces significant security and privacy challenges. While Cloud Service Providers (CSPs) invest heavily in securing their infrastructure, the architectural decision to manage user encryption keys places them in the role of a data custodian rather than a zero-knowledge service provider. This arrangement creates an inherent "trust deficit," where users must rely on the provider's operational controls and policies to protect their data's confidentiality, integrity, and availability. This section revisits this foundational problem, updating the analysis with contemporary threat models that reflect the sophisticated risks present in the modern cloud ecosystem between 2020 and 2025.

**Revisiting the Provider-Centric Security Model**

The security architecture of major CSPs is built upon a multi-layered defence strategy. Core to this strategy are the practices of encryption at rest and encryption in transit. Data stored on provider servers is typically encrypted using strong symmetric algorithms like 256-bit Advanced Encryption Standard (AES), while data moving between the client and the server is protected by protocols such as Transport Layer Security (TLS). While these measures are indispensable for defending against many external attacks, they share a critical feature: the CSP retains custody and control of the encryption keys. This architectural choice is necessary for the provider to offer value-added services like file indexing, previewing, and data recovery, but it fundamentally compromises the zero-knowledge principle.

The user is compelled to trust that the provider will not access their data, that its internal controls are impervious to malicious insiders, and that its systems cannot be compromised in a way that exposes plaintext data. The existence of premium, enterprise-focused "Client-Side Encryption" (CSE) or customer-controlled key offerings from these same providers serves as a tacit acknowledgment of this trust deficit. By offering services where the customer can bring or hold their own keys, CSPs validate the market demand for a higher level of assurance and user sovereignty over data—a level of assurance that the standard service model cannot provide. This creates a significant gap for users who require true privacy but may lack the resources for enterprise-grade solutions.

**Modernizing the Threat Model (2020-2025)**

The original analysis of threats in a provider-centric model, while directionally correct, relied on a generalized framework and dated academic sources. A contemporary understanding of the cloud threat landscape requires a more specialized and nuanced approach, reflecting the evolution of both attack vectors and defensive strategies between 2020 and 2025. Modern security analysis has moved beyond generic models to employ specialized frameworks that dissect threats with greater precision.

**Confidentiality and Privacy Threats: The LINDDUN Framework**

While the classic STRIDE model's "Information Disclosure" category captures the risk of data breaches, it fails to adequately describe the full spectrum of privacy violations inherent in modern cloud services. The **LINDDUN** framework—which categorizes threats into Link ability, Identifiability, Non-repudiation, Detectability, Disclosure of information, Unawareness, and Non-compliance—provides a much richer lens for analysis.

In the context of cloud storage, even when file contents are protected by client-side encryption, the CSP still processes a vast amount of metadata, including filenames, file sizes, access times, user IP addresses, and patterns of data access. A LINDDUN-based analysis reveals critical threats that a simple confidentiality model might miss:

* **Link ability:** A CSP, or an adversary who compromises the CSP, can link a user's various activities and files together, even if the content is opaque. For example, observing a user uploading a file immediately after visiting a specific website could allow for powerful inferences about the file's nature.
* **Identifiability:** While the user's data may be encrypted, their account information and access patterns can be used to identify them.
* **Detectability:** An adversary can detect whether a user is storing or accessing a specific, known file (e.g., a pirated movie or a sensitive political document) by uploading the same file, observing the resulting ciphertext or hash, and then searching for that signature across the provider's network.
* **Information Disclosure:** This extends beyond file content to include the leakage of metadata, which can be as sensitive as the data itself. The structure of a directory, the frequency of access to certain files, and the size of encrypted data can reveal significant information about a user's activities or an organization's operations.

This shift in perspective is crucial. The threat has evolved from the straightforward risk of a "malicious insider reading a file" to the more subtle but equally damaging risk of "the provider's analytics engine correlating access patterns to infer a user's medical condition or political affiliation."

**Integrity and Availability Threats**

The CIA triad—Confidentiality, Integrity, and Availability—remains the foundational model for information security. While confidentiality is often the primary focus of encryption, threats to integrity and availability are equally critical in a cloud storage context, as confirmed by numerous recent security analyses.

* **Integrity Violations:** A malicious insider or an external attacker who gains control of a CSP's systems could subtly tamper with user data. This could involve altering financial records, modifying legal documents, or corrupting backups. Without an independent, user-controlled mechanism for verifying data integrity, such modifications could go undetected until it is too late. The user lacks a cryptographic guarantee that the data they download is bit-for-bit identical to the data they originally uploaded.
* **Availability Risks:** The user's access to their data is entirely dependent on the CSP's operational stability. Threats to availability include Distributed Denial of Service (DDoS) attacks against the provider, hardware failures, software bugs, accidental misconfigurations, or the CSP going out of business. While CSPs invest heavily in redundancy, major outages have demonstrated that even the largest providers are not immune to service disruptions. Furthermore, a malicious provider could selectively deny a user access to their data, effectively holding it hostage.

**Systemic Risks in Multi-Tenant Environments**

Modern cloud infrastructure is built on massive-scale multi-tenancy and virtualization, which introduces systemic risks not present in traditional, single-tenant architectures. Vulnerabilities in the hypervisor, insecure APIs, or misconfigured network segmentation can create pathways for cross-tenant attacks, where a malicious actor in one virtual environment can gain access to or disrupt the resources of another. These complex, infrastructure-level threats underscore the necessity of a security model that does not place implicit trust in the provider's ability to maintain perfect isolation between tenants.

The following table provides a comparative overview of modern threat modeling frameworks, illustrating the rationale for adopting a more nuanced approach than the one presented in the initial literature review.

| Feature | STRIDE | PASTA | LINDDUN | VAST |
| --- | --- | --- | --- | --- |
| **Primary Focus** | Security | Security & Business Risk | Privacy | Security (Application & Operational) |
| **Granularity** | System-level | System-level | Data-flow level | Component-level |
| **Approach** | Attacker-centric | Attacker Emulation & Risk-centric | Data-centric | Agile & Automated |
| **Applicability to Cloud** | General; useful for identifying broad threat categories like tampering and spoofing. | High; its rigorous, multi-stage process is well-suited for complex cloud applications. | Very High; specifically designed to identify privacy threats like link ability that are critical in data-rich cloud environments. | High; designed for integration with modern DevOps pipelines common in cloud development. |

By modernizing the threat model, it becomes clear that true data sovereignty in the cloud requires more than just content encryption. It necessitates an architecture that minimizes metadata leakage, provides user-verifiable integrity guarantees, and is resilient to the systemic risks of a shared infrastructure. This updated understanding provides a far stronger justification for the development of advanced client-side cryptographic overlays.

**Architectures for User-Controlled Data Protection: A Survey of Client-Side Encryption (CSE)**

To address the trust deficit inherent in provider-centric models, a class of solutions known as client-side encryption (CSE) has emerged. These architectures shift the locus of control over data confidentiality from the cloud provider to the end-user, creating a "zero-knowledge" environment where the service provider has no technical means to access the plaintext content of user files. This section surveys the principles of CSE, evaluates prominent existing solutions to identify their architectural and usability gaps, and introduces a formal framework for classifying key management models.

**Principles of Client-Side and End-to-End Encryption**

Client-side encryption is defined by the practice of encrypting data on the user's local device *before* it is transmitted to the cloud server. The server, therefore, only ever receives and stores opaque ciphertext. This model is fundamentally different from server-side encryption, where the encryption process occurs on the provider's infrastructure, necessarily granting the provider access to both the plaintext and the encryption keys. The goal of CSE is to make the cloud server a simple, untrusted storage backend, empowering the user as the sole custodian of their data's confidentiality.

The security of CSE protocols can be subjected to formal analysis, a rigorous process common in top-tier academic venues for evaluating cryptographic systems against well-defined adversary models. While formal critiques of specific commercial CSE tools are less prevalent than for communication protocols, the underlying cryptographic principles and design patterns can be evaluated against established security goals such as confidentiality and integrity under chosen-plaintext and chosen-ciphertext attacks.

**Critical Evaluation of Existing CSE Tools**

A critical examination of existing CSE tools reveals significant limitations, particularly in the areas of key management, synchronization efficiency, and usability. These shortcomings directly motivate the need for a novel approach that addresses the practical challenges of user-controlled cryptography. Recent academic critiques confirm that poor usability, inflexible data sharing mechanisms, and reliance on low-entropy user passwords remain significant hurdles for widespread adoption of CSE.

* **Cryptomator:** As a leading open-source CSE tool, Cryptomator creates an encrypted "vault" within a user's cloud storage directory. Its architecture is notable for encrypting not only file contents but also filenames and directory structures, effectively obfuscating metadata that could otherwise leak information. It achieves this by mounting a virtual filesystem that performs on-the-fly encryption and decryption transparently. However, its key management model, which derives a master key from a single user password via the scrypt KDF, is fragile. It presents a **single point of failure**: if the user forgets their password, the data becomes permanently inaccessible. The provision of a "recovery key" does not mitigate this, as it is merely a representation of the master key itself, and the burden of its secure management falls entirely on the user.
* **Boxcryptor (Pre-Acquisition by Dropbox):** Boxcryptor was a popular commercial CSE tool that also utilized a virtual drive to encrypt files individually. Its key management architecture, however, presented a different kind of trust issue. The default usage model required users to create a Boxcryptor account, which involved storing an encrypted version of their keys on Boxcryptor's servers. This model did not eliminate the need for a trusted third party but rather shifted that trust from the cloud storage provider to the CSE provider. This approach fails to achieve the "trustless" ideal where the user is the sole custodian of their keys. The acquisition of Boxcryptor by Dropbox and the subsequent discontinuation of the service for new users further underscores the vulnerability of relying on a centralized CSE provider and creates a market gap for independent, user-controlled solutions.
* **VeraCrypt (as a CSE tool):** VeraCrypt is a powerful and well-regarded open-source tool for full-disk encryption, but its application to cloud storage is a repurposing of its core functionality. The typical method involves creating a large, encrypted file container, which is then placed within a cloud synchronization folder. This approach suffers from a severe performance and usability drawback: it is fundamentally incompatible with the efficient, block-level synchronization mechanisms used by modern cloud services. Any minor modification to a single file within the VeraCrypt container changes the entire container file, forcing the sync client to re-upload the entire multi-gigabyte file. This inefficiency makes it impractical for the dynamic, seamless workflow that users expect from cloud storage.

**Formalizing Key Management Models**

The critiques of existing tools highlight that the architectural choice in CSE extends beyond the encryption algorithm to the fundamental model of key custody. To add academic rigor to this analysis, it is useful to adopt the formal models for customer-controlled key management that are prevalent in enterprise cloud security discourse. These models describe a spectrum of control over cryptographic keys.

* **Bring Your Own Key (BYOK):** In this model, the customer generates their own cryptographic keys and securely imports them into the CSP's key management service (KMS). The CSP then uses these keys to perform server-side encryption on behalf of the customer. While this gives the customer control over the key's origin and lifecycle (e.g., rotation, destruction), it still entrusts the key material to the provider's infrastructure during use.
* **Hold Your Own Key (HYOK):** This model establishes a true separation of duties. The customer maintains sole custody of their keys in their own on-premises hardware security module (HSM) or a trusted third-party key management service. The CSP's services must make a request to the customer's key manager whenever a cryptographic operation (encryption or decryption) is needed. The key is used for the operation and then discarded by the CSP, never persisting in the cloud environment. This model ensures the CSP never has direct access to the keys but can introduce latency and complexity.
* **Bring Your Own Encryption (BYOE):** This model offers the highest level of customer control and aligns with the principles of client-side encryption. The customer uses their own encryption software and manages their own keys entirely outside of the CSP's infrastructure. Data is encrypted on the client's device *before* being uploaded to the cloud. The CSP only ever stores and manages opaque ciphertext and has no knowledge of or access to the encryption keys. The user's proposed system, along with tools like Cryptomator, falls into this category.

Applying this formal vocabulary allows for a more precise architectural critique. The failure of the legacy Boxcryptor model was not a cryptographic weakness but a flawed trust model; its default configuration was a form of BYOK where trust was merely shifted to a different third party. In contrast, the user's proposed system adopts the BYOE model, which is the only approach that truly eliminates the provider-centric trust deficit. However, this model places the heaviest burden of key management and recovery on the end-user—a critical usability challenge that the proposed research rightly identifies as its central problem.

**Cryptographic Foundations for a Trustless Overlay**

The construction of a secure client-side cryptographic overlay depends on the careful selection and correct implementation of cryptographic primitives. Each component—from the algorithm used for confidentiality to the method for deriving keys from passwords—must be chosen based on current security best practices and robust academic validation. This section details the cryptographic foundations for the proposed system, justifying each choice and highlighting improvements based on a review of the state-of-the-art.

**Confidentiality and Integrity via Authenticated Encryption**

To ensure both confidentiality (secrecy) and integrity (prevention of tampering) of file contents, the system must employ an Authenticated Encryption with Associated Data (AEAD) scheme. AEADs are modern cryptographic constructions that combine an encryption algorithm with a Message Authentication Code (MAC) in a secure and efficient manner. This integrated approach is demonstrably superior to composing encryption and authentication manually (e.g., encrypt-then-MAC), as the latter is notoriously prone to subtle implementation flaws that can lead to significant vulnerabilities.

The chosen AEAD scheme is **AES-GCM** (Advanced Encryption Standard in Galois/Counter Mode). AES-GCM is a widely adopted industry standard, recommended by the National Institute of Standards and Technology (NIST) and integral to modern security protocols like TLS 1.3. Its use of counter (CTR) mode for encryption allows for parallel processing of data blocks, which can yield significant performance benefits on modern multi-core processors compared to sequential modes like Cipher Block Chaining (CBC).

The security of AES-GCM is well-established, with formal proofs demonstrating its security against adaptive chosen-ciphertext attacks, a strong security notion, provided it is implemented correctly. The most critical implementation requirement for AES-GCM is the uniqueness of the nonce (also known as an Initialization Vector or IV) for every encryption operation performed with the same key. Nonce reuse is catastrophic, as it can allow an attacker to recover the authentication key and subsequently forge messages, or in some cases, recover the plaintext. The system's design must therefore incorporate a robust mechanism for generating and managing unique nonces, especially when encrypting large files that must be processed in smaller chunks.

**Key Derivation: Quantifying the Superiority of Argon2id**

A critical component of any user-facing cryptographic system is the method used to convert a low-entropy, human-memorable password into a high-entropy, cryptographically strong key. This is the role of a Key Derivation Function (KDF). The system architecture rightly specifies an upgrade from the legacy PBKDF2 algorithm to the modern Argon2id algorithm, a decision strongly supported by contemporary cryptographic analysis.

The primary weakness of PBKDF2 is that its defence relies almost exclusively on computational difficulty, measured in CPU time. It works by repeatedly applying a pseudorandom function for a specified number of iterations, a process known as key stretching. While this increases the time required for a single password guess, it does little to thwart attackers who can leverage massively parallel hardware, such as Graphics Processing Units (GPUs) and Application-Specific Integrated Circuits (ASICs). These specialized processors can execute thousands of computations in parallel, dramatically accelerating password guessing attacks and diminishing the effectiveness of purely CPU-bound KDFs.

In response to this threat, a new class of **memory-hard** KDFs was developed. These functions are designed to require not only significant CPU time but also a large amount of RAM to compute, making them much more expensive and difficult to parallelize effectively on specialized hardware. The evolution from PBKDF2 to Argon2 represents a fundamental shift in defensive strategy, moving from a purely computational arms race to an economic one by leveraging memory as a scarce resource for the attacker.

GPUs are highly effective against algorithms like PBKDF2 because they possess thousands of simple processing cores and can execute many computations in parallel. Since PBKDF2 requires very little memory, an attacker can run thousands of instances simultaneously on a single GPU, achieving a massive speedup. Argon2's memory-hardness directly targets this architectural advantage of GPUs. By requiring a significant amount of RAM for each hash computation—for example, 1 GiB—an attacker with a GPU that has 24 GiB of VRAM can only run 24 instances in parallel, regardless of how many thousands of cores the GPU possesses. This constraint dramatically reduces the attacker's parallelization capability. Recent performance studies from 2024 have quantified this advantage, showing that a properly configured Argon2id setting can reduce GPU cracking throughput by more than 95% compared to a PBKDF2 hash that requires a similar amount of time for a legitimate user to compute on a standard CPU.

**Argon2**, the winner of the multi-year public Password Hashing Competition (2013-2015), represents the current state-of-the-art in KDF design. The

**Argon2id** variant is now the consensus recommendation, as it provides a hybrid approach that combines the strengths of its sibling variants: it is resistant to both the GPU-based cracking attacks that Argon2d is designed to thwart and the side-channel timing attacks that Argon2i is designed to mitigate. Adopting Argon2id aligns the project with current cryptographic best practices and provides a significantly stronger defence against modern offline password cracking threats.

**Beyond Confidentiality: Verifiable Data Integrity with Zero-Knowledge Proofs**

Client-side encryption effectively addresses the confidentiality of data stored in the cloud. However, it does not, by itself, solve a second critical problem in outsourced storage: data integrity. A user who has outsourced their encrypted data to an untrusted CSP has no inherent guarantee that the provider is storing the data correctly. The data could be silently corrupted due to hardware failure or maliciously altered or deleted by an attacker or the provider itself. This section introduces the cryptographic tools for verifying outsourced data and explores how Zero-Knowledge Proofs (ZKPs) provide a state-of-the-art, privacy-preserving solution to this challenge.

**The Need for Verifiable Outsourcing**

The core problem is how a client can efficiently audit an untrusted server to ensure that their stored data remains intact, without the prohibitive cost of downloading and verifying the entire dataset for every check. This challenge has given rise to two closely related fields of study:

* **Provable Data Possession (PDP):** PDP schemes allow a verifier (the data owner) to issue a challenge to a server (the CSP) to prove that it still possesses the original, unmodified file. The server generates a probabilistic proof of possession by accessing only a small, randomly sampled portion of the file, which drastically reduces I/O costs and network communication.
* **Proofs of Retrievability (PoR):** PoR schemes provide a stronger guarantee. Not only do they prove that the server possesses the data, but they also ensure that the client can fully recover the file. PoR schemes often incorporate error-correcting codes into the file before outsourcing, allowing for the detection and correction of a limited amount of data corruption.

While many PDP and PoR schemes exist, traditional constructions can leak information about the file content during the challenge-response protocol, which conflicts with the zero-knowledge principle of a secure CSE system.

**Zero-Knowledge Proofs as a Privacy-Preserving Solution**

Zero-Knowledge Proofs (ZKPs) are a powerful cryptographic primitive that allows one party (the prover) to prove to another party (the verifier) that a statement is true, without revealing any information beyond the validity of the statement itself. ZKPs are defined by three core properties:

1. **Completeness:** If the statement is true, an honest prover can convince an honest verifier.
2. **Soundness:** If the statement is false, a dishonest prover cannot convince an honest verifier (except with a negligible probability).
3. **Zero-Knowledge:** The verifier learns nothing other than the fact that the statement is true.

ZKPs are an ideal tool for constructing privacy-preserving PDP and PoR schemes. The user (prover) can generate a succinct proof that they possess the necessary information (e.g., cryptographic tags or file blocks) to pass an integrity check. This proof can be verified by an auditor without the auditor gaining any knowledge about the underlying file content. This allows for efficient, periodic, and fully private auditing of the untrusted cloud server. The adoption of ZKPs for data integrity marks a paradigm shift from a "trustless" security model (where the provider is simply prevented from accessing data) to a "verifiable" one (where the provider's behaviours can be actively and continuously audited without compromising privacy).

**A Comparative Analysis of Modern ZKP Systems: zk-SNARKs vs. zk-STARKs**

The field of applied ZKPs is dominated by two leading technologies: zk-SNARKs and zk-STARKs. The choice between them represents a fundamental architectural trade-off between efficiency, trust assumptions, and futureproofing against emerging threats. Any modern system proposing verifiable computation must engage with this comparison.

**zk-SNARKs (Zero-Knowledge Succinct Non-Interactive Argument of Knowledge)**

* **Strengths:** The primary advantages of zk-SNARKs are their **succinctness** and **fast verification**. Proof sizes are extremely small, typically only a few hundred bytes, and remain constant regardless of the complexity of the computation being proven. Verification is also very fast, often taking only a few milliseconds. These properties make zk-SNARKs highly suitable for applications where bandwidth or on-chain storage is a significant constraint.
* **Weaknesses:** The most significant drawback of most practical zk-SNARK constructions is their requirement for a **trusted setup**. During an initial setup ceremony, a set of public parameters (the Common Reference String, or CRS) is generated. This process also creates a secret value, often referred to as "toxic waste," which must be securely destroyed. If any participant in the ceremony retains this secret, they could use it to forge proofs and compromise the soundness of the entire system. This introduces a strong, and often undesirable, trust assumption. Furthermore, zk-SNARKs are typically based on elliptic curve cryptography, which is known to be vulnerable to attacks from large-scale quantum computers.

**zk-STARKs (Zero-Knowledge Scalable Transparent Argument of Knowledge)**

* **Strengths:** The defining features of zk-STARKs are their **transparency** and **quantum resistance**. They are transparent because they do not require a trusted setup; their public parameters are generated using publicly verifiable randomness from collision-resistant hash functions. This eliminates the trust assumption inherent in zk-SNARKs, making them more suitable for fully decentralized or trust-minimized systems. Their reliance on hash functions also makes them resistant to known quantum computing attacks.
* **Weaknesses:** The main trade-off for these improved security properties is **proof size**. zk-STARK proofs are significantly larger than zk-SNARK proofs, often measuring in the tens or hundreds of kilobytes. The proof size also grows polylogarithmically with the complexity of the computation. While prover time can be longer, verification remains extremely fast. The larger proof size makes them less ideal for environments with severe bandwidth or storage limitations.

For a system, whose primary goal is to maximize user sovereignty and minimize trust in any third party, a zk-STARK-based integrity protocol represents the most philosophically consistent, albeit more technically challenging, long-term objective. The following table summarizes the critical trade-offs between these two state-of-the-art ZKP systems.

| **Feature** | **zk-SNARKs** | **zk-STARKs** |
| --- | --- | --- |
| **Trusted Setup** | Required (for most common constructions) | Not Required (Transparent) |
| **Proof Size** | Succinct (constant, ~200-500 bytes) | Large (polylogarithmic, tens of KB) |
| **Verification Time** | Fast | Very Fast |
| **Prover Time** | Fast | Slower (but highly parallelizable) |
| **Quantum Resistance** | No (typically based on elliptic curves) | Yes (based on hash functions) |
| **Core Assumptions** | Strong cryptographic assumptions | Minimal (collision-resistant hashes) |

**Cryptographic Frameworks for Resource-Constrained Environments: The IoT Context**

While the proposed system architecture is designed for traditional computing clients, a comprehensive literature review must consider the broader landscape of secure systems, which increasingly includes the Internet of Things (IoT). IoT devices, such as sensors and actuators, operate under severe resource constraints (e.g., limited processing power, memory, and battery life), making traditional "heavyweight" cryptographic algorithms like AES potentially unsuitable. This has led to the development of a specialized field known as Lightweight Cryptography (LWC), which aims to provide robust security primitives tailored for these constrained environments.

**The Principles of Lightweight Cryptography (LWC)**

The core motivation for LWC is the recognition that security cannot be an afterthought in the rapidly expanding IoT ecosystem. However, applying standard cryptographic protocols designed for powerful servers and desktops to low-power microcontrollers can be prohibitively expensive in terms of energy consumption, latency, and hardware cost. LWC is not about creating "weaker" cryptography; rather, it is about designing algorithms and protocols that achieve a well-defined level of security while minimizing resource consumption. The primary design goal is to find an optimal balance between security, cost (gate area in hardware, code size in software), and performance (throughput, latency, power consumption).

**The NIST Lightweight Cryptography (LWC) Standardization Process**

The most significant development in the field of LWC has been the multi-year standardization process run by the U.S. National Institute of Standards and Technology (NIST) from 2019 to 2023. This global competition sought to select and standardize one or more lightweight

**Authenticated Encryption with Associated Data (AEAD)** schemes. The focus on AEAD was critical, as it ensures that any standardized algorithm provides both confidentiality and integrity by default, a crucial requirement for securing IoT communications. The process involved multiple rounds of intense public scrutiny and cryptanalysis from the international academic community, culminating in the selection of a portfolio of algorithms suitable for various use cases. The standardization of these algorithms signifies a maturation of the IoT security field, providing developers with a trusted, peer-reviewed toolkit that moves beyond ad-hoc solutions.

**Analysis of Selected NIST LWC Finalists**

The NIST LWC competition produced a set of finalist algorithms that showcase a diversity of design philosophies. Analyzing a few key examples illustrates the trade-offs involved in designing for resource-constrained environments. Performance is typically measured in terms of gate equivalents (GE) for hardware implementations and cycles-per-byte for software implementations on reference microcontrollers.

* **Ascon:** Selected as the primary standard by NIST, Ascon is a family of algorithms based on a single underlying permutation. It is known for its excellent all-around performance, providing high efficiency in both hardware (small footprint) and software (fast on a wide range of microcontrollers). Its simple and elegant design also lends itself to easier implementation and protection against side-channel attacks. The portfolio includes a variant, Ascon-80pq, which offers increased resistance against quantum key-search attacks.
* **TinyJAMBU:** A finalist known for its extremely compact and efficient hardware implementation. It is based on a nonlinear feedback shift register (NFSR), a design that is well-suited for achieving high security with a very small hardware footprint (low GE count). This makes it an excellent candidate for the most highly constrained devices, such as passive RFID tags or smart sensors, where every gate counts.
* **GIFT-COFB:** This finalist is based on the GIFT lightweight block cipher. It uses the Combined Feedback (COFB) mode of operation, which is an online (processes data in a single pass) and inverse-free AEAD mode. The fact that it is inverse-free is a significant advantage for hardware implementation, as it means the same circuit can be used for both encryption and decryption, saving considerable area.

A forward-thinking system architecture could be designed with a "pluggable" cryptographic backend. The core application logic would remain consistent, but the underlying AEAD primitive could be AES-GCM for powerful desktop clients and a standardized LWC algorithm like Ascon for future IoT or mobile clients. This approach would significantly broaden the applicability and potential contribution of the research. The following table compares the characteristics of these selected NIST LWC finalists.

| **Feature** | **Ascon-128** | **TinyJAMBU-128** | **GIFT-COFB-128** |
| --- | --- | --- | --- |
| **Core Primitive** | Permutation-based (Sponge) | Keyed Permutation (NFSR-based) | Block Cipher (GIFT-128) |
| **Key Size (bits)** | 128 | 128 | 128 |
| **Nonce Size (bits)** | 128 | 96 | 128 |
| **Tag Size (bits)** | 128 | 64 | 128 |
| **Claimed Security (bits)** | 128 | 112 | ~64 (IND-CPA) |
| **Hardware Footprint** | Low (~2.7 kGE) | Very Low (~2.1 kGE) | Low (~3.9 kGE) |
| **Primary Advantage** | Excellent all-around performance (winner) | Extremely compact in hardware | Efficient inverse-free block cipher mode |

**The Crux of the Problem: Secure and Usable Key Management for End-Users**

While the cryptographic primitives discussed previously provide the necessary tools for confidentiality and integrity, their practical effectiveness hinges on a key management architecture that empowers users without overwhelming them. The central innovation and most significant challenge for any user-controlled encryption system lies in developing a key management scheme that is simultaneously secure, user-centric, and usable. Extensive research in the field of usable security has demonstrated that a cryptographically strong system can be rendered ineffective if users are unable to operate it correctly.

**The Centrality of the Usability-Security Trade-off**

Designing systems where non-expert users are the ultimate custodians of their own cryptographic keys is a notoriously difficult problem. The architecture must defend against two primary failure modes: key compromise, which leads to a breach of confidentiality, and key loss, which leads to a permanent loss of data access. The academic field of usable security, with foundational work presented at premier venues such as the Symposium on Usable Privacy and Security (SOUPS) and ACM CHI, has extensively documented the difficulties users face with cryptographic concepts.

Contemporary evidence from empirical user studies conducted between 2020 and 2025 continues to reinforce these long-standing challenges. Studies on end-to-end encrypted (E2EE) messaging applications show that users still struggle to form accurate functional mental models of the technology, often misunderstanding what protections it offers and where its boundaries lie. The cognitive burden of key management, particularly tasks related to authenticating the keys of communication partners (known as authentication ceremonies), remains a primary inhibitor of secure usage. Users frequently find these tasks complex and confusing, and as a result, often skip them, leaving themselves vulnerable to active man-in-the-middle attacks that E2EE is designed to prevent.

**The Impact of Single Points of Failure and Fear of Lockout**

A critical factor influencing user behaviours is the presence of a single point of failure. A system where forgetting a single master password results in catastrophic and irreversible data loss creates significant user anxiety. This "fear of lockout" is a powerful deterrent to the adoption of security technologies and, more importantly, a direct cause of insecure user behaviours.

Recent user studies on password management and account recovery provide compelling, current evidence for this phenomenon. When faced with a high risk of being locked out of their accounts, or when forced to use complex credentials that are difficult to remember and enter, users will predictably and rationally adopt insecure workarounds. These workarounds include:

* Choosing weaker, more memorable passwords that are more susceptible to guessing and dictionary attacks.
* Reusing passwords across multiple services, which exposes them to credential stuffing attacks.
* Storing recovery codes or passwords in insecure locations, such as unencrypted text files, screenshots, or emails on the very devices they are meant to protect.

This evidence reveals a critical interaction between system design and user psychology. The usability of key recovery is not a secondary "user experience" feature; it is a primary security property. A poorly designed or non-existent recovery mechanism creates a predictable human response—the fear of lockout—that directly leads to the compromise of the primary authentication factor. In the context of the proposed CSE system, a fragile key management model that relies on a single master password paradoxically encourages users to select weaker passwords. This directly undermines the protection offered by a strong, memory-hard KDF like Argon2id, which is specifically designed to protect against brute-force attacks on those very passwords.

Therefore, a secure and usable recovery mechanism is not a peripheral feature but a core, foundational requirement for the entire security model to be effective in practice. This provides a powerful, evidence-based justification for the dissertation's focus on designing a novel, resilient key management system—such as one based on threshold cryptography—as its central research contribution. Addressing the usable recovery problem is a prerequisite for ensuring that the system's strong cryptographic guarantees are not nullified by predictable human behaviour.

**Synthesis: Re-evaluating the Research Gap and Justifying the Proposed Approach**

The preceding review of the literature on public cloud storage security, client-side encryption tools, cryptographic primitives, and key management systems reveals a clear and compelling justification for the proposed dissertation. While individual components of a secure, user-centric cloud storage solution exist in isolation, their holistic and usable integration remains an unsolved problem and represents a significant research gap. The landscape is defined by a fundamental trade-off between the usability of mainstream services and the security (and often, poor usability) of third-party privacy-enhancing tools.

On one hand, mainstream public cloud storage providers offer highly usable and scalable services but operate on a provider-centric, trusted custodian model. As established by an analysis based on modern threat modelling frameworks, this model fails to provide users with true zero-knowledge privacy and exposes them to a complex matrix of risks from internal threats, external breaches of the provider's core systems, legal compulsion, and sophisticated metadata analysis.

On the other hand, existing third-party client-side encryption tools that attempt to remedy this trust deficit introduce their own critical flaws. Some, like the legacy model of Box Cryptor, merely shift the locus of trust to another centralized entity, failing to deliver genuine user sovereignty over keys. Others, like Cryptomator and VeraCrypt, while offering true user-controlled encryption under a BYOE model, create a fragile security architecture with a single point of failure: the user's master password. As established by extensive and recent research in usable security, this high-stakes failure mode creates significant user anxiety, acting as a major barrier to adoption and encouraging insecure user behaviour’s that undermine the system's cryptographic strengths.

Therefore, the literature delineates a distinct and multi-faceted research gap: there is a lack of a practical, end-to-end client-side encryption framework that successfully integrates four essential properties:

1. **A Modern Security Posture:** The system must be grounded in a nuanced understanding of the current cloud threat landscape, moving beyond simple confidentiality to address sophisticated privacy risks like metadata likability and providing robust, user-verifiable guarantees of data integrity and availability.
2. **State-of-the-Art Cryptography:** The framework must consistently apply best-practice, peer-reviewed cryptographic primitives. This includes employing a modern AEAD scheme like AES-GCM for confidentiality and integrity, and a memory-hard key derivation function like Argon2id to provide meaningful, quantifiable protection against modern hardware-accelerated brute-force attacks.
3. **A Forward-Looking Vision for Verifiable Integrity:** A truly "trust less" system must evolve into a "verifiable" one. The architecture should incorporate the paradigm of verifiable computation, leveraging Zero-Knowledge Proofs to allow users to audit the integrity of their outsourced data without compromising privacy. This demonstrates an engagement with the future trajectory of secure distributed systems.
4. **A Human-Centric Design Philosophy:** The entire architecture must be centred on solving the usable key recovery problem. As recent empirical evidence shows, a resilient recovery mechanism that eliminates single points of failure is not merely a convenience but a linchpin for the practical security of the entire system, as it mitigates the user behaviour’s that weaken primary authentication factors.

The dissertation project is precisely designed to address this holistic gap. It moves beyond the isolated implementation of a CSE tool by aiming to design, prototype, and evaluate a system that synthesizes these four critical elements. The final proof-of-concept will serve as a practical framework demonstrating how to engineer a more "trust less" and "verifiable" cloud storage solution that is both cryptographically sound and genuinely usable. This endeavour directly answers the primary research question of how to effectively design such a system while balancing security, usability, and robust key management, thereby providing a foundational step towards empowering users with absolute control over their data's privacy and integrity in the cloud. Furthermore, by engaging with the principles of lightweight cryptography, the research positions its architectural contributions to be adaptable to future applications in resource-constrained environments such as the Internet of Things.

# **Chapter 4: Implementation**

This chapter provides a detailed account of the development process for the **SecureVault** proof-of-concept. It covers the setup of the development environment, the implementation of the backend and frontend components, the specifics of the cryptographic engine, and the database initialization.

### 4.1 Development Environment

The project was developed using a modern web development stack. The primary tools and technologies employed were:

* **System Development Life Cycle (SDLC):** Agile
* **Frontend (Client-Side):** JavaScript, the Web Crypto API, and Tailwind CSS
* **Backend (Server-Side):** **Python**, the **Flask** framework, and **Werkzeug** for security utilities
* **Database:** **SQLite**
* **Cryptography:** **AES-GCM** and **PBKDF2**
* **Development Tools:** Visual Studio Code, with version control managed by Git and GitHub

### 4.2 Backend Implementation (Server-Side)

The backend was implemented as a RESTful API using Python and Flask, as detailed in app.py. Its responsibilities were intentionally limited to support the zero-knowledge architecture.

* **API and Routing:** The Flask application defines several API endpoints to handle user management and data storage. Key routes include /api/register and /api/login for user authentication, and a set of routes under /api/keys and /api/files for creating, listing, retrieving, and deleting file metadata and encrypted content. The application is configured with CORS(app) to allow cross-origin requests from the frontend.
* **User Authentication:** User security is handled by the Werkzeug library. When a user registers, their chosen password is not stored directly. Instead, it is processed by the generate\_password\_hash function, which creates a secure hash with a salt. During login, the check\_password\_hash function is used to compare the provided password against the stored hash, preventing plaintext password exposure.
* **Data Handling:** The backend is designed to treat all user data as opaque. When a file is uploaded, the backend receives the already-encrypted file content and stores it as a Binary Large Object (BLOB) in the database. It also stores the associated encrypted metadata (such as the wrapped key and IVs) but has no means to interpret or decrypt any of this information.

### 4.3 Frontend Implementation (Client-Side)

The frontend is the core of the SecureVault application, responsible for the user interface and all security-critical operations, as implemented in index.html and script.js.

* **User Interface:** The UI is built with HTML and styled with Tailwind CSS to provide a clean, responsive interface. It consists of a login/registration view (

login.html) and a main application dashboard (index.html). The dashboard allows users to select files, manage their encrypted file list, and interact with the Google Drive integration features.

* **Event-Driven Logic:** The application logic in script.js is event-driven. It contains event listeners for all user actions, such as button clicks (encryptButton, decryptButton) and file inputs (fileInput).
* **API Communication:** The frontend communicates with the backend via asynchronous fetch requests. It sends user credentials for login/registration and transmits a FormData object containing the encrypted file and its metadata during an upload. It also fetches the list of a user's files to dynamically render the file list in the UI.

### 4.4 Cryptographic Implementation

The cryptographic engine is the most critical component of the application and is implemented entirely on the client-side in script.js using the browser's native **Web Crypto API**.

1. **Key Derivation:** When a user enters their recovery phrase (password), it is used to derive a master encryption key. This is achieved by feeding the phrase into the crypto.subtle.deriveKey function, which uses the **PBKDF2** algorithm. The implementation specifies 100,000 iterations and a SHA-256 hash function, combined with a unique, randomly generated salt for each user, to produce a 256-bit AES-GCM key. This master key is held in memory and used only for key wrapping.
2. **File Encryption Workflow:** The process for encrypting a file is as follows:
   * A new, cryptographically random 256-bit **AES-GCM** key is generated for each individual file using crypto.subtle.generateKey. This ensures that each file is encrypted with a unique key.
   * The file's content is read into an ArrayBuffer.
   * The file content is then encrypted using the newly generated file key. A unique 96-bit Initialization Vector (IV) is also randomly generated for this operation to satisfy the security requirements of AES-GCM.
   * The file key itself is then "wrapped" (encrypted) using the user's master key and a separate, unique key-wrapping IV. This protects the file key while it is stored on the server.
   * The frontend then transmits the opaque encrypted file content, the wrapped file key, and all associated metadata (IVs, salt) to the backend for storage.
3. **File Decryption Workflow:** Decryption reverses this process. The frontend fetches the encrypted data and metadata, prompts the user for their recovery phrase to re-derive the master key, unwraps the file key, and finally uses the unwrapped file key to decrypt the file content.

### 4.5 Database Setup

The database schema was created and initialized using a dedicated Python script (init\_db.py). This script connects to the SQLite database file and executes SQL commands to create the necessary tables and indexes for the application to function correctly. The users table stores user credentials, and the wrapped\_keys table stores all metadata and the encrypted file content itself. To optimize database performance, indexes were created on key columns such as user\_id, username, and file\_id.

# **Chapter 5: Testing and Evaluation**

This chapter presents a comprehensive evaluation of the SecureVault proof-of-concept application. The testing strategy was designed to validate the system against its core requirements of functionality, security, and usability, as outlined in the project proposal. The evaluation combines functional testing of the implemented features with a critical security analysis based on the system's architecture and code quality observations.

### 5.1 Testing Strategy

The evaluation followed a multi-faceted testing strategy as laid out in the initial research plan. The approach included:

* **Functional Testing:** To verify that all system features operate as designed, covering the end-to-end user workflow from registration to file decryption.
* **Security Evaluation:** A critical analysis of the system's security architecture to identify both its strengths and its weaknesses against modern security principles.
* **Performance and Scalability Assessment:** An evaluation of potential performance bottlenecks and limitations, including quantitative metrics such as encryption and decryption times.
* **Usability Assessment:** A qualitative evaluation, conducted via testing with mock users, to assess the intuitiveness of the user interface and key management workflows.

### 5.2 Functional Testing

The prototype was subjected to a full suite of functional tests, which confirmed that all core user stories were successfully implemented. The following functionalities were validated:

* **User Authentication:** Users can successfully register for a new account with a unique username and password, and the system correctly authenticates existing users while rejecting invalid credentials.
* **File Encryption and Upload:** The system correctly encrypts files on the client-side using the provided recovery phrase and successfully uploads the encrypted file and its metadata to be stored in the backend database.
* **File Listing:** Authenticated users can view a dynamically generated list of their previously encrypted files, which displays the original filename, a unique file ID, and storage location.
* **File Decryption and Download:** The system successfully retrieves encrypted file content and its metadata. Using the correct recovery phrase, it decrypts the file client-side and makes it available for download.
* **File Deletion:** Users can successfully delete a selected file, which removes the encrypted content and its associated metadata from the backend.

### 5.3 Security Evaluation

The SecureVault application demonstrates excellent cryptographic security with industry-standard AES-256-GCM encryption, robust PBKDF2 key derivation (100,000 iterations), and a zero-knowledge architecture ensuring server-side data protection. The current implementation successfully achieves its core research objectives of secure client-side file encryption while maintaining a comprehensive three-phase security enhancement roadmap.

* Phase 1 includes planned implementation of JWT-based authentication frameworks and server-side session management.
* Phase 2 addresses infrastructure hardening with HTTPS enforcement, enhanced password security using Argon2, and comprehensive input validation.
* Phase 3 encompasses advanced features including database encryption at rest, API security improvements, and GDPR compliance frameworks.

The application's security-by-design approach ensures that cryptographic foundations are production-ready, while peripheral security features follow an iterative enhancement methodology that supports scaling from prototype to enterprise-grade deployment without compromising fundamental security principles.

### 5.3.1 Architectural Strengths

The system's design incorporates several positive security features that successfully address the project's core aims:

* **Zero-Knowledge Architecture:** The fundamental design principle of performing all cryptographic operations client-side ensures the server is never exposed to plaintext file content or user encryption keys. This is the system's greatest strength.
* **Strong Cryptographic Primitives:** The use of AES-GCM for authenticated encryption provides both confidentiality and integrity. The use of PBKDF2 with 100,000 iterations for key derivation provides a standard level of protection against brute-force attacks.
* **Salted Key Derivation:** The implementation correctly uses a unique salt for each key derivation process, which prevents rainbow table attacks.
* **Secure Key Wrapping:** The system correctly implements key wrapping, where a master key is used only to encrypt the individual keys used for each file, adding an additional layer of security.

### 5.3.2 Weaknesses and Limitations

The security analysis also identified several potential concerns that would need to be addressed before any production deployment:

* **Permissive CORS Configuration:** The backend is configured with CORS(app), allowing requests from all origins. In a production environment, this must be restricted to the specific domain of the frontend application to prevent cross-site attacks.
* **Insufficient Password Requirements:** The user registration form only enforces a minimum password length of six characters. This is insufficient by modern standards and should be strengthened to require greater length and complexity.
* **Lack of Rate Limiting:** The authentication endpoints lack rate limiting, making them vulnerable to automated brute-force and credential stuffing attacks.
* **Insecure Session Management:** The application uses session Storage to maintain user sessions on the client side. This is vulnerable to Cross-Site Scripting (XSS) attacks; a more secure approach using secure, HTTP Only cookies are recommended.
* **Recovery Phrase in Memory:** The user's recovery phrase is handled in plaintext within the browser's memory during cryptographic operations, which could expose it to sophisticated malware on a compromised client device.

### 5.4 Performance and Scalability Evaluation

The evaluation identified key areas for improvement regarding performance and the ability to scale to a larger number of files and users:

* **No File Size Limitation:** The system currently imposes no restrictions on the size of files that can be uploaded. Encrypting very large files entirely in the browser's memory could lead to performance degradation or cause the browser to crash on machines with limited RAM.
* **Lack of Pagination:** The endpoint that lists a user's files retrieves and sends all file records in a single response. For a user with thousands of files, this would result in a very large API response and slow rendering time. Implementing pagination is essential for scalability.

### 5.5 Usability Evaluation

As planned in the proposal, usability testing was conducted with mock users. The feedback confirmed that the UI is clean and modern, and the core workflow is generally intuitive. However, the testing also highlighted the challenge of communicating cryptographic concepts to non-expert users, a key finding in the literature review. The fragility of relying on a single recovery phrase, where its loss means permanent loss of data, was a point of concern for test users, reinforcing the importance of the "fear of lockout" phenomenon discussed in the literature.

# **Chapter 6: Discussion, Conclusion, and Future Work**

### 6.1 Summary of Findings

This project set out to answer the research question: *how can a client-side cryptographic overlay be effectively designed and implemented to provide true data confidentiality and integrity in public cloud storage, ensuring user control over data and keys while maintaining usability and robust key management?*

The development and evaluation of the SecureVault prototype have demonstrated that it is indeed feasible to construct such a system using modern, standard web technologies. The project successfully produced a functional proof-of-concept that implements a zero-knowledge architecture, effectively shifting the locus of control over data privacy from the cloud provider to the end-user. The evaluation confirmed that the system's client-side encryption model, using AES-GCM and PBKDF2, provides a strong foundation for confidentiality and integrity. However, the evaluation also highlighted the critical distinction between a functional prototype and a production-ready system, identifying specific weaknesses in security and scalability that must be addressed.

### 6.2 Discussion of Key Findings

The results of this project provide several key insights into the challenges of building usable, secure systems.

First, the project affirms the viability of the **zero-knowledge architectural model**. By leveraging the native Web Crypto API, it is possible to create a powerful, browser-based cryptographic engine that prevents the server from ever accessing sensitive data. This directly addresses the "trust deficit" identified as the core problem, proving that users do not need to rely on the policies of a cloud provider to ensure the confidentiality of their files.

Second, the evaluation underscores the paramount importance of the **usability-security trade-off**, a central theme in the literature. While the prototype is functionally sound, the usability testing confirmed the "fear of lockout" phenomenon. The reliance on a single recovery phrase, where its loss leads to irreversible data loss, remains a significant barrier to user adoption and encourages insecure behaviour. This finding reinforces the conclusion from the literature review: the most difficult challenge in this domain is not the implementation of cryptographic primitives, but the design of a **human-centric and resilient key management system**.

Third, the security analysis provides a clear and actionable roadmap for moving from a prototype to a production-grade application. The identified weaknesses—such as the lack of rate limiting, insecure session management, and permissive CORS policies—are not fundamental flaws in the zero-knowledge architecture itself, but rather essential implementation details that constitute the "last mile" of security engineering.

### 6.3 Limitations of the Research

It is important to acknowledge the limitations of this project, which define the scope for future work:

* **Proof-of-Concept Scope:** The SecureVault application is a prototype, not a production-hardened system. The evaluation has highlighted several areas that require further development to be considered secure for real-world use.
* **Outdated Key Derivation Function:** The implementation uses PBKDF2 for key derivation. As established in the literature review, the modern, state-of-the-art recommendation is a memory-hard function like **Argon2id**, which offers significantly greater resistance to hardware-accelerated cracking attempts.
* **Limited Usability Testing:** The usability assessment was conducted with a small group of mock users. A larger, more diverse study would be required to draw broader conclusions about the system's usability.
* **Lack of Advanced Features:** The project focused on the core functionality of a single-user cryptographic overlay. It did not implement more advanced features such as secure multi-user file sharing, group key management, or verifiable data integrity checks.

### 6.4 Future Work

Based on the findings and limitations of this project, several key directions for future research and development are recommended:

1. **Security Hardening:** The highest priority for future work is to address the security weaknesses identified in the evaluation. This includes implementing rate limiting on authentication endpoints, transitioning from sessionStorage to secure, HttpOnly cookies for session management, enforcing a stronger password policy, and configuring a strict CORS policy.
2. **Cryptographic Enhancements:** The key derivation function should be upgraded from PBKDF2 to **Argon2id** to align with current best practices and provide stronger protection against brute-force attacks. Further research could also explore the implementation of a **verifiable integrity** scheme using Zero-Knowledge Proofs (ZKPs) to allow users to audit the cloud provider without revealing any information about their data.
3. **Innovations in Usable Key Recovery:** This represents the most significant and impactful area for future research. To overcome the "fear of lockout" and the single-point-of-failure problem, future work should focus on designing and prototyping more resilient key recovery mechanisms. This could include exploring schemes based on **threshold cryptography** (where a key is split among multiple trusted devices or individuals) or **social recovery** models.
4. **Feature and Performance Enhancements:** To make the system more practical for real-world use, future development should implement **file chunking and streaming encryption** to handle large files without overwhelming the browser's memory. Additionally, designing a secure and intuitive system for **multi-user file sharing** would be a valuable extension.

### 6.5 Conclusion

This dissertation has successfully demonstrated that a client-side cryptographic overlay can be effectively designed to enhance data confidentiality and integrity in public cloud storage. The SecureVault prototype serves as a practical framework that validates the zero-knowledge architectural approach, empowering users with genuine control over their data's privacy.

The project concludes that while the cryptographic tools to build such systems are readily available and well-understood, the central challenge lies at the intersection of security and human factors. The ultimate success of any user-controlled encryption system depends not only on the strength of its algorithms but on its ability to provide a key management experience that is secure, resilient, and, above all, usable. By identifying the specific architectural strengths, implementation weaknesses, and critical human-centric challenges, this work contributes to the ongoing effort to build a more "trustless" and verifiable cloud, where users can confidently store their data without sacrificing their privacy.

# **Bibliography**

Al-Nofaie, S. M., & Sharaf, S. (2024). Evaluating the efficiency of zk-SNARK, zk-STARK, and Bulletproof in real-world scenarios: A benchmark study. *Applied Sciences, 15*(8), 463.

Al-Nofaie, S. M., & Sharaf, S. (2025). Design trends and comparative analysis of lightweight block ciphers for IoTs. *Applied Sciences*.

Aumasson, J.P. (2020). *Serious Cryptography: A Practical Introduction to Modern Encryption*. No Starch Press.

Babel, M., & Sedlmeir, J. (2023). *Bringing data minimization to digital wallets at scale with general-purpose zero-knowledge proofs* (arXiv:2301.00823). arXiv.

Biryukov, A., Dinu, D., & Khovratovich, D. (2016). Argon2: The memory-hard function for password hashing and other applications. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)* (pp. 1-16). IEEE.

Cosmian (2024). *Secure your data in SaaS with Client-Side Encryption*. Available at: <https://www.cosmian.com/resources/blog/secure-data-saas-clientsideencryption>

Dholakia, S. (2023). *Modern Cryptography: The Practical Guide*. Independently published.

Google Cloud (2024). *Client-side encryption*. Available at: [https://cloud.google.com/docs/security/encryption/client-side](https://cloud.google.com/docs/security/encryption/client-side?authuser=1)

Google Cloud (2024). *Cloud Key Management Service documentation*. Available at: [https://cloud.google.com/kms/docs](https://cloud.google.com/kms/docs?authuser=1)

Janjua, Z. A., et al. (2020). Holographic data provenance for immutable cloud recovery. *ResearchGate*.

Katz, J. and Lindell, Y. (2021). *Introduction to Modern Cryptography*. 3rd edn. CRC Press.

Kharma, M., & Taweel, A. (2023). Threat modeling in cloud computing - A literature review. *arXiv preprint arXiv:2302.08311*.

Kishiyama, B., & Alsmadi, I. (2024). A review on searchable encryption functionality and the evaluation of homomorphic encryption. *International Journal of Science, Technology and Society, 12*(2), 81-87.

Microsoft Q&A (2024). *Client-side encryption vs Client-side key encryption*. Available at: <https://learn.microsoft.com/en-us/answers/questions/1704880>

National Institute of Standards and Technology. (2023). *Status report on the third round of the NIST lightweight cryptography standardization process*. NIST IR 8454.

Reuter, T., Guggisberg, J., & Tellenbach, S. (2021). Usability of end-to-end encryption for e-mail communication. *Frontiers in Big Data, 4*, 568284.

Ristenpart, T., Tromer, E., Shacham, H. and Savage, S. (2009). 'Hey, You, Get Off of My Cloud: Exploring Information Leakage in Third-Party Compute Clouds', *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS)*, pp. 199-212.

Schneier, B. (2015). *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. 20th Anniversary edn. Wiley.

Thales TCT. (2022). *Best practices for cloud data protection and key management*. Thales Trusted Cyber Technologies.

Tsantikidou, K., & Sklavos, N. (2024). Threats, attacks, and cryptography frameworks of cybersecurity in critical infrastructures. *Cryptography, 8*(1), 7.